RÉSUMÉ THÉORIQUE – FILIÈRE DÉVELOPPEMENT DIGITAL

Option – Applications mobiles

M209 –Acquérir les bases de développement Android

50 heures

SOMMAIRE

1. DÉCOUVRIR LE DÉVELOPPEMENT MOBILE

- Introduire les différentes plateformes de développement mobile
- Découvrir l'écosystème de développement mobile

2. MAÎTRISER LA PLATEFORME ANDROID

- Préparer l'environnement de développement
- S'initier à Android Studio
- Maîtriser la génération d'une application mobile

3. CODER EN JAVA

- Coder une application en JAVA
- Réaliser des interfaces graphiques simples
- Maîtriser Git

MODALITÉS PÉDAGOGIQUES

1

LE GUIDE DE SOUTIEN

Il contient le résumé théorique et le manuel des travaux pratiques

2

LA VERSION PDF

Une version PDF est mise en ligne sur l'espace apprenant et formateur de la plateforme WebForce Life

3

DES CONTENUS TÉLÉCHARGEABLES

Les fiches de résumés ou des exercices sont téléchargeables sur WebForce Life

4

DU CONTENU INTERACTIF

Vous disposez de contenus interactifs sous forme d'exercices et de cours à utiliser sur WebForce Life

5

DES RESSOURCES EN LIGNES

Les ressources sont consultables en synchrone et en asynchrone pour s'adapter au rythme de l'apprentissage

PARTIE 1

Découvrir le développement mobile

Dans ce module, vous allez :

- Explorer les bases du développement mobile
- Acquérir des connaissances sur les approches de développement mobile
- Découvrir l’écosystème de développement mobile

8 heures

CHAPITRE n° 1

Introduire les différentes plateformes de développement mobile

Ce que vous allez apprendre dans ce chapitre :

- Les systèmes d'exploitation mobile
- Les approches de développement mobile
- La composition des architectures Android et iOS

5 heures

CHAPITRE n° 1

Introduire les différentes plateformes de développement mobile

1. **Présentation des systèmes d'exploitation mobile**
2. Approche de développement mobile
3. Composition de l'architecture d'Android et iOS

01 – Introduire les différentes plateformes de développement mobile

Présentation des systèmes d'exploitation mobile

Objectif

- Ce chapitre permet d'acquérir les compétences théoriques et pratiques sur les **systèmes d'exploitation mobiles** et les **approches de développement mobile** afin de pouvoir implémenter des applications sur des dispositifs **mobiles.**

- Ce chapitre aborde dans un 1er lieu quelques **généralités**, et présente ensuite **les systèmes d'exploitation mobile** suivi des **approches de développement mobile.**

PARTIE 1

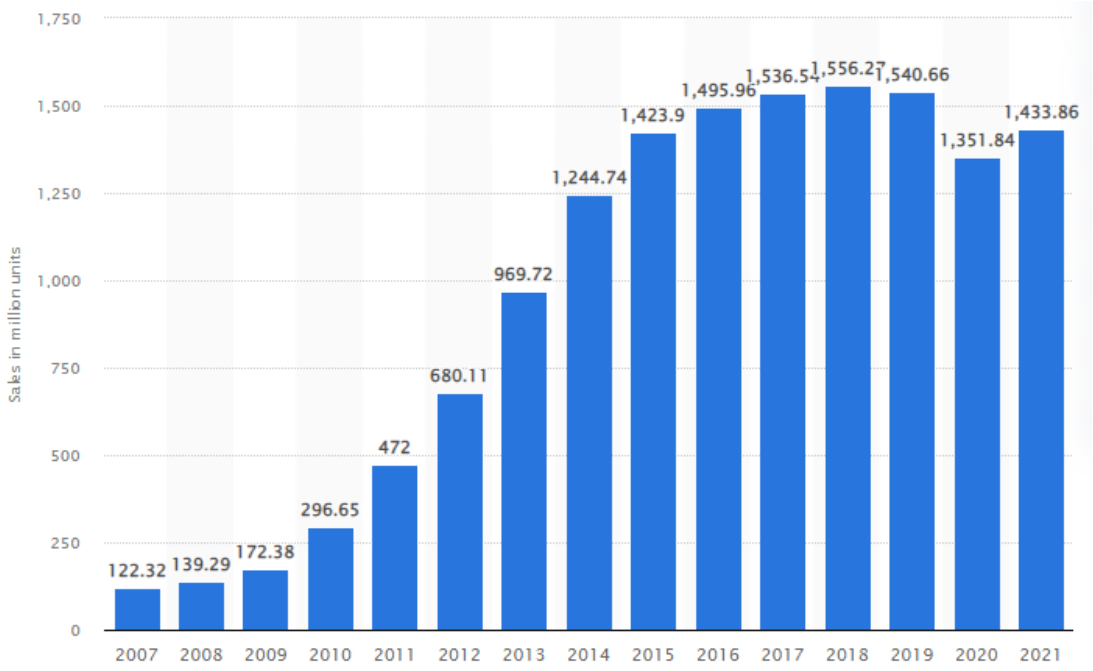
Téléphonie mobile : Smartphone

- Un smartphone, ordiphone ou téléphone intelligent, est un téléphone mobile disposant aussi des fonctions d'un assistant numérique personnel (PDA) ;
- La saisie des données se fait par le biais d'un écran tactile ou d'un clavier ;
- Il fournit des fonctionnalités basiques comme : l'agenda, le calendrier, la navigation sur le web, la consultation de courrier électronique, de messagerie instantanée, le GPS, etc.

PARTIE 1

Nombre de smartphones vendus aux utilisateurs finaux dans le monde de 2007 à 2021

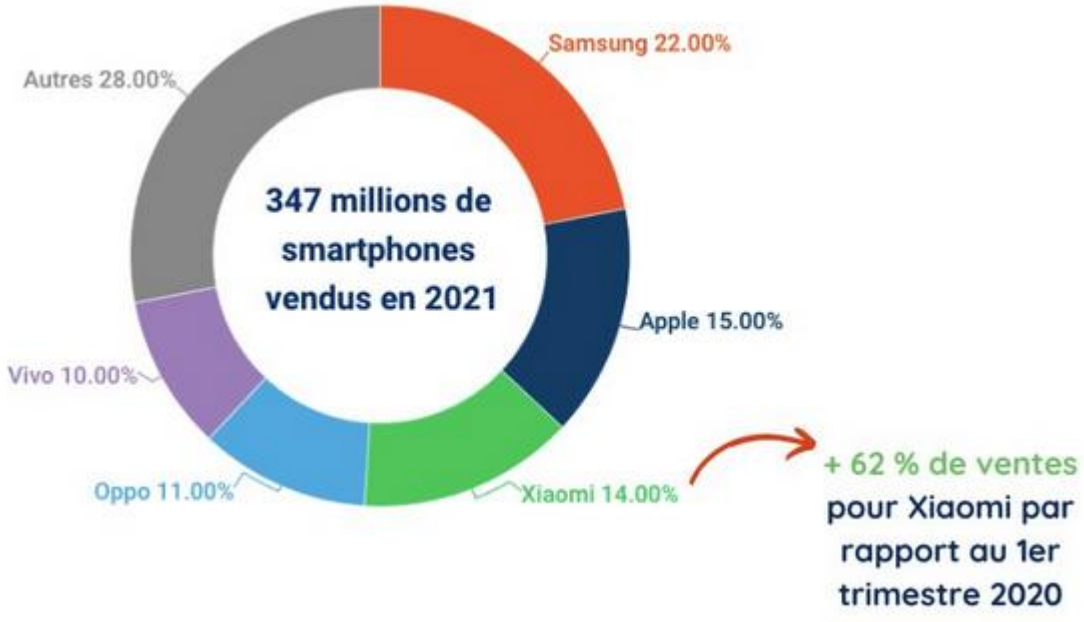
En 2021, les vendeurs de smartphones ont vendu environ 1,43 milliard de smartphones dans le monde.



© Statista 2022

Top 5 des marques de téléphone les plus vendues en 2021

Presque 350 millions de smartphones ont été vendus en 2021 dans le monde, selon l'analyste Canalys. Des ventes en hausse de 27 % par rapport à l'année 2020, marquée négativement par la Covid-19.

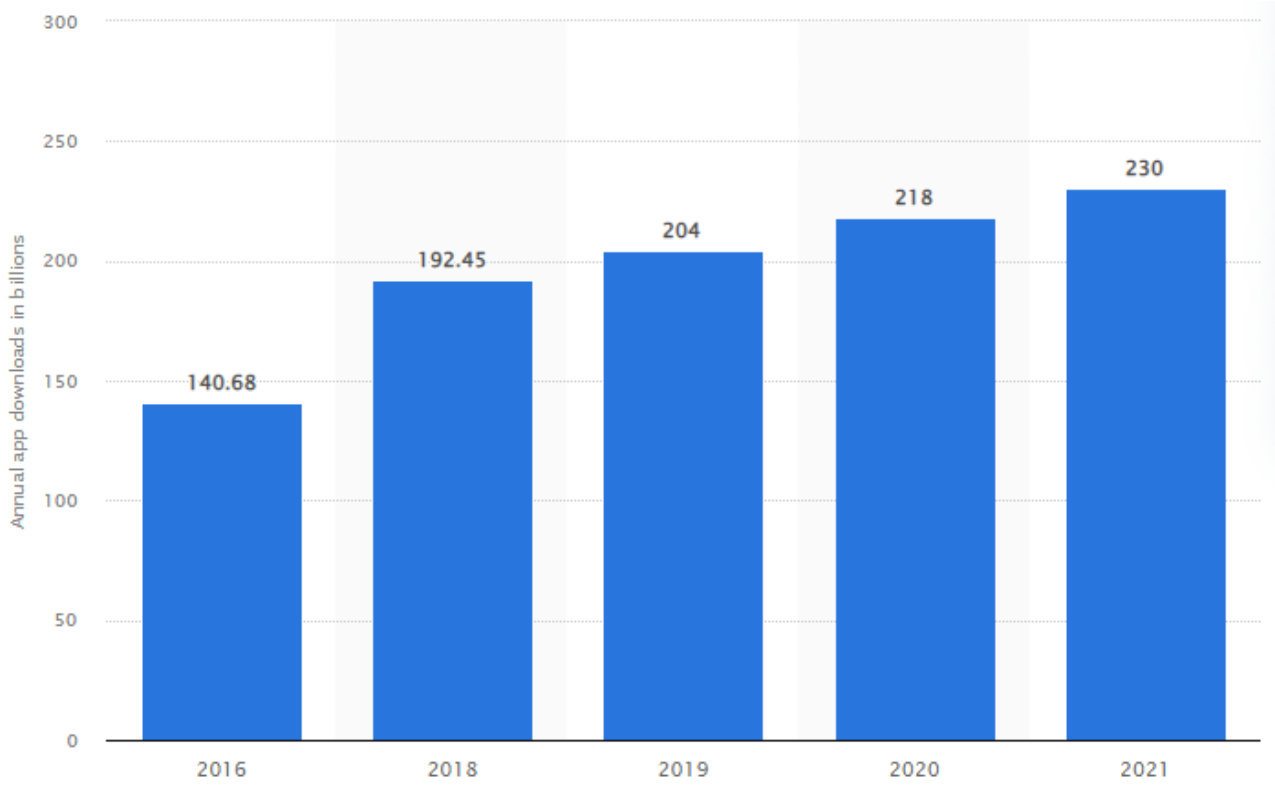


Réf : reassurez-moi

PARTIE 1

Nombre de téléchargements d'applications mobiles dans le monde de 2016 à 2021(en milliards)

Le nombre de téléchargements d'applications mobiles dans le monde n'a cessé d'augmenter à partir de 2016, pour dépasser les 200 milliards en 2019. Au cours de la dernière année mesurée, les consommateurs ont téléchargé 230 milliards d'apps mobiles sur leurs appareils connectés, soit une augmentation de plus de 63 % par rapport aux 140,7 milliards de téléchargements d'apps en 2016.



© Statista 2022

PARTIE 1

Applications mobiles

Une application mobile s'exécute sur un support matériel mobile (smartphones, tablettes, iPads) :

- **Ressources limitées**
  - Batterie (énergie), interface graphique, CPU, périphériques d'IO, …
- **Supports physiques (matériels) très divers**
  - De très élémentaire à très évolué (à présenter par la suite).
- **Utilisation ubiquitaire**
  - Ubiquité géographique
  - Ubiquité des utilisateurs

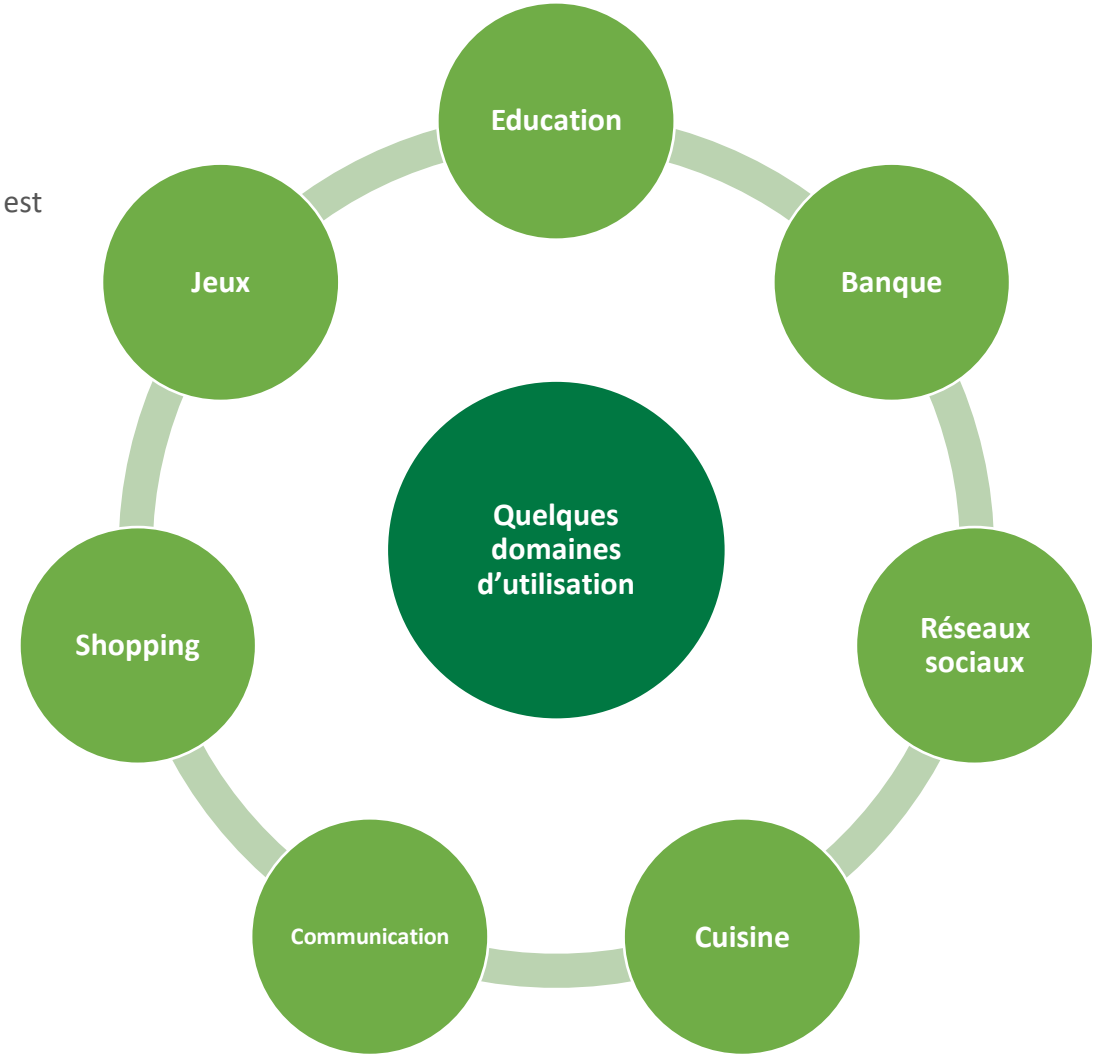
Remarques

- **L'informatique ubiquitaire** combine les objets du monde réel à ceux du monde virtuel et préente la capacité de localiser des objets et des personnes mais aussi de reconnaître la voix, les gestes et les mouvements.

PARTIE 1

Domaines d'utilisations des applications mobiles

- L'importance des applications mobiles dans la vie quotidienne.
- L'importance des téléphones mobiles dans notre vie et nos activités quotidiennes est indéniablement sans fin.



Logiciel open source

- Un logiciel libre (open source) est un logiciel informatique dont le code source est librement **accessible**, **modifiable** et **redistribuable**. Le développement collaboratif et le partage des compétences informatiques sont généralement associés aux communautés de développeurs Open Source.
- L'une des principales différences, opposant les logiciels libres aux logiciels propriétaires, est que le code source de ces derniers est "fermé". En effet, l'accès au code source est impossible ou très limité. Cela signifie que les utilisateurs paient une licence juste pour utiliser le logiciel, tel qu'il est, puisque le code source reste inaccessible.
- **Par exemple**, comparons deux systèmes d'exploitation professionnels que vous connaissez bien : Microsoft Windows est un logiciel propriétaire, tandis que RedHat Enterprise Linux est un logiciel libre. Tous deux sont très populaires et largement utilisés dans les entreprises de nos jours.

Logiciel open source

L'organisation à but non lucratif Open Source Initiative a défini 10 principes à respecter pour qu'un logiciel informatique soit qualifié d'Open Source :

1. La distribution gratuite, c'est-à-dire que tout le monde peut vendre ou distribuer le logiciel : aucune redevance ne doit être versée à l'auteur ;
2. Le code source doit être accessible, distribuable et lisible ;
3. Les travaux dérivés et les modifications sont autorisés et doivent être distribués selon les mêmes termes que la licence de la version originale du logiciel ;
4. Intégrité du code source de l'auteur : la distribution de code source modifié peut être restreinte, uniquement si des correctifs sont distribués avec le code source, afin de modifier le programme ;
5. Pas de discrimination contre les individus ou les groupes ;
6. Pas de discrimination à l'égard des domaines d'activité ;
7. L'ensemble des droits appliqués au logiciel sont également applicables à tous ses utilisateurs ;
8. La licence ne doit pas être spécifique à un produit : le logiciel reste libre, même s'il est séparé de son logiciel de distribution ;
9. La licence ne doit pas fixer de restrictions sur les autres logiciels distribués avec le même logiciel sous licence ;
10. La licence doit être neutre du point de vue technologique.

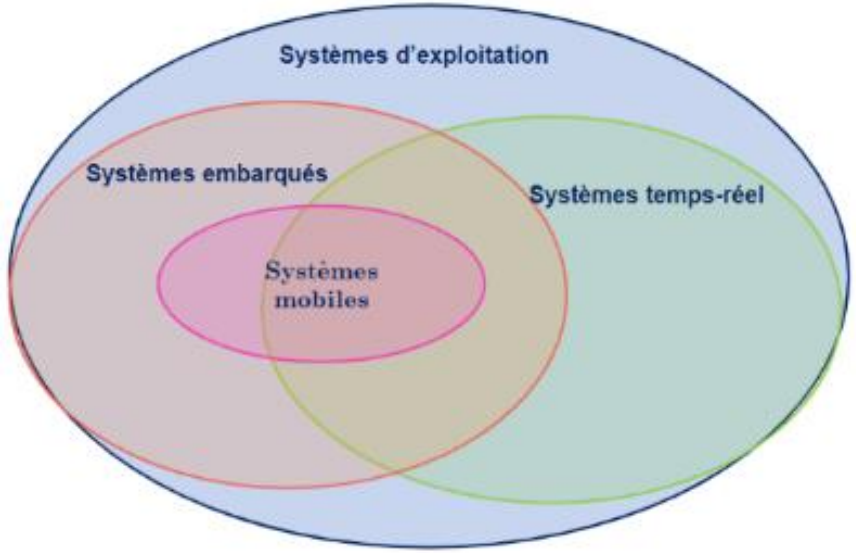
Système d'exploitation mobile

Un **système d'exploitation mobile** est un **système d'exploitation** qui permet **d'exécuter d'autres logiciels d'application** sur **des appareils mobiles**. Il s'agit du même type de logiciel que les célèbres systèmes d'exploitation pour ordinateurs tels que Linux et Windows, mais ils sont désormais **plus légers** et **plus simples** dans une certaine mesure.

- Un système d'exploitation mobile est un **système d'exploitation embarqué** conçu pour fonctionner sur **un appareil mobile**. Ce type de système d'exploitation se concentre entre autres sur la gestion de la connectivité sans fil et celle des différents types d'interface.
- Les systèmes d'exploitation que l'on trouve sur les smartphones comprennent Symbian OS, iPhone OS, BlackBerry de RIM, Windows Mobile, Palm WebOS, Android et Maemo. Android, WebOS et Maemo sont tous dérivés de Linux. L'iPhone OS est issu de BSD et NeXTSTEP, qui sont liés à Unix.

Remarques

- Une brève présentation de ses systèmes sera abordée dans les diapositives qui suivent, et dans la suite de ce cours nous allons nous focaliser sur les deux systèmes d'exploitation mobile les plus utilisés, à savoir Android et iOS.



Réf : Sghaier- Systèmes d'exploitation pour mobiles

PARTIE 1

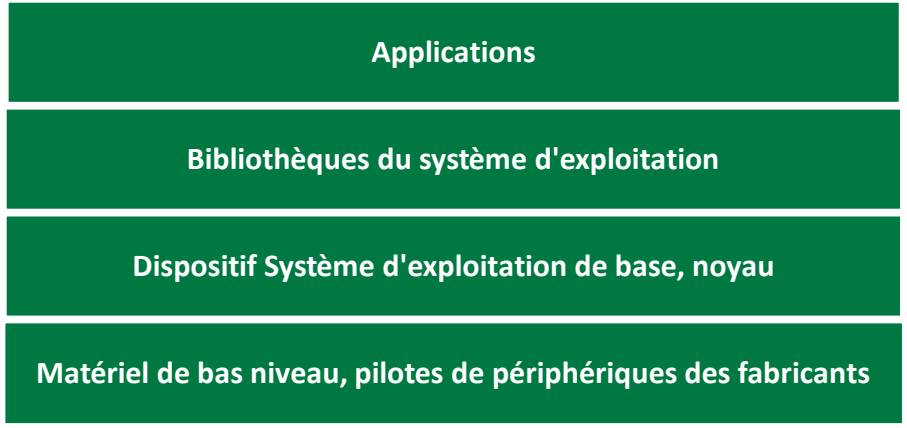
Système d'exploitation mobile

Un système d'exploitation mobile, c'est l'ensemble des programmes responsables de :

- la gestion des opérations (processus, ordonnancement, E/S, fichiers, multimédias...) ;
- le contrôle ;
- la coordination ;
- l'utilisation du matériel ;
- le partage des ressources d'un dispositif mobile entre divers programmes tournant sur ce dispositif.

Un système d'exploitation mobile est une plateforme logicielle sur laquelle d'autres programmes appelés programmes d'application peuvent fonctionner sur des appareils mobiles tels que PDA, téléphones cellulaires, smartphones et etc.

Architecture générale



Réf : Sghaier- Systèmes d'exploitation pour mobiles

Contraintes des systèmes d'exploitation mobiles

Les dispositifs mobiles ont des contraintes et des restrictions sur leur caractéristique physique telles que :

- la taille de l'écran ;
- la mémoire ;
- la puissance de traitement, etc ;
- la faible disponibilité de batterie ;
- la quantité limitée de capacités informatiques et de communication.

Ainsi, ils ont besoin de différents types de systèmes d'exploitation en fonction des capacités qu'ils supportent. Par exemple, un OS PDA est différent d'un système d'exploitation de Smartphone.

Un **système d'exploitation mobile** est un logiciel embarqué responsable de la gestion des opérations, du contrôle, de la coordination de l'utilisation du matériel entre les différents programmes d'application et le partage des ressources d'un dispositif mobile.

Réf : Sghaier- Systèmes d'exploitation pour mobiles

Plateformes populaires de l'OS mobile

- **Système d'exploitation Android** : Le système d'exploitation Android est le système d'exploitation le plus populaire aujourd'hui. Il s'agit d'un système d'exploitation mobile basé sur le noyau Linux et les logiciels libres. Le système d'exploitation Android a été développé par Google. Le premier appareil Android a été lancé en 2008.
- **Bada (Samsung Electronics)** : Bada est un système d'exploitation mobile de Samsung qui a été lancé en 2010. Le Samsung wave a été le premier mobile à utiliser le système d'exploitation Bada. Le système d'exploitation Bada offre de nombreuses fonctionnalités mobiles, telles que les graphiques en 3D, l'installation d'applications et le toucher multipoint.
- **BlackBerry OS :** Le système d'exploitation BlackBerry est un système d'exploitation mobile développé par Research In Motion (RIM). Ce système d'exploitation a été conçu spécifiquement pour les appareils portatifs BlackBerry. Ce système d'exploitation est avantageux pour les utilisateurs professionnels car il permet la synchronisation avec Microsoft Exchange, Novell GroupWise, Lotus Domino et d'autres logiciels professionnels lorsqu'il est utilisé avec le BlackBerry Enterprise Server.Traduit avec www.DeepL.com/Translator (version gratuite).
- **iPhone OS / iOS :** L'iOS a été développé par la société Apple pour être utilisé sur ses appareils. Le système d'exploitation iOS est le système d'exploitation le plus populaire aujourd'hui. C'est un système d'exploitation très sûr. Le système d'exploitation iOS n'est pas disponible pour d'autres mobiles.

Plateformes populaires de l'OS mobile

- **Symbian OS :** Le système d'exploitation Symbian est un système d'exploitation mobile qui offre un haut niveau d'intégration avec la communication. Le système d'exploitation Symbian est basé sur le langage java. Il combine un intergiciel de communication sans fil et une fonctionnalité de gestion des informations personnelles (PIM). Le système d'exploitation Symbian a été développé par Symbian Ltd en 1998 pour l'utilisation des téléphones mobiles. À l'époque, Nokia a été la première entreprise à commercialiser le système d'exploitation Symbian sur son téléphone mobile.
- **Windows Mobile OS** : Le système d'exploitation Windows Mobile OS est un système d'exploitation mobile qui a été développé par Microsoft. Il a été conçu pour les PC de poche et les mobiles intelligents.
- **Harmony OS :** Le système d'exploitation Harmony est le dernier système d'exploitation mobile qui a été développé par Huawei pour l'utilisation de ses appareils. Il est conçu principalement pour les appareils IoT.
- **Palm OS :** Le système d'exploitation Palm est un système d'exploitation mobile qui a été développé par Palm Ltd pour être utilisé sur des assistants numériques personnels (PAD). Il a été introduit en 1996. Palm OS est également connu sous le nom de Garnet OS.
- **WebOS (Palm/HP)** : Le WebOS est un système d'exploitation mobile qui a été développé par Palm. Il est basé sur le noyau Linux. HP utilise ce système d'exploitation dans ses mobiles et ses tablettes tactiles.

Open Handset Alliance™

- Est une coalition qui a vu le jour fin 2007, qui regroupe 84 entreprises technologiques et mobiles.
- A pour objectif de créer et de promouvoir le système Android comme système ouvert et gratuit dans le monde du mobile.
    - Google est l'acteur majeur.
- Adresse web : http://www.openhandsetallinace.com

Présentation d'Android

- Système d'exploitation mobile basé sur le noyau Linux ;
- Il a été développé par **l'Open Handset Alliance**, qui était dirigée par Google et d'autres entreprises ;
- Le système d'exploitation Android est basé sur Linux et peut être programmé en C/C++, mais la plupart des applications sont développées en Java (Java accède aux bibliothèques C via JNI, abréviation de Java Native Interface) ;
- Interface utilisateur pour les écrans tactiles ;
- Utilisé sur plus de 80% de tous les smartphones ;
- Dispositifs tels que les montres, les téléviseurs, et les voitures ;
- Plus de 2 millions des applications Android dans Google Play Store ;
- Hautement personnalisable pour les périphériques / par les vendeurs ;
- Open source.

Présentation de iOS

- Système d'exploitation mobile développé par Apple Inc ;
- Publié en 2007 pour l'iPhone, l'iPod Touch et l'Apple TV ;
- iOS est la version mobile d'Apple du système d'exploitation OS X utilisé dans les ordinateurs Apple ;
- Contrairement aux autres grands systèmes d'exploitation, iOS gère le dispositif matériel et fournit les technologies nécessaires à la création d'applications sur la plateforme. Quelques applications système par défaut sont livrées avec les appareils, telles que Mail, Calendar, Calculator, Phone, Safari, et ainsi de suite, qui sont généralement utilisées par les utilisateurs.
- Il n'est pas possible d'exécuter iOS et Mac OS X sur un autre matériel que celui d'Apple, et il est restreint d'utiliser iOS sur un autre appareil mobile que celui d'Apple pour des raisons de sécurité et de commerce.

Comparaison des principaux systèmes d'exploitation pour les appareils mobiles

	Android	iOS	Windows Phone
Editer par	Google	Apple	Microsoft
Environnement de développement	Android Studio	XCode	Visual Studio
Langages de programmation	JAVA, KOTLIN	Objective-C, Swift	C#, VB.net
Interface graphique	XML	CocoaTouch	XAML
Fichier exécutable	.apk	.app	.xap
Applications sur	Google Play	Apple-iTunes	Windows Store
Machine virtuelle	Dalvik VM / Android Run Time	Non	CLR
Développement sur	Multiplateforme	Mac OS X	Windows
Open source	Oui	Non	Non

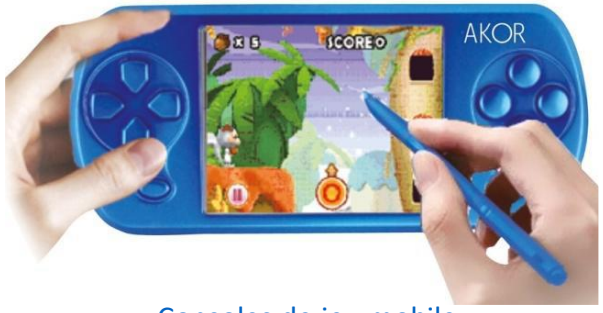
Dispositifs mobiles

PDA

Smartphone

Lecteurs multimédias

Tablette



Consoles de jeu mobile

Smartwatch

PARTIE 1

QCM

1. **Un système d'exploitation (OS) construit exclusivement pour les appareils mobiles tels que les smartphones, les tablettes, les PDA, etc. Il est similaire à un système d'exploitation standard mais il est relativement simple et léger.**

a. Système d'exploitation Windows
b. Système d'exploitation Linux
c. Système d'exploitation Macintosh
d. Système d'exploitation mobile

QCM

1. **Un système d'exploitation (OS) construit exclusivement pour les appareils mobiles tels que les smartphones, les tablettes, les PDA, etc. Il est similaire à un système d'exploitation standard mais il est relativement simple et léger.**

a. Système d'exploitation Windows
b. Système d'exploitation Linux
c. Système d'exploitation Macintosh
d. Système d'exploitation mobile

QCM

2. Il est créé par Google, et est l'un des systèmes d'exploitation mobile les plus couramment installés sur les appareils mobiles, avec le soutien de divers fabricants d'appareils. Il s'agit d'un système d'exploitation open source, ce qui signifie que les développeurs ont accès à du matériel non verrouillé pour développer de nouveaux programmes.

a. Android
b. Windows
c. Linux
d. Xamarin

QCM

2. **Il est créé par Google, et est l'un des systèmes d'exploitation mobile les plus couramment installés sur les appareils mobiles, avec le soutien de divers fabricants d'appareils. Il s'agit d'un système d'exploitation open source, ce qui signifie que les développeurs ont accès à du matériel non verrouillé pour développer de nouveaux programmes.**

   a. **Android**
   b. Windows
   c. Linux
   d. Xamarin

QCM

3. Il est développé par Apple et distribué exclusivement pour le matériel Apple (iPhone, iPad, etc.). L'une de ses caractéristiques bien connues est sa gestuelle multi-touch.

a. iOS (iPhone OS)
b. Windows OS
c. Android OS
d. Blackberry OS

QCM

3. **Il est développé par Apple et distribué exclusivement pour le matériel Apple (iPhone, iPad, etc.). L'une de ses caractéristiques bien connues est sa gestuelle multi-touch.**

   a. **iOS (iPhone OS)**
   b. Windows OS
   c. Android OS
   d. Blackberry OS

QCM

4. Il est développé par Microsoft Corporation. Parmi ses caractéristiques, citons les applications qui se mettent à jour en temps réel et la prise en charge complète des produits Microsoft tels que MS Office.

a. iOS (iPhone OS)
b. Windows OS
c. Android OS
d. Blackberry OS

QCM

4. Il est développé par Microsoft Corporation. Parmi ses caractéristiques, citons les applications qui se mettent à jour en temps réel et la prise en charge complète des produits Microsoft tels que MS Office.

a. iOS (iPhone OS)
b. Windows OS
c. Android OS
d. Blackberry OS

QCM

5. **Il s'agit d'un système d'exploitation abandonné, utilisé en dernier lieu par Nokia pour ses smartphones, car ils représentent le marché le moins important des systèmes d'exploitation mobile.**

a. Blackberry OS
b. Windows OS
c. Symbian OS
d. Chrome OS

QCM

5. Il s'agit d'un système d'exploitation abandonné, utilisé en dernier lieu par Nokia pour ses smartphones, car ils représentent le marché le moins important des systèmes d'exploitation mobile.

a. Blackberry OS
b. Windows OS
c. Symbian OS
d. Chrome OS

CHAPITRE n° 1

Introduire les différentes plateformes de développement mobile

1. Présentation des systèmes d'exploitation mobile
2. **Approche de développement mobile**
3. Composition de l'architecture d'Android et iOS

Types des approches de développement mobile

Il existe quatre approches pour le développement des applications mobiles :

- **Développement d'applications natives**
- **Développement d'applications hybrides**
- **Développement d'applications multiplateformes**
- **Application web progressive**

Types des approches de développement mobile

Développement d'applications natives

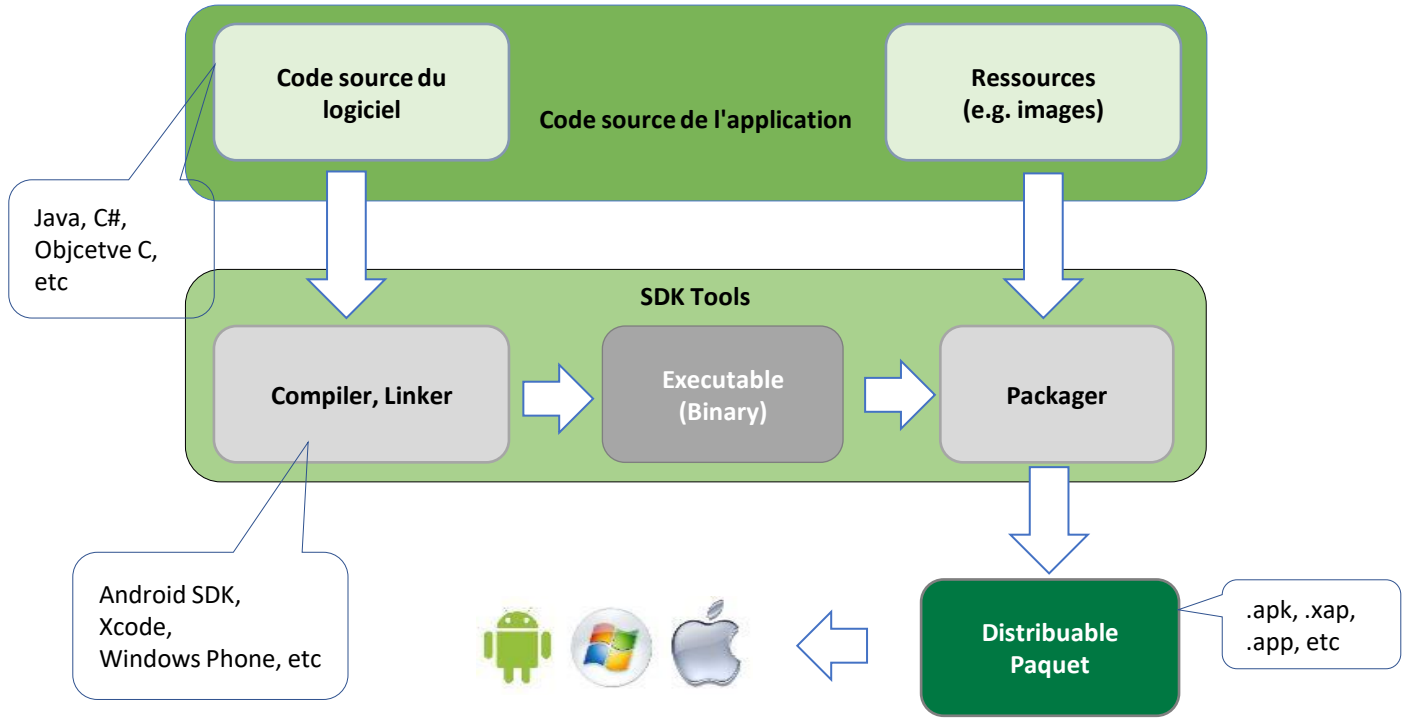
- Les applications natives sont des applications spécifiquement **développées pour un système d'exploitation.**
- Les systèmes d'exploitation les plus connus pour le mobile sont iOS et Android.
- Si vous souhaitez développer une application native compatible avec iOS et Android, il faudra **développer deux applications complètement différentes :**
    - une première pour iOS, en langage Swift ou Objective-C ;
    - une seconde pour Android, en langage Kotlin ou Java.

PARTIE 1

Types des approches de développement mobile

Approche native : Architecture

- Développement spécifique à chaque plateforme ;
- Nécessite un SDK + connaissance d'un langage spécifique ;
- Fonctionnement en mode online et offline ;
- Utilisation de toutes les fonctionnalités du mobiles (GPS, voix, notifications, contacts...).



Ref : 10.14569/IJACSA.2017.080215

Types des approches de développement mobile

Approche native : Avantages et Inconvénients

Avantages	Inconvénients
▪ Multitude d'API pour accéder à toutes les fonctionnalités des périphériques mobiles comme la caméra, les capteurs, l'accès au réseau, GPS, stockage de fichiers, base de données, SMS et e-mail.	▪ Applications natives sont plus difficiles à développer et nécessitent un haut niveau d'expérience.
▪ Meilleures performances par rapport aux applications web, aux applications hybride et aux applications multiplateformes.	▪ Elles doivent être développées séparément pour chaque plateforme, ce qui augmente les temps de développement, le coût et les efforts.
• Apparence native, interface fluide et conviviale.	▪ Restrictions et les coûts associés au développement et le déploiement de certaines plateformes (e.g. la licence de développement Apple et l'approbation d'Apple pour distribuer les applications à la boutique iTunes Apps).

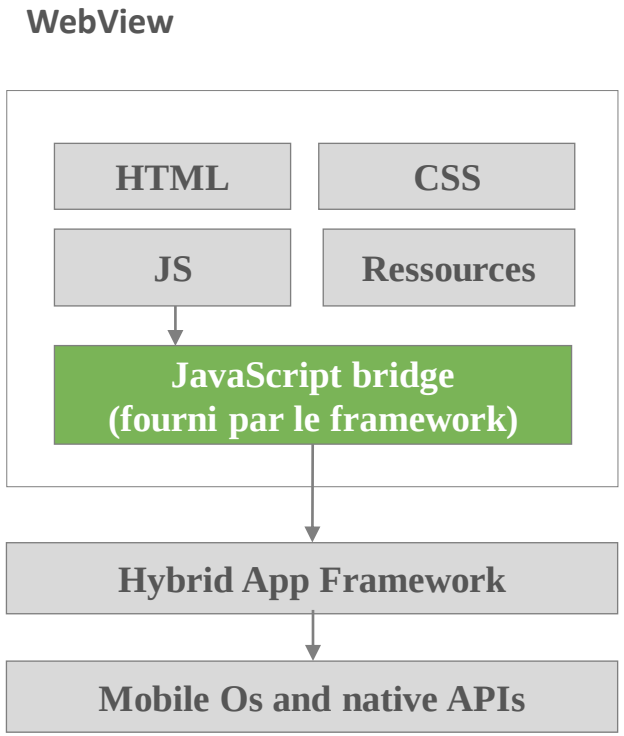
Types des approches de développement mobile

Développement d'applications hybrides

- Une application mobile hybride est une application dont **le code source permet de générer deux applications** : l'une pour Android et l'autre pour iOS.
- Elle s'appuie pour cela sur **des technologies « Web »** telles que le Javascript, HTML ou encore CSS.
- Son contenu s'exécute dans **un « conteneur » natif, une webview** (c'est-à-dire une page internet), grâce à un Framework (Cordova, Capacitor), pour pouvoir être utilisée comme une application native.
- Il s'agit donc, comme toute application Web, d'un code « interprété » qui nécessite d'être traduit par la machine à chaque exécution, ce qui réduit la performance de l'application.
- Parmi les plus connus, on retrouve PhoneGap et Ionic.

Types des approches de développement mobile

Approche hybride : Architecture



Ref : 10.14569/IJACSA.2017.080215

- Encapsulation d'une WebApp dans une MobileApp.
- Ce modèle de développement reste un compromis. (Il existe de nombreuses façons d'attaquer une WebView WebKit ; cependant, le contrôle qui peut être exploité ne permet généralement pas d'interagir avec le code Java. Il s'agit donc d'un vecteur d'attaque très intéressant.
- Utilisation d'un langage interprété (ex : Java Script).
- Le pont (bridge) permet aux applications HTML/JavaScript d'accéder directement au code natif

Types des approches de développement mobile

Approche hybride : Avantages et Inconvénients

Avantages	Inconvénients
▪ Coûts de développement plus faibles	▪ Performances plus lentes
▪ Support de différents OS	▪ Les fonctionnalités du système d'exploitation sont limitées
▪ Réutilisation du code	▪ Aucune interaction avec d'autres applications natives
▪ Développement rentable	▪ L'interface utilisateur n'aura pas l'apparence de l'application native. Pour obtenir une apparence native, le style propre à la plateforme pourrait être nécessaire.
▪ Grandes capacités de personnalisation	

Types des approches de développement mobile

Développement d'applications multiplateformes

- Les applications multiplateformes (cross-platform) ressemblent beaucoup aux applications hybrides.

- Cependant, contrairement aux applications hybrides dont le code s'exécute dans une « webview » native, les applications cross-platform produisent **un même code source qui, compilé, produit deux applications natives**. Cela induit une meilleure performance et une expérience utilisateur plus proche du natif.

- Les technologies utilisées pour développer une application cross-platform sont Appcelerator, Titanium, Xamarin, React Native et Flutter ou encore Kotlin multiplateforme.

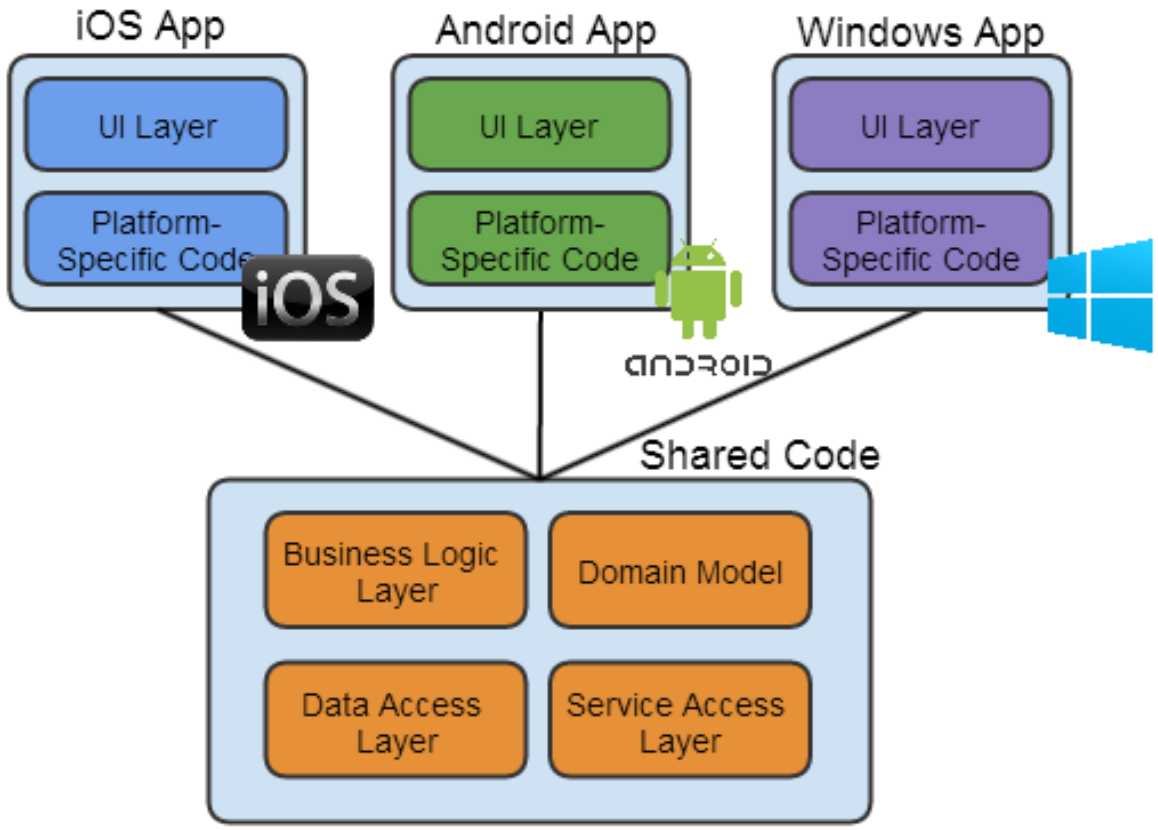
Types des approches de développement mobile

Approche multiplateformes : Architecture

Le code de l'application doit être décomposé en plusieurs parties :

- **Code commun (Sahed Code)** - un projet commun qui contient du code réutilisable pour stocker les données de la tâche ; expose une classe Modèle et une API pour gérer l'enregistrement et le chargement des données.

- **Code spécifique à la plate-forme (Platform Specific Code)** - projets spécifiques à la plate-forme qui mettent en œuvre une interface utilisateur native pour chaque système d'exploitation, en utilisant le code commun comme "back end".



Ref : microsoft - Xamarin

PARTIE 1

Types des approches de développement mobile

Approche multiplateformes : Avantages et Inconvénients

Avantages	Inconvénients
▪ La création d'applications basées sur des applications multiplateformes permet de gagner du temps et de l'argent.	▪ Performances plus lentes.
▪ Vous pouvez ajouter des fonctionnalités rapidement, car le code de base ne nécessite qu'une petite mise au point.	▪ Accès limité aux fonctionnalités du système d'exploitation.
▪ Les performances de l'interface utilisateur correspondent à celles des applications natives, car elles partagent une base de code similaire.	▪ Faible interaction avec les autres applications natives.
▪ Si votre budget est faible au départ, le choix de la multiplateforme est idéal	

Types des approches de développement mobile

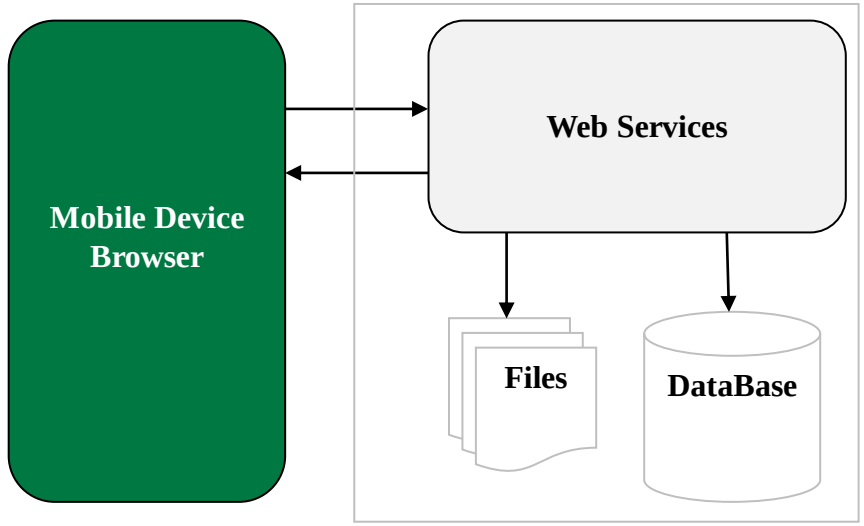
Application web progressive

- Avec l'approche de développement web, l'application s'exécute dans le navigateur du terminal mobile et utilise les technologies web standard telles que HTML5, CSS3 et JavaScript.
- Les applications web mobiles n'ont pas accès aux fonctions de la plateforme car elles reposent uniquement sur le navigateur et sur les normes web associées.
- Les applications web mobiles ne sont pas distribuées via un magasin d'applications. Elles sont accessibles via un lien sur un site Web ou un signet dans le navigateur du terminal mobile de l'utilisateur.

Types des approches de développement mobile

Approche web progressive : Architecture

- Navigateur web ;
- Langage du web (HTML/JS/CSS/PHP ...) ;
- Nécessite obligatoirement une connexion internet ;
- Limites aux possibilités du navigateur.



Ref : 10.14569/IJACSA.2017.080215

PARTIE 1

Types des approches de développement mobile

Approche web progressive : Avantages et Inconvénients

Avantages	Inconvénients
▪ Facile à apprendre et à développer en utilisant les technologies web.	▪ Les applications web ne sont pas disponibles dans les boutiques en ligne.
▪ Tout le traitement est effectué sur le serveur et seule l'interface utilisateur est envoyée au mobile.	▪ Une connexion Internet est nécessaire pour faire fonctionner l'application.
▪ Le maintien de l'application est simple parce que les mises à jour de l'application et les données sont effectuées sur le serveur.	▪ Les applications web ne peuvent pas accéder au logiciel de l'appareil mobile et aux matériels tels que la caméra, et les capteurs, GPS, etc.
▪ La même application est développée une fois et peut fonctionner sur différentes plateformes en utilisant les navigateurs web mobiles.	▪ Une application web pourrait souffrir du mauvais fonctionnement en raison de la connexion et des délais réseau.
	▪ Le développeur d'applications a moins de contrôle sur la manière dont les différents navigateurs interprètent le contenu.

Critères pour sélectionner une approche afin de développer une application mobile

- Budget
- Délai de mise sur le marché
- Portée (couverture du marché)
- Performances de l'application
- UI et UX
- Fonctionnalités

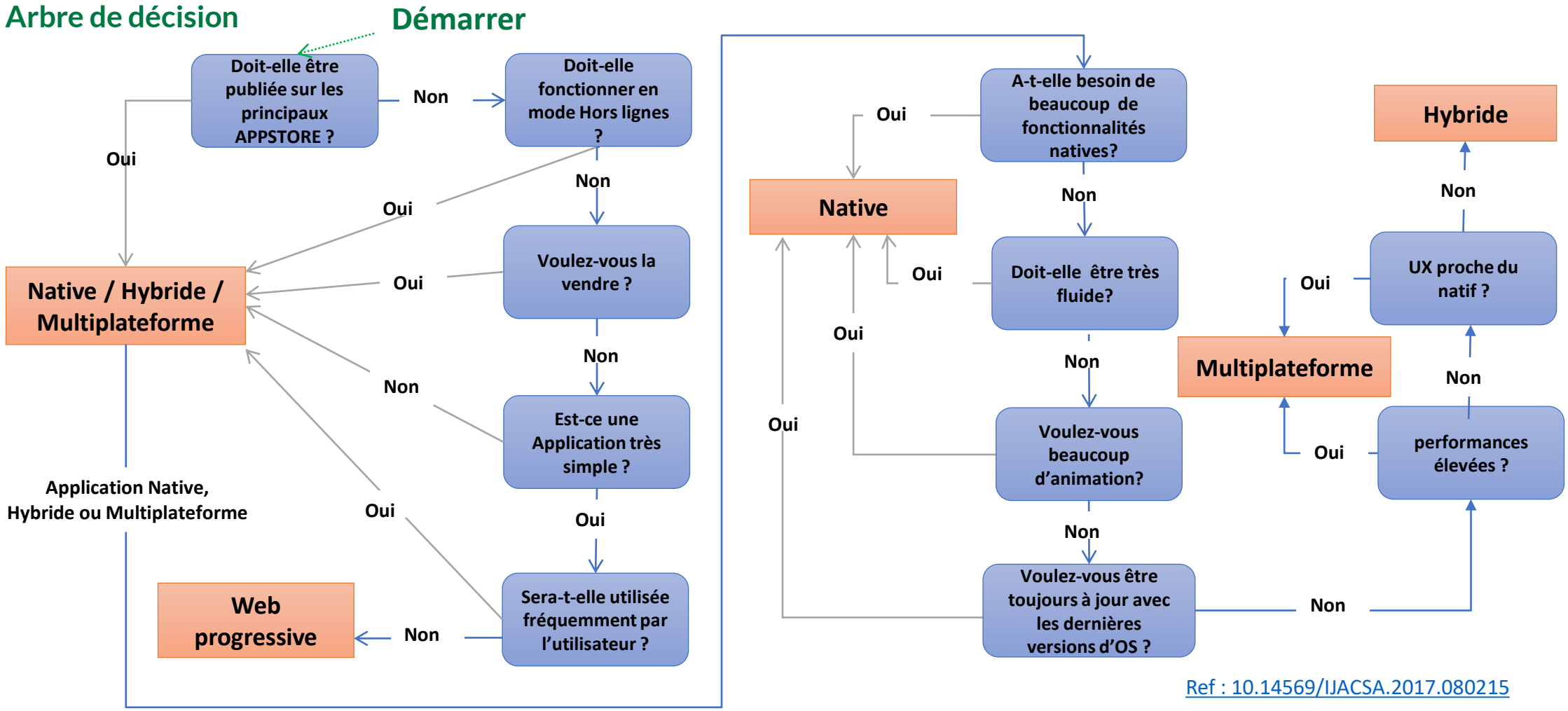
Dans la diapositive suivante, nous allons situer les approches de développement mobile vis-à-vis des critères de sélection, afin de dresser par la suite un arbre de décision. Ainsi, cela permettra de faciliter aux développeurs et aux décideurs le choix de l'approche adéquate pour la mise en place d'une application mobile spécifique.

Critères pour sélectionner une approche afin de développer une application mobile

	Native	Hybride	Multiplateformes
Budget moindre à l'achat	✗	✓	✓
Plein usage de fonctionnalités du terminal mobile	✓	✗	✗
Performance élevée	✓	✗	✓
Expérience utilisateur optimisée pour chaque OS	✓	✓ *	✓ *
Maintenabilité et évolutivité	✓	✓ **	✓ **
Planning serré de mise à disposition	✗	✓	✓

Réf : inflexsys

* : Moyennant un effort de développement supplémentaire pour adapter l'application aux standards de navigation spécifiques à chaque OS
** : Moyennant des montées de versions plus fréquentes



Ref : 10.14569/IJACSA.2017.080215

PARTIE 1

QCM

1. **Il s'agit d'une application qui peut fonctionner sur n'importe quel appareil fonctionnant sous Windows. La prise en charge de plusieurs langages vous permet de développer des applications mobiles en utilisant votre langage de codage préféré.**

a. Plate-forme universelle Windows (UWP)

b. Développement d'applications multiplateformes

c. Développement natif

d. Développement d'applications mobiles

QCM

1. **Il s'agit d'une application qui peut fonctionner sur n'importe quel appareil fonctionnant sous Windows. La prise en charge de plusieurs langages vous permet de développer des applications mobiles en utilisant votre langage de codage préféré.**

a. **Plate-forme universelle Windows (UWP)**

b. Développement d'applications multiplateformes

c. Développement natif

d. Développement d'applications mobiles

QCM

2. Parmi les facteurs suivants, quels sont ceux à prendre en compte lors de la conception d'une application mobile ?

a. Plateformes et compatibilité des appareils
b. Taille de l'écran
c. Interaction avec l'utilisateur ,
d. Gestion des ressources
e. Tous les choix

QCM

2. Parmi les facteurs suivants, quels sont ceux à prendre en compte lors de la conception d'une application mobile ?

a. Plateformes et compatibilité des appareils
b. Taille de l'écran
c. Interaction avec l'utilisateur
d. Gestion des ressources
e. Tous les choix

QCM

3. Un type de développement d'applications qui crée des applications pour une plateforme ou un appareil spécifique. Elle interagit et/ou tire parti des fonctionnalités et des ressources qui sont généralement disponibles pour sa plateforme mère.

a. Plate-forme universelle Windows (UWP)

b. Développement d'applications multiplateformes

c. Développement natif

d. Développement d'applications mobiles

QCM

3. Un type de développement d'applications qui crée des applications pour une plateforme ou un appareil spécifique. Elle interagit et/ou tire parti des fonctionnalités et des ressources qui sont généralement disponibles pour sa plateforme mère.

a. Plate-forme universelle Windows (UWP)

b. Développement d'applications multiplateformes

c. Développement natif

d. Développement d'applications mobiles

QCM

4. Laquelle des affirmations suivantes est FAUSSE à propos des plateformes de développement d'applications natives pour les appareils mobiles ?

a. iOS dispose de l'IDE XCode, ainsi que du kit de développement logiciel iOS (iOS SDK) qui utilise les langages de programmation Objective-C et Swift.

b. Android dispose du bien nommé Android Studio comme plateforme native, ainsi que du SDK Android qui utilise le langage de programmation Java.

c. Windows utilise Visual Studio IDE pour le développement d'applications natives, ainsi que les API Windows qui utilisent le langage de programmation C#.

d. Toutes les affirmations sont VRAIES

QCM

4. Laquelle des affirmations suivantes est FAUSSE à propos des plateformes de développement d'applications natives pour les appareils mobiles ?

a. iOS dispose de l'IDE XCode, ainsi que du kit de développement logiciel iOS (iOS SDK) qui utilise les langages de programmation Objective-C et Swift.

b. Android dispose du bien nommé Android Studio comme plateforme native, ainsi que du SDK Android qui utilise le langage de programmation Java.

c. Windows utilise Visual Studio IDE pour le développement d'applications natives, ainsi que les API Windows qui utilisent le langage de programmation C#.

d. Toutes les affirmations sont VRAIES

QCM

5. Il s'agit d'un type de développement d'applications qui permet de créer des applications qui fonctionnent, ou sont compatibles, avec plusieurs plates-formes et appareils.

a. Développement multiplateforme

b. Développement natif

c. Développement d'applications mobiles

d. Développement de la programmation

QCM

5. Il s'agit d'un type de développement d'applications qui permet de créer des applications qui fonctionnent, ou sont compatibles, avec plusieurs plates-formes et appareils.

a. Développement multiplateforme

b. Développement natif

c. Développement d'applications mobiles

d. Développement de la programmation

QCM

6. **Xamarin est le seul grand framework qui se compile en binaires natifs respectifs pour une application 100 % native ; le codage se fait à l'aide de quel langage de programmation ?**

a. C#
b. VB.net
c. Java
d. Javascript

QCM

6. **Xamarin est le seul grand Framework qui se compile en binaires natifs respectifs pour une application 100 % native ; le codage se fait à l'aide de quel langage de programmation ?**

a. C#

b. VB.net

c. Java

d. Javascript

QCM

7. Lequel des avantages suivants du développement multiplateforme est FAUX ?

a. Une base de code unique est utilisée pour le développement de plusieurs plateformes/appareils.

b. Le code est correctement compilé en utilisant des bibliothèques de code natif pour assurer la compatibilité des applications.

c. Il permet de réduire le débogage et la vérification des erreurs et d'améliorer la maintenabilité.

d. Elle permet de gagner du temps et de l'argent grâce à des concepts de développement tels que l'utilisation de principes de conception et la réutilisation du code.

e. Aucun des choix

QCM

7. Lequel des avantages suivants du développement multiplateforme est FAUX ?

a. Une base de code unique est utilisée pour le développement de plusieurs plateformes/appareils.

b. Le code est correctement compilé en utilisant des bibliothèques de code natif pour assurer la compatibilité des applications.

c. Il permet de réduire le débogage et la vérification des erreurs et d'améliorer la maintenabilité.

d. Elle permet de gagner du temps et de l'argent grâce à des concepts de développement tels que l'utilisation de principes de conception et la réutilisation du code.

e. Aucun des choix

CHAPITRE n° 1

Introduire les différentes plateformes de développement mobile

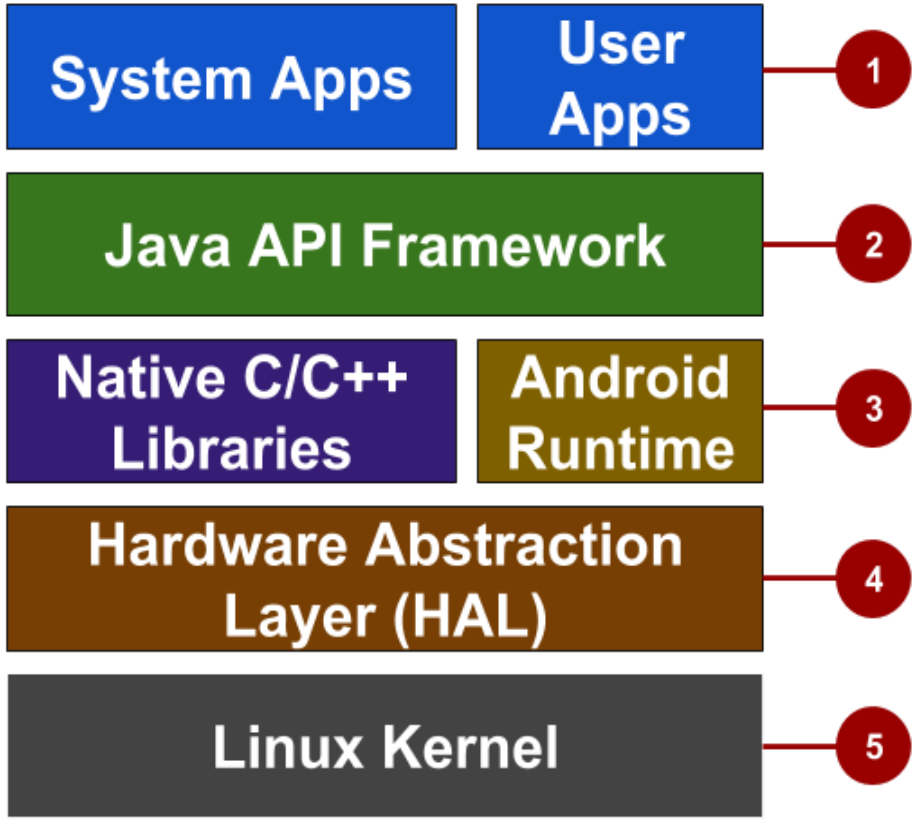
1. Présentation des systèmes d'exploitation mobile
2. Approche de développement mobile
3. Composition de l'architecture d'Android et iOS

Présentation de l'architecture Android

L'architecture d'Android est composée de 5 couches :

1. Applications système et utilisateur ;
2. API du système d'exploitation Android dans le Framework Java ;
3. Exposition des API natives ; exécution des applications ;
4. Exposer les capacités matérielles du dispositif ;
5. Noyau Linux.

Une description de chaque couche sera présentée dans les diapositives qui suivent.



Réf : Android developer

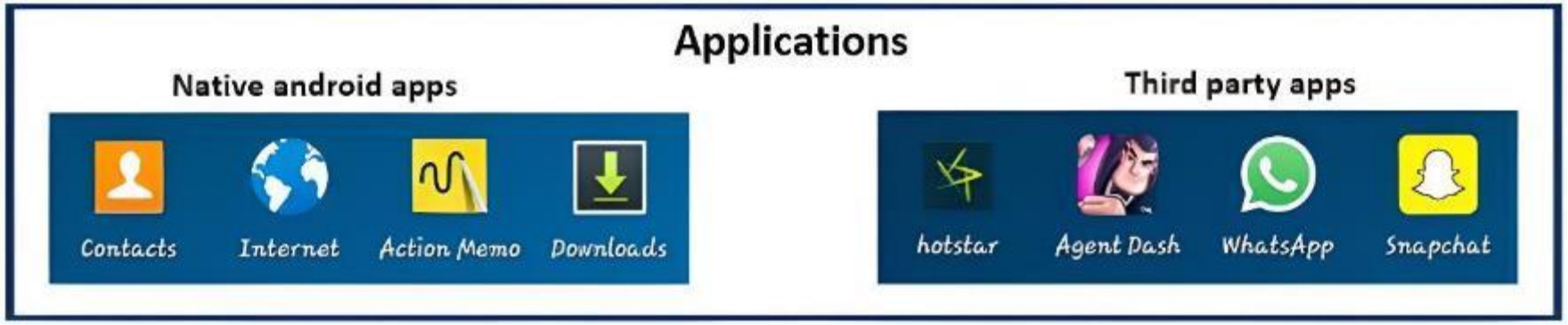
PARTIE 1

Architecture Android : Applications système et utilisateur

- Les applications système n'ont pas de statut particulier.
- Les applications système fournissent des capacités clés aux développeurs d'applications.

Exemple :

- Votre application peut utiliser une application système pour délivrer un message SMS ou passer un appel téléphonique.

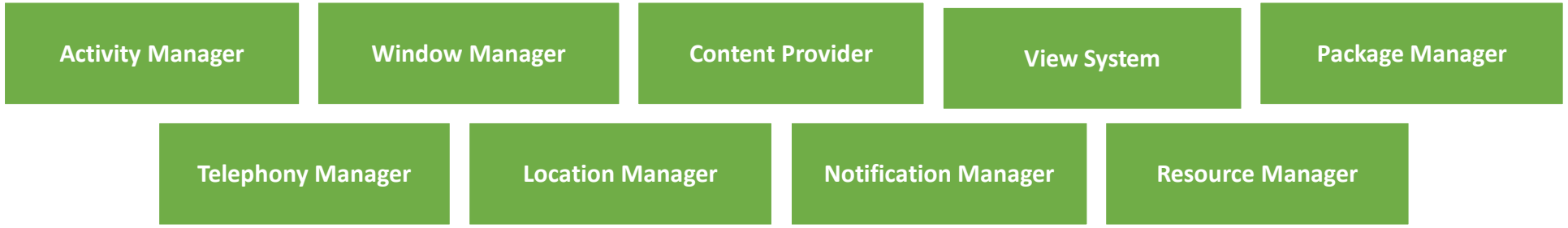


Architecture Android : Java API Framework

L'ensemble des fonctionnalités du système d'exploitation Android est mis à votre disposition par le biais d'API écrites en langage Java :

- Hiérarchie des classes de vue pour créer des écrans d'interface utilisateur ;
- Gestionnaire de notifications ;
- Gestionnaire d'activités pour les cycles de vie et la navigation.

Java API Framework

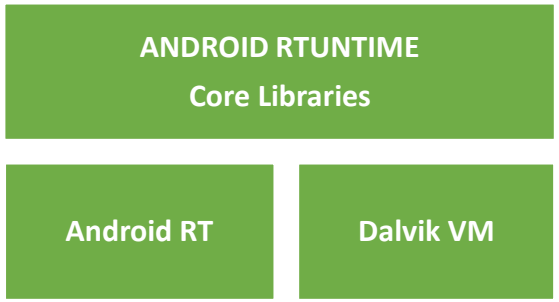


Architecture Android : Android runtime

Chaque application s'exécute dans son propre processus avec sa propre instance du Runtime Android.

Bien qu'Android soit développé en Java, la couche d'exécution de l'architecture Android est constituée de la **machine virtuelle Dalvik (DVM)**, des bibliothèques Java de base et, récemment, d'une nouvelle machine virtuelle appelée **Android runtime (ART).**

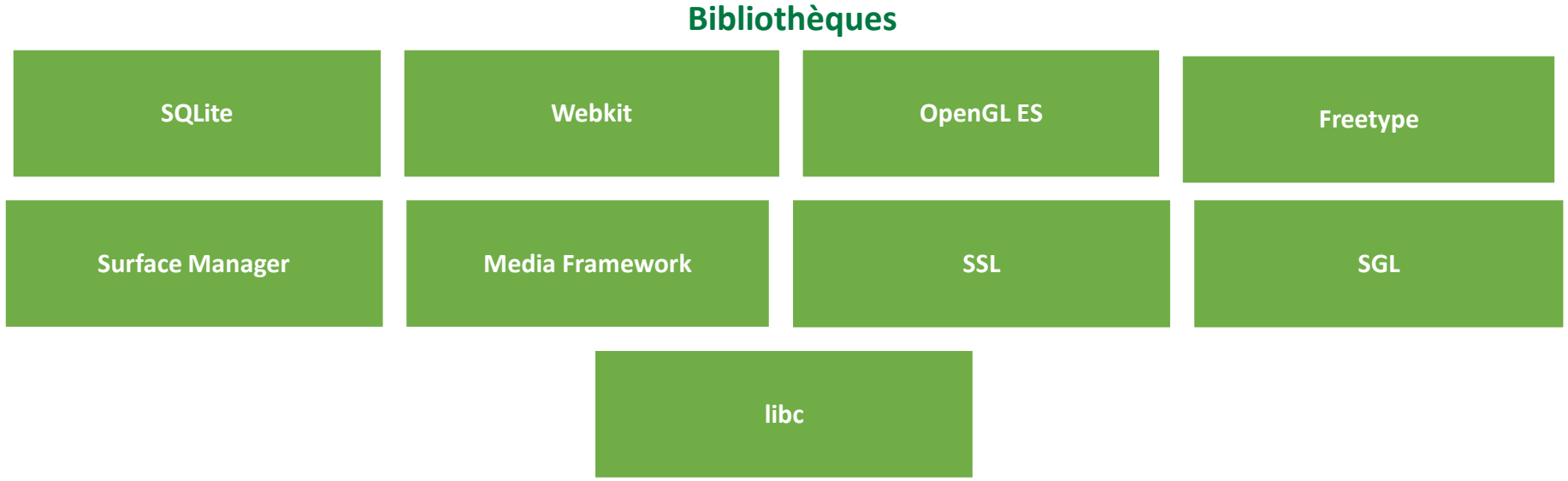
La DVM exécute des applications programmées en Java. La DVM ne prétend pas être une machine virtuelle Java (JVM) pour des raisons de licence, mais elle remplit la même fonction. La raison en est que Dalvik est optimisé pour fonctionner sur des appareils de petite taille avec une mémoire limitée. Pour des raisons de performances, la DVM n'est démarrée qu'une seule fois. Chaque nouvelle instance de celle-ci est clonée par un service système appelé Zygote. Le schéma suivant présente la structure du runtime Android :



PARTIE 1

Architecture Android : Bibliothèques C/C++

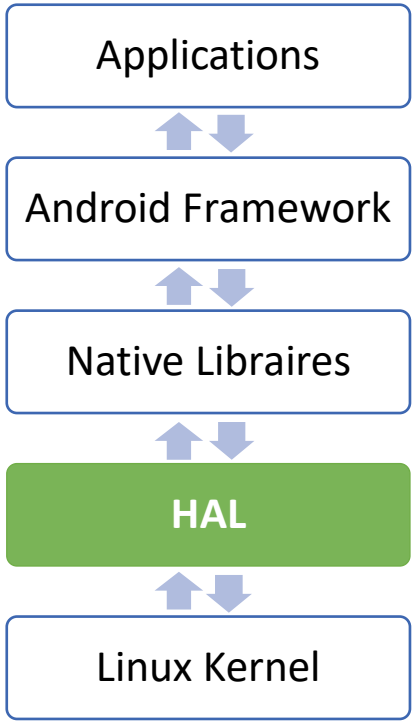
Les bibliothèques C/C++ de base donnent accès aux principaux composants et services natifs du système Android.



Architecture Android : Couche d'abstraction matérielle (HAL)

Interfaces standard qui exposent les capacités matérielles des dispositifs sous forme de bibliothèques.

Exemples : Caméra, module Bluetooth



Architecture Android : Noyau Linux

- Threading et gestion de la mémoire de bas niveau ;
- Fonctions de sécurité ;
- Pilotes.

Noyau Linux

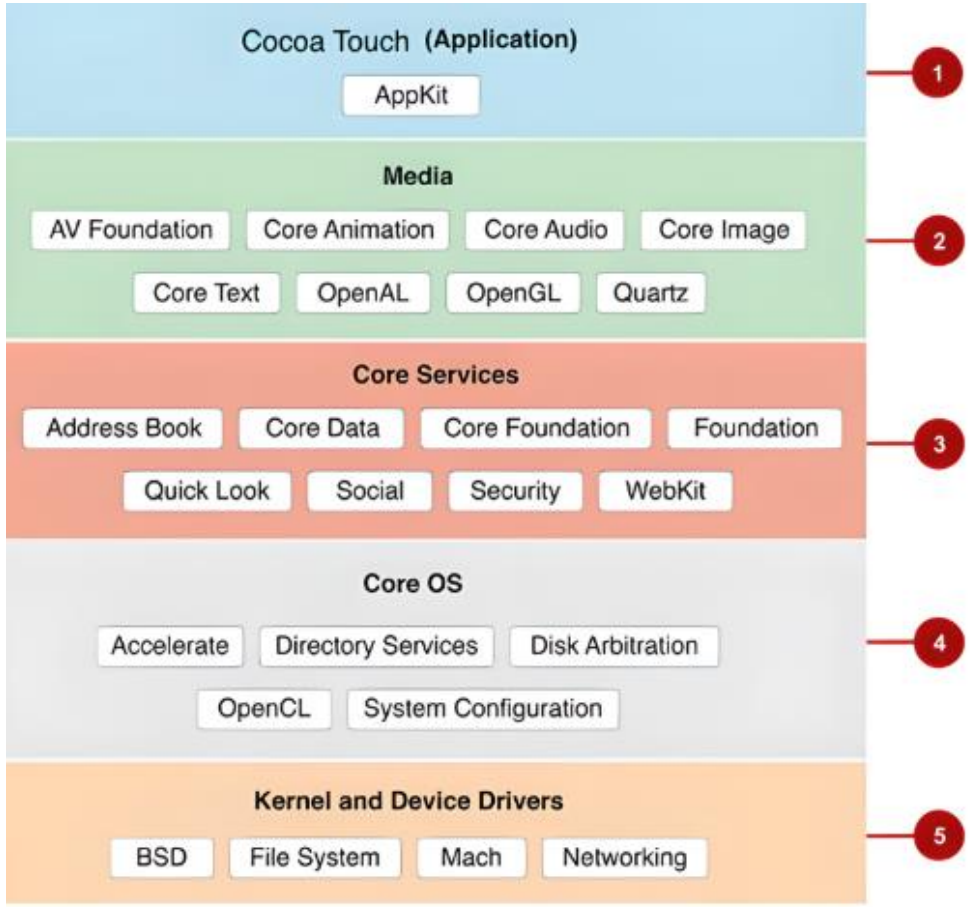
Display Driver	**Camera Driver**	**Audio Driver**	**Bindler (IPC) Driver**	**Flash Memory Driver**
Process Management	Wifi Driver	Bluetooth Driver	Memory Mangement	Power Management

Présentation de l'architecture iOS

L'architecture de iOS est composée de 5 couches :

1. Couche Cocoa Touch ;
2. Couche média ;
3. Couche de services de base ;
4. Couche de base du système d'exploitation ;
5. Noyau et pilotes de périphériques.

Une description de chaque couche sera présentée dans les diapositives qui suivent.



Réf: dotnettricks

PARTIE 1

Architecture iOS : Cocoa Touch Layer

La couche Cocoa Touch comprend un ensemble crucial de cadres (Frameworks), écrits en Objective-C, et développés sur la base de l'API Mac OS X Cocoa. L'apparence de toute application que vous voyez dans iOS est développée à l'aide du cadre Cocoa Touch. Les notifications, le multitâche, les entrées spécifiques au toucher, tous les services système de haut niveau et d'autres technologies clés sont pris en charge par cette couche, qui fournit également une infrastructure de base pour une application.

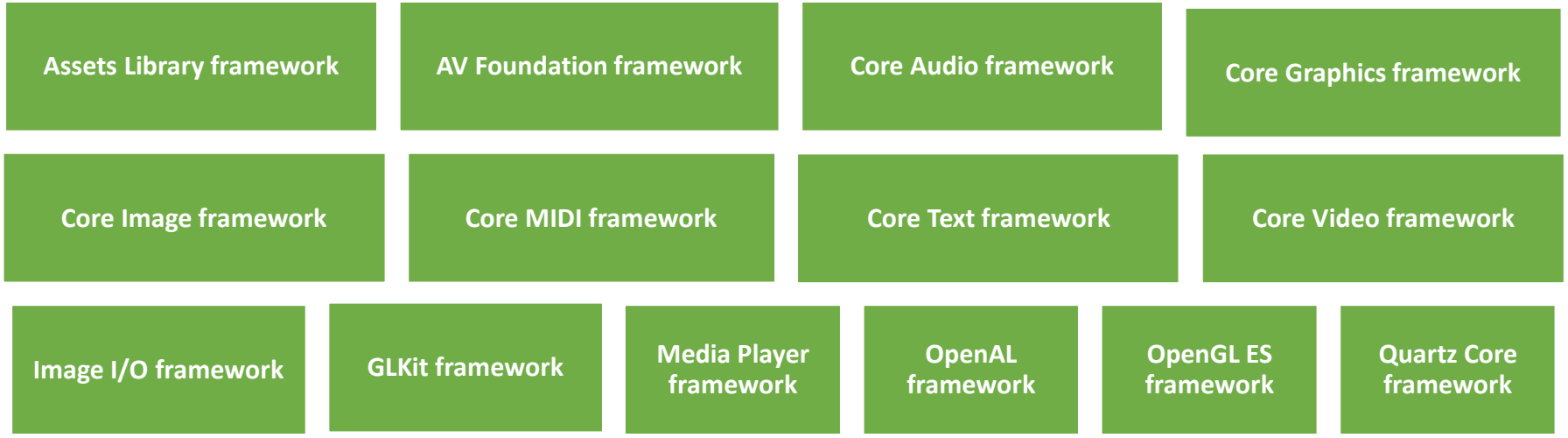
Voici la liste des cadres importants qui sont largement utilisés dans cette couche :

Address Book UI framework	Event Kit UI framework	Game Kit framework	iAd framework
Map Kit framework	Message UI framework	Twitter framework	UIKit framework

Architecture iOS : Media Layer

Nous commentons souvent les expériences multimédia, notamment **la clarté du son** et la **qualité de la vidéo**. Ce rôle est essentiellement joué par la couche média de la pile iOS, qui fournit à l'iOS des capacités audio, vidéo, graphiques et AirPlay (over-the-air).

Comme pour la couche Cocoa Touch, la couche média comprend un ensemble de cadres qui peuvent être utilisés par les développeurs.



Architecture iOS : Core Services Layer

La couche des services de base fournit les services fondamentaux que toutes les applications peuvent utiliser. Comme les autres couches, la couche des services de base fournit une liste de cadres :

Accounts framework	Address Book framework	Ad Support framework	CFNetwork framework	Core Data framework	Core Foundation framework	Core Location framework
Core Media framework	Core Motion framework	Core Telephony framework	Event Kit framework	Foundation framework	Mobile Core Services framework	Newsstand Kit framework
	Pass Kit framework	Quick Look framework	Social framework	Store Kit framework	System Configuration framework	

Architecture iOS : Core OS Layer

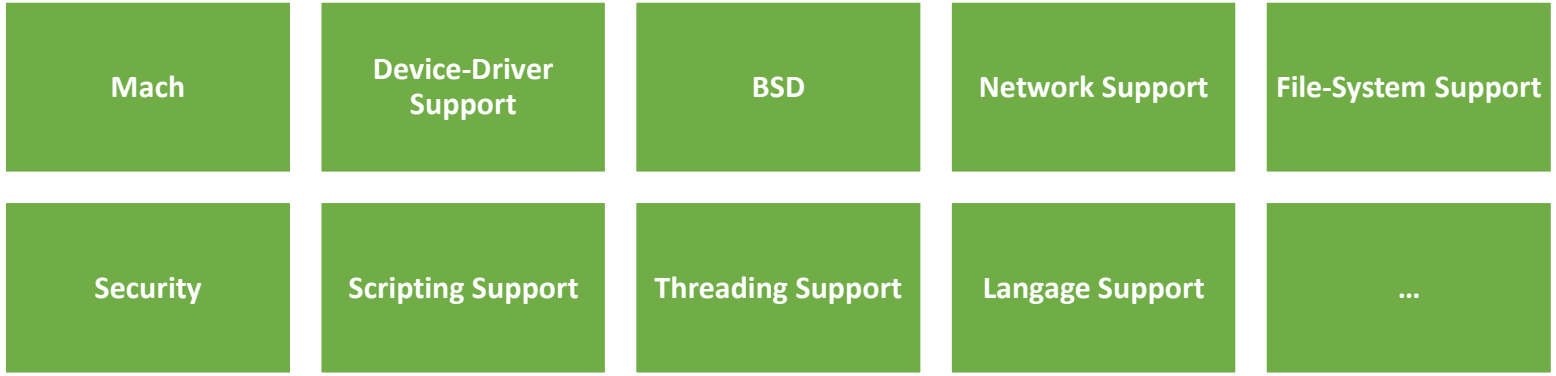
Le système d'exploitation principal contient des services et des technologies fondamentales de bas niveau pour les utilisateurs finaux. Il comprend le noyau d'OS X. Il exploite les lectures d'E/S entre les UC et le périphérique. Il s'agit de la couche qui se trouve au-dessus du matériel du périphérique, qui fournit un réseau de bas niveau, l'accès aux accessoires externes et les services système fondamentaux tels que la gestion de la mémoire, les fichiers système, etc.

Core OS contient les frameworks suivants :

- **Accelerate framework**
- **Core Bluetooth framework**
- **External Accessory framework**
- **Generic Security Services framework**
- **Security framework**

Architecture iOS : Noyau et pilotes de périphériques

La couche la plus basse d'OS X comprend le noyau, les pilotes et les parties BSD (Berkeley Software Distribution) du système et repose principalement sur des technologies open source. OS X étend cet environnement de bas niveau avec plusieurs technologies d'infrastructure de base qui vous permettent de développer plus facilement des logiciels.



CHAPITRE n° 2

Découvrir l'écosystème de développement mobile

Ce que vous allez apprendre dans ce chapitre :

- Environnements de développement
- Utilisation des langages et librairies appropriés (KOTLIN, JAVA, Swift, Objectif C)

3 heures

CHAPITRE n° 2

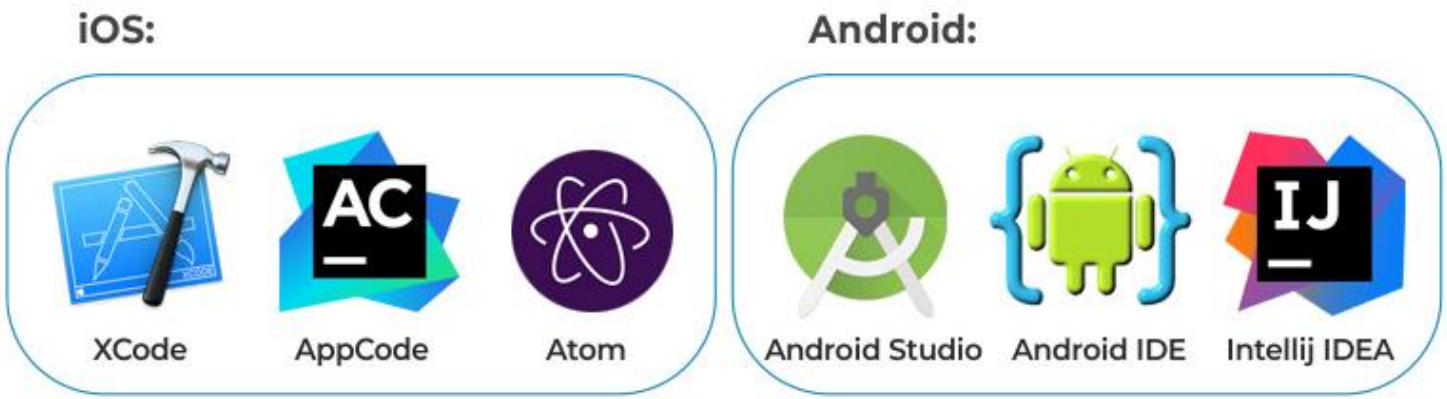
Découvrir l’écosystème de développement mobile

1. Environnements de développement
2. Langages et librairies appropriées

Développement d'applications natives

- **Android Studio** est l'environnement de développement intégré (IDE) officiel du système d'exploitation Android de Google, construit sur le logiciel IntelliJ IDEA de JetBrains et conçu spécifiquement pour le développement Android (https://developer.android.com/studio), fournit un environnement de développement intégré complet comprenant un éditeur de code avancé et un ensemble de modèles d'application. En outre, il contient des outils de développement, de débogage, de test et de performance qui facilitent le développement d'applications. Vous pouvez tester vos applications avec une large gamme d'émulateurs préconfigurés ou sur votre propre appareil mobile, créez des applications de production et les publier sur le magasin Google Play.

- **Xcode** est la suite d'environnement de développement intégré (IDE) développée par Apple pour le développement d'applications iOS (https://developer.apple.com/xcode/).



Développement d'applications hybrides

- **Ionic** est un cadre de développement d'applications mobiles HTML5 destiné à la création d'applications mobiles hybrides (https://ionicframework.com/).

- **Apache Cordova** est un cadre de développement mobile à code source ouvert. Il permet d'utiliser les technologies Web standard (HTML5, CSS3 et JavaScript) pour le développement multiplateforme. Les applications s'exécutent dans des wrappers ciblés pour chaque plateforme et s'appuient sur des liaisons API conformes aux normes pour accéder aux capacités de chaque appareil, telles que les capteurs, les données, l'état du réseau, etc. (https://cordova.apache.org/).

- **Visual studio code**, également appelé VS Code, est un éditeur de code source créé par Microsoft pour Windows, Linux et MacOs. Les fonctionnalités comprennent la prise en charge du débogage, la coloration syntaxique, la complétion de code intelligente, les snippets, le refactoring de code et Git intégré. Les utilisateurs peuvent modifier le thème, les raccourcis clavier, les préférences et installer des extensions qui ajoutent des fonctionnalités supplémentaires. (https://code.visualstudio.com/).

Développement d'applications multiplateformes

- **React Native** est un cadre logiciel UI open-source créé par **Meta Platforms**, Inc. Il est utilisé pour développer des applications pour Android, Android TV, iOS, MacOs, tvOS, Web, Windows et UWP en permettant aux développeurs d'utiliser le Framework React en même temps que les capacités de la plateforme native. Il est également utilisé pour développer des applications de réalité virtuelle (https://reactnative.dev/)

- **Xamarin** est un cadre d'interface utilisateur multiplateforme open source de **Microsoft** permettant de créer des applications iOS, Android et Windows avec .NET à partir d'une base de code unique et partagée (https://dotnet.microsoft.com/en-us/apps/xamarin)

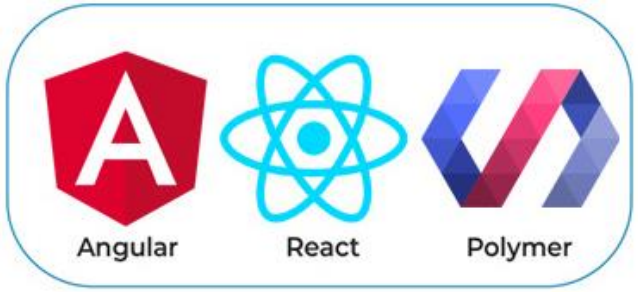
- **Flutter** est un cadre de **développement d’applications multiplateforme**, conçu par **Google**, dont la première version a été publiée sous forme de projet open source à la fin de l’année 2018. Flutter met à disposition une grande variété de bibliothèques d’éléments d’IU standard pour Android et iOS (https://flutter.dev/)

Application web progressive

- **Angular** est un Framework open source JavaScript développé par Google. Ce Framework est utilisé pour développer des applications web et mobile. Avec cette technologie, on réalise des interfaces de type monopage ou "one page" qui fonctionnent sans rechargement de la page web, développé par **Google** en 2009 (https://angular.io/).

- **React** (également connu sous le nom de React.js ou ReactJS) est une bibliothèque JavaScript frontale gratuite et open-source permettant de créer des interfaces utilisateur basées sur des composants d'interface utilisateur. Elle est maintenue par Meta (anciennement Facebook) et une communauté de développeurs individuels et d'entreprises (https://reactjs.org/).

- **Polymer** est une bibliothèque JavaScript open-source permettant de créer des applications Web à l'aide de composants Web. La bibliothèque est développée par des développeurs et des contributeurs de Google sur GitHub. Les principes de conception moderne sont mis en œuvre dans le cadre d'un projet distinct utilisant les principes de conception Material Design de Google (https://polymer-library.polymer-project.org/).



CHAPITRE n° 2

Découvrir l’écosystème de développement mobile

1. Environnements de développement
2. Langages et librairies appropriées

Langages de programmation d'applications Android

Une majorité des apps développées pour Android sont des apps natives ; celles-ci sont développées en **Java** et, depuis 2017, en **Kotlin**. **Google** encourage l'utilisation de **Kotlin** pour le développement des apps mobiles.

Java

Java est un langage de programmation orienté objet créé par **James Gosling** et **Patrick Naughton**, employés de Sun Microsystems, avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld**.**

Kotlin

En 2011, **JetBrains** a annoncé le développement du langage de programmation **Kotlin**, qui constituait une alternative à l'écriture de code dans des langages comme Java ou Scala s'exécutant sur la machine virtuelle Java. Six ans plus tard (2017), **Google** a annoncé que Kotlin devenait un outil de développement officiellement pris en charge par le système d'exploitation Android.

Langages de programmation d'applications iOS

Une majorité des apps développées pour iOS sont des apps natives. Celles-ci sont développées en **Objective-C** et, depuis 2015, en **Swift**. Apple a imposé l'utilisation de **Swift** pour le développement des apps. Cela serait facile pour ceux qui ont une certaine expérience des langages de programmation orientés objet.

Objective-C

Objective-C est un sur-ensemble strict du C et une augmentation de celui-ci ; c'est un langage orienté objet qui ajoute des messages de type Smalltalk (langage de programmation orienté objet, dynamiquement typé et réactif) au langage de programmation C. Il a été créé par Brad Cox et Tom Love au début des années 1980. Cela signifie que le compilateur Objective-C peut également compiler des programmes C.

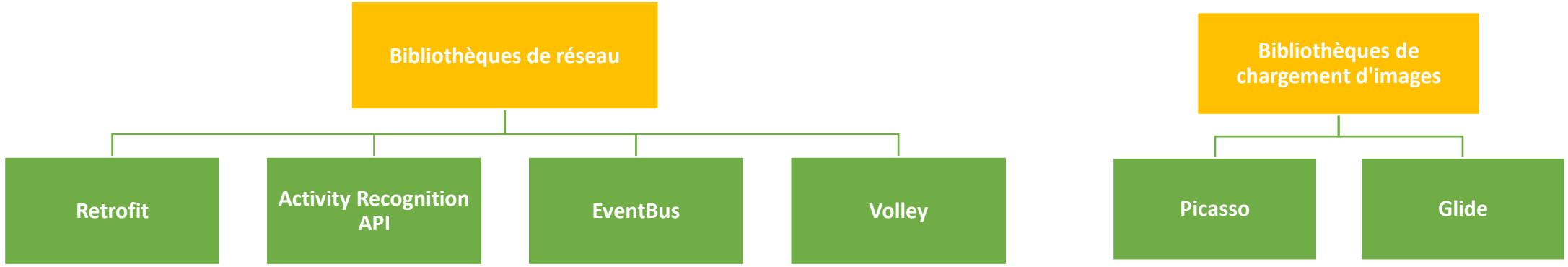
Swift

Swift est un nouveau langage de programmation créé par Apple Inc. spécifiquement pour iOS, OS X et watchOS, et est potentiellement un remplacement de l'Objective-C dans le futur. Il a été publié pour la première fois le 2 juin 2014, avec une version stable le 15 septembre 2015.

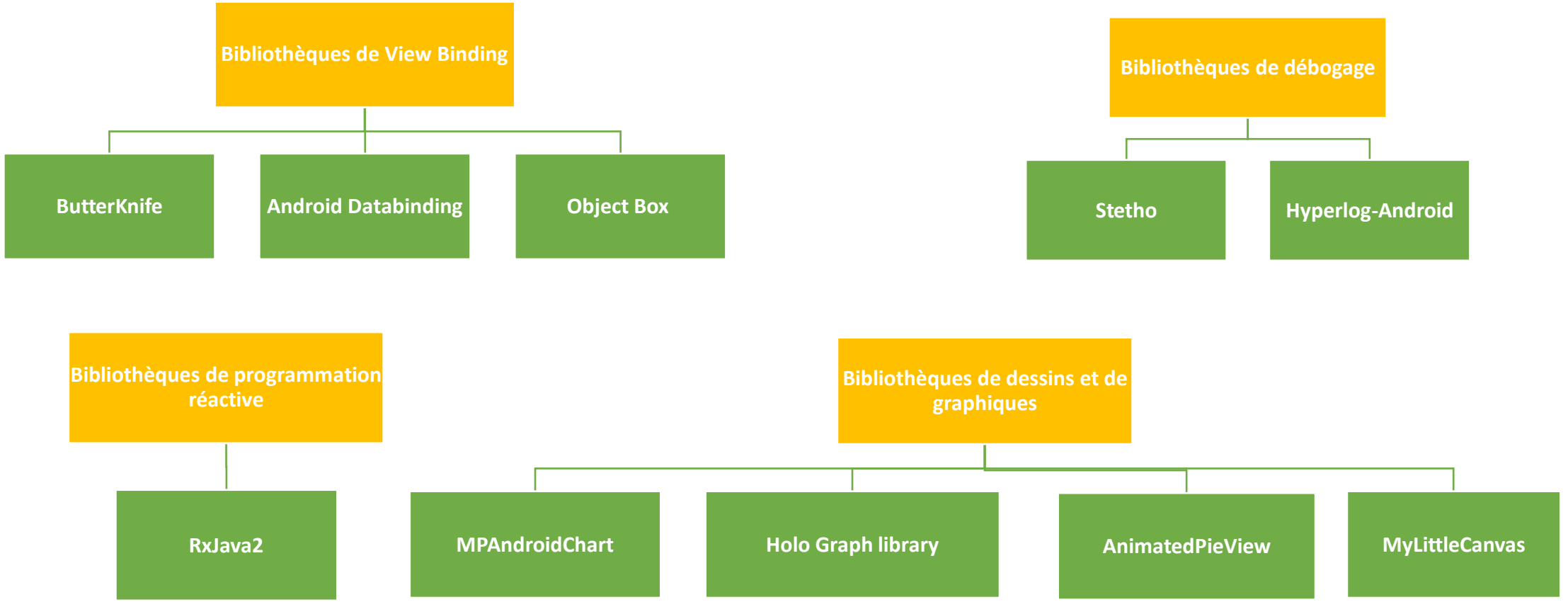
Librairies appropriées Android

L'une des principales raisons d'utiliser les bibliothèques est d'accélérer et d'optimiser le processus de développement d'applications Android, c'est-à-dire de développer une application étonnante sans efforts inutiles ou supplémentaires. Ces bibliothèques fournissent des codes de base pré-écrits et d'autres éléments importants qui peuvent être utilisés instantanément plutôt que d'effectuer des tâches à partir de zéro.

Les bibliothèques populaires qui sont un outil indispensable pour presque toutes les entreprises de développement d'applications mobiles Android sont :



Librairies appropriées Android



Librairies appropriées Android

Librairies appropriées iOS

- **SwiftyJSON**: permet de traiter facilement des données JSON en Swift.
- **Alamofire** : une bibliothèque de mise en réseau HTTP écrite en Swift.
- **SVProgressHud** : un HUD (Hookup Dating) propre et facile à utiliser destiné à afficher la progression d'une tâche en cours sur iOS et tvOS.
- **MJRefresh** : Une façon simple d'utiliser la fonction pull-to-refresh.
- **Masonry :** Masonry est un cadre de mise en page léger qui enveloppe AutoLayout avec une syntaxe plus agréable. Masonry a son propre DSL de mise en page qui fournit une manière chaînable de décrire les NSLayoutConstraints qui résulte en un code de mise en page qui est plus concis et lisible. Masonry est compatible avec iOS et Mac OS X.

QCM

1. Un environnement de programmation qui a été packagé comme un programme d'application.

a. Langage de programmation

b. SDK Android

c. Visual studio

d. Environnement de développement intégré

QCM

1. Un environnement de programmation qui a été packagé comme un programme d'application.

a. Langage de programmation

b. SDK Android

c. Visual studio

d. Environnement de développement intégré

QCM

2. Il s'agit de l'IDE officiel pour le développement d'applications Android, basé sur IntelliJ IDEA (un IDE Java).

a. Visual studio

b. Android studio

c. Netbeans

d. Eclipse

QCM

2. Il s'agit de l'IDE officiel pour le développement d'applications Android, basé sur IntelliJ IDEA (un IDE Java).

a. Visual studio

b. Android studio

c. Netbeans

d. Eclipse

QCM

3. Il comprend tout ce qui peut être utilisé pour créer des applications étonnantes pour iPhone et iPad. Il permet aux utilisateurs de créer des applications et de les exécuter directement sur leurs appareils Apple.

a. Xcode

b. Android SDK

c. Swift

d. Visual studio Xamarin

QCM

3. Il comprend tout ce qui peut être utilisé pour créer des applications étonnantes pour iPhone et iPad. Il permet aux utilisateurs de créer des applications et de les exécuter directement sur leurs appareils Apple.

a. Xcode

b. Android SDK

c. Swift

d. Visual studio Xamarin

QCM

4. Le langage de programmation utilisé pour créer des applications dans iOS (créé par Apple pour le développement de iOS, OS X et watchOS).

a. Xcode
b. Swift
c. iPhone OS
d. iTouch

QCM

4. Le langage de programmation utilisé pour créer des applications dans iOS (créé par Apple pour le développement de iOS, OS X et watchOS).

a. Xcode

b. Swift

c. iPhone OS

d. iTouch

QCM

5. Il s'agit d'un IDE extensible complet permettant de créer des applications modernes pour Windows, Android et iOS, ainsi que des applications Web et des services en nuage.

a. Visual studio SDK

b. Android SDK

c. iPhone OS SDK

d. Windows Phone SDK

QCM

5. Il s'agit d'un IDE extensible complet permettant de créer des applications modernes pour Windows, Android et iOS, ainsi que des applications Web et des services en nuage.

a. Visual studio SDK

b. Android SDK

c. iPhone OS SDK

d. Windows Phone SDK

QCM

6. … est un informaticien canadien, surtout connu pour être le fondateur et le principal concepteur du langage de programmation Java.

a. James Arthur Gosling
b. Dennis Ritchie
c. Anders Hejlsberg
d. Bjarne Stroustrup

QCM

6. …est un informaticien canadien, surtout connu pour être le fondateur et le principal concepteur du langage de programmation Java.

a. James Arthur Gosling

b. Dennis Ritchie

c. Anders Hejlsberg

d. Bjarne Stroustrup

QCM

7. C# est un langage de programmation moderne orienté objet développé en 2000 par _____ chez Microsoft pour rivaliser avec Java (dont il est assez proche). Il a été créé parce que Sun, (racheté plus tard par Oracle)

a. James Arthur Gosling

b. Dennis Ritchie

c. Anders Hejlsberg

d. Bjarne Stroustrup

QCM

7. C# est un langage de programmation moderne orienté objet développé en 2000 par _____ chez Microsoft pour rivaliser avec Java (dont il est assez proche). Il a été créé parce que Sun, (racheté plus tard par Oracle)

a. James Arthur Gosling

b. Dennis Ritchie

c. Anders Hejlsberg

d. Bjarne Stroustrup

PARTIE 2

Maîtriser la plateforme Android

Dans ce module, vous allez :

- Préparer l'environnement de développement
- Vous initier à Android Studio
- Maîtriser la génération d'une application mobile

12 heures

CHAPITRE n° 1

Préparer l’environnement de développement

Ce que vous allez apprendre dans ce chapitre :

- Utilisation de Android studio
- Installation de l’SDK Android
- Installation de l’émulateur

3 heures

CHAPITRE n° 1

Préparer l’environnement de développement

1. Installation d’Android studio
2. Installation de l’SDK Android
3. Installation de l’émulateur

Qu'est-ce qu'Android Studio ?

- Environnement de développement intégré (IDE) Android ;
- Modèles de projets et d'activités ;
- Éditeur de mise en page ;
- Outils de test ;
- Construction basée sur Gradle ;
- Console de journalisation et débogueur ;
- Émulateurs.

Exigences techniques

La liste des exigences techniques essentielles pour le développement d'Android avec Android Studio et ses outils connexes. Il s'agit toutefois du strict minimum.

- **La configuration requise pour Windows est la suivante :**
  - Microsoft® Windows® 7/8/10/11 (64-bit) ;
  - 4 Go de RAM au minimum ; 8 Go de RAM recommandés ;
  - 2 Go d'espace disque disponible au minimum ; 4 Go recommandés (500 Mo pour l'environnement de développement intégré (IDE) + 1,5 Go pour le kit de développement Android Software (SDK) et l'image système de l'émulateur) ;
  - Résolution d'écran minimale de 1 280 x 800.

- **La configuration requise pour Mac est la suivante**
  - Mac® OS X® 10.10 (Yosemite) ou supérieur, jusqu'à 10.14 (macOS Mojave) ;
  - 4 Go de RAM au minimum ; 8 Go de RAM recommandés ;
  - 2 Go d'espace disque disponible au minimum ; 4 Go recommandés (500 Mo pour l'IDE + 1,5 Go pour le SDK Android et l'image système de l'émulateur).

Exigences techniques

- **Les exigences pour Linux sont les suivantes :**
    - Bureau GNOME ou KDE ;
    - Testé sur gLinux basé sur Debian ;
    - Distribution 64 bits capable d'exécuter des applications 32 bits ;
    - Bibliothèque GNU C (glibc) 2.19 ou ultérieure ;
    - 4 Go de RAM au minimum ; 8 Go de RAM recommandés ;
    - 2 Go d'espace disque disponible au minimum ; 4 Go recommandés (500 Mo pour l'IDE + 1,5 Go pour le SDK Android et l'image système de l'émulateur) ;
    - Résolution d'écran minimale de 1 280 x 800.

Aperçu de l'installation Android Studio

- Mac, Windows, ou Linux.
- Téléchargez et installez Android Studio à partir de: https://developer.android.com/studio/
- **Voir TP 1 : Android Studio et Hello World**

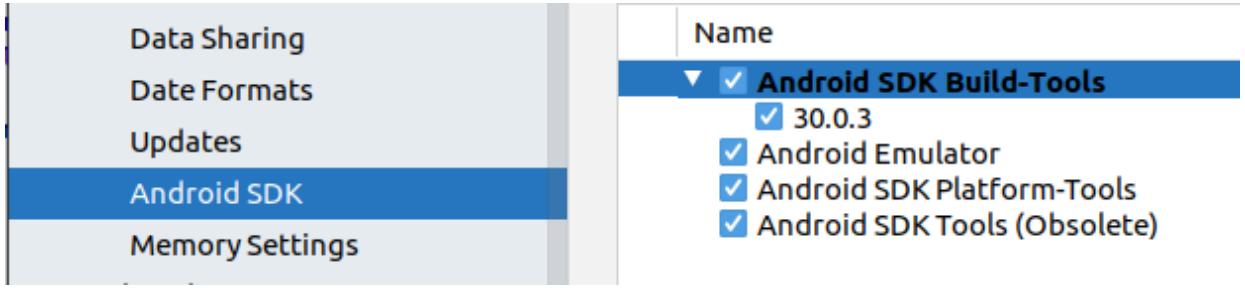
CHAPITRE n° 1

Préparer l’environnement de développement

1. Installation d’Android studio
2. **Installation de l’SDK Android**
3. Installation de l’émulateur

SDK Android

- Android SDK signifie Android Software Development Kit qui est développé par Google pour Android Platform ;
- SDK Android permet de créer facilement des applications Android ;
- Android SDK est un ensemble de bibliothèques et d'outils de développement logiciel essentiels au développement d'applications Android ;
- Ces outils assurent un déroulement fluide du processus de développement (développement, débogage, test, …) ;
- Android SDK est compatible avec tous les systèmes d'exploitation tels que Windows, Linux, MacOs, etc…
- Android SDK se compose d'un ensemble complet d'outils de développement et de débogage :
  - Outil de construction du SDK Android
  - Émulateur Android
  - Outils de la plate-forme SDK Android
  - Outils SDK Android



PARTIE 2

SDK Android offre :

Outils de construction du SDK Android

- Android SDK Build-Tools est un composant du SDK Android nécessaire à la construction d'applications Android. Il est installé dans le répertoire <sdk>/build-tools/
- Il comprend l'ensemble complet des outils de développement et de débogage pour Android.

Emulateur Android

- L'émulateur Android simule des appareils Android sur l'ordinateur afin que de tester l'application sur une variété d'appareils et de niveaux d'API Android sans avoir besoin d'avoir chaque appareil physique.

Android SDK Platform-Tools

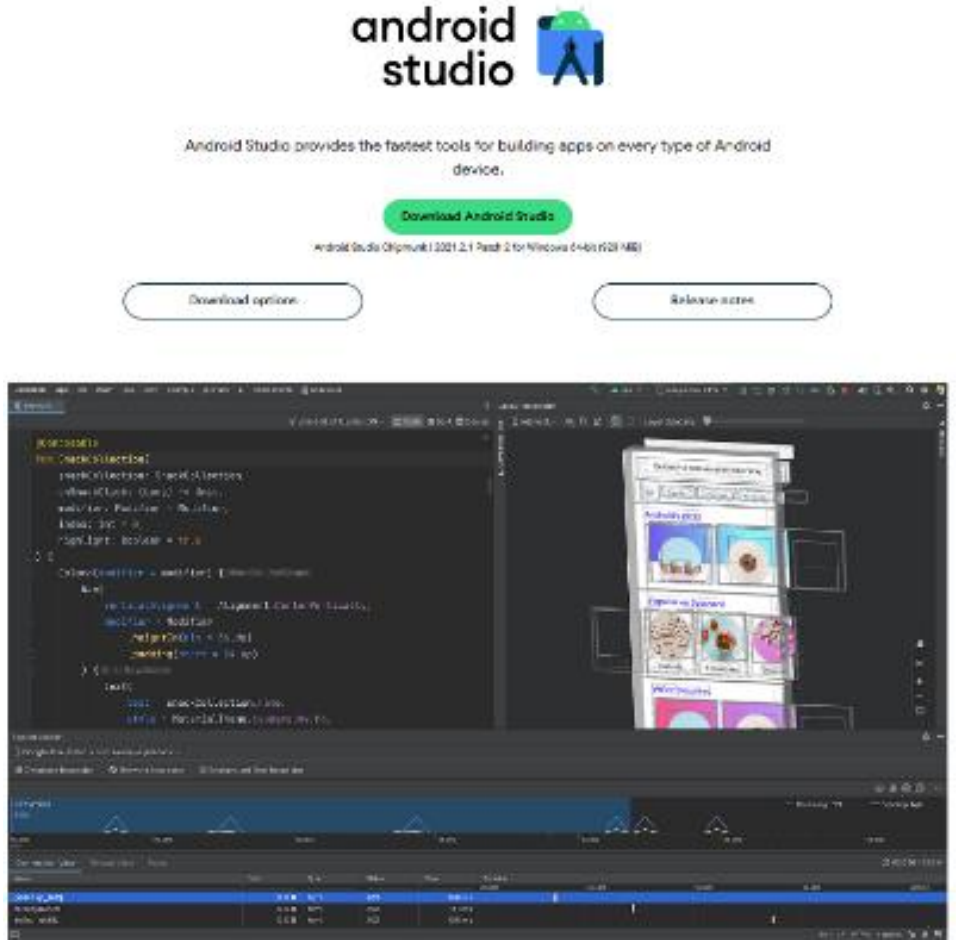
- Android SDK Platform-Tools est un composant du SDK Android. Il comprend des outils qui s'interfacent avec la plate-forme Android, tels que adb, fastboot et systrace.

Android SDK Tools

- Android SDK Tools est un composant du SDK Android. Il comprend un ensemble complet d'outils de développement et de débogage pour Android, et est inclus dans Android Studio. Le SDK Tools comprend également des outils de test et d'autres utilitaires nécessaires au développement d'une application.

Aperçu de l'installation de SDK Android

- Mac, Windows, ou Linux
- Téléchargez et installez Android Studio à partir de: https://developer.android.com/studio/
- **Voir TP 1 : Android Studio et Hello World**



CHAPITRE n° 1

Préparer l’environnement de développement

1. Installation d”Android studio
2. Installation de l’SDK Android
3. **Installation de l’émulateur**

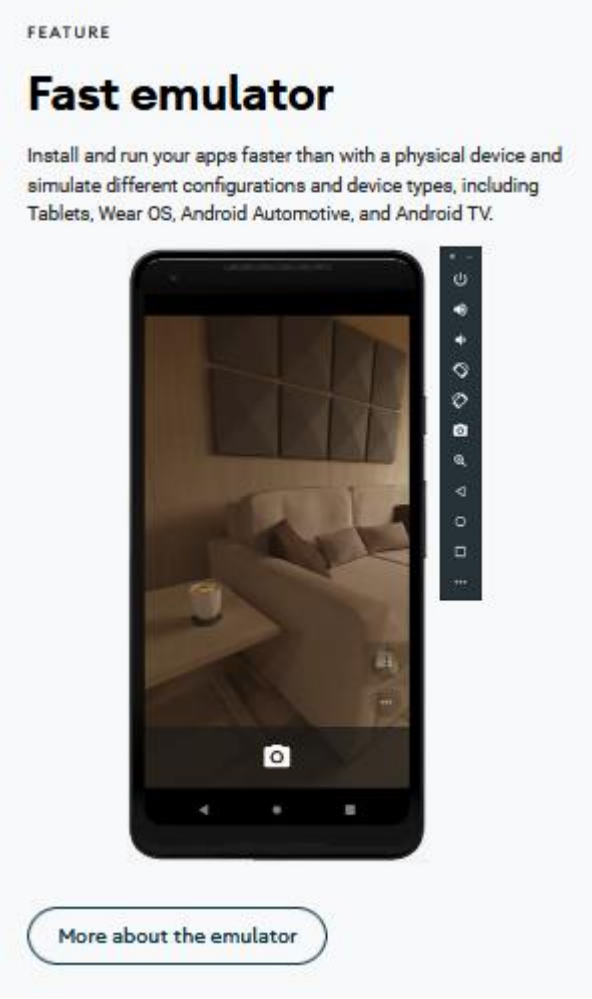
Emulateur Android

- L'émulateur Android simule des appareils Android sur ordinateur afin de tester l’application sur une variété d'appareils et de niveaux d'API Android sans avoir besoin de posséder chaque appareil physique.
- L'émulateur offre presque toutes les fonctionnalités d'un véritable appareil Android. Permet de simuler des appels téléphoniques et des messages texte entrants, spécifier l'emplacement de l'appareil, simuler différentes vitesses de réseau, simuler la rotation et d'autres capteurs matériels, accéder au Google Play Store, et bien plus encore.
- L'émulateur Android a des exigences supplémentaires au-delà de la configuration de base requise pour Android Studio, qui sont décrites ci-dessous :
    - SDK Tools 26.1.1 ou supérieur ;
    - Processeur 64 bits ;
    - Windows : CPU avec support UG (unrestricted guest).

    HAXM 6.2.1 ou supérieur (HAXM 7.2.0 ou supérieur recommandé)

Aperçu de l'installation de l'émulateur

- Mac, Windows, ou Linux
- Téléchargez et installez Android Studio à partir de: https://developer.android.com/studio/
- **Voir TP 1 : Android Studio et Hello World**



CHAPITRE n° 2

S’initier à Android Studio

Ce que vous allez apprendre dans ce chapitre :

- Découvrir l’interface d’Android Studio
- Créer un projet
- Comprendre la structure d’un projet Android
- Lancer une application Android (sur un émulateur, sur un vrai device)
- Déboguer et lire les logs

5 heures

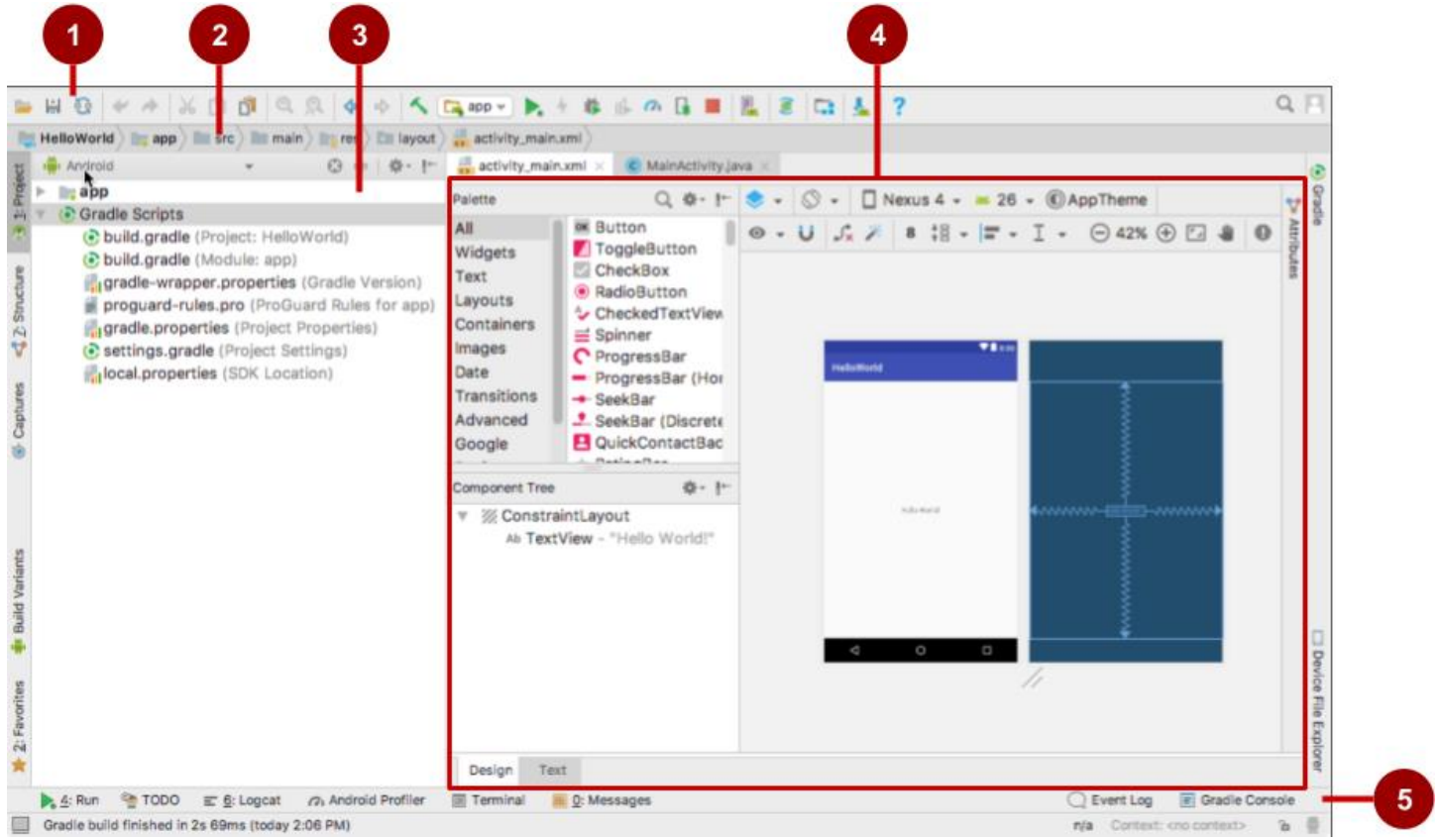
CHAPITRE n° 2

S’initier à Android Studio

1. **Interface d’Android Studio**
2. Création de projet
3. Structure d’un projet Android
4. Lancer une application Android (sur un émulateur, sur un vrai device)
5. Déboguer et lire les logs.

PARTIE 2

Aperçu



1. Barre d'outils ;
2. Barre de navigation ;
3. Volet du projet ;
4. Éditeur ;
5. Onglets pour les autres volets.

CHAPITRE n° 2

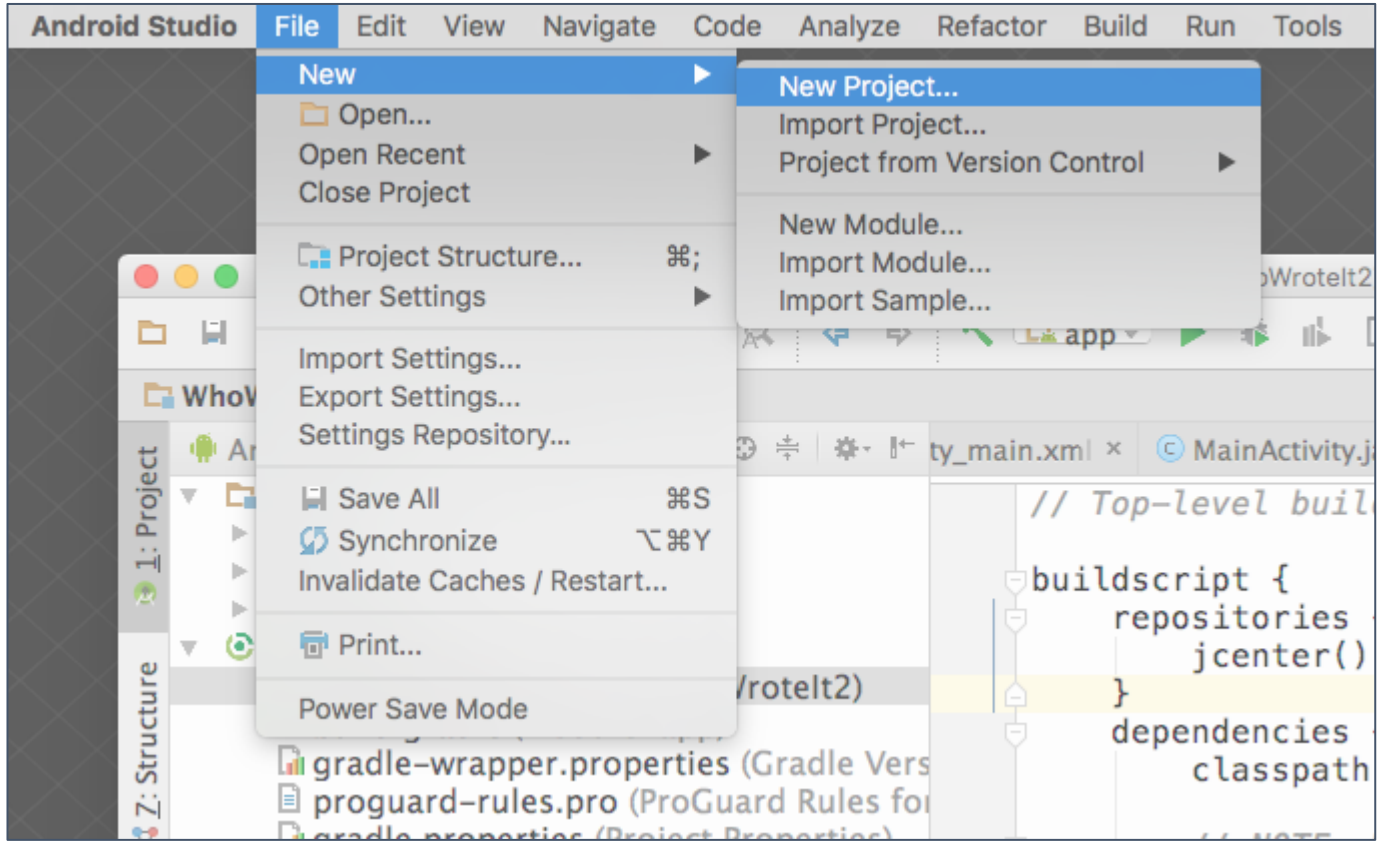
S’initier à Android Studio

1. Interface d’Android Studio
2. **Création de projet**
3. Structure d’un projet Android
4. Lancer une application Android (sur un émulateur, sur un vrai device)
5. Déboguer et lire les logs.

Démarrer Android Studio

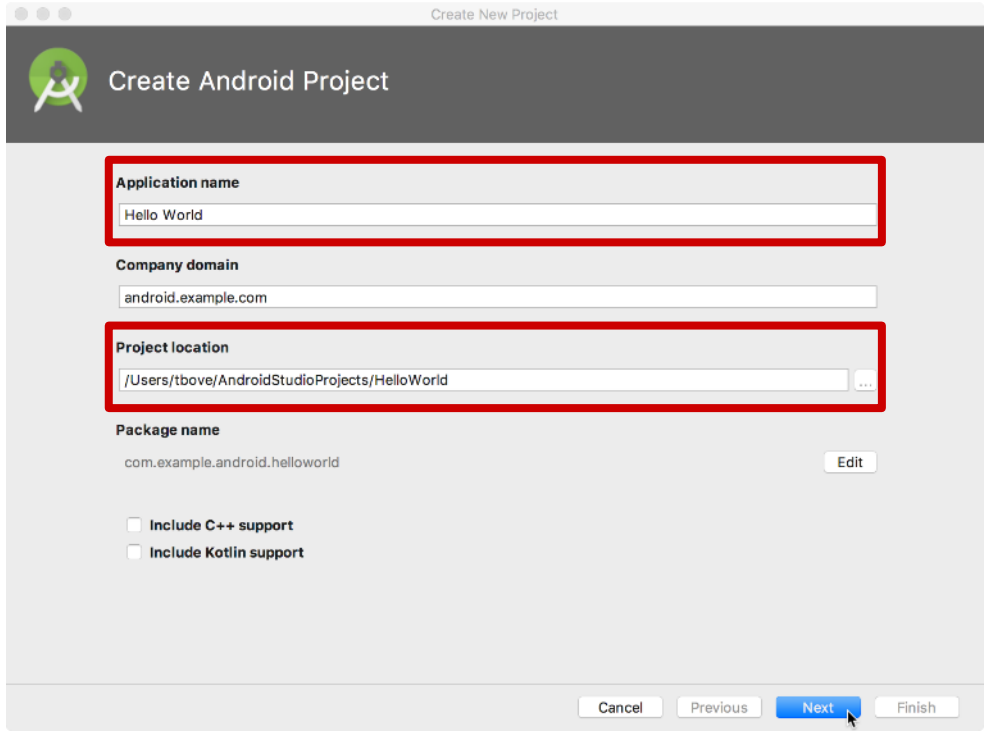
PARTIE 2

Créer un projet dans Android Studio



PARTIE 2

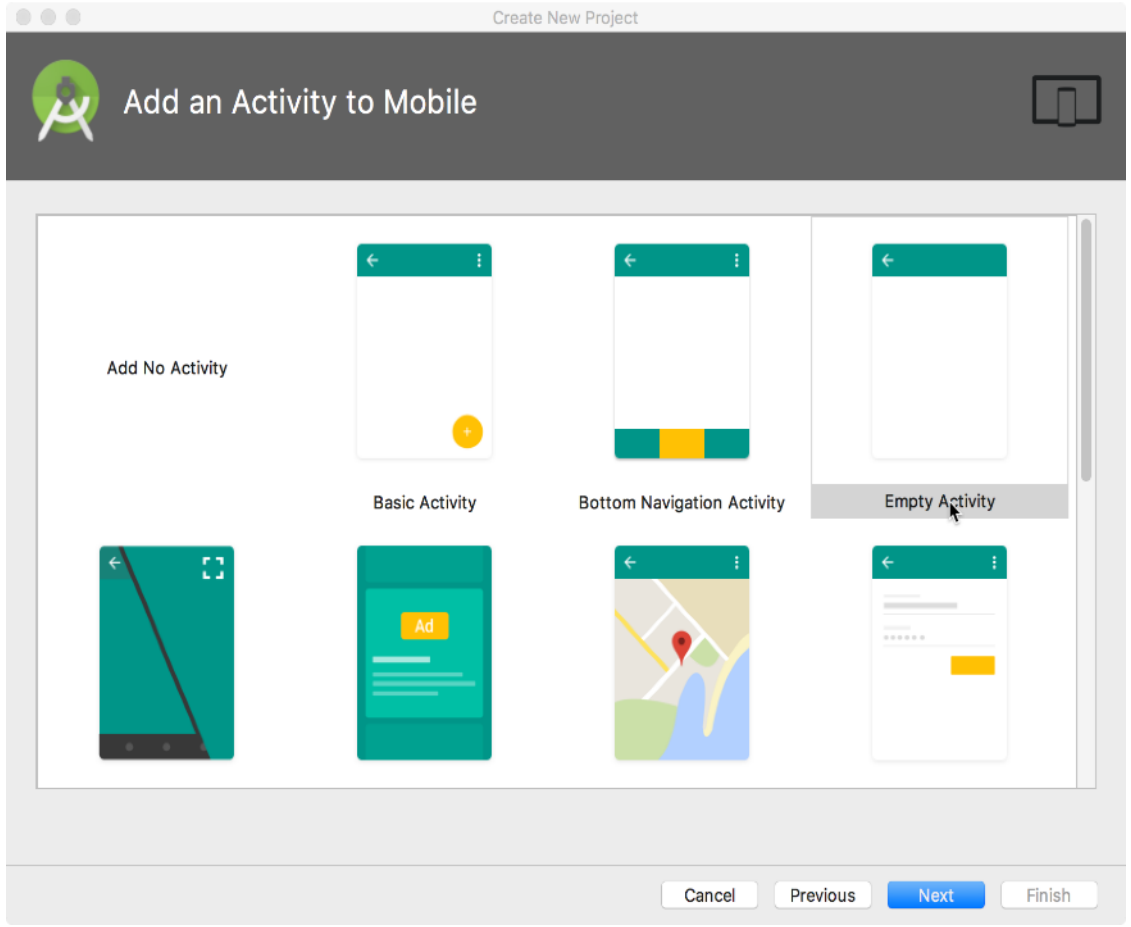
Nommez votre application



Modèle d'activité de sélection

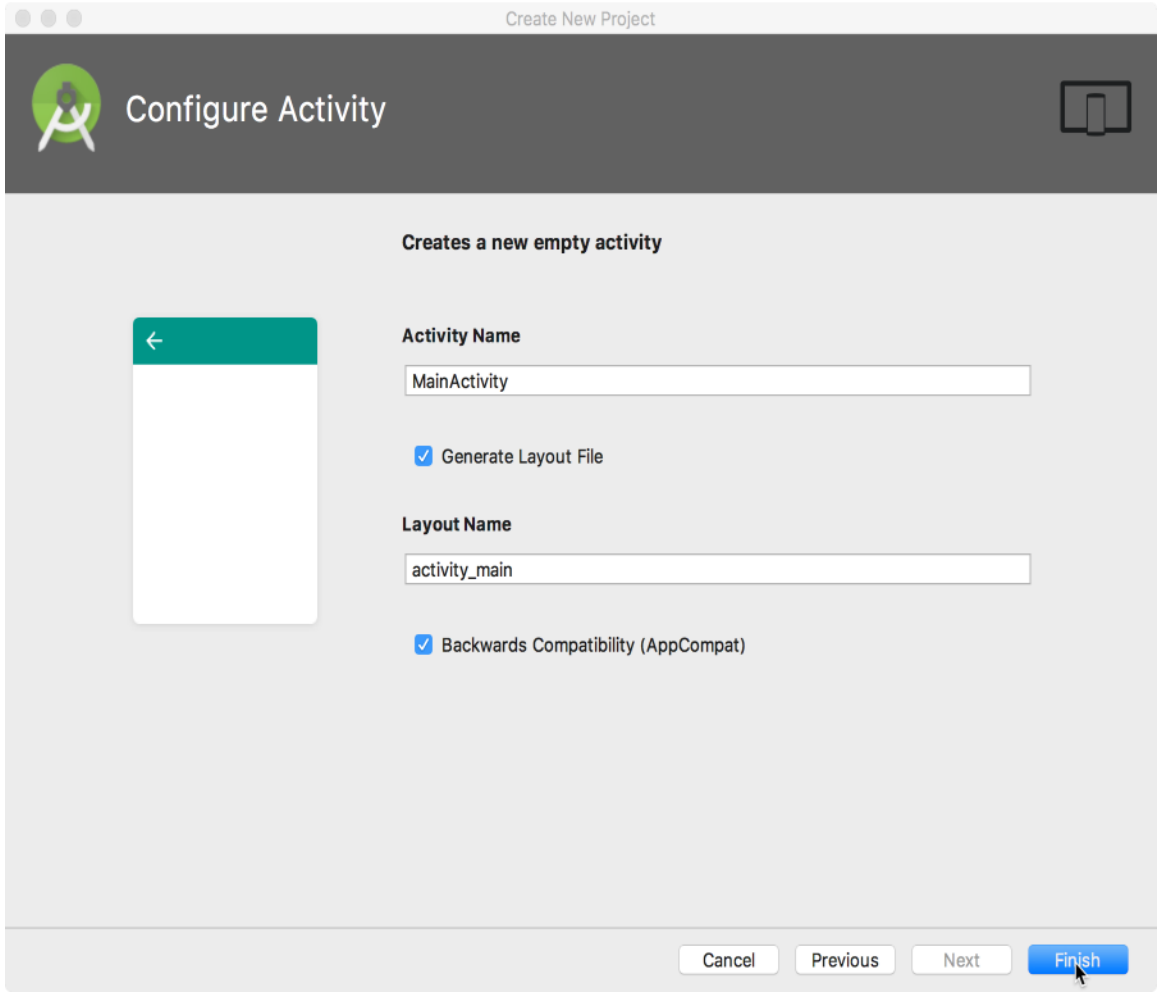
- Choisissez des modèles pour les activités courantes, comme les cartes ou les tiroirs de navigation ;

- Choisissez Activité vide ou Activité de base pour les activités simples et personnalisées.



Nommez votre activité

- Bonne pratique :
    - Nommez l'activité principale MainActivity ;
    - Nommez la mise en page activity_main.
- Utiliser AppCompat
- Générer un fichier de mise en page est pratique



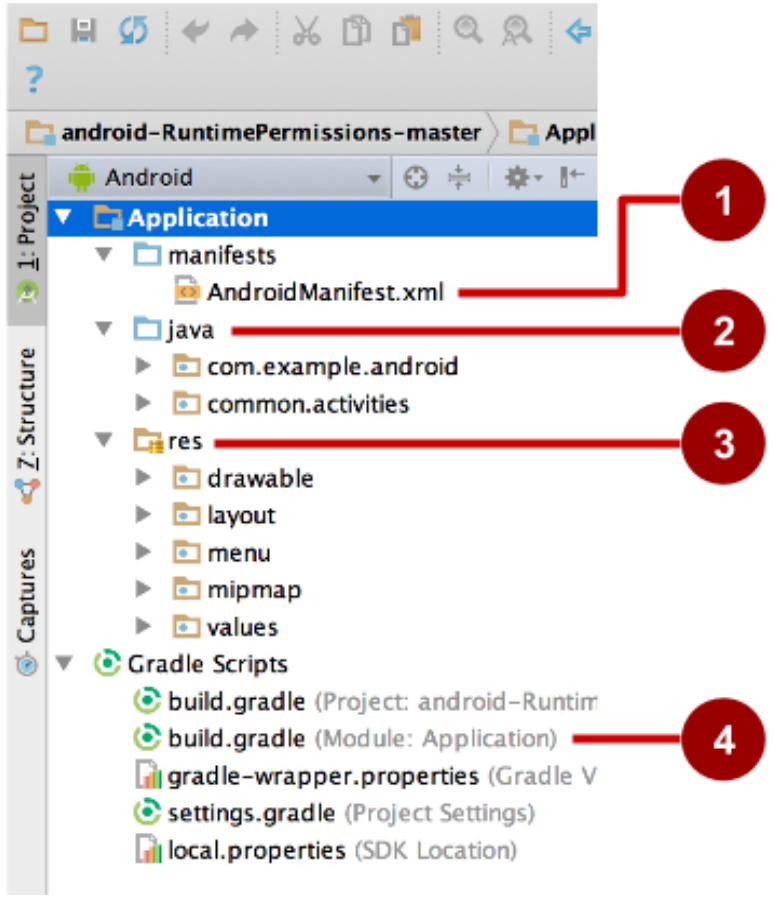
CHAPITRE n° 2

S’initier à Android Studio

1. Interface d’Android Studio
2. Créer un projet
3. **Structure d’un projet Android**
4. Lancer une application Android (sur un émulateur, sur un vrai device)
5. Déboguer et lire les logs.

Nommez votre activité

1. **Manifests -** Fichier manifeste Android - description de l'application lue par le runtime Android;
2. **java** - Paquets de code source Java;
3. **res** - Ressources (XML) - mise en page, chaînes de caractères, images, dimensions, couleurs...;
4. **build.gradle** - Fichiers de construction Gradle.



PARTIE 2

Système de construction Gradle

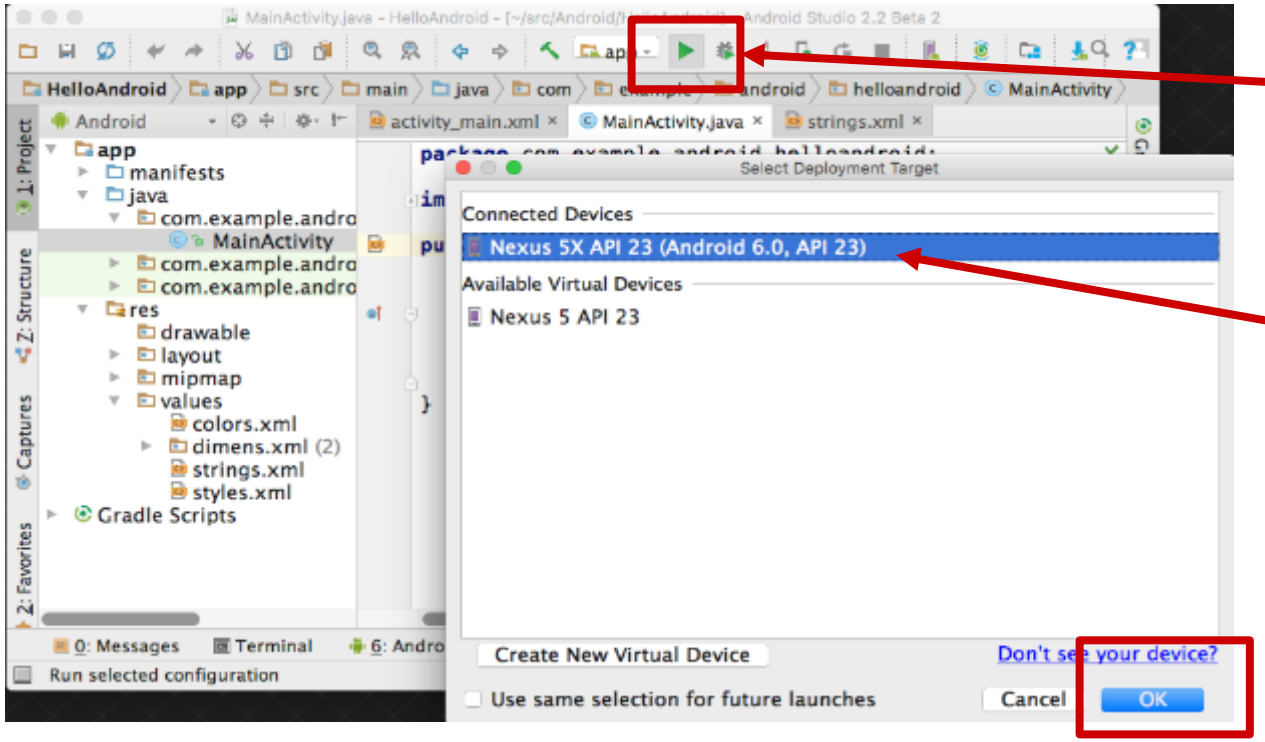
- Sous-système de construction moderne dans Android Studio.
- Trois build.gradle :
  - projet ;
  - module ;
  - paramètres.;
- Il n'est généralement pas nécessaire de connaître les détails de bas niveau de Gradle.
- Pour en savoir plus sur gradle, consultez le site https://gradle.org/.

CHAPITRE n° 2

S’initier à Android Studio

1. Interface d’Android Studio
2. Créer un projet
3. La structure d’un projet Android
4. **Lancer une application Android (sur un émulateur, sur un vrai device)**
5. Déboguer et lire les logs.

Exécuter l'application



1. Exécuter

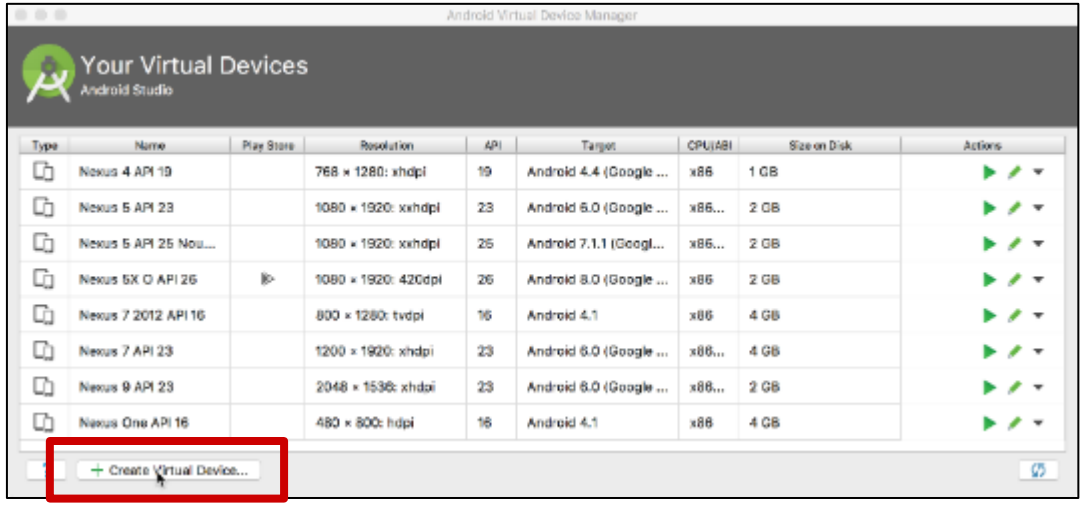
2. Sélectionner le dispositif virtuel ou physique

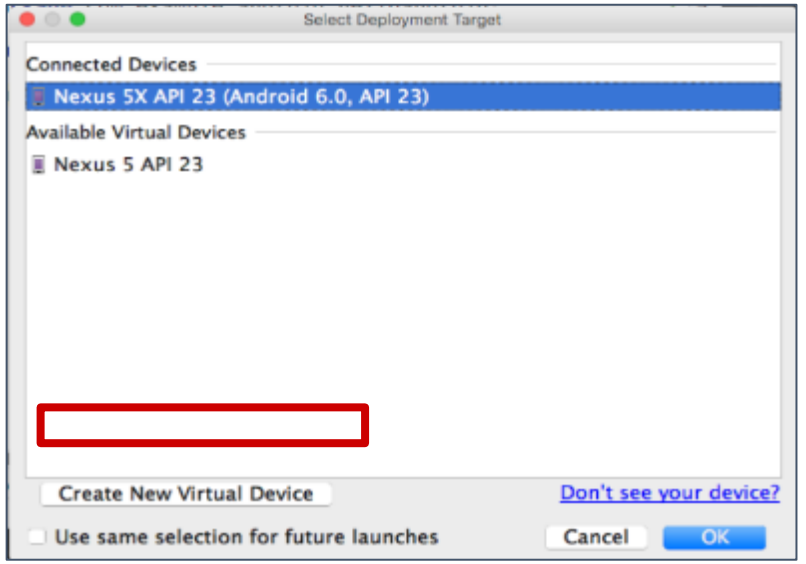
OK

Créer un dispositif virtuel

- Utilisez des émulateurs pour tester l'application sur différentes versions d'Android et différents facteurs de forme.

Tools > Android > AVD Manager **ou :**

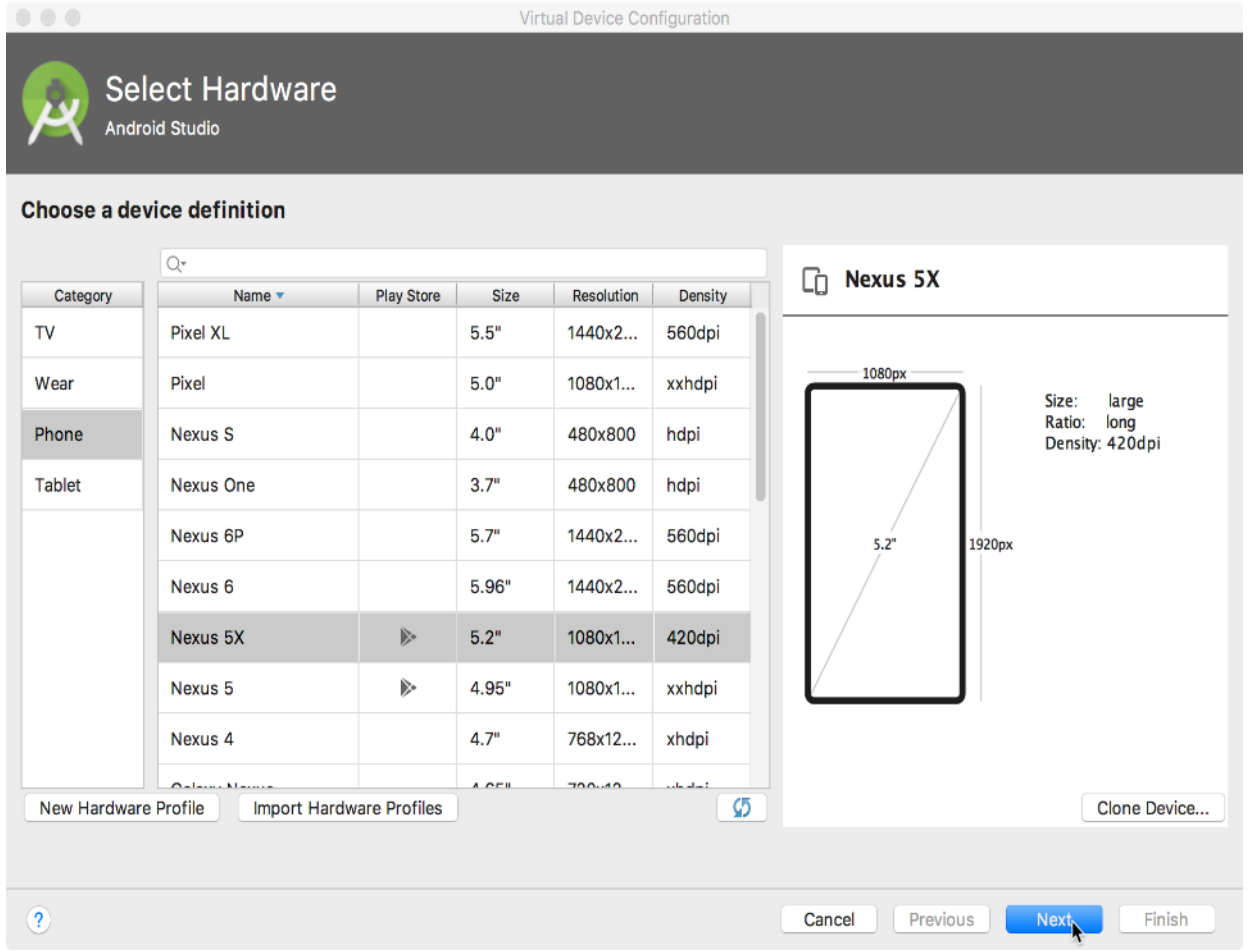




PARTIE 2

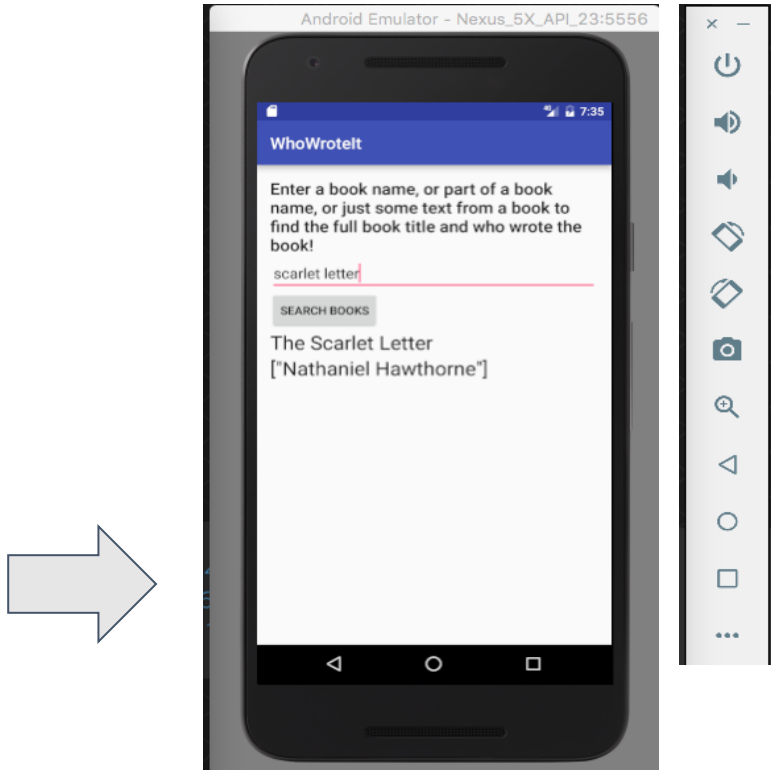
Configurer le dispositif virtuel

- Choisir le matériel ;
- Choisir la version d'Android ;
- Finaliser.



PARTIE 2

Exécuter sur un dispositif virtuel



Exécution sur un dispositif physique

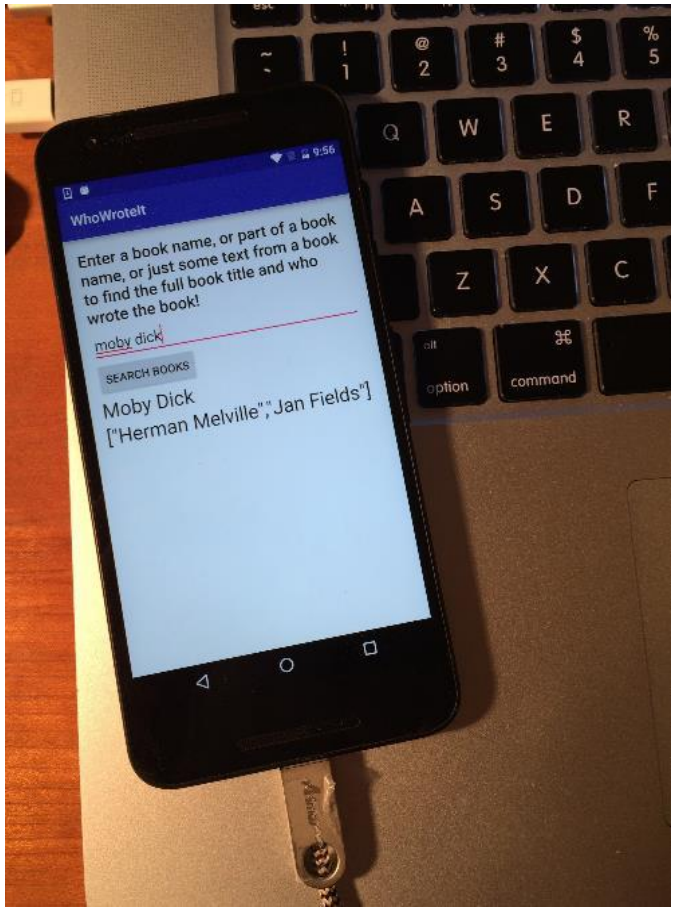
1. Activez les options du développeur :
   a. **Paramètres > À propos du téléphone**
   b. Appuyez sept fois sur le **numéro de construction (Build number)**
2. Activez le débogage USB :
   a. Réglages > Options du développeur > Débogage USB
   (En Anglais : **Settings > Developer Options > USB Debugging**)
3. Connecter le téléphone à l'ordinateur avec un câble.

Configuration supplémentaire de Windows/Linux :

- Utilisation de périphériques matériels : http://developer.android.com/tools/device.html

Pilotes Windows :

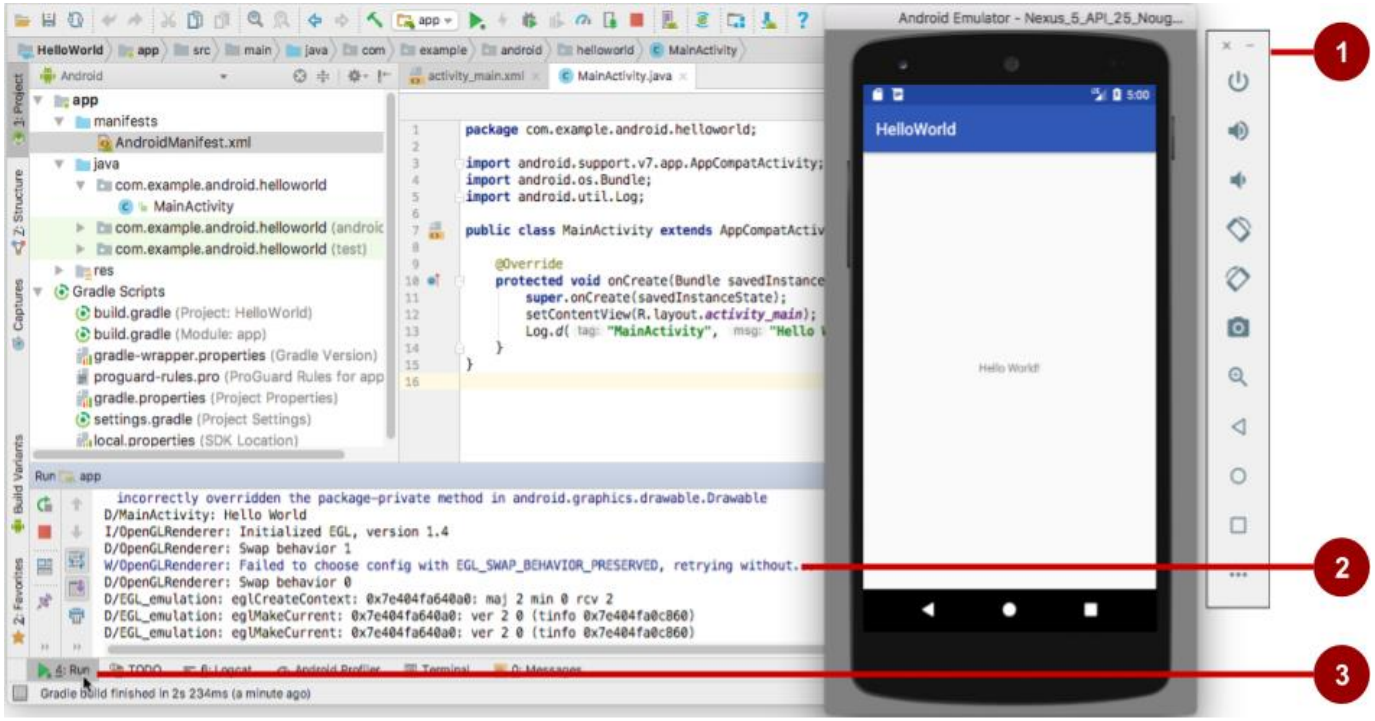
- Pilotes USB OEM : http://developer.android.com/tools/extras/oem-usb.html



PARTIE 2

Obtenez un retour d'information pendant l'exécution de votre application

1. Emulateur exécutant l'application ;
2. Volet d'exécution ;
3. Onglet "Exécuter" pour ouvrir ou fermer le volet "Exécuter".



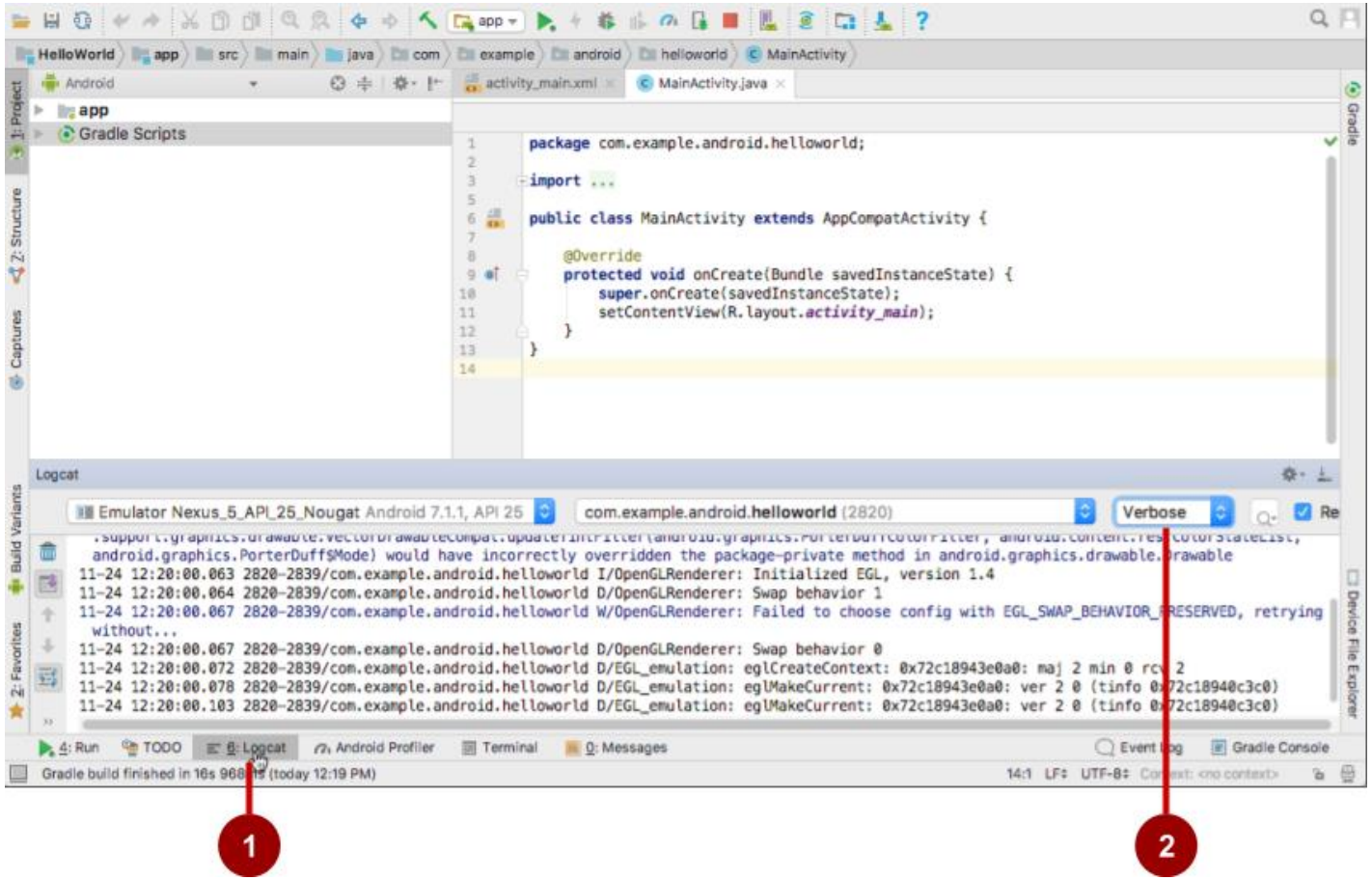
Ajouter la journalisation à votre application

1. Lorsque l'application s'exécute, le volet **Logcat** affiche des informations ;
2. Ajoutez à votre application des instructions de journalisation qui apparaîtront dans le volet **Logcat ;**
3. Définissez des filtres dans le volet **Logcat** pour voir ce qui est important pour vous ;
4. Recherche à l'aide de balises.

Le volet Logcat

1. Onglet Logcat pour afficher le volet Logcat

2. Menu du niveau du journal



Déclaration de journalisation

```
import android.util.Log;

// Utiliser le nom de la classe comme balise
private static final String TAG =
    MainActivity.class.getSimpleName();

// Afficher le message dans l'Android Monitor, volet logcat
// Log.<log-level>(TAG, "Message");
Log.d(TAG, "Creating the URI… ");
```

OFPPT
La Voie de l'Avenir

WEBFORCE
BE THE CHANGE

CHAPITRE n° 2
S'initier à Android Studio

1. Interface d'Android Studio
2. Créer un projet
3. La structure d'un projet Android
4. Lancer une application Android (sur un émulateur, sur un vrai device)
5. **Déboguer et lire les logs**

Bogues

- Résultat incorrect ou inattendu, valeurs erronées.
- Crashes, exceptions, blocages, fuites de mémoire.
- Causes :
    - Erreur de conception ou d'implémentation humaine > Corrigez votre code ;
    - Défaut logiciel, mais dans les bibliothèques > Contournez la limitation ;
    - Défaut ou limitation du matériel -> Faites fonctionner le système avec ce qui est disponible.

Débogage

- Trouver et corriger les erreurs ;
- Corriger un comportement inattendu et indésirable ;
- Les tests unitaires permettent d'identifier les bogues et d'éviter la régression ;
- Les tests utilisateurs permettent d'identifier les bogues d'interaction.

Outils de débogage Android Studio

Android Studio possède des outils qui vous aident à :

- Identifier les problèmes ;
- Trouver où dans le code source le problème est créé ;
- Afin que vous puissiez le corriger.

Ajouter des messages de journal à votre code

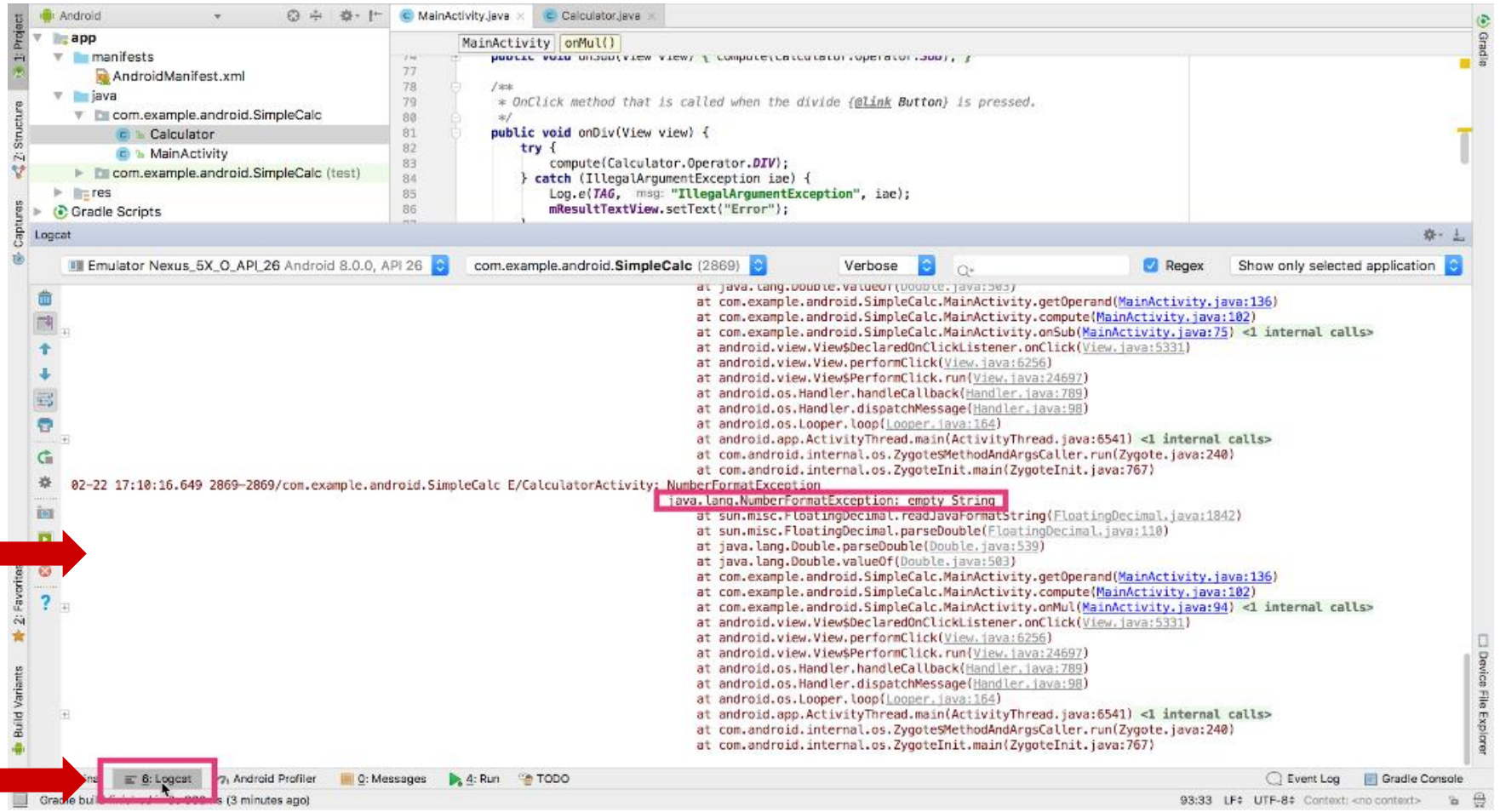
```
import android.util.Log;

// Utiliser le nom de la classe comme balise
private static final String TAG =
    MainActivity.class.getSimpleName();

// Afficher le message dans l'Android Monitor, volet logcat
// Log.<log-level>(TAG, "Message");
Log.d(TAG, "Hello world");
```

PARTIE 2

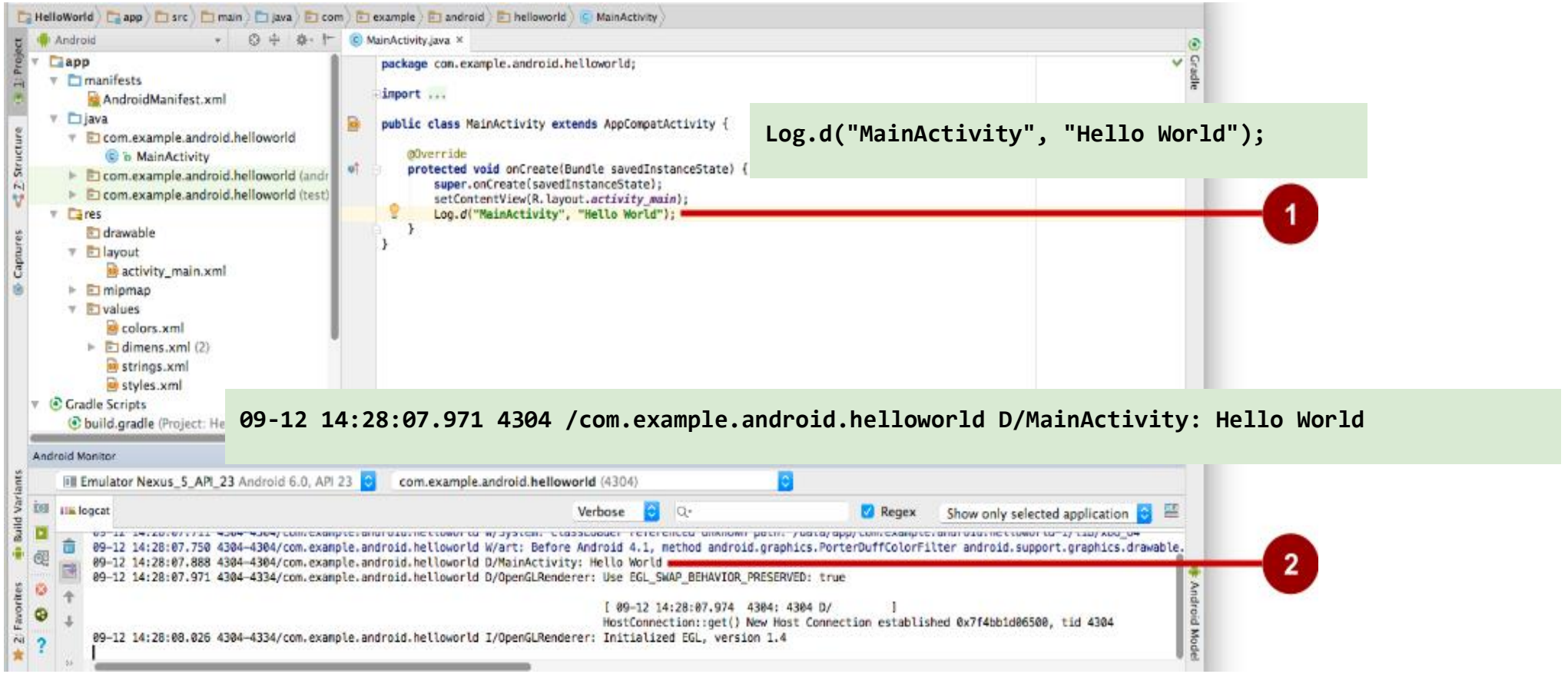
Ouvrir le volet Logcat

Panel Logcat

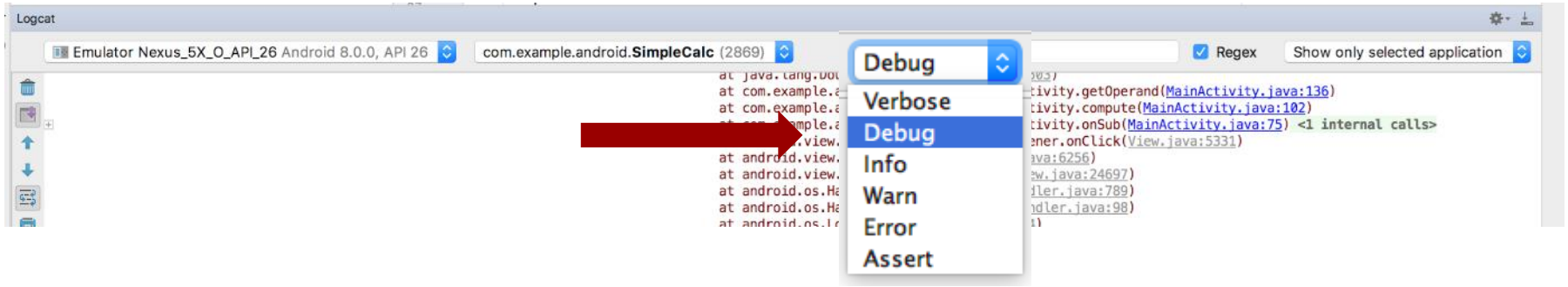
Onglet Logcat



Inspecter les messages de journalisation



Choisissez le niveau de journalisation visible



Le niveau de journalisation affiche les journaux dont le niveau est égal ou supérieur au niveau sélectionné.

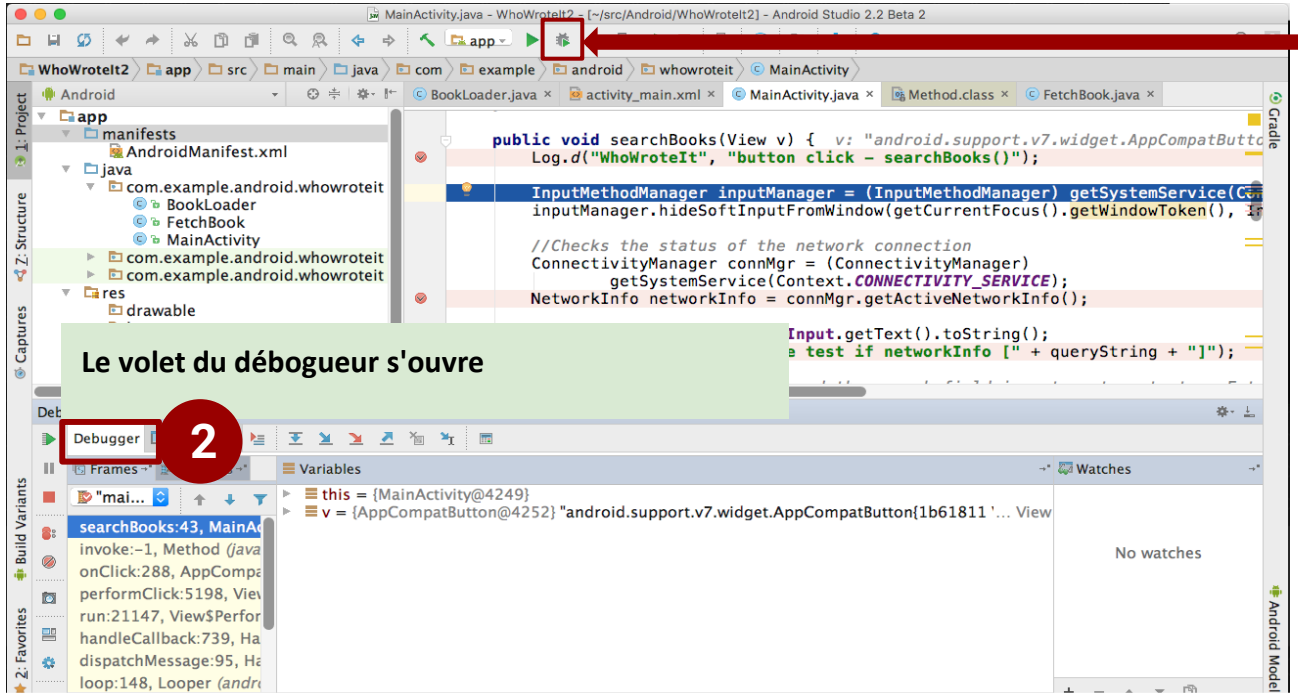
Niveaux du journal

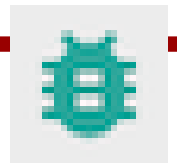
- **Verbose** - Toutes les déclarations de journal verbeux et le système complet ;
- **Debug** - Tous les journaux de débogage, les valeurs des variables, les notes de débogage ;
- **Info** - Informations d'état, telles que la connexion à la base de données ;
- **Warning** - Comportement inattendu, problèmes non fatals ;
- **Error** - Conditions d'erreur sérieuses, exceptions, plantages uniquement.

Débogage avec Android Studio

- Exécution en mode débogage avec débogueur intégré ;
- Définir et configurer des points d'arrêt ;
- Arrêter l'exécution aux points d'arrêt ;
- Inspecter les cadres de la pile d'exécution et les valeurs des variables ;
- Modifier les valeurs des variables ;
- Parcourir le code ligne par ligne ;
- Pause et reprise d'un programme en cours d'exécution.

Exécuter en mode débogage



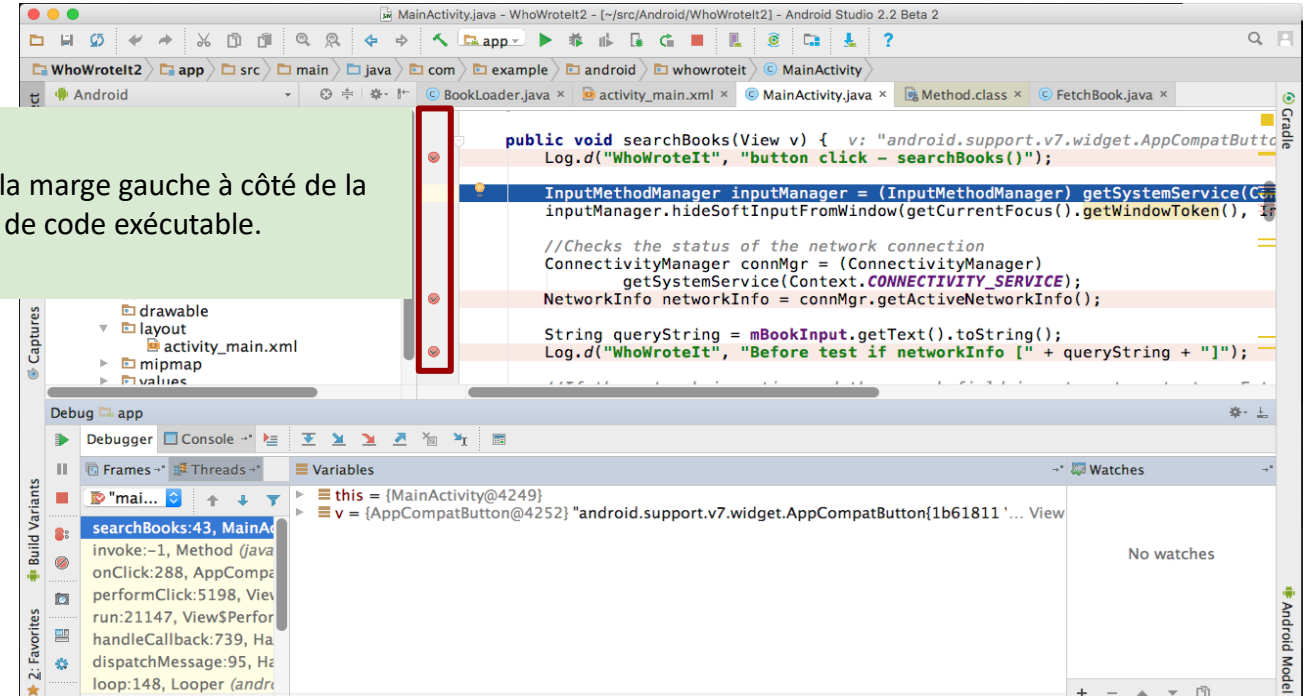


1

Menu:
Run > Debug 'your app'

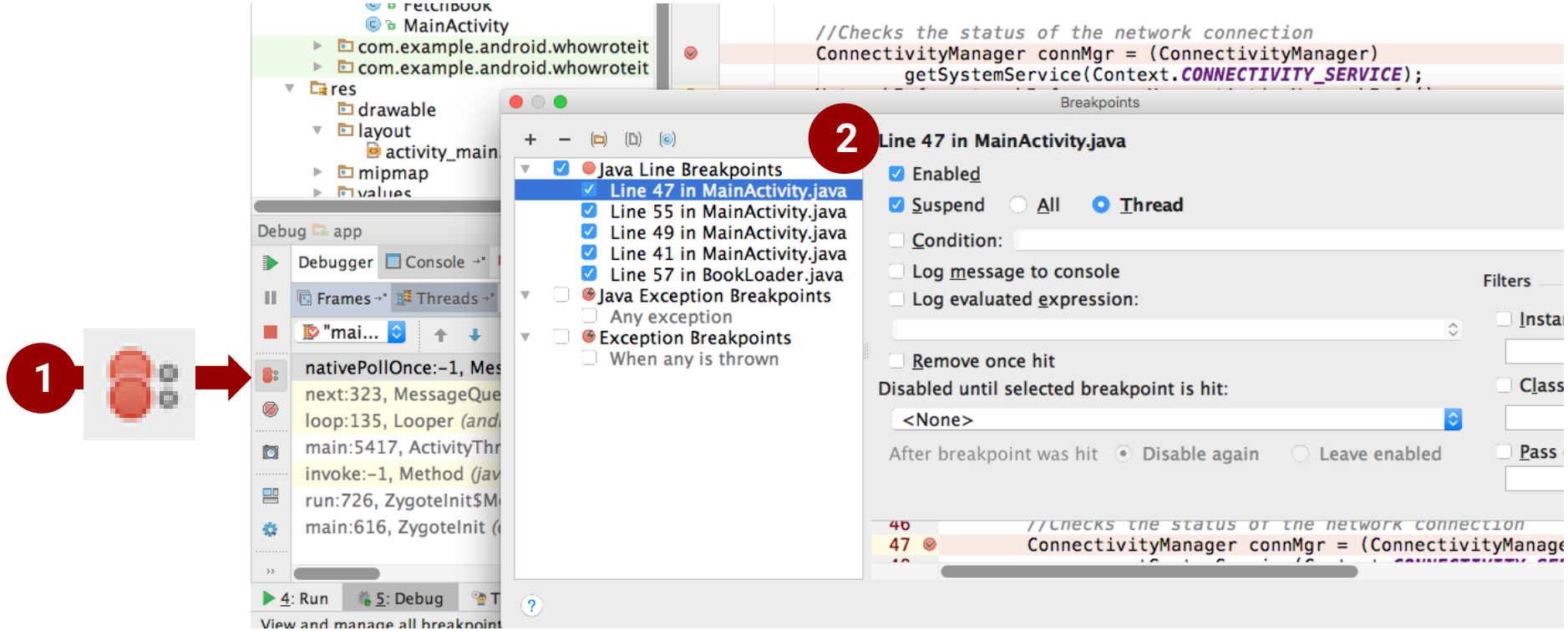
Définir des points d'arrêt

Cliquez dans la marge gauche à côté de la ligne de code exécutable.



PARTIE 2

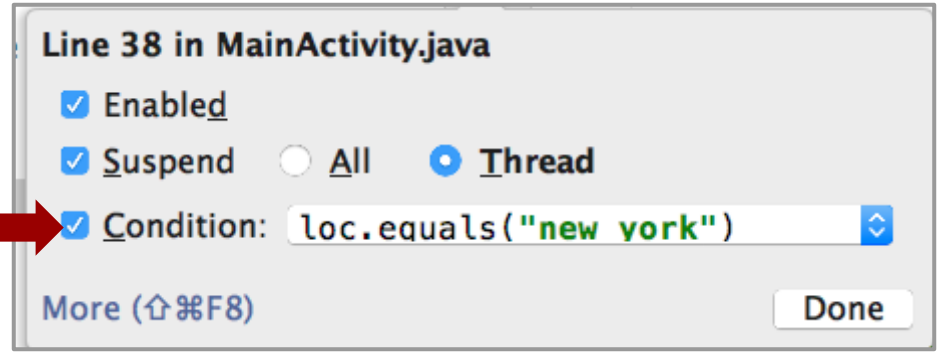
Modifier les propriétés du point d'arrêt



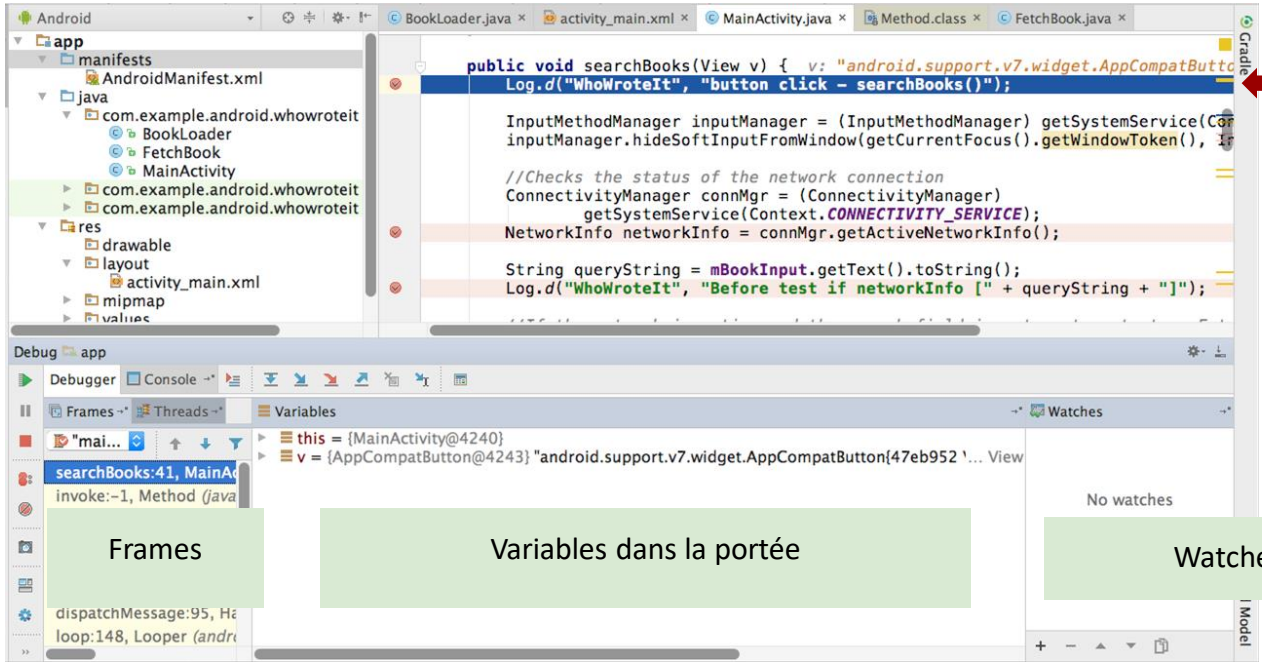
PARTIE 2

Rendre les points d'arrêt conditionnels

- Dans la boîte de dialogue des propriétés ou en cliquant avec le bouton droit de la souris sur le point d'arrêt existant ;
- Toute expression Java qui renvoie un booléen ;
- La complétion de code vous aide à écrire les conditions.

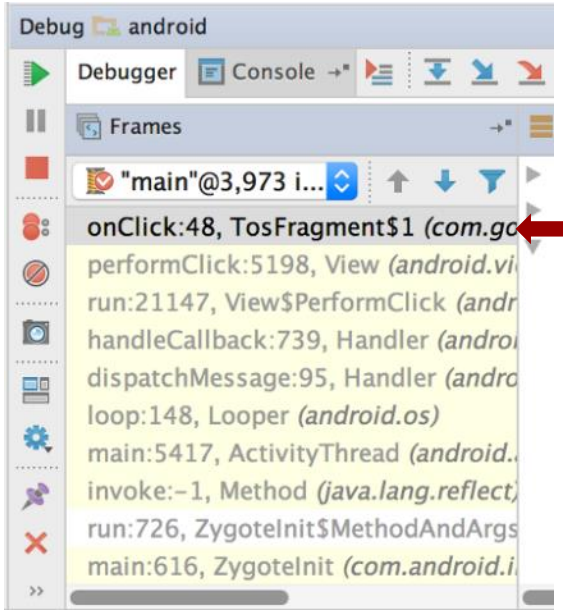


Exécuter jusqu'à ce que l'application s'arrête au point d'arrêt



PARTIE 2

Inspecter frames

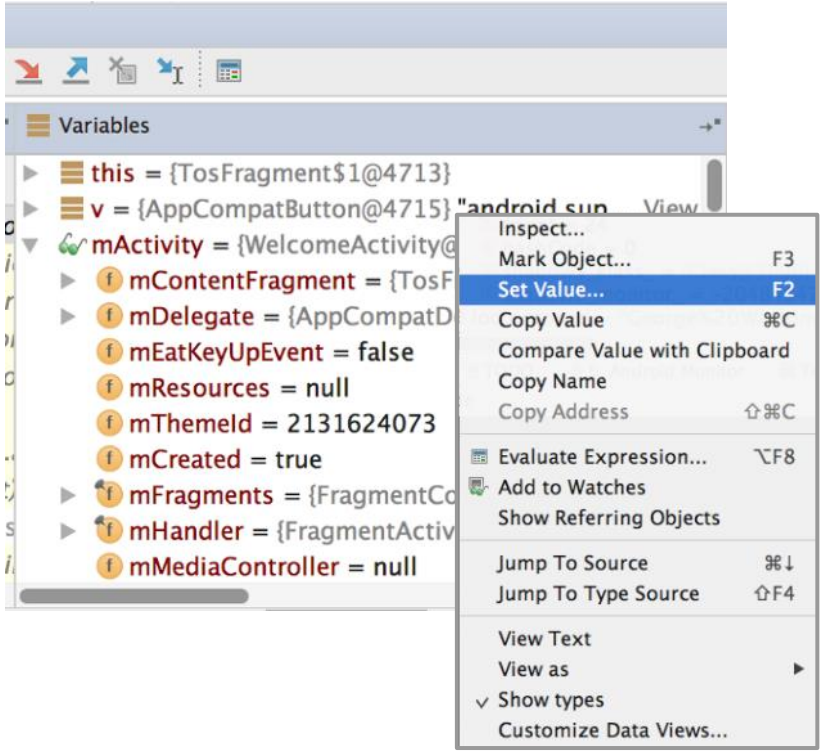


Le cadre supérieur est l'endroit où l'exécution est arrêtée dans votre code.

PARTIE 2

Inspecter et modifier les variables

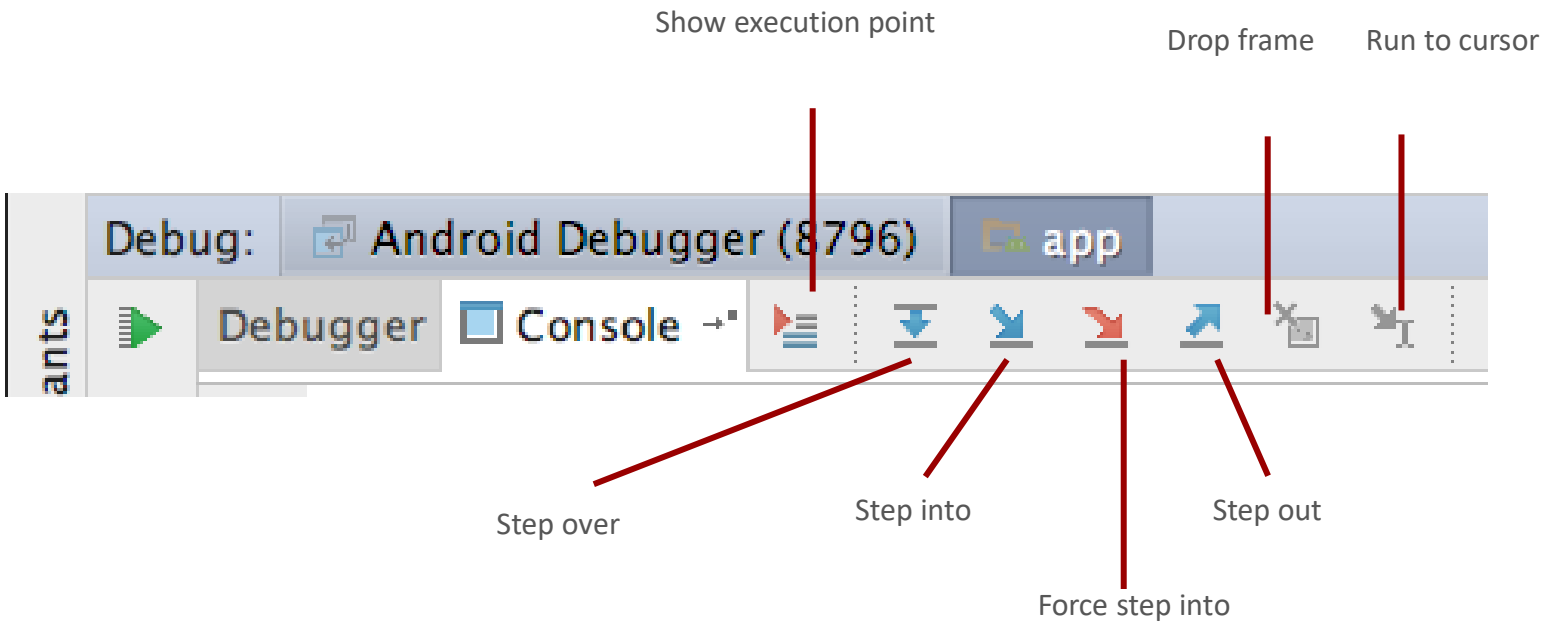
Clic droit sur la variable pour afficher le menu.



Commandes de base pour les pas

Step Over	F8	Passer à la ligne suivante dans le fichier courant
Step Into	F7	Passer à la ligne exécutée suivante
Force Step Into	⇧F7	Entrez dans une méthode d'une classe dans laquelle vous n'entrez pas normalement, comme une classe standard du JDK.
Step Out	⇧F8	Passer à la première ligne exécutée après le retour de la méthode actuelle
Run to Cursor	⌥F9	Aller à la ligne où se trouve le curseur dans le fichier

Se déplacer dans le code

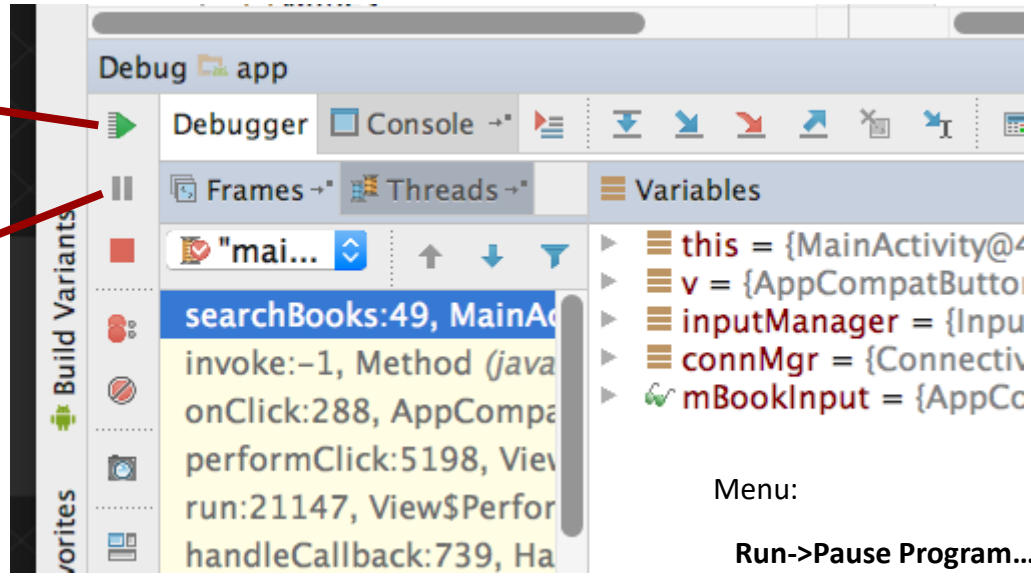


Reprise et pause

Resume

Pause

Mute all breakpoints



Menu:

Run->Pause Program…
Run->Resume Program...

PARTIE 2

CHAPITRE n°3

Maîtriser la génération d’une application mobile

Ce que vous allez apprendre dans ce chapitre :

- Fonctionnement de build d’une application mobile
- Manipulation des flavors (release, debug…)

4 heures

OFPPT
La Voie de l'Avenir

WEBFORCE
BE THE CHANGE

CHAPITRE n° 3

Maîtriser la génération d'une application mobile

1. Fonctionnement de build d'une application mobile
2. Manipulation des flavors (release, debug…)

Configuration de build

- Le système de construction d'Android compile les ressources de l'application et le code source, et les regroupe dans des APK ou des paquets d'applications Android que vous pouvez tester, déployer, signer et distribuer.
- Android Studio utilise Gradle, une boîte à outils de construction avancée, pour automatiser et gérer le processus de construction, tout en vous permettant de définir des configurations de construction personnalisées et flexibles
- Chaque configuration de construction peut définir son propre ensemble de code et de ressources, tout en réutilisant les parties communes à toutes les versions de votre application.
- Le plugin Android pour Gradle fonctionne avec la boîte à outils de construction pour fournir des processus et des paramètres configurables qui sont spécifiques à la construction et au test des applications Android.

Configuration de build

- Gradle et le plugin Android fonctionnent indépendamment d'Android Studio. Cela signifie que la construction d'une applications Android à partir d'Android Studio, de la ligne de commande de la machine, ou sur des machines où Android Studio n'est pas installé (comme les serveurs d'intégration continue).

- Si vous n'utilisez pas Android Studio, vous pouvez apprendre à construire et à exécuter votre application à partir de la ligne de commande. Le résultat de la construction est le même, que vous construisiez un projet à partir de la ligne de commande, sur une machine distante ou en utilisant Android Studio.

Remarques

- Comme Gradle et le plugin Android fonctionnent indépendamment d'Android Studio, vous devez mettre à jour les outils de construction séparément. Lisez les notes de version pour savoir comment mettre à jour Gradle et le plugin Android.

PARTIE 2

Le processus de construction

- Le processus de construction fait intervenir de nombreux outils et processus qui convertissent le projet en un paquet d'applications Android (APK) ou un paquet d'applications Android (AAB).
- Le processus de construction est très flexible, il est donc utile de comprendre son fonctionnement.

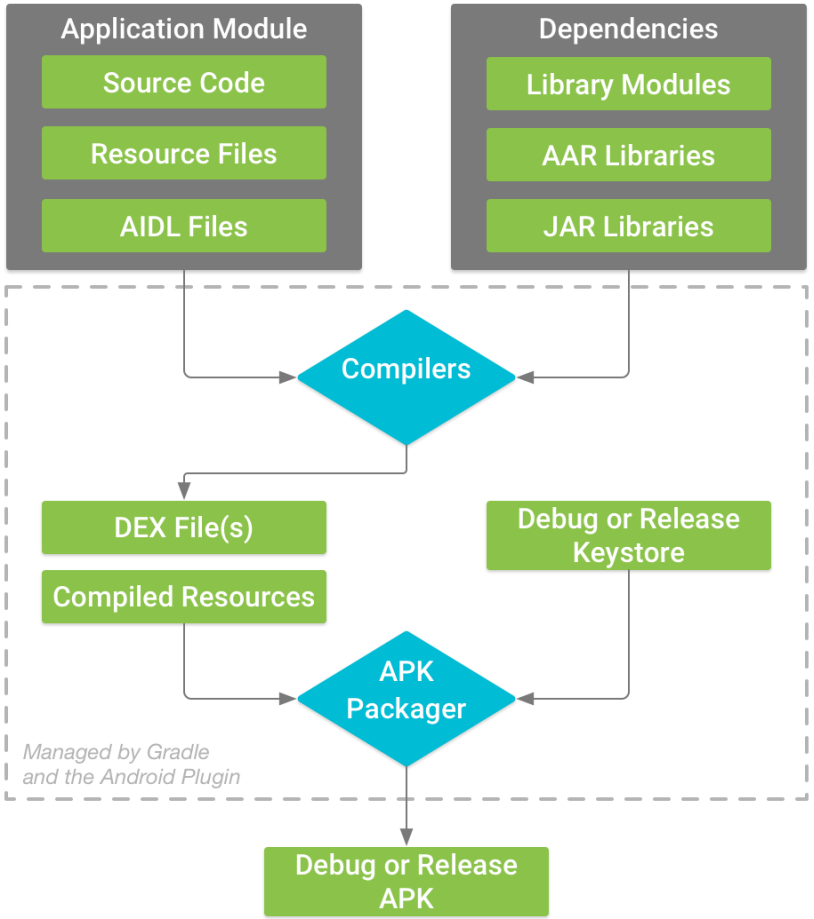


Figure 1. Le processus de construction d'un module d'application Android typique (Réf : Android dev).

PARTIE 2

Le processus de construction

Le processus de construction d'un module d'application Android typique, comme le montre la figure 1, suit les étapes générales suivantes :

1. Les compilateurs convertissent le code source en fichiers DEX (Dalvik Executable), qui incluent le bytecode qui fonctionne sur les appareils Android, et tout le reste en ressources compilées.

2. Le packager combine les fichiers DEX et les ressources compilées dans un APK ou un AAB, en fonction de la cible de construction choisie. Avant que votre application ne puisse être installée sur un appareil Android ou distribuée dans une boutique, telle que Google Play, l'APK ou l'AAB doit être signé.

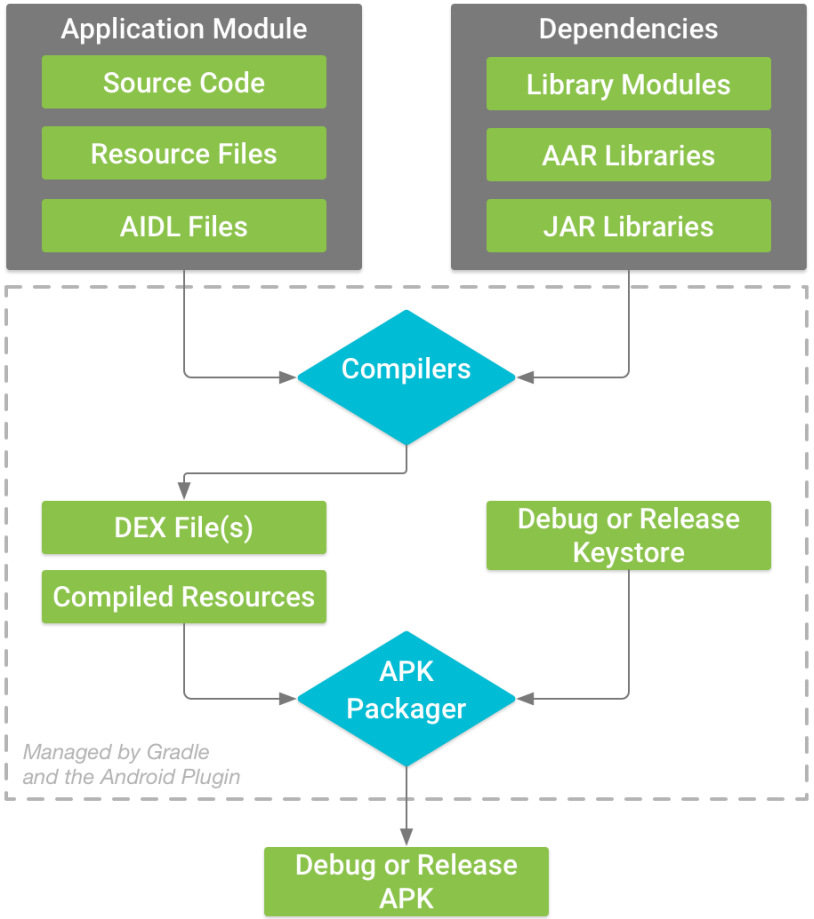


Figure 1. Le processus de construction d'un module d'application Android typique (Réf : Android dev).

Le processus de construction

3. Le packager signe l'APK ou AAB en utilisant le keystore debug ou release :

   a. Si vous créez une version de débogage de votre application, c'est-à-dire une application destinée uniquement à être testée et profilée, le packager signe votre application avec le keystore de débogage. Android Studio configure automatiquement les nouveaux projets avec un keystore de débogage.

   b. Si vous créez une version de votre application destinée à être diffusée en externe, le packager signe votre application avec le keystore de diffusion que vous devez configurer. Pour créer un keystore de version, lisez la section consacrée à la signature de l'application dans Android Studio.

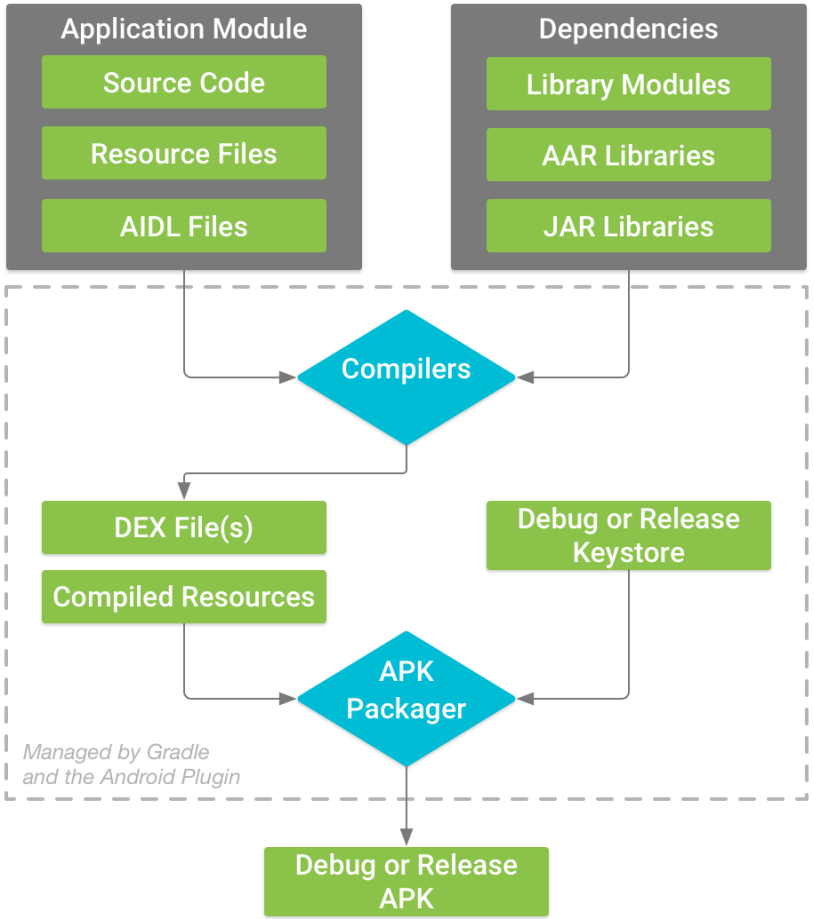


Figure 1. Le processus de construction d'un module d'application Android typique (Réf : Android dev).

Le processus de construction

4. Avant de générer la APK final, le packager utilise l'outil zipalign pour optimiser votre application afin qu'elle utilise moins de mémoire lors de son exécution sur un appareil.

À la fin du processus de construction, vous disposez d'un APK ou d'un AAB de débogage ou de publication de votre application que vous pouvez utiliser pour déployer, tester ou publier auprès d'utilisateurs externes.

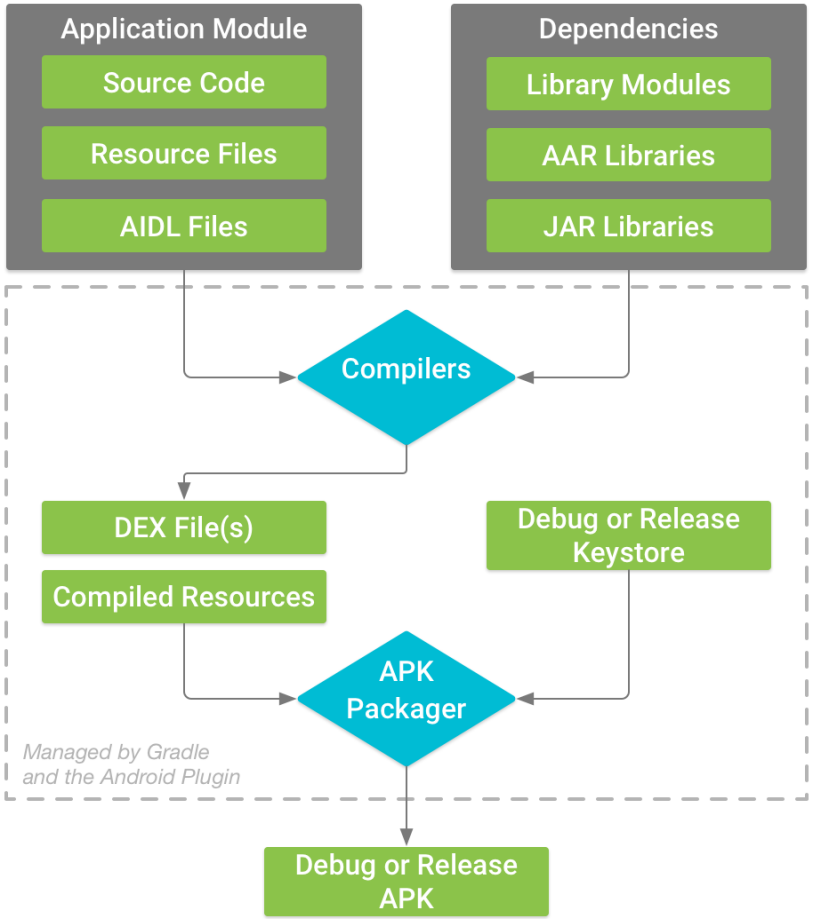


Figure 1. Le processus de construction d'un module d'application Android typique (Réf : Android dev).

Génération de l'APK

Android Studio vous permet de créer deux types de fichiers APK :

- Les premiers sont les fichiers **APK de débogage (**Unsigned APK**)** qui sont générés uniquement à des fins de test. Ils fonctionneront sur votre mobile Android. Cependant, ils ne peuvent pas être téléchargés sur le Play Store ou mis à la disposition du public.

- Deuxièmement, vous pouvez générer des fichiers **APK signés**. Les fichiers APK signés sont utiles lorsque vous avez testé votre application et qu'elle est prête à être téléchargée sur le Play Store et mise à la disposition du grand public.

Génération de l’APK de débogage

- Unsigned APK, comme son nom l'indique, signifie qu'il n'est pas signé par un Keystore. Un Keystore est essentiellement un fichier binaire qui contient un ensemble de clés privées. Chaque application que nous installons via l'App Store de Google Play doit être signée par un Keystore. L’APK signé est simplement l’APK non signé qui a été signé via l'outil JDK jarsigner.

- Lorsque les développeurs développent une application Android pour les utilisateurs finaux, ils doivent tester l'application pour laquelle ils partagent l’APK avec leurs clients afin d'obtenir un retour d'information et d'apporter les améliorations nécessaires. Il est un peu long et difficile de générer un APK signé à chaque fois pour le partager, car cela nécessite un Keystore (unique). C'est pourquoi, la plupart du temps, nous préférons générer un APK non signé car il est facile à générer.

PARTIE 2

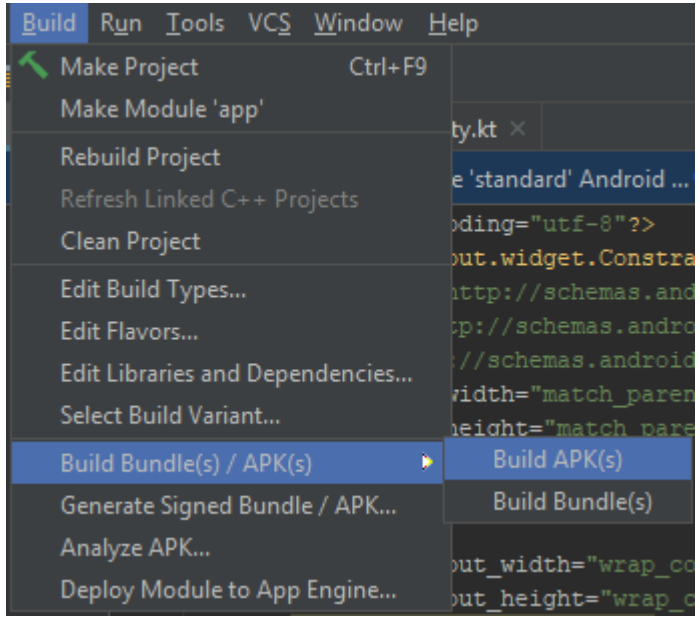
Génération de l'APK de débogage

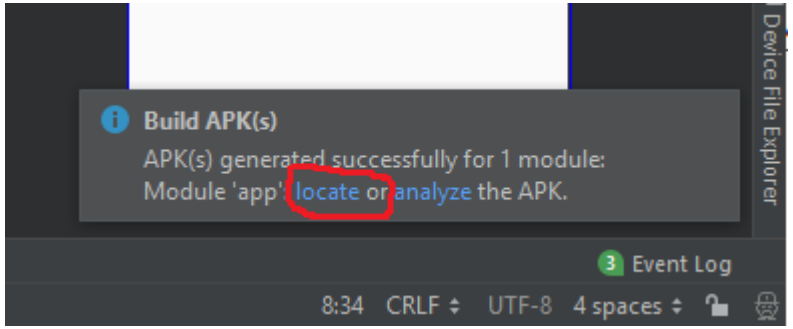
Étape 1 : Construisez votre projet puis allez dans le menu Build > Build Bundle(s)/APK(s) > Build APK(s) comme indiqué dans la 1ère image ;

Étape 2 : Vous verrez que Gradle est en train de construire. Attendez 3 à 4 minutes pour que la construction soit terminée ;

Étape 3 : Après l'exécution de la compilation Gradle, dans la partie inférieure droite, vous verrez quelque chose comme indiqué dans la 2ème image ;

Étape 4 : Cliquez ensuite sur localiser et vous verrez votre fichier Apk comme (app-debug.apk) et maintenant vous pouvez partager ce fichier Apk avec n'importe qui.





CHAPITRE n° 3

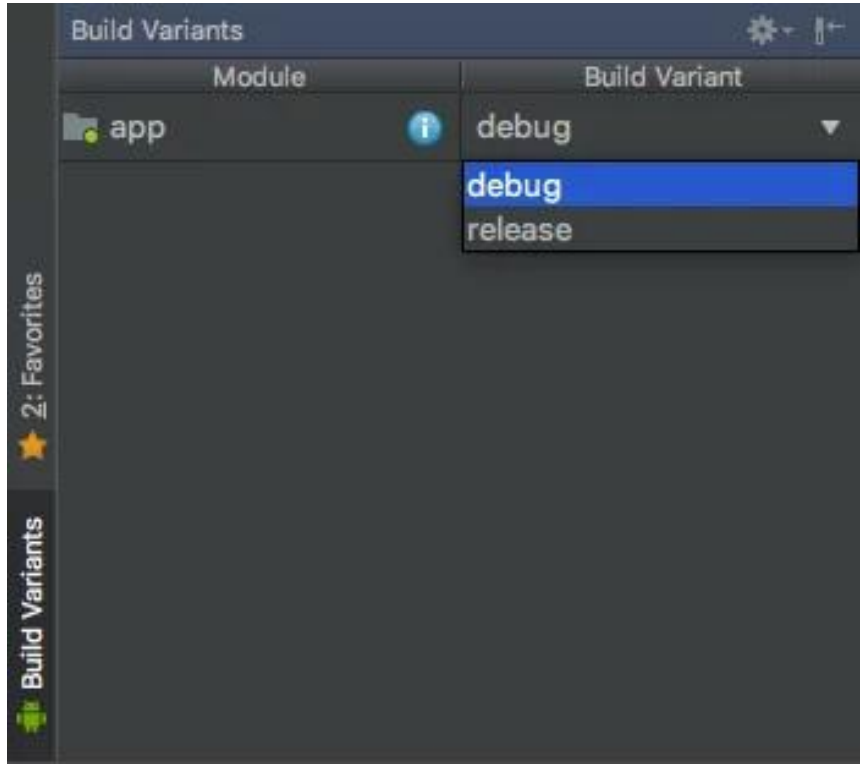
Maîtriser le débogage d’une application mobile

1. Maîtriser la génération d’une application mobile
2. **Manipulation des flavors**

Types de builds Android

Une fois le nouveau projet créé, il se compose par défaut de deux types/variantes de construction - debug, release.

- **debug** est le type de construction qui est utilisé lorsque nous exécutons l'application depuis l'IDE directement sur un appareil.
- **release** est le type de build qui nécessite de signer l'APK. Les builds release sont destinés à être téléchargés sur le play store.



Types de builds Android

- Dans le build.gradle par défaut, seul le bloc de type release build est écrit :

```
buildTypes {
      release {
         minifyEnabled false
         proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
      }
   }
```

- Définir signingConfigs dans build.gradle :

```
signingConfigs {
          release {
                    storeFile file("release-key.keystore")
                    storePassword 'password'
                    keyAlias 'alias'
                    keyPassword 'key'
          }
}
```

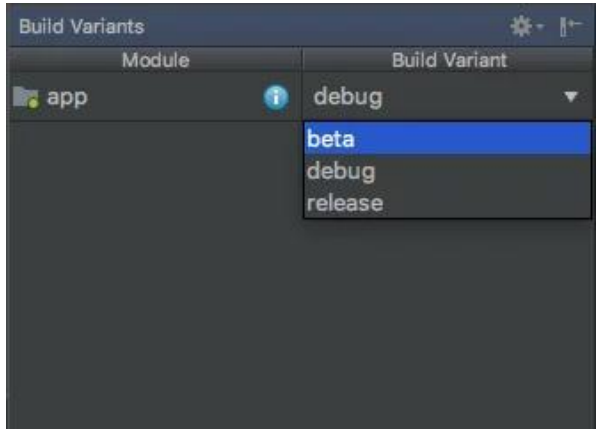
Remarques

- Assurez-vous que vous avez créé un fichier clé signé avec le nom de la clé de sortie et le mot de passe ci-dessus à partir de Build | Generate Signed APK pour que le code ci-dessus fonctionne.

Types de builds Android

- Ajoutons de nouveaux types de construction et plus de propriétés au buildConfig :

```
buildTypes {
        release {
            signingConfig signingConfigs.release
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
            'proguard-rules.pro'
        }
        debug{
            applicationIdSuffix ".debug"
            versionNameSuffix "-debug"
        }
        beta{
            signingConfig signingConfigs.release
            applicationIdSuffix ".beta"
            versionNameSuffix "-beta"
        }
}
```

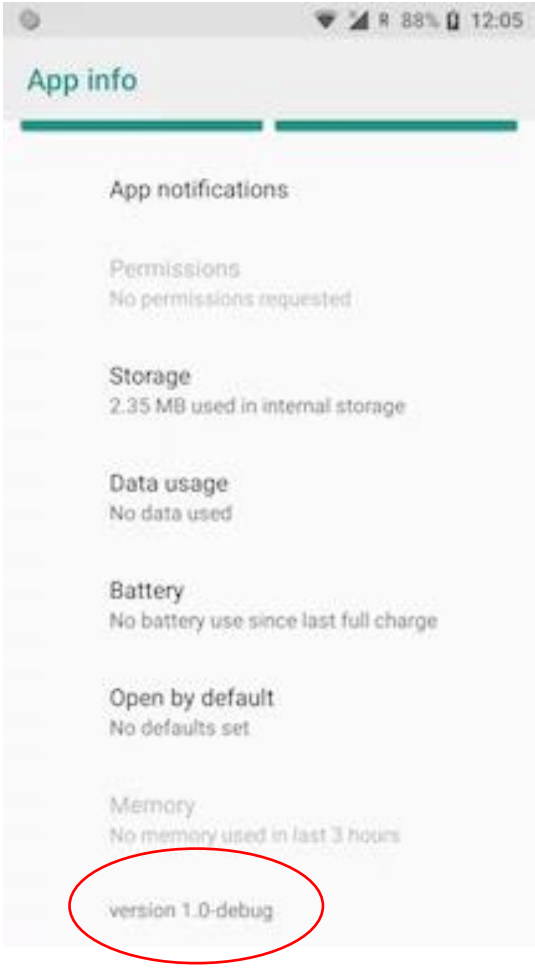


Remarques

- suffixe applicationId ajoute la chaîne à l'applicationId de l'application.
- versionName fait la même chose sur le nom de la version présente dans la defaultConfig.

Types de builds Android

- Après l'exécution de la version de débogage, allez vers **Paramètres > Applications > Nom de notre application**. L'écran des informations sur l'application montre le numéro de version en bas de page. Ceci est utile pour différencier les différentes constructions.



BuildConfig

- La classe **BuildConfig.java** est générée automatiquement lorsque différents buildFlavors sont créés.
- Nous pouvons définir les champs **BuildConfig** dans notre **build.gradle.**

```
buildTypes {
        release {
                signingConfig signingConfigs.release
                minifyEnabled false
                proguardFiles getDefaultProguardFile('proguard-android.txt'),
                        'proguard-rules.pro'
        }
        debug{
                applicationIdSuffix ".debug"
                versionNameSuffix "-debug"
                buildConfigField "String", "TYPE", '"I AM A DEBUG" '
        }
        beta{
                signingConfig signingConfigs.release
                applicationIdSuffix ".beta"
                versionNameSuffix "-beta"
                buildConfigField "String", "TYPE", '"I AM A BETA" '
        }
}
```

Configurer les variantes de construction

- Les variantes de construction (build) permettent de créer différentes versions de l'application à partir d'un seul projet.
- Chaque variante de construction représente une version différente de l'application construite. Par exemple, **une version gratuite** de l'application, avec un contenu limité, et **une autre version payante** qui en comprend davantage. Ou la création des différentes versions de l'application qui ciblent différents appareils, en fonction du niveau d'API ou d'autres variations de l'appareil.

```
productFlavors{
        free{
                applicationId "com.project.androidproductflavors.free"
        }
        paid{
                applicationId "com.project.androidproductflavors.paid"
        }
}
```

Configurer les variantes de construction

Les flavors des produits sont définis dans le bloc **Android** du fichier **build.gradle** des modules d'application. C'est ainsi qu'ils peuvent être définis :

```
android {...
   defaultConfig {...}
   productFlavors {
      flavor1 {...}
      flavor2 {...}
      flavor3 {...}
   }
   ...
   buildTypes{...}
}
```

Nous avons créé 3 flavors appelés flavor1, flavor2 et flavor3. Par défaut, Android a 2 types de débogage appelés debug et release (d'autres peuvent être ajoutés comme jnidebug), une variante de construction est une combinaison du type de construction et de la flavor du produit.

Donc maintenant il y aura 6 variantes de construction : flavor1-debug, flavor1-release, flavor2-debug, flavor2-release, flavor3-debug et flavor3-release.

À l'intérieur de ces blocs de flavor se trouvent des propriétés et des méthodes spécifiques aux flavor, comme applicationId (le nom du paquetage de l'application), versionCode, buildConfigField, etc.

Configurer les variantes de construction

- **Exemple :**

```
productFlavors {
    // Définir des flavors de produits distinctes pour le développement et la production.
    dev {
        // dev utilise minSDKVersion = 21 pour permettre au plugin Android gradle
        // de pré-dexer chaque module et de produire un APK qui peut être testé sur
        // Android Lollipop sans processus de fusion dex fastidieux.
        minSdkVersion 21
    }
    prod {
        // La version minSdkVersion réelle de l'application.
        minSdkVersion 15
    }
}
```

Comprendre ce que sont les variantes de compilation

- Les variantes de compilation sont le résultat de différentes combinaisons de types de produit et de types de compilation.
- Les types de produit sont considérés comme des attributs côté utilisateur, et les types de compilation comme des attributs côté développeur.
- Les variantes de compilation représentent les différentes combinaisons "type de produit/type de compilation" possibles. Elles sont donc nommées selon le schéma <product-flavor><build-type>. Par exemple, avec les types de compilation debug et release, et les types de produit free et paid, vous obtiendrez les variantes de compilation suivantes :
  - freeDebug
  - freeRelease
  - paidDebug
  - paidRelease

Remarques

- Vous n'allez pas configurer les variantes de compilation directement : vous allez définir un ensemble de types de produit et un ensemble de types de compilation, ce qui déterminera les variantes de compilation.

Configurer des types de produit

Les types de produit correspondent aux attributs de l'application côté utilisateur, dans la mesure où ils représentent généralement les versions de l'application disponibles.

Pour créer **la version gratuite** et **la version payante** de l'application, vous devez ajouter les deux types de produit correspondants et les attribuer à un groupe de types.

Pour ajouter les types de produit, ouvrez le fichier build.gradle au niveau de l'application (app > build.gradle dans la vue Project) et collez ce code dans le bloc android {}.

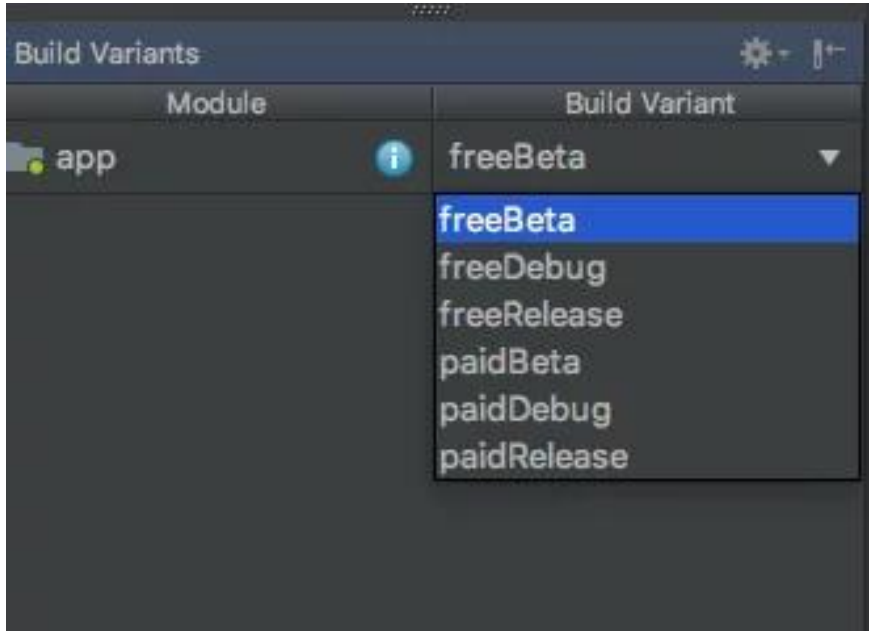
```
flavorDimensions "app_type"
productFlavors {
  free {
     dimension "app_type"
      }
  paid {
     dimension "app_type"
  }
}
```

Ce code effectue les opérations suivantes :

- Il crée un groupe de types appelé app_type ;
- Il crée deux types de produit, représentés par les blocs demo {} et full {} ;
- Il attribue les deux types de produit au groupe app_type (facultatif, s'il n'y a qu'un seul groupe de types).

Configurer des types de produit

Maintenant, la synchronisation du gradle vous donnera les flavors de produit suivantes :



Remarques

- Les variantes de build Android combinent les types de build et les flavors des produits. Elles créent une matrice de toutes les combinaisons.

Ajouter des dépendances pour les flavors

Vous pouvez ajouter différentes dépendances pour une flavor de produit spécifique.

Il suffit d'utiliser la syntaxe <flavorName>Compile 'group:name:x.y.z' :

```
android {
   ...
   productFlavors {
      flavor1 {
        //.....
      }
      flavor2 {
        //.....
      }
   }
}

...

dependencies {

   compile 'com.android.support:appcompat-v7:24.2.0'

   // Ajouter une dépendance juste pour flavor1
   flavor1Compile 'group:name:x.y.z'

   // Ajouter une dépendance juste pour flavor2
   flavor2Compile 'group:name:x.y.z'
}
```

OFPPT
La Voie de l'Avenir

WEBFORCE
BE THE CHANGE

PARTIE 3
Coder en JAVA

Dans ce module, vous allez :

- Coder une application en JAVA
- Réaliser des interfaces graphiques simples
- Maîtriser Git

30 heures

CHAPITRE n° 1

Coder une application en JAVA

Ce que vous allez apprendre dans ce chapitre :

- Introduction des notions de base en JAVA
- Intégration des concepts POO et JAVA
- Gestion des exceptions en JAVA
- Gestion des entrées et sorties
- Manipulation des collections

14 heures

OFPPT
La Voie de l'Avenir

WEBFORCE
BE THE CHANGE

CHAPITRE n° 1

Coder une application en JAVA

Rapide historique de Java

- **Origine**
  - Créé par Sun Microsystems ;
  - Cible : les systèmes embarqués (véhicules, électroménager, etc) utilisant des langages dédiés incompatibles entre eux.

- **Dates clés**

1995	mai : premier lancement commercial du JDK 1.0
1996	janvier : JDK 1.0.1 septembre : lancement du JDC
1997	Java Card 2.0 février : JDK 1.1
1998	décembre : lancement de J2SE 1.2 et du JCP Personal Java 1.0
1999	décembre : lancement J2EE 1.2
2000	mai : J2SE 1.3
2001	J2EE 1.3
2002	février : J2SE 1.4
2003	J2EE 1.4
2004	septembre : J2SE 5.0
2005	Lancement du programme Java Champion
2006	mai : Java EE 5 décembre : Java SE 6.0
2007	Duke, la mascotte de Java est sous la licence Free BSD
2008	décembre : Java FX 1.0
2009	février : JavaFX 1.1 juin : JavaFX 1.2 décembre : Java EE 6

Rapide historique de Java

- **Dates clés (Suite)**

2010	janvier : rachat de Sun Microsystems par Oracle avril : JavaFX 1.3
2011	juillet : Java SE 7 octobre : JavaFX 2.0
2012	août : JavaFX 2.2
2013	juin : Java EE 7
2014	mars : Java SE 8, JavaFX 8
2017	septembre Java SE 9, Java EE 8
2018	mars : Java SE 10 septembre : Java SE 11
2019	mars : Java SE 12 septembre : Java SE 13, Jakarta EE 8
2020	mars : Java SE 14 septembre : Java SE 15 décembre : Jakarta EE 9
2021	mars : Java SE 16 mai : Jakarta EE 9.1 septembre : Java SE 17
2022	mars : Java SE 18, Jakarta EE 10
2019	mars : Java SE 12 septembre : Java SE 13, Jakarta EE 8
2020	mars : Java SE 14 septembre : Java SE 15 décembre : Jakarta EE 9
2021	mars : Java SE 16 mai : Jakarta EE 9.1 septembre : Java SE 17
2022	**mars : Java SE 18, Jakarta EE 10**

Les caractéristiques de JAVA

Parmi les caractéristiques qui ont contribué au succès de Java :

1. Java est Portable (grâce à JVM) ;
2. Java est simple (pas de pointeurs, héritage multiple) ;
3. Fortement typé ;
4. Gestion automatique de la mémoire (GC) ;
5. Java est Sûr ;
6. Multitâche.

Remarque

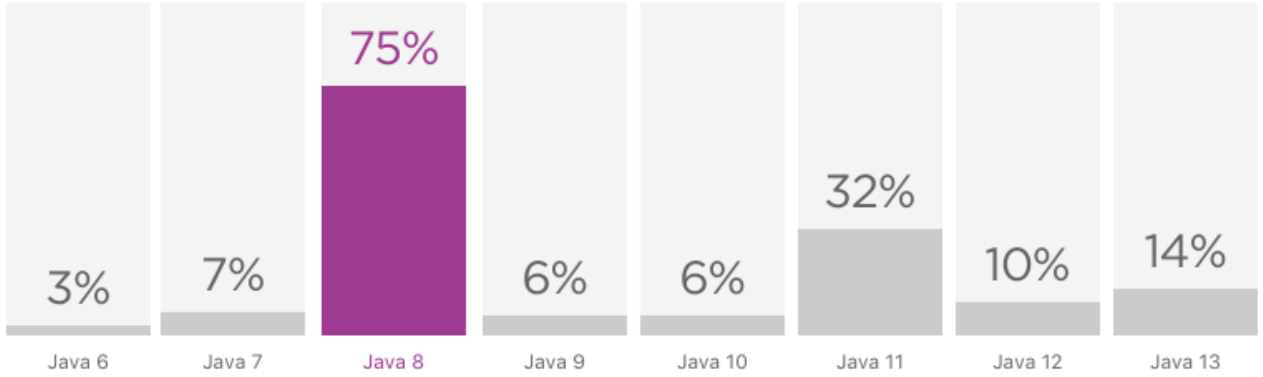
- Un langage de programmation est dit **fortement typé** lorsqu'il garantit que les types de données employées décrivent correctement les données manipulées. Par opposition, un langage sans typage fort est dit faiblement typé.

PARTIE 3

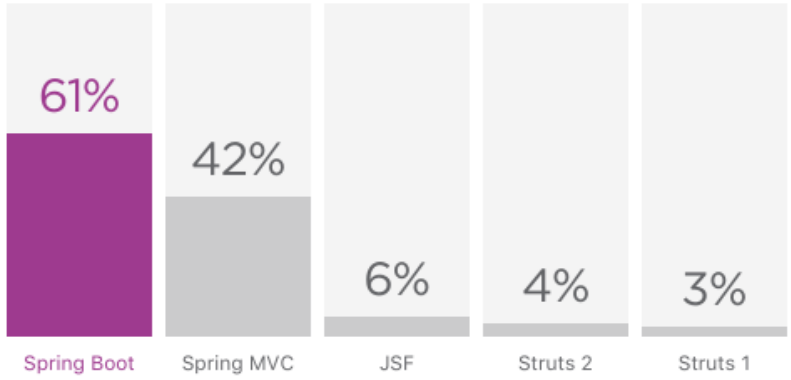
Popularité de Java

Présente dans de très nombreux domaines d'application : des serveurs d'applications aux téléphone portables et cartes à puces (JME) (Java Micro Edition).

Popularité des versions Java en 2020



Top 5 des frameworks Web



Réf : developpez.com

PARTIE 3

Popularité de Java

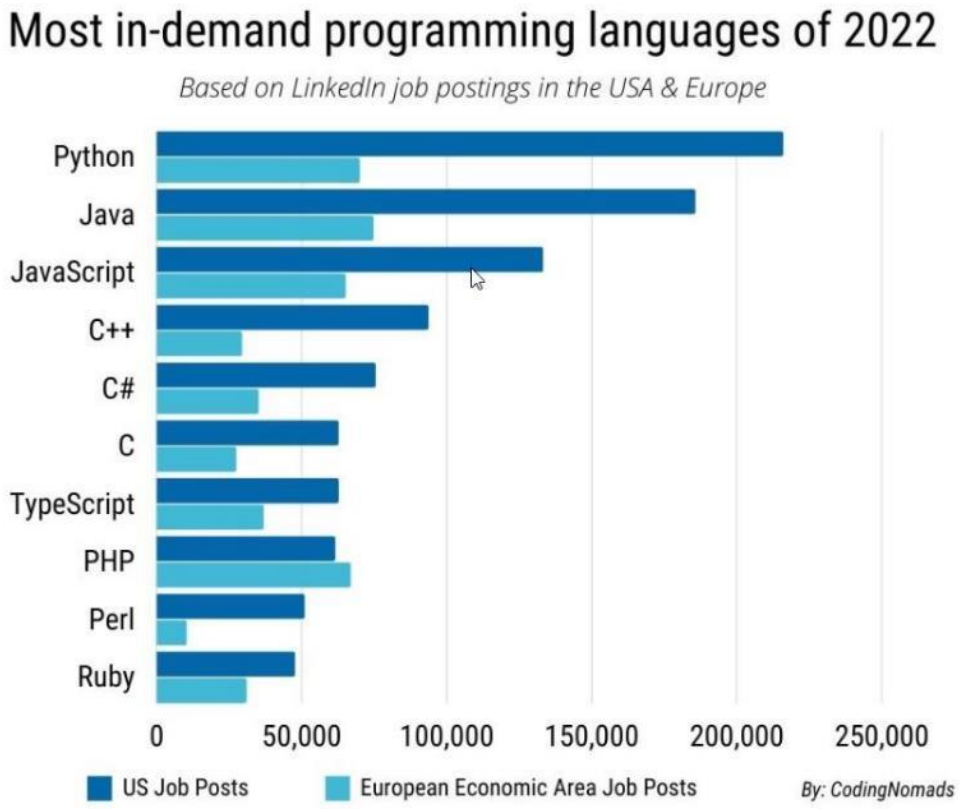
Size of programming language communities in Q3 2021

Active software developers, globally, in millions (n=12,506)

		Most popular in	Least popular in
Javascript*	16.4 M	Web, backend	DS/ML, embedded
Python	11.3 M	DS/ML, IoT apps	Mobile, AR/VR
Java	9.6 M	Mobile, desktop	DS/ML, web
C/C++	7.5 M	Embedded, IoT apps	Web, mobile
PHP	7.3 M	Web, backend	DS/ML, mobile
C#	7.1 M	AR/VR, desktop, games	DS/ML, mobile
Visual development tools	3.6 M	Desktop, AR/VR	Cloud, web
Kotlin	2.9 M	Mobile, AR/VR	DS/ML, desktop
Swift	2.5 M	Mobile, AR/VR	Backend, desktop
Go	2.0 M	Backend, apps for 3rd-party ecosystems	Games, web
Dart	1.4 M	Mobile	Web
Objective C	1.4 M	AR/VR	Desktop, games
Ruby	1.4 M	IoT, backend	DS/ML, web
Rust	1.1 M	AR/VR, embedded	Mobile, web
Lua	0.8 M	AR/VR, IoT, games	Mobile, desktop

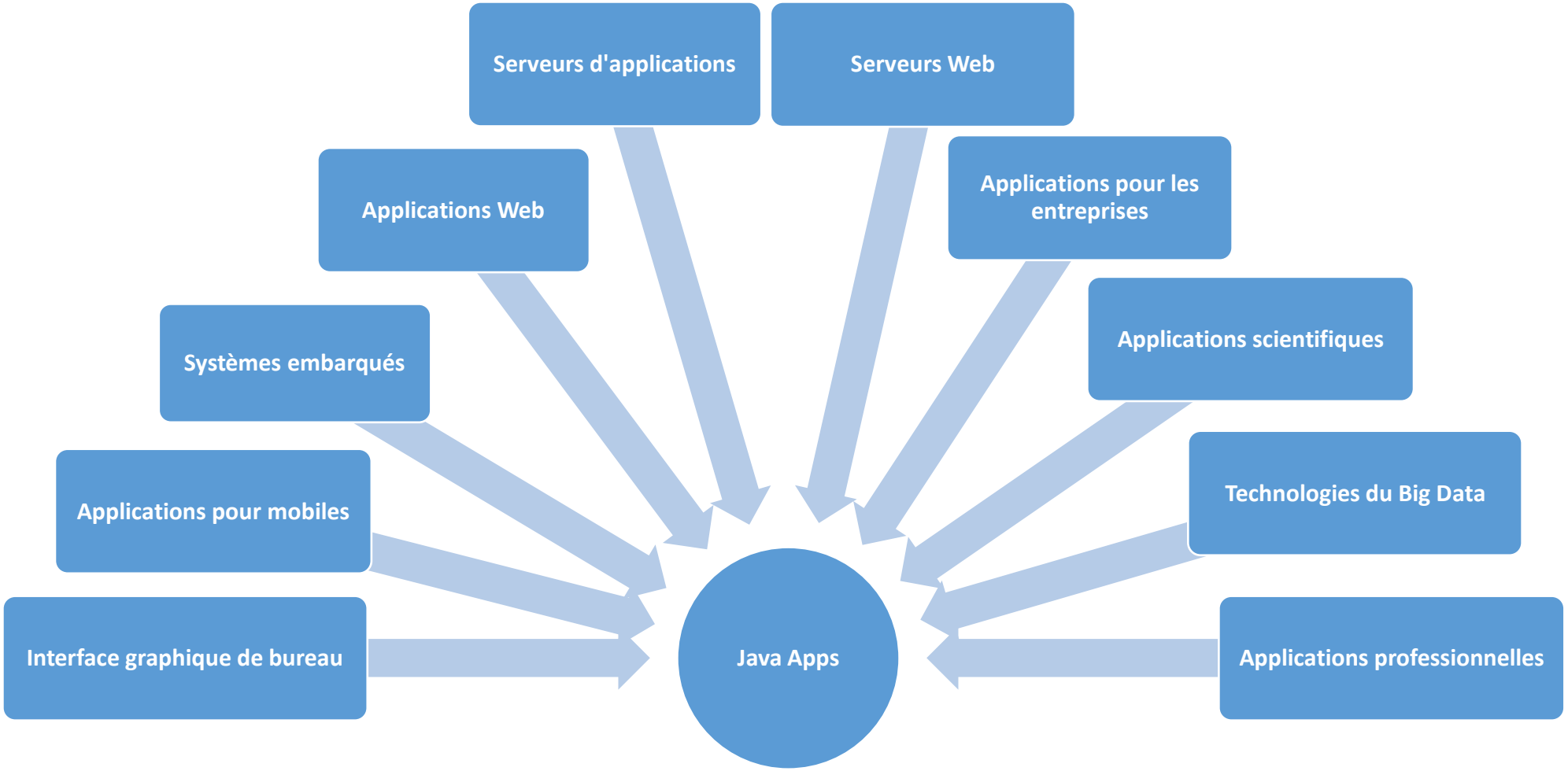
/DATA

Popularité des langages dans les offres d'emploi en 2022



© SlashData

Applications Java



Principe de fonctionnement de Java

- **Source Java**
  - Fichier utilisé lors de la phase de programmation ;
  - Le seul fichier réellement intelligible par le programmeur.

- **Byte-Code Java**
  - Code objet destiné à être exécuté sur toute « Machine Virtuelle » Java ;
  - Provient de la compilation du code source.

- **Machine Virtuelle Java**
  - Programme interprétant le Byte-Code Java et fonctionnant sur un système d'exploitation particulier.

Remarque

- Il suffit de disposer d'une « Machine Virtuelle » Java pour pouvoir exécuter tout programme Java même s 'il a été compilé avec un autre système d'exploitation

Principales étapes d'un développement

- **Création du code source**

À partir des spécifications (par exemple en UML).

Outil : éditeur de texte, IDE

- **Compilation en Byte-Code**

À partir du code source.

Outil : compilateur Java

- **Diffusion sur l'architecture cible**

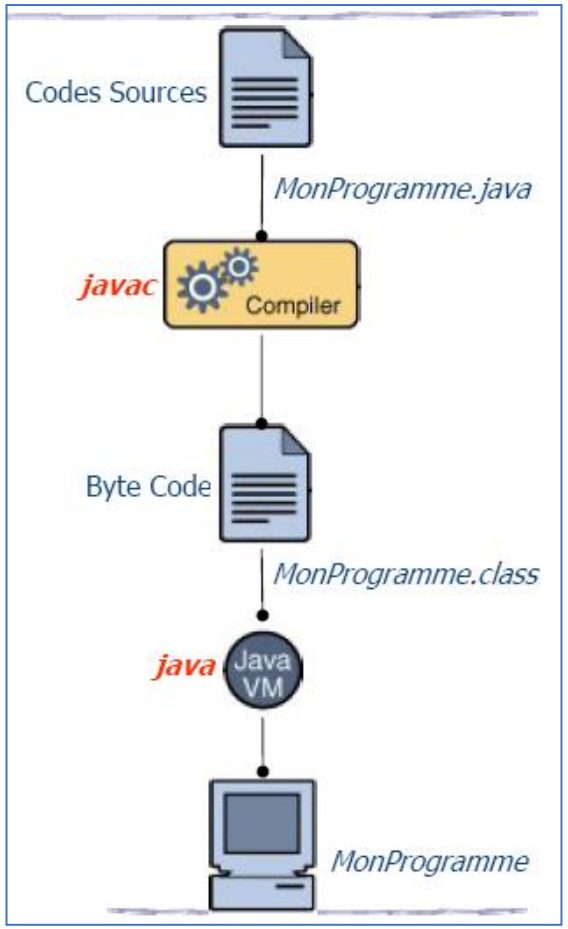
Transfert du Byte-Code seul.

Outils : réseau, disque, etc..

- **Exécution sur la machine cible**

Exécution du Byte-Code.

Outil : Machine Virtuelle Java



Java et ses versions ...

- **Plateformes Java**
  - **Java 2 Micro Edition (J2ME)** : contient le nécessaire pour développer des applications capables de fonctionner dans des environnements limités tels que les assistants personnels (PDA), les téléphones portables ou les systèmes de navigation embarqués.
  - **Java 2 Standard Edition (J2SE)** : contient le nécessaire pour développer des applications et des applets. Cette édition reprend le JDK 1.0 et 1.1.
  - **Java 2 Enterprise Edition (J2EE)** : contient un ensemble de plusieurs API permettant le développement d'applications destinées aux entreprises tel que JDBC pour l'accès aux bases de données, EJB pour développer des composants orientés métiers, Servlet / JSP pour générer des pages HTML dynamiques, etc. Cette édition nécessite le J2SE pour fonctionner.
- **Différentes finalités**
  - SDK (Software Development Kit) fournit un compilateur et une machine virtuelle.
  - JRE (Java Runtime Environment) fournit uniquement une machine virtuelle. Idéal pour le déploiement de vos applications
- **Version actuelle de Java**
  - Actuellement « Java SE 18 (22 mars 2022) ».

Les outils ...

Simples éditeurs ou environnements de développement :

Apache NetBeans IDE

CHAPITRE n° 1

Coder une application en JAVA

1. Introduction à JAVA
2. **Introduction des notions de base en JAVA**
3. Programmation OO en JAVA
4. Gestion des exceptions en JAVA
5. Gestion des entrées et sorties
6. Manipulation des collections

Premier exemple de programme en Java

```
public class PremierProg {
        public static void main (String[] argv) {
                System.out.println("Ola, mon Premier Programme");
        }
}
```

- **public class PremierProg**

Nom de la classe.

- **public static void main**

La fonction principale équivalent à la fonction main du C/C++.

- **String[] argv**

Permet de récupérer des arguments transmis au programme au moment de son lancement.

- **System.out.println("Ola … ")**

Méthode d'affichage dans la fenêtre console.

Mise en œuvre

- **Pas de séparation entre définition et codage des opérations**

Un seul fichier « **NomDeClasse.java** » ;

Pas de fichier d'en tête comme C/C++.

- **Compilation**

javac NomDeClasse.java ou **javac *.java** quand plusieurs classes

Génération d'un fichier Byte-Code « NomDeClasse.class ».

- **Exécution**

java NomDeClasse

Choisir la classe principale à exécuter.

Types primitifs de Java

- Ne sont pas des objets !!!
- Occupent une place fixe en mémoire réservée à la déclaration.
- Types primitifs
  - Entiers : **byte (1 octet) - short (2 octets) - int (4 octets) – long (8 octets)**
  - Flottants : **float (4 octets) - double (8 octets)**
  - Booléens : **boolean (true ou false)**
  - Caractères : **char (codage Unicode sur 16 bits)**
- Chacun des types simples possède un alter-ego objet disposant de méthodes de conversion (à voir dans la partie Classes et Objets).
- L'autoboxing introduit depuis la version 5.0 convertit de manière transparente les types primitifs en références.

Initialisation et constantes

Initialisation

- Une variable peut recevoir une valeur au moment de sa déclaration :

```
int n = 15;

boolean b = true;
```

- Cette instruction joue le même rôle :

```
int n;

n = 15;

boolean b;

b = true;
```

Constantes

Ce sont des variables dont la valeur ne peut affectée qu'une fois.

Elles ne peuvent plus être modifiées.

Elles sont définies avec le mot clé **final :**

```
final int n = 5;

final int t;

...

t = 8;

n = 10; // erreur : n est déclaré final
```

Structure de contrôle

1. **Choix**

- Si alors sinon : « **if condition {…} else {…} »**

2. **Itérations**

- Boucle : « **for (initialisation ; condition ; modification) { … } »**
- Boucle (for each) : « for (Type var : Collection) { … } »
- Tant que : « **while (condition) { … } »**
- Faire jusqu'à : « **do { … } while (condition) »**

3. **Sélection bornée**

- Selon faire : « **switch ident { case valeur0 : … case valeur1 : … default: …} »**
- Le mot clé **break demande à sortir du bloc.**

Remarque

- Il n'y a pas de mot clé « then » dans la structure Choix.

Remarque

- Penser à vérifier si break est nécessaire dans chaque case.

PARTIE 3

Structure de contrôle

- Exemple : structure de contrôle

```
public class SwitchBreak {

    public static void main (String[] argv)  {
        int n = ...;
        System.out.println("Valeur de n :" + n);
        switch(n) {
            case 0 : System.out.println("nul");
                     break;
            case 1 :
            case 2 : System.out.println("petit");
            case 3 :
            case 4 :
            case 5 : System.out.println("moyen");
                     break;
            default : System.out.println("grand");
        }
        System.out.println("Adios...");
    }
}
```

Faisons varier n :

Valeur de n : 0
nul
Adios...

Valeur de n : 1
petit
moyen
Adios...

Valeur de n : 6
grand
Adios...

Opérateurs sur les types primitifs

- **Opérateurs arithmétiques**

Unaires : « +a, -b »

Binaires : « a+b, a-b, a*b, a%b »

Incrémentation et décrémentation : « a++, b-- »

Affectation élargie : « +=, -=, *=, /= »

- **Opérateurs comparaisons**

« a==b, a!=b, a>b, a<b, a>=b, a<=b »

- **Opérateurs logiques**

Et : « a && b », « a & b »

Ou : « a || b », « a | b »

- **Conversion de type explicite (cast) « (NouveauType)variable »**

Remarque

Attention : erreur
boolean t = true;
if (t == true) {...}

Préférer :
boolean t = true;
if (t) {...}

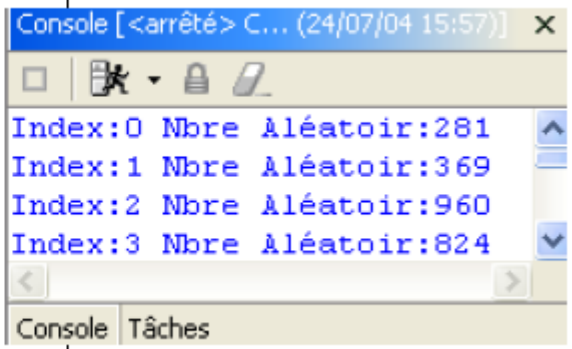
Opérateurs sur les types primitifs

- Exemple : simulation du Loto.

Pas optimisé mais montre l'utilisation des concepts précédents :

```
public class ExempleTypesPrimitifs {

    public static void main (String[] argv) {
        int compteur = 0;

        while(compteur != 100) {
            // Prend un nombre aléatoire
            double nbreAleatoir = Math.random() * 1000;

            // Etablie un index de 0 à 10
            int index = compteur % 10;

            // Construction de l'affichage
            System.out.println("Index:" + index +
                "Nbre Aléatoir:" + (int)nbreAleatoir);

            // Incrémentation de la boucle
            compteur+= 1;
        }
    }
}
```

A voir plus tard...



PARTIE 3

Affectation, recopie et comparaison

- **Affecter et recopier un type primitif**

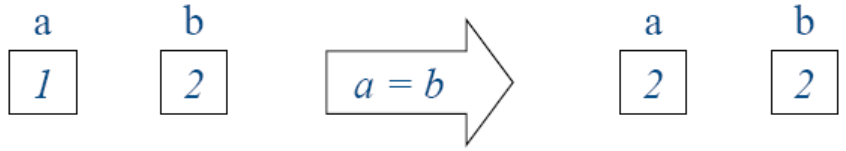
« a=b » signifie a prend la valeur de b ;

a et b sont distincts ;

Toute modification de a n'entraîne pas celle de b.

- **Comparer un type primitif**

« a == b » retourne « true » si les valeurs de a et b sont identiques.



Les tableaux en Java

- Les tableaux sont considérés comme des objets ;
- Fournissent des collections ordonnées d'éléments ;
- Les éléments d'un tableau peuvent être :
    - Des variables d'un type primitif (int, boolean, double, char, …) ;
    - Des références sur des objets (à voir dans la partie Classes et Objets).

- Création d'un tableau
    1. Déclaration = déterminer le type du tableau ;
    2. Dimensionnement = déterminer la taille du tableau ;
    3. Initialisation = initialiser chaque case du tableau.

Les tableaux en Java : Déclaration

1. Déclaration

- La déclaration précise simplement le type des éléments du tableau :

```
int[] tab;
```

tab → *null*

- Peut s'écrire également :

```
int tab[];
```

Remarque

- **Une déclaration de tableau ne doit pas préciser de dimensions :**

```
int tab[5]; // Erreur
```

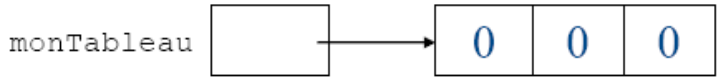
Les tableaux en Java : Dimensionnement

2. Dimensionnement

- Le nombre d'éléments du tableau sera déterminé quand l'objet tableau sera effectivement créé en utilisant le mot clé **new.**
- La taille déterminée à la création du tableau est fixe, elle ne pourra plus être modifiée par la suite.
- Longueur d'un tableau : « tab.**length » :**

```
int[] tab;          // Déclaration
tab = new int[3];   // Dimensionnement
```

- La création d'un tableau par **new :**
  - Alloue la mémoire en fonction du type de tableau et de la taille ;
  - Initialise le contenu du tableau à 0 pour les types simples.



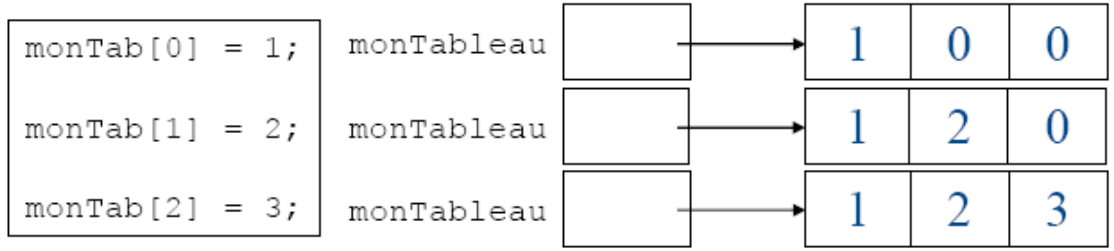
Les tableaux en Java : Initialisation

3. Initialisation

- Comme en C/C++ les indices commencent à zéro.
- L'accès à un élément d'un tableau s'effectue suivant cette forme :

monTab[varInt]; // varInt >= 0 et <monTab.length

- Java vérifie automatiquement l'indice lors de l'accès (exception levée) :



- Autre méthode : en donnant explicitement la liste de ses éléments entre {…} :

int[] monTab = {1, 2, 3}

- Est équivalent à :

monTab = new int[3];

monTab[0] = 1; monTab[1] = 2; monTab[2] = 3;

PARTIE 3

Les tableaux en Java : Synthèse

1. Déclaration

```
int[] tab;
```

2. Dimensionnement

```
tab = new int[3];
```

3. Initialisation

```
tab[0] = 1;
tab[1] = 2;
tab[2] = 3;
```

Ou 12 et 3

```
int[] monTab = {1, 2, 3};
```

```
for (int current : tab) {
    System.out.println(curent);
}
```

```
for (int i = 0; i < tab.length; i++) {
    System.out.println(tab[i]);
}
```

Les tableaux en Java : Multidimensionnels

Tableaux dont les éléments sont eux mêmes des tableaux :

- Déclaration :

  type[ ][ ] tab;

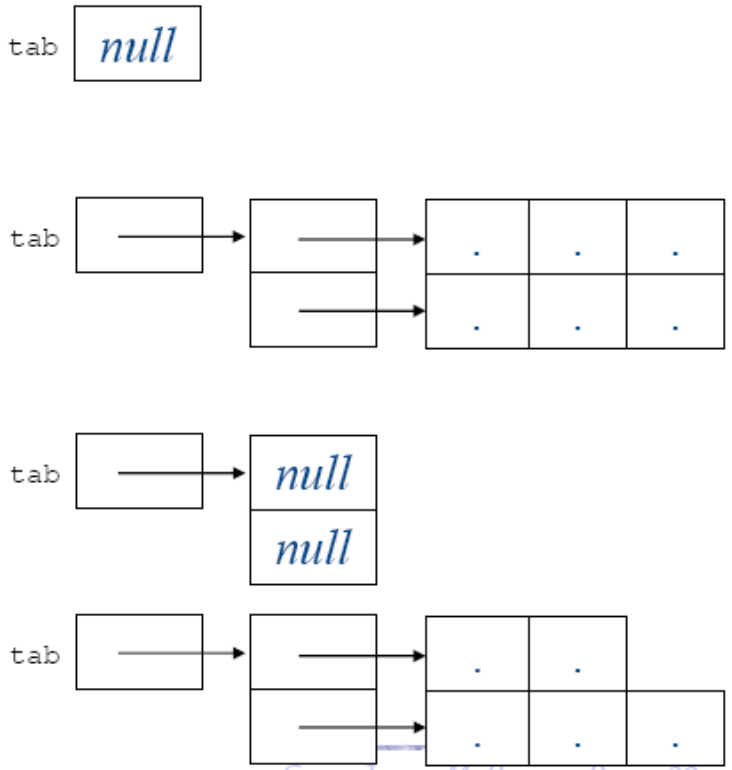
- Tableaux rectangulaires
  - Dimensionnement :

    tab = new type[2][3];
- Tableaux non-rectangulaires
  - Dimensionnement :

    tab = new type[2][ ];

    tab[0] = new type[2];

    tab[1] = new type[3];



Petite précision du «System.out.println(…) »

- Usages : affichage à l'écran :
  - « System.out.println(…) » : revient à la ligne ;
  - « System.out.print(…) » : ne revient pas à la ligne.
- Différentes sorties possibles :
  - « out » sortie standard ;
  - « err » sortie en cas d'erreur (non temporisée).
- Tout ce que l'on peut afficher…
  - Objets, nombres, booléens, caractères, …
- Tout ce que l 'on peut faire …
  - Concaténation sauvage entre types et objets avec le « + » System.out.println("a=" + a + "donc a < 0 est " + a < 0).

Commentaires et mise en forme

- Documentation des codes sources :
  - Utilisation des commentaires :

```
// Commentaire sur une ligne complète
int b = 34; // Commentaire après du code
/* Le début du commentaire
** Je peux continuer à écrire …
Jusqu'à ce que le compilateur trouve cela */
```

  - Utilisation de l'outil Javadoc .
- Mise en forme
  - Facilite la relecture ;
  - Crédibilité assurée ;
  - Indentation à chaque ;
  - Niveau de bloc.

Convention de codage

La plupart des projets OpenSource suivent ces instructions.

- Chaque conditionnelle contient les parenthèses ouvrantes et fermantes .

```
//Correct
If (expression) {
    // le code
}

//Incorrect
If (expression)
    //le code
```

Convention de codage

- Les noms des fonctions et des paramètres, ne doivent pas avoir de préfixe, commencent par une minuscule et chaque partie de mot est en majuscule.

```
public class MaClass  {
        private String maChaine;
        public void maMethode (String monParametre)
        {

        }
}
```

Convention de codage

- La version de l'application (du code) est précisée dans chaque fichier d'extension .java :

@version 2.6

- Le nom de l'auteur est précisé :

@author lachgar

- Chaque importation de paquetage (paquet) Java est pleinement qualifiée :

```
// Correct
import  java.util.Date;
import java.net.HttpURLConnection;
```

```
//Incorrect
import  java.util.*;
import java.net.*:
```

Les conventions Java

- **Fichiers**

Les classes doivent être regroupées en packages (paquetage ou paquets).

- **Sources**

Il est recommandé de ne pas dépasser 2 000 lignes de code par fichier.

- **Formatage**

L'espace est autorisé après les virgules, avant et après les accolades, avant et après chaque opérateur, après les mots réservés du langage et entre une méthode et la parenthèse des ses paramètres.

```
// Correct
maMethode (a, b, c, d);
for (i = 0; i < 100; i++) {
    ++count;
    (MaClasse)maVariable.get(i);
```

```
//Incorrect
MaMethode (a,b,c,d);
for (i=0; i<100;i++) {
++ count;
(MaClasse) maVariable.get(i);
```

Les conventions Java

- **Nommage**

Les noms utilisés doivent être explicites, c'est-à-dire que le nom doit expliquer le contenu de l'objet, le rôle de la méthode…

- **Package/Paquetage**

Les noms des paquetage doivent être en minuscules :

```
//Correct
package com.monprojet.monpaquet;
```

```
//Incorrect
package Com.MonProjet.MonPaquet;
```

- **Classes et interfaces**

Les noms des classes doivent être en minuscules, hormis les initiales des mots qui les composent :

```
//Correct
Class  MaClasseFavorite;
```

```
//incorrect
class  maClassefavorite;
```

Les conventions Java

- **Méthodes**

Les noms des méthodes doivent être en minuscules hormis les initiales des mots qui composent les mots (sauf la première lettre) :

```
//Correct
public void maMethodeFavorite ( ) {
```

```
//Incorrect
public void mamethodefavorite ( ) {
```

Les accesseurs directs (getters et setters) des attributs d'une classe doivent être préfixés d'un get pour la lecture et d'un set pour l'écriture. Le préfixe is doit être utilisé pour la méthodes qui retournent un booléan :

```
//Correct
public int getNiveau() {
Public void setNiveau (int niveau) {
Public boolean isVisible ( ) {
```

```
//Incorrect
public  int recupererLeNiveau ( ) {
public void ecrireLeNiveau (int niveau) {
public boolean estIlVisible ( ) {
```

Nous pouvons également utiliser d'autres mots pour les recherches, les suppressions, les ajouts, la fermeture de connexion… (find, delete, add, close ..).

Les conventions Java

- **Attributs, variables et paramètres**

Les attributs des classes, les variables ainsi que les paramètres des méthodes doivent être en minuscules hormis initiales des mots qui les composent (sauf le premier). Les variables de boucle doivent porter une seule lettre: i, j, k … Les signes dollars ($) et soulignement (_) sont proscrits :

```
//Correct
Voiture prochaineVoiture = voitures.get (this.id + 1)
float  laTaille = 145.5;

//Incorrect
Voiture a = voitures.get (this.id + 1)
float  la_Taille = 145.5;
```

Les conventions Java

- **Collections**

Les collections doivent être nommée au pluriel :

```
//Correct
Vector comptes;
Collection banques;
Objet [ ] mesObjets;
```

```
//Incorrect
Vector compte;
Collection banque;
Object  [ ] monObjet;
```

- **Constantes**

Les noms des constantes doivent être écrits entièrement en majuscules. Le séparateur de mot est le caractère de soulignement (underscore) :

```
//Correct
static final int LOG_CONSOLE = 1;
```

```
//Incorrect
static final int LOGCONSOLE = 1;
static final int console_Log = 1;
static final int Console_LOG = 1;
```

Les conventions Java

- Les commentaires sont essentiels dans un code source. Ils permettent de documenter le projet à l'intérieur même du code source en vue de la génération de la documentation via l'outil JavaDoc. Il existe en Java deux types de commentaires :
  - Les commentaires mono-ligne qui permettent de désactiver tout ce qui apparait sur la même ligne //;
  - Les commentaires multilignes qui permettent de désactiver tout le code qui se trouve entre les deux délimiteurs /* */.

```
/*
* La classe MaClase permet telles fonctionnalités …
*/
public class MaClasse {
//  Recuperer un objet de la collection
monFichier = (Fichier)fichiers.get ( ( int ) item.getIdFichier( ) );
```

Les conventions Java

- **Déclaration**

Les variables doivent être déclarées ligne par ligne ;

L'initialisation doit se faire lors de la déclaration lorsque cela est possible ;

Les noms des méthodes sont accolés à la parenthèse ouvrante listant leurs paramètres. Aucun espace ne doit y être inséré :

```
//Correct
int  niveau = 10;
void maMethode( ) {
```

```
//Incorrect
int niveau;
niveau = 10;
void maMethode ( ) {
```

Les conventions Java

- **Ordre**

L'ordre de déclaration des entités du code source doit être le suivant (qui est plus ou moins naturel) :

- Les attributs de la classe (1-> statiques, 2->publique, 3->protégés, 4-> privés) ;
- Les méthodes de la classe (1-> statiques, 2->publique, 3->protégés, 4-> privés).

- **Instructions**

Une ligne de code ne peut contenir qu'une seule instruction :

```
//Correct
count++;
i--;
println (''Bonjour'');
```

```
//Incorrect
count++; i--; println(''Bonjour '');
```

CHAPITRE n° 1

Coder une application en JAVA

1. Introduction à JAVA
2. Introduction des notions de base en JAVA
3. **Programmation OO en JAVA**
4. Gestion des exceptions en JAVA
5. Gestion des entrées et sorties
6. Manipulation des collections

Programmation Structurée VS POO

- **Objectifs de la POO**
  - Facilité la réutilisation de code, encapsulation et abstraction ;
  - Facilité de l'évolution du code ;
  - Améliorer la conception et la maintenance des grands systèmes ;
  - Programmation par « composants ». Conception d'un logiciel à la manière de la fabrication d'une voiture.

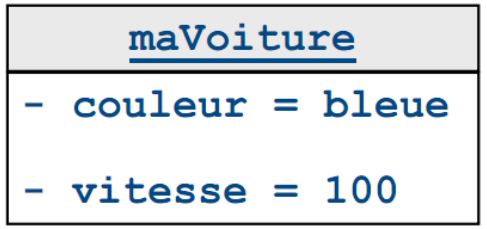
- **Programmation Structurée**
  - Unité logique : le module ;
  - Une zone pour les variables ;
  - Une zone pour les fonctions :
  - Chaque fonction résout une partie du problème ;
  - Structuration « descendante » du programme.

Principes POO : programmation par objets

Unité logique : **l'objet**

Objet est défini par

- un état ;
- un comportement ;
- une identité.



État : représenté par des attributs (variables) qui stockent des valeurs.

Comportement : défini par des méthodes (procédures) qui modifient des états.

Identité : permet de distinguer un objet d'un autre objet.

Principes POO : programmation par objets

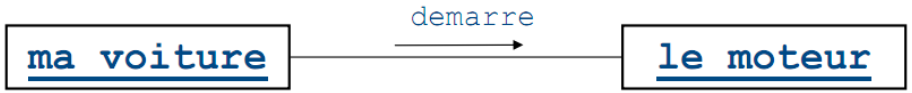
Les objets communiquent entre eux par des messages.

Un objet peut recevoir un message qui déclenche :

- une méthode qui modifie son état;

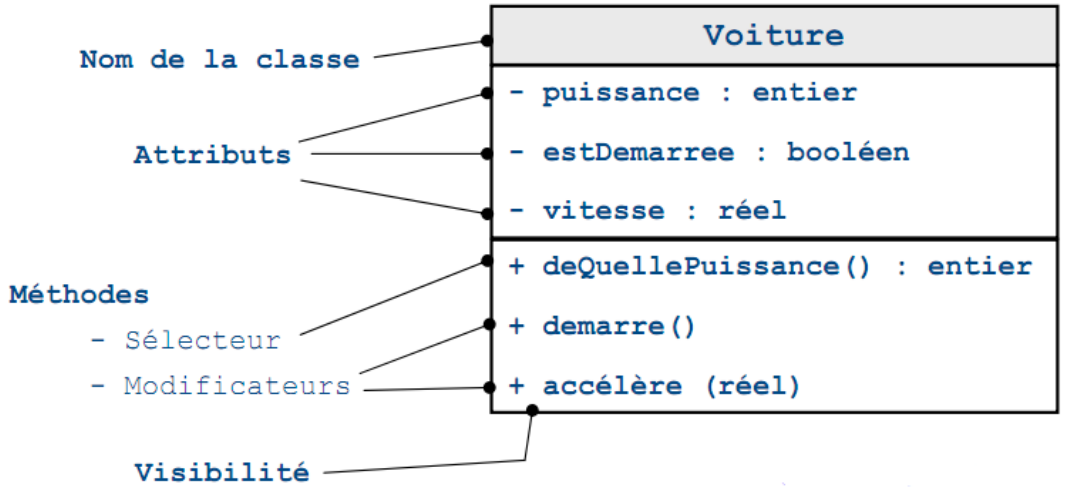
et / ou

- une méthode qui envoie un message à un autre objet.



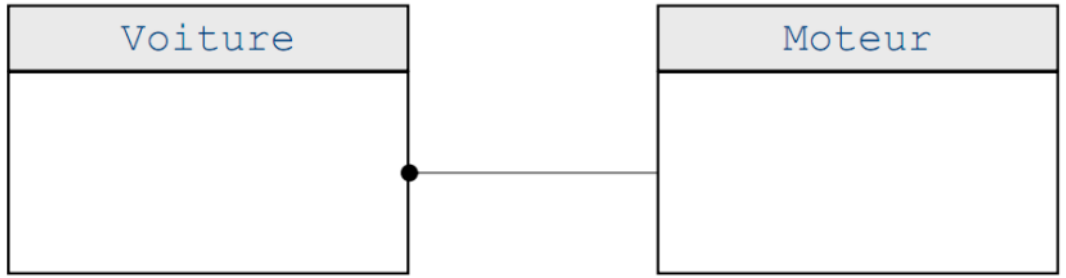
Principes POO : notion de classe

- Les objets qui ont les mêmes états et les mêmes comportements sont regroupés : c'est une classe ;
- Les classes servent de « **moules** » pour la création des objets ;
- Un objet est une instance d'une classe ;
- Un programme OO est constitué de classes qui permettent de créer des objets qui s'envoient des messages.



Principes POO

- L'ensemble des interactions entre les objets défini un algorithme.
- Les relations entre les classes reflètent la décomposition du programme :



Classe et définition

Une classe est constituée de :

- Données, ce qu'on appelle des **attributs.**
- Procédures et/ou des fonctions ce qu'on appelle des **méthodes.**

Une classe est un modèle de définition pour des objets :

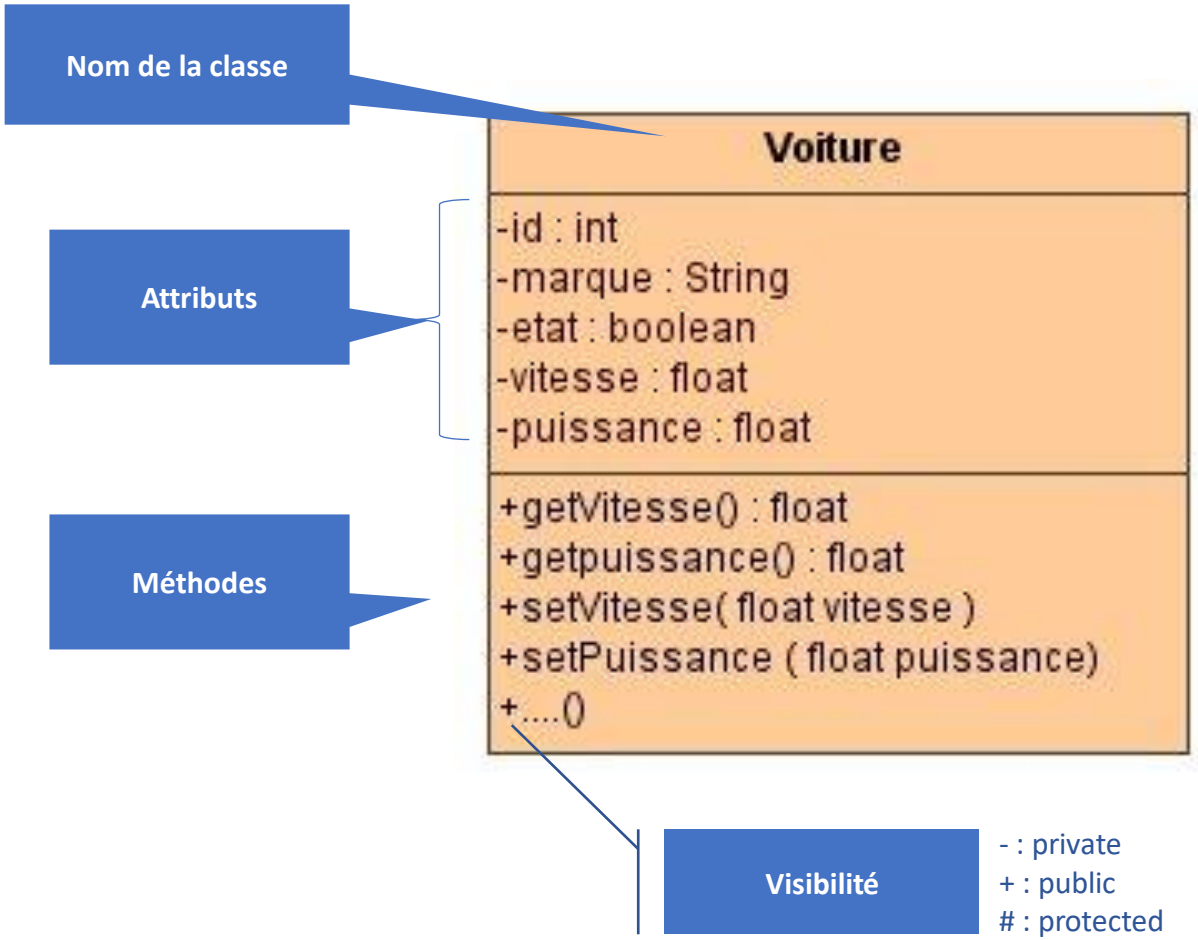
- Ayant même structure (même ensemble d'attributs) ;
- Ayant même comportement (même méthodes) ;
- Ayant une sémantique commune.

Les **objets sont des représentations dynamiques du modèle** défini pour eux au travers de la classe (**instanciation).**

Une classe permet d'**instancier (créer) plusieurs objets.**

Chaque objet est instance d'une classe et une seule.

Classe et notation UML



Codage de la classe «Voiture »

```
public class Voiture {
        private int id;
        private String marque;
        private float vitesse;
        private float puissance;
        private boolean etat;

        public int getId() {
                return id;
        }
        public String getMarque() {
                return marque;
        }
        public void setMarque(String marque) {
                this.marque = marque;
        }
public float getVitesse() {
                return vitesse;
}
public void setVitesse(float vitesse) {
                this.vitesse = vitesse;
}
public float getPuissance() {
                return puissance;
}
public void setPuissance(float puissance) {
                this.puissance = puissance;
}
 public boolean isEtat() {
                return etat;
}
public void setEtat(boolean etat) {
                this.etat = etat;
}
}
```

Nom de la classe

Attributs

Méthodes

Visibilité

Classe et visibilité des attributs

Caractéristique d'un attribut

- Variables « globales » de la classe.
- Accessibles dans toutes les méthodes de la classe :

```
public class Voiture {
        private int id;
        private String marque;
        private float vitesse;
        private float puissance;
        private String etat;

        public int getId() {
                return id;
        }
        public String getMarque() {
                return marque;
        }
        public void setMarque(String marque) {
                this.marque = marque;
        }
```

Attributs visibles dans les méthodes

Distinction entre attributs et variables

Caractéristique d'une variable :

- Visible à l'intérieur du bloc qui le définit :

```
public void accelere (float v) {
    if (isEtat)
    {
                float tolerance;
                tolerance = v + 100;
                this.vitesse = this.vitesse + tolerance;
    }
}
```

Variable visible uniquement dans cette méthode

Variable peut être définie n'importe où dans un bloc

Rappel : Conventions en Java : de la rigueur et de la classe …

Conventions de noms :

- CeciEstUneClasse ;
- celaEstUneMethode(…) ;
- jeSuisUneVariable ;
- JE_SUIS_UNE_CONSTANTE ;

Un fichier par classe, une classe par fichier :

- Classe « Voiture » décrite dans le fichier Voiture.java.
- Il peut exceptionnellement y avoir plusieurs classes par fichier (cas des Inner classes) .

Remarque

- Respecter les minuscules et les majuscules des noms

Objet et définition

Un objet est instance d'une seule classe :

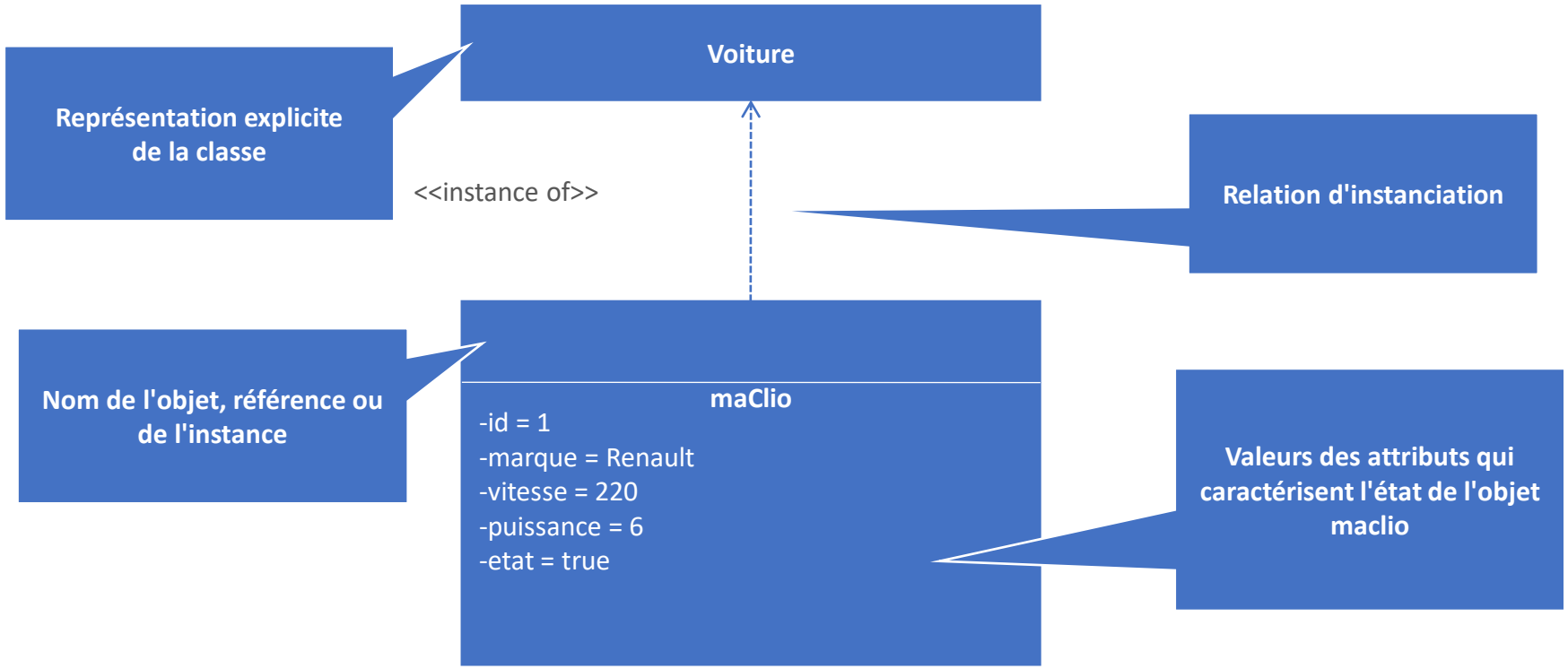
- Se conforme à la description que celle-ci fournit ;
- Admet une valeur propre à l'objet pour chaque attribut déclaré dans la classe ;
- Les valeurs des attributs caractérisent l'état de l'objet ;
- Possibilité de lui appliquer toute opération (méthode) définie dans la classe ;

Tout objet est manipulé et identifié par sa référence :

- Utilisation de pointeur caché (plus accessible que le C++) ;
- On parle indifféremment d'instance, de référence ou d'objet.

Objet et notation UML

maClio est une instance de la classe Voiture :



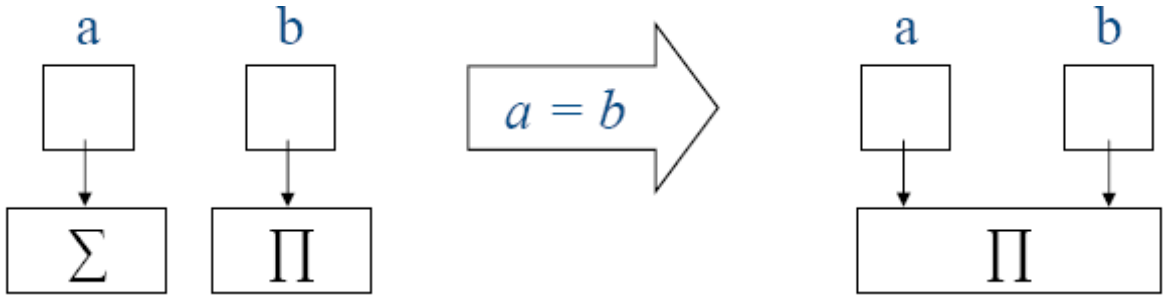
Affectation et comparaison

Affecter un objet

- « a = b » signifie a devient identique à b ;
- Les deux objets a et b sont identiques et toute modification de a entraîne celle de b.

Comparer deux objets

- « a == b » retourne « true » si les deux objets sont identiques ;
- C'est-à-dire si les références sont les mêmes, cela ne compare pas les attributs.



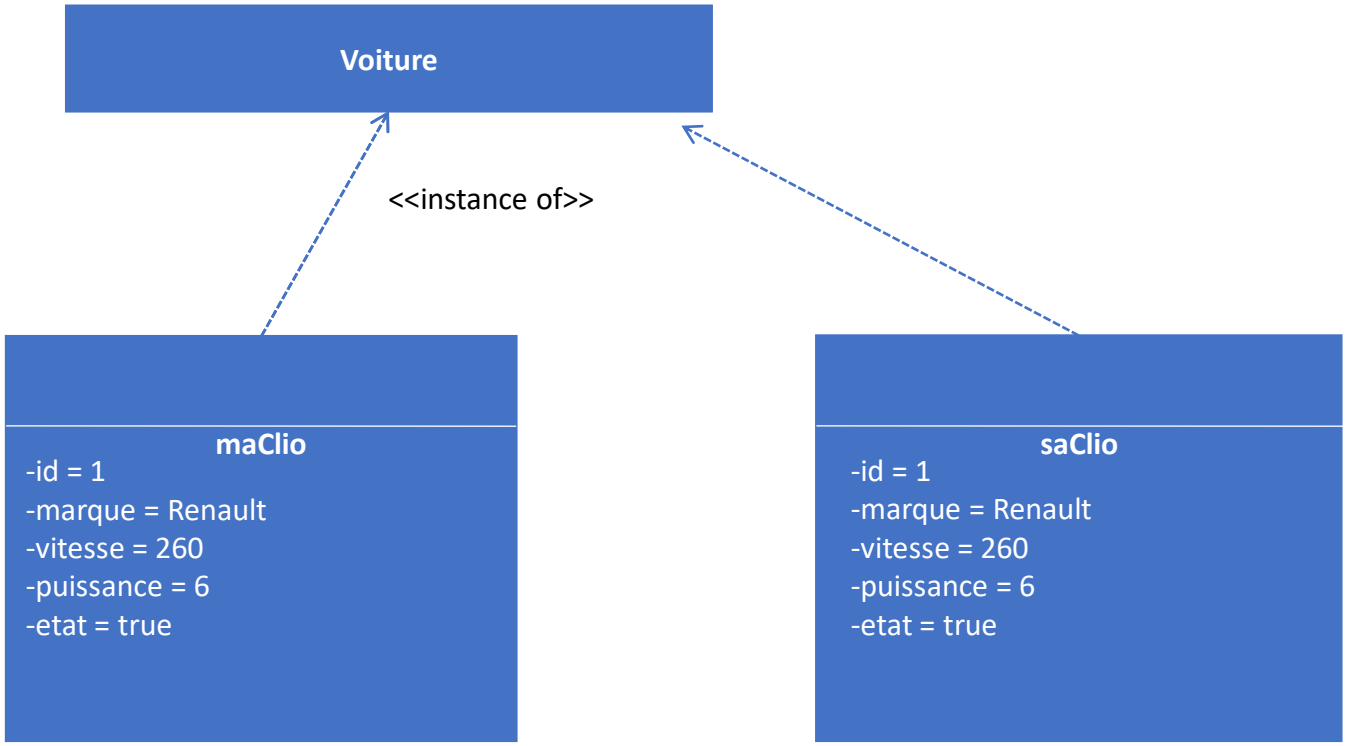
Affectation et comparaison

L'objet **maClio et saClio ont les mêmes attributs (états** identiques) mais ont des références différentes.

- **maClio != saClio**

Remarque

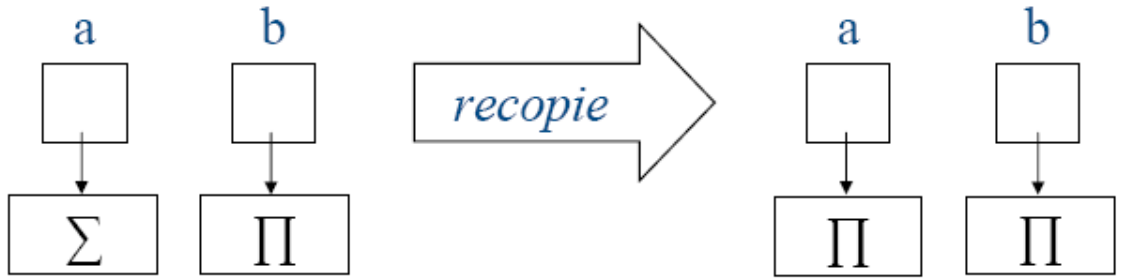
- Le test de comparaison (== et !=) entre objets ne concerne que les référence et non les attributs !



Affectation et comparaison

Recopier les attributs d'un objet « clone() » :

- Les deux objets a et b sont distincts ;
- Toute modification de a n'entraîne pas celle de b.



Comparer le contenu des objets : « equals(Object o) » :

- Renvoyer « true » si les objets a et b peuvent être considérés comme identique au vu de leurs attributs.

Cycle de vie d'un objet

Création

- Usage d'un Constructeur ;
- L'objet est créé en mémoire et les attributs de l'objet sont initialisés.

Utilisation

- Usage des Méthodes et des Attributs (non recommandé) ;
- Les attributs de l'objet peuvent être modifiés ;
- Les attributs (ou leurs dérivés) peuvent être consultés.

Remarque

L'utilisation d'un objet non construit provoque une exception de type NullPointerException.

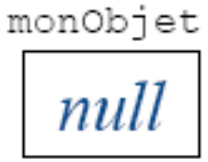
Destruction et libération de la mémoire lorsque

- Usage (éventuel) d'un Pseudo-Destructeur ;
- L'objet n'est plus référencé, la place mémoire occupée est récupérée.

Création d'objets : déroulement

La création d'un objet à partir d'une classe est appelée une **instanciation.**

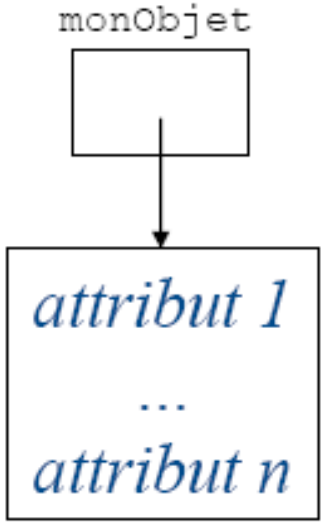
L'objet créé est une **instance de la classe.**



Déclaration

- Définit le nom et le type de l 'objet ;
- Un objet seulement déclaré vaut « **null** ».

  (mot réservé du langage)



Création et allocation de la mémoire

- Appelle de méthodes particulières : les constructeurs ;
- La création réserve la mémoire et initialise les attributs.

Renvoi d'une référence sur l'objet maintenant créé monObjet != **null.**

PARTIE 3

Création d'objets : réalisation

La création d'un nouvel objet est obtenue par l'appel à :

new Constructeur(paramètres)

- Il existe un constructeur par défaut qui ne possède pas de paramètres (si aucun autre constructeur avec paramètre n'existe) :

Les constructeurs portent le même nom que la classe :

Déclaration

Création et allocation mémoire

```
public class TestMaVoiture {

    public static void main (String[] argv)  {

        // Déclaration puis création
        Voiture maVoiture;
        maVoiture = new Voiture();

        // Déclaration et création en une seule ligne
        Voiture maSecondeVoiture = new Voiture();
    }
}
```

Création d'objets : réalisation

Exemple : construction d'objets.

```
public class TestMaVoiture {

    public static void main (String[] argv)  {

        // Déclaration puis création
        Voiture maVoiture;
        maVoiture = new Voiture();

        // Déclaration d'une deuxième voiture
        Voiture maVoitureCopie;
        // Attention!! pour l'instant maVoitureCopie vaut null


        // Test sur les références.
        if (maVoitureCopie == null) {

            // Création par affectation d'une autre référence
            maVoitureCopie = maVoiture;
            // maVoitureCopie possède la même référence que maVoiture
        }
        ...
    }}
```

Déclaration

Affectation par référence

Le constructeur de « Voiture »

Actuellement

- On a utilisé le constructeur par défaut sans paramètres ;
- On ne sait pas comment se construit la « Voiture » ;
- Les valeurs des attributs au départ sont indéfinies et identique pour chaque objet (puissance, etc..).

Rôle du constructeur en Java

- Effectuer certaines initialisations nécessaires pour le nouvel objet créé.

Toute classe Java possède au moins un constructeur

- Si une classe n'est pas définie explicitement de constructeur, un constructeur par défaut sans arguments et qui n'effectue aucune initialisation particulière est invoqué.

Remarque

- Les constructeurs portent le même nom que la classe et n'ont pas de valeur de retour.

Le constructeur de « Voiture »

Le constructeur de Voiture

- Initialise vitesse à zéro ;
- Initialise marque à «bMW » ;
- Initialise état à faux ;
- Initialise puissance à la valeur passée en paramètre du constructeur.

```
public class Voiture {
    private int id;
    private String marque;
    private float vitesse;
    private float puissance;
    private boolean etat;

    public Voiture(float puissance) {
        this.marque = "bMW";
        this.vitesse = 0;
        this.puissance = puissance;
        this.etat = false;
    }
```

Constructeur avec paramètre

Construire une «Voiture » de 7 CV

Création de la Voiture

- Déclaration de la variable maVoiture ;
- Création de l'objet avec la valeur 7 en paramètre du constructeur.

Déclaration

Création et allocation mémoire avec *Voiture(int)*

```
public class TestMaVoiture {

    public static void main(String[] argv)   {

        // Déclaration puis création
        Voiture maVoiture;

        maVoiture = new Voiture(7);

        Voiture maSecVoiture;
        // Sous entendu qu'il existe
        // explicitement un constructeur : Voiture(int)

        maSecVoiture = new Voiture(); // Erreur
    }
}
```

Constructeur sans arguments

Utilité

- Lorsque l'on doit créer un objet sans pouvoir décider des valeurs de ses attributs au moment de la création ;
- Il remplace le constructeur par défaut qui est devenu inutilisable dès qu'un constructeur (avec paramètres) a été défini dans la classe.

```
public class TestMaVoiture {

    public static void main (String[] argv)  {

        // Déclaration puis création
        Voiture maVoiture;
        maVoiture = new Voiture(7);
        Voiture maSecVoiture;
        maSecVoiture = new Voiture(); // OK
    }
}
```

```
public class Voiture {
    private int id;
    private String marque;
    private float vitesse;
    private float puissance;
    private boolean etat;

    public Voiture() {
        this.marque = "bMW";
        this.vitesse = 0;
        this.puissance = 7;
        this.etat = false;
    }

    public Voiture(float puissance) {
        this.marque = "bMW";
        this.vitesse = 0;
        this.puissance = puissance;
        this.etat = false;
    }
```

Constructeurs multiples

Intérêts

- Possibilité d'initialiser un objet de plusieurs manières différentes ;
- On parle alors de **surchage (overloaded) ;**
- Le compilateur distingue les constructeurs en fonction :
  - de la position des arguments ;
  - du nombre ;
  - du type.

```
public Voiture() {
    this.puissance = 7;
}

public Voiture(float puissance) {
    this.puissance = puissance;
}

public Voiture(float puissance, boolean etat) {

    this.puissance = puissance;
    this.etat = etat;
}
```

Remarque

- Chaque constructeur possède le même nom (le nom de la classe).

Accès aux attributs

Pour accéder aux données d'un objet on utilise une notation pointée :

identificationObjet.nomAttribut

```
public class TestMaVoiture {

    public static void main (String[] argv)  {

        // Déclaration puis création
        Voiture v1 = new Voiture();
        Voiture v2 = new Voiture();

        // Accès aux attributs en écriture
        v1.puissance = 110;

        // Accès aux attributs en lecture
        System.out.println("Puissance de v1 = "  + v1.puissance);
    }
}
```

Remarque

- Il n'est pas recommandé d'accéder directement aux attributs d'un objet.

Envoi de messages : appel de méthodes

Pour « demander » à un objet d'effectuer un traitement il faut lui **envoyer un message :**

- Un message est composé de trois parties ;
- Une référence permettant de désigner l'objet à qui le message est envoyé ;
- Le nom de la méthode ou de l'attribut à exécuter ;
- Les éventuels paramètres de la méthode :

```
identificationObjet.nomDeMethode("Paramètres éventuels")
```

Envoi de message similaire à un appel de fonction :

- Le code défini dans la méthode est exécuté ;
- Le contrôle est retourné au programme appelant.

Envoi de messages : appel de méthodes

Remarque

- Ne pas oublier les parenthèses pour les appels aux méthodes

```
public class TestMaVoiture {

    /**
     * @param args
     */
    public static void main(String[] args) {

        //Décalaration puis Création
        Voiture maVoiture = new Voiture();
        //La voiture démarre
        maVoiture.setEtat(true);

        if(maVoiture.getVitesse() == 4)
        {
            System.out.println("Pas très rapide");
        }
        //La voiture accélère
        maVoiture.setVitesse(12.5f);

    }

}
```

**Envoi d'un message à l'objet maVoiture
Appel d'un modificateur**

**Envoi d'un message à l'objet maVoiture
Appel d'un sélecteur**

PARTIE 3

Envoi de messages : passage de paramètres

Un paramètre d'une méthode peut être

- Une variable de type simple ;
- Une référence d'un objet typée par n'importe quelle classe.

En Java tout est passé par valeur

- Les paramètres effectifs d'une méthode ;
- La valeur de retour d'une méthode (si différente de void).

Les types simples

- Leur valeur est recopiée ;
- Leur modification dans la méthode n'entraîne pas celle de l'original.

Les objets

- Leur modification dans la méthode entraîne celle de l'original !
- Leur référence est recopiée et non pas les attributs

L'objet « courant »

L'objet « courant » est désigné par le mot clé **this :**

- Permet de désigner l'objet dans lequel on se trouve ;
- **Self ou current dans d'autres langages ;**
- Désigne une référence particulière qui est un membre caché.

Utilité de l'objet « courant » :

- Rendre explicite l'accès aux propres attributs et méthodes définies dans la classe ;
- Passer en paramètre d'une méthode la référence de l'objet courant.

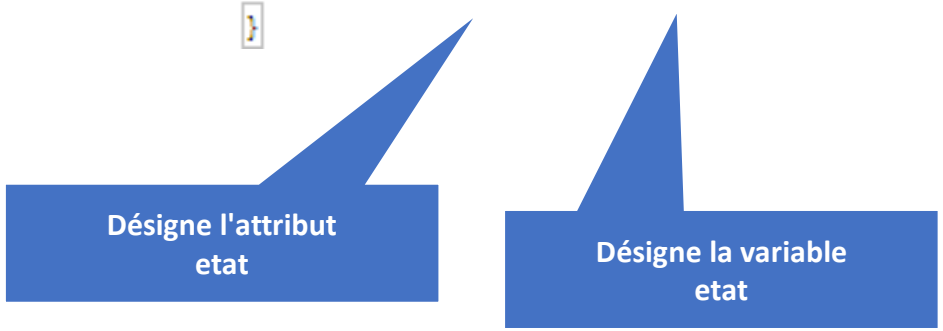
Remarque

Ne pas tenter d'affecter une nouvelle valeur à this !!!!
this = ... ; // Ne pas y penser.

L'objet « courant » : attributs et méthodes

Désigne des variables ou méthodes définies dans une classe

```
public Voiture(float puissance, boolean etat) {
    this.puissance = puissance;
    this.etat = etat;
}
```



Remarque

- this n'est pas nécessaire lorsque les identificateurs de variables ne présentent aucun équivoque.

Le retour d'UML…

- Association : les objets sont sémantiquement liés.

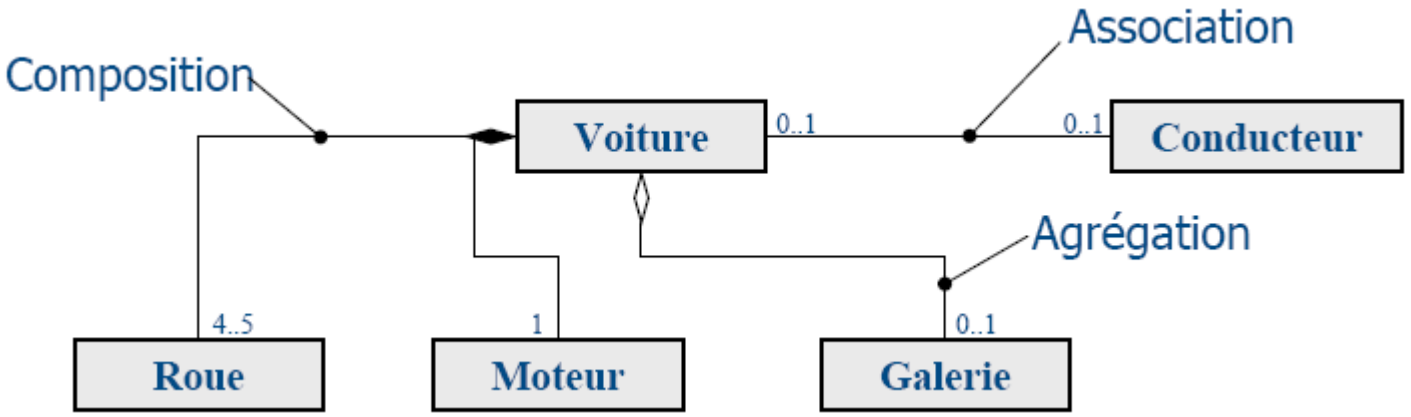
Exemple : une Voiture est conduite par un Conducteur :

- Agrégation : cycle de vie indépendant.

Exemple : une Voiture et une Galerie :

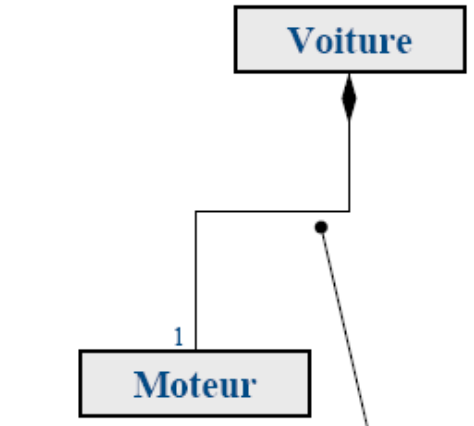
- Composition : cycle de vie identiques.

Exemple : voiture possède un moteur qui dure la vie de la voiture :



Codage de l'association : composition

L'objet de classe Voiture peut envoyer des messages à l'objet de classe Moteur : Solution 1.



A discuter : si le moteur d'une voiture est « mort », il y a la possibilité de le changer

Attribut qui stocke la référence du moteur

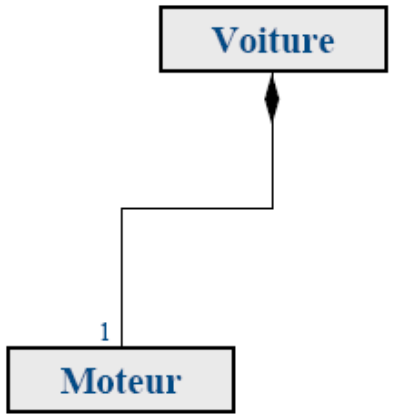
```
public class Voiture {

    private Moteur leMoteur;
    ...

    public Voiture(int p) {
        leMoteur = new Moteur(p);
        ...
    }
    ...
}
```

Création de l'objet Moteur

PARTIE 3

Codage de l'association : composition

L'objet de classe Moteur n'envoie pas de message à l'objet de classe Voiture : Solution 1



Attribut qui stocke la puissance

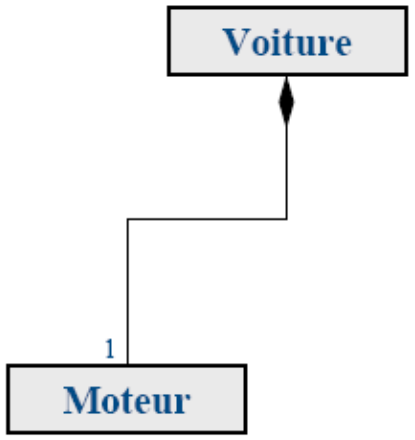
```
public class Moteur {

    private int puissance;
    ...

    public Moteur(int p) {
        puissance = p;
        ...
    }
    ...
}
```

La puissance est donnée lors de la construction

Codage de l'association : composition

Il peut être nécessaire que les deux objets en composition s'échangent des messages : Solution 2.

- L'objet Voiture « voit » l'obiet Moteur :



Attribut qui stocke la référence du Moteur

```
public class Voiture {

    private Moteur leMoteur;
    ...

    public Voiture(int p) {
        leMoteur = new Moteur(p,this);
        ...
    }
    ...
}
```

Création de l'objet Moteur

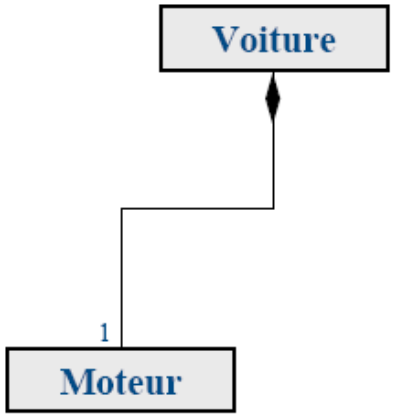
Transmission de la référence de l'objet courant

PARTIE 3

Codage de l'association : composition

Il peut être nécessaire que les deux objets en composition s'échangent des messages : Solution 2

- L'objet Moteur « voit » l'objet Voiture.



Attribut qui stocke la puissance

```
public class Moteur {

    private int puissance;
    private Voiture laVoiture;
    ...

    public Moteur(int p, Voiture v) {
        puissance = p;
        laVoiture = v;
        ...
    }
    ...
}
```

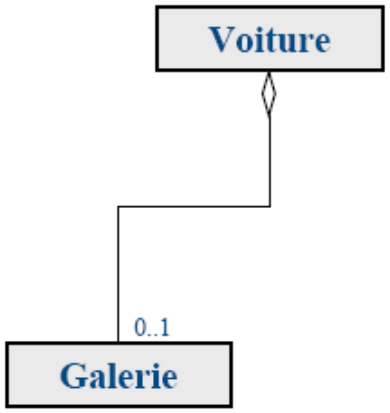
La puissance est donnée lors de la construction

Référence d'un objet Voiture en paramètre

PARTIE 3

Codage de l'association : agrégation

L'objet de classe Galerie n'envoie pas de message à l'objet de classe Voiture :



Attribut qui stocke une référence de *Galerie*

```
public class Voiture {

    private Galerie laGalerie;
    ...

    public Voiture(Galerie g) {
        laGalerie = g;
        ...
    }
    ...
}
```

Un objet *Galerie* est transmis au moment de la construction de Voiture

PARTIE 3

Destruction et ramasse-miettes

La destruction des objets se fait de manière implicite.

Le ramasse-miettes ou Garbage Collector se met en route.

- Automatiquement ;
    - Si plus aucune variable ne référence l'objet ;
    - Si le bloc dans lequel il est défini se termine ;
    - Si l'objet a été affecté à « null ».
- Manuellement :
    - Sur demande explicite par l'instruction « System.gc() ».

Un pseudo-destructeur « protected void finalize() » peut être défini explicitement par le programmeur.

- Il est appelé juste avant la libération de la mémoire par la machine virtuelle, mais on ne sait pas quand ;
- Conclusion : pas très sûr !

Remarque

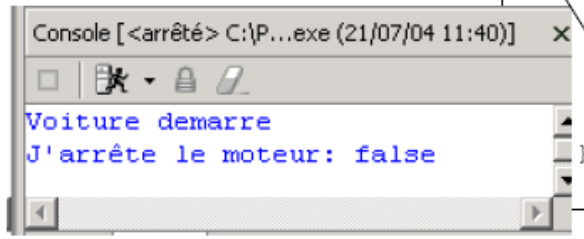
- Préférez définir une méthode et de l'invoquer avant que tout objet ne soit plus référencé : detruit().

Destruction et ramasse-miettes

```
public class Voiture {

    private boolean estDemarree;
    ...

    protected void finalize() {
        estDemarree = false;
        System.out.println("Moteur arrêté");
    }
    ...
}
```

```
public class TestMaVoiture {

    public static void main (String[] argv) {
        // Déclaration puis création de maVoiture
        Voiture maVoiture = new Voiture();
        maVoiture.demarre();
        // maVoiture ne sert plus à rien
        maVoiture = null;

        // Appel explicite du garbage collector
        System.gc();

        // Fin du programme
        System.exit(0);
        System.out.println("Message non visible");
    }
```

Force le programme à se terminer



Remarque

- Pour être sûr que finalize s'exécute il faut absolument appeler explicitement System.gc().

PARTIE 3

Gestion des objets

- Afficher son type et sa place mémoire : System.out.println() :

```
System.out.println(maVoiture) // Voiture@119c082
```

- Récupérer son type : méthode « Class getClass() » :

```
maVoiture.getClass(); // Retourne un objet de type Class
Class classVoiture = maVoiture.getClass(); // Class est une classe!!!
```

- Tester son type : opérateur « instanceof » mot clé « class » :

```
if (maVoiture instanceof Voiture) {...} // C'est vrai
```

ou

```
if (maVoiture.getClass() == Voiture.class) {...} // C 'est vrai
// également
```

Surcharge

- La surcharge overloading n'est pas limitée aux constructeurs, elle est possible également pour n'importe quelle méthode.
- Possibilité de définir des méthodes possédant le même nom mais dont les arguments diffèrent.
- Quand une méthode surchargée est invoquée, le compilateur sélectionne automatiquement la méthode dont le nombre et le type des arguments correspondent au nombre et au type des paramètres passés dans l'appel de la méthode.

Remarque

- Des méthodes surchargées peuvent avoir des types de retour différents à condition qu'elles aient des arguments différents.

Surcharge

- Exemple : une voiture surchargée.

```
public class Voiture {
    private int id;
    private String marque;
    private float vitesse;
    private float puissance;
    private boolean etat;

    public void accelre (float v){
        if(etat)
        {
            this.vitesse = this.vitesse + v;
        }
    }

    public void accelre (int v){
        if(etat)
        {
            this.vitesse = this.vitesse + v;
        }
    }
```

```
public class TestMaVoiture {

    /**
     * @param args
     */
    public static void main(String[] args) {

        //Décalaration puis Création
        Voiture maVoiture = new Voiture();

        maVoiture.accelre(12.56f);
        maVoiture.accelre(12);
        maVoiture.acc

    }

}
```

accelre(float v) void - Voiture
accelre(int v) void - Voiture

Constructeurs multiples : le retour

Appel explicite d'un constructeur de la classe à l'intérieur d'un autre constructeur

- Doit se faire comme première instruction du constructeur ;
- Utilise le mot-clé this (paramètres effectifs).

Exemple :

- Implantation du constructeur sans paramètre de Voiture à partir du constructeur avec paramètres.

```
public class Voiture {
    ...
    public Voiture() {
        this(7, new Galerie());
    }

    public Voiture(int p) {
        this(p, new Galerie());
    }

    public Voiture(int p, Galerie g) {
        puissance = p;
        moteur = new Moteur(puissance);
        galerie = g;
        ...
    }
...
```

Encapsulation

Possibilité d'accéder aux attributs d'une classe Java mais pas recommandé car contraire au principe d'encapsulation.

- Les données (attributs) doivent être protégés et accessibles pour l'extérieur par des sélecteurs.

Possibilité d'agir sur la visibilité des membres (attributs et méthodes) d'une classe vis à vis des autres classes.

Plusieurs niveaux de visibilité peuvent être définis en précédant d'un modificateur la déclaration d'un attribut, méthode ou constructeur.

- Private ;
- Public ;
- Protected (A revoir dans la partie suivante).

Encapsulation : visibilité des membres d'une classe

	+ public	- private
classe	La classe peut être utilisée par n'importe quelle classe	Utilisable uniquement par les classes définies à l'intérieur d'une autre classe. Une classe privée n'est utilisable que par sa classe englobante
attribut	Attribut accessible partout où sa classe est accessible. N'est pas recommandé du point de vue encapsulation	Attribut restreint à la classe où est faite la déclaration
méthode	Méthode accessible partout où sa classe est accessible.	Méthode accessible à l'intérieur de la définition de la classe

Encapsulation

```
public class Voiture {

    private int puissance;
    ...

    public void demarre() {
        ...
    }

    private void makeCombustion() {
        ...
    }
}
```

Remarque

- Une méthode privée ne peut plus être invoquée en dehors du code de la classe où elle est définie.

```
public class TestMaVoiture {

    public static void main (String[] argv) {
        // Déclaration puis création de maVoiture
        Voiture maVoiture = new Voiture();

        // Démarrage de maVoiture
        maVoiture.demarre();

        maVoiture.makeCombustion(); // Erreur
    }
}
```

Les chaînes de caractères «String »

- Ce sont des objets traités comme des types simples ...

- Initialisation :

```
String maChaine = "Bonjour!"; // Cela ressemble à un type simple
```

- Longueur :

```
maChaine.length(); // Avec les parenthèses car c'est une méthode
```

- Comparaison :

```
maChaine.equals("Bonjour!"); // Renvoi vrai
```

- Concaténation :

```
String essai = "ess" + "ai";
String essai = "ess".concat("ai");
```

Remarque

- Faites attention à la comparaison de chaînes de caractères.
  maChaine == "toto";
  Comparaison sur les références !

Les Chaînes modifiables «StringBuffer »

- Elles sont modifiables par insertion, ajouts, conversions, etc.
- On obtient une « StringBuffer » avec ses constructeurs :

```
StringBuffer mCM = new StringBuffer(int length);
StringBuffer mCM = new StringBuffer(String str);
```

- On peut les transformer en chaînes normales String :

```
String s = mCM.toString();
```

- On peut leur ajouter n'importe (surcharge) quoi :

```
mCM.append(...); // String, int, long, float, double, boolean, char
```

- On peut leur insérer n'importe (surcharge) quoi :

```
mCM.insert(int offset, ...); // String, int, long, float, double, boolean, char
```

Les chaînes décomposables « StringTokenizer »

- Elles permettent la décomposition en mots ou éléments suivant un délimiteur :

  this is a test => this

  is

  a

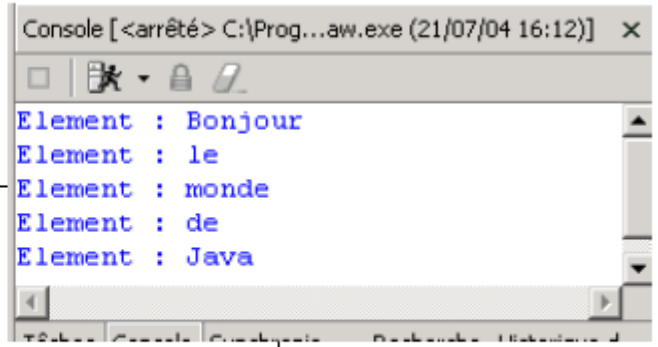
  test

- On obtient une « StringTokenizer » avec ses constructeurs :

  StringTokenizer mCM = new StringTokenize(String str); // Délimiteur = blanc

  StringTokenizer rMCM = new StringTokenizer(String str, String delim);

```
StringTokenizer st =
    new StringTokenizer("Bonjour,
        le monde|de|Java",", |");

while(st.hasMoreElements())
    System.out.println("Element : " + st.nextElement());
```



PARTIE 3

Variables de classe

Il peut être utile de définir pour une classe des attributs indépendamment des instances : nombre de Voitures créées.

Utilisation des Variables de classe comparables aux « variables globales » :

Usage des **variables de classe :**

- Variables dont il n'existe qu'un seul exemplaire associé à sa classe de définition ;
- Variables existent indépendamment du nombre d'instances de la classe qui ont été créés ;
- Variables utilisables même si aucune instance de la classe n'existe.

Variables et Constantes de classe

Exemple : constantes de classe.

```
public class Voiture {

    public static final int PTAC_MAX = 3500;
    private int poids;
    public static int nbVoitureCreees;
    ...

    public Voiture(int poids, ...) {
        this.poids = poids;
        nbVoitureCrees++;
        ...
    }
}
```

Dangereux car possibilité de modification extérieure...

```
public class TestMaVoiture {

    public static void main (String[] argv) {
        // Déclaration puis création de maVoiture
        Voiture maVoiture = new Voiture(2500);
        ...

        System.out.println("Poids maxi:" +
            Voiture.PTAC_MAX);
        System.out.println(Voiture.nbVoitureCreees);
        ...
    }
}
```

Utilisation de Variables et Constantes de classe par le nom de la classe Voiture

PARTIE 3

Méthodes de classe

Usage :

- Ce sont des méthodes qui ne s'intéressent pas à un objet particulier ;
- Utiles pour des calculs intermédiaires internes à une classe ;
- Utiles également pour retourner la valeur d'une variable de classe en visibilité private.

Elles sont définies comme les méthodes d'instances, mais avec le mot clé **static**

```
public static double vitesseMaxToleree() {
        return vitesseMaxAutorisee*1.10;
}
```

Pour y accéder, il faut utiliser non pas un identificateur d'objet mais le nom de la classe (idem variables de classe)

```
Voiture.vitesseMaxToleree()
```

Méthodes de classe : erreur classique

Exemple (suite) : méthode de classe

```
public class Voiture {

    private Galerie laGalerie;
    ...

    public Voiture(Galerie g) {
        laGalerie = g;
        ...
    }

    public static boolean isGalerieInstall() {
        return (laGalerie != null)
    }
}
```

Déclaration d'un objet Galerie non statique

Erreur : Utilisation d'un attribut non statique dans une zone statique

Remarque

- On ne peut pas utiliser dans un bloc statique (bloc, méthode) les variables d'instance (variable non statique)

Méthodes de classe

Rappel : les types simples (int, double, etc.) possèdent un alter-ego objet disposant de méthodes de conversion

Par exemple la classe **Integer « encapsule » le type int :**

- Constructeur à partir d'un int ou d'une chaîne de caractères :

```
public Integer(int value);
public Integer(String s);
```

- Disponibilité de méthodes qui permettent la conversion en type simple :

```
Integer valueObjet = new Integer(123);
int valuePrimitif = valueObjet.intValue();
```

Ou

```
int valuePrimitif = valueObjet; (AutoBoxing)
```

- Des méthodes de classe très utiles qui permettent à partir d'une chaîne de caractères de transformer en type simple ou type object :

```
String maValueChaine = new String("12313");
int maValuePrimitif = Integer.parseInt(maValueChaine);
```

Remarque

- Attention aux erreurs de conversion. Retour d'une exception.

Les tableaux en Java : application Objets

1. Déclaration

```
Voiture[] tab;
```

2. Dimensionnement

```
tab = new Voiture[3];
```

3. Initialisation

```
tab[0] = new Voiture(5);
tab[1] = new Voiture(7);
tab[2] = new Voiture(8);
```

```
for (int i = 0; i < tab.length; i++) {
        System.out.println(tab[i].demarre());
}
```

Ou 12 et 3

```
Voiture[] monTab = {
        new Voiture(5),
        new Voiture(7),
        new Voiture(8)
};
```

Varargs : passage de paramètres en nombre indéfini

- **Varargs est une nouveauté Java 5 permettant de passer en** paramètre un nombre indéfini de valeurs de même type.
- Pour ceux qui ont connu le langage Turbo Pascal, l'équivalent du System.out.println() le permettait déjà.
- Avant la version Java 5, il fallait passer en paramètre un tableau d'un type donné pour réaliser la même chose :

```
public ajouterPassager(String[] tab)
```

- La syntaxe de varargs est la suivante : utilisation de « … » :

```
public ajouterPassager(String... tab)
```

Varargs : passage de paramètres en nombre indéfini

- Du côté de la méthode où le varargs est défini, les données sont manipulées comme un tableau :

```
public ajouterPassager(String... tab) {
        for (String current : tab) {
            System.out.println(current)
        }
}
```

- Du côté client qui fait un appel à la méthode, les données peuvent être envoyées comme un :

Tableau

```
String passagers = {"Tony", "Luck" , "John"};
maVoiture.ajouterPassager(passagers);
```

- Ensemble de paramètres :

```
maVoiture.ajouterPassager("Tony", "Luck", "John");
```

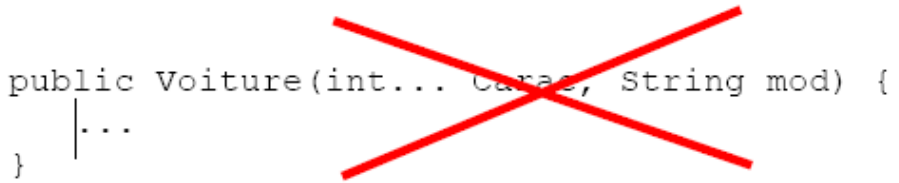
Varargs : passage de paramètres en nombre indéfini

Comme un varargs est considéré comme un tableau le contenu peut être vide :

```
public Voiture(int... carac) {
        ...
}
public static void main(String[] argv) {
        new Voiture();
}
```

Si un varargs est accompagné d'un ou plusieurs autres paramètres, le varargs doit obligatoirement être placé en dernier :

```
public Voiture(String mod, int... carac) {
    ...
}
```



Varargs : passage de paramètres en nombre indéfini

Problématiques liées à la surcharge d'une méthode utilisant un varargs

- Dans le cas de la surcharge d'une méthode la méthode contenant le varargs a la priorité la plus faible :

```
public class Voiture {
    public Voiture(int... carac) {
    }

    public Voiture(int caract1, int caract2) {
    ...
    }

    public static void main(String[] argv) {
        new Voiture(12, 23);
        new Voiture(12);
    }
}
```

Bloc Static / Non Static

```
package test;

public class Salle {
	private int id;
	private String code;

	{
		System.out.println("Bloc non static");
	}

	static {
		System.out.println("Bloc static");
	}

	public Salle(int id, String code) {
		this.id = id;
		this.code = code;
		System.out.println("Constructeur");
	}

	public static void main(String[] args) {
		new Salle(1, "CC31");
	}
}
```

Résultat d'exécution :
Bloc static
Bloc non static
Constructeur

Remarque

- Dans un bloc static on ne peut pas appeler les attributs non static.

Héritage - définition et intérêts

Héritage

- Technique offerte par les langages de programmation pour construire une classe à partir d'une (ou plusieurs) autre classe en partageant ses attributs et opérations.

Intérêts

- **Spécialisation, enrichissement : une nouvelle classe réutilise les** attributs et les opérations d 'une classe en y ajoutant et/ou des opérations particulières à la nouvelle classe.
- **Redéfinition : une nouvelle classe redéfinit les attributs et** opérations d'une classe de manière à en changer le sens et/ou le comportement pour le cas particulier défini par la nouvelle classe.
- **Réutilisation : évite de réécrire du code existant et parfois on ne** possède pas les sources de la classe à hériter.

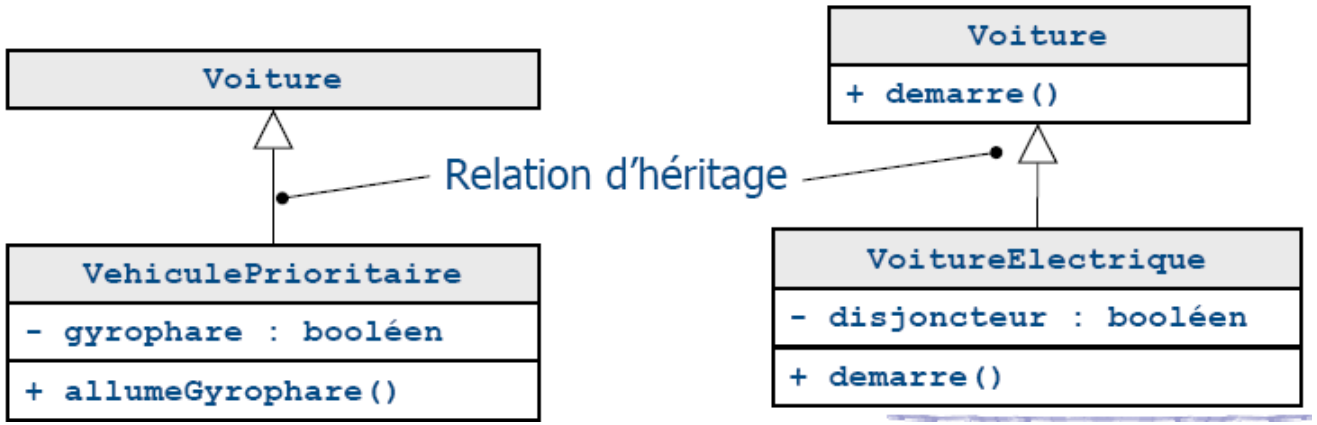
Héritage - définition et intérêts

Un véhicule prioritaire est une voiture avec un gyrophare :

- Un véhicule prioritaire répond aux mêmes messages que la Voiture ;
- On peut allumer le gyrophare d'un véhicule prioritaire.

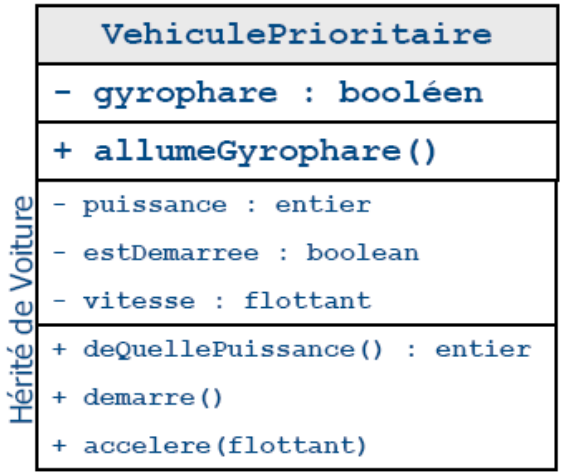
Une voiture électrique est une voiture dont l'opération de démarrage est différente :

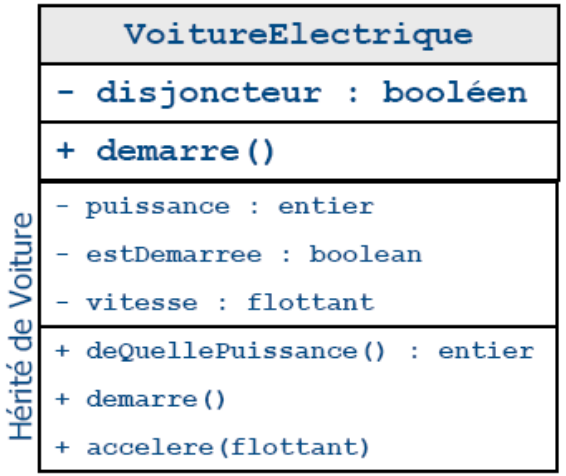
- Une voiture électrique répond aux même messages que la Voiture ;
- On démarre une voiture électrique en activant un disjoncteur.



Classes et sous-classes

Un objet de la classe **VehiculePrioritaire** ou **VoitureElectrique** est aussi un objet de la classe Voiture donc il dispose de tous les attributs et opérations de la classe Voiture :





Classes et sous-classes : terminologie

Définitions

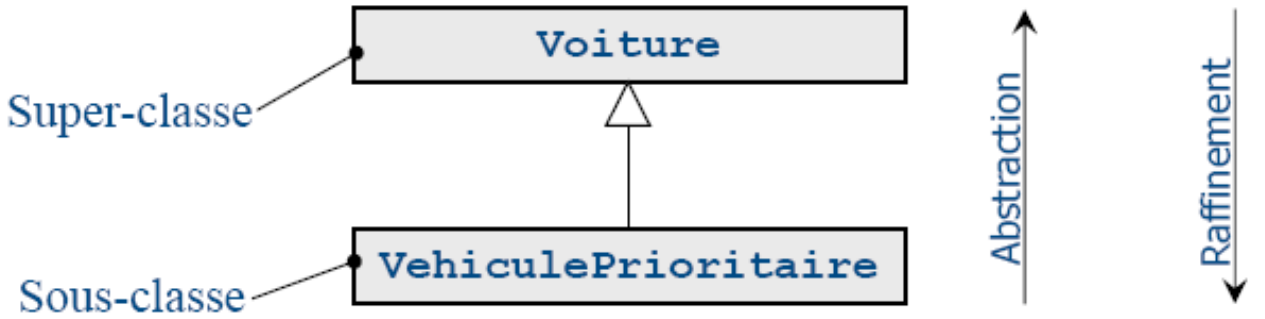
- La classe VehiculePrioritaire **hérite de la classe Voiture ;**
- Voiture est la **classe mère et VehiculePrioritaire la classe fille ;**
- Voiture est la **super-classe de la classe VehiculePrioritaire ;**
- VehiculePrioritaire est une **sous-classe de Voiture.**

Attention

- Un objet de la classe VehiculePrioritaire ou VoitureElectrique est forcément un objet de la classe Voiture ;
- Un objet de la classe Voiture n'est pas forcément un objet de la classe VehiculePrioritaire ou VoitureElectrique.

Généralisation et Spécialisation

La généralisation exprime une relation « **est-un** » **entre une** classe et sa super-classe :



L'héritage permet :

- de **généraliser dans le sens abstraction ;**
- de **spécialiser dans le sens raffinement.**

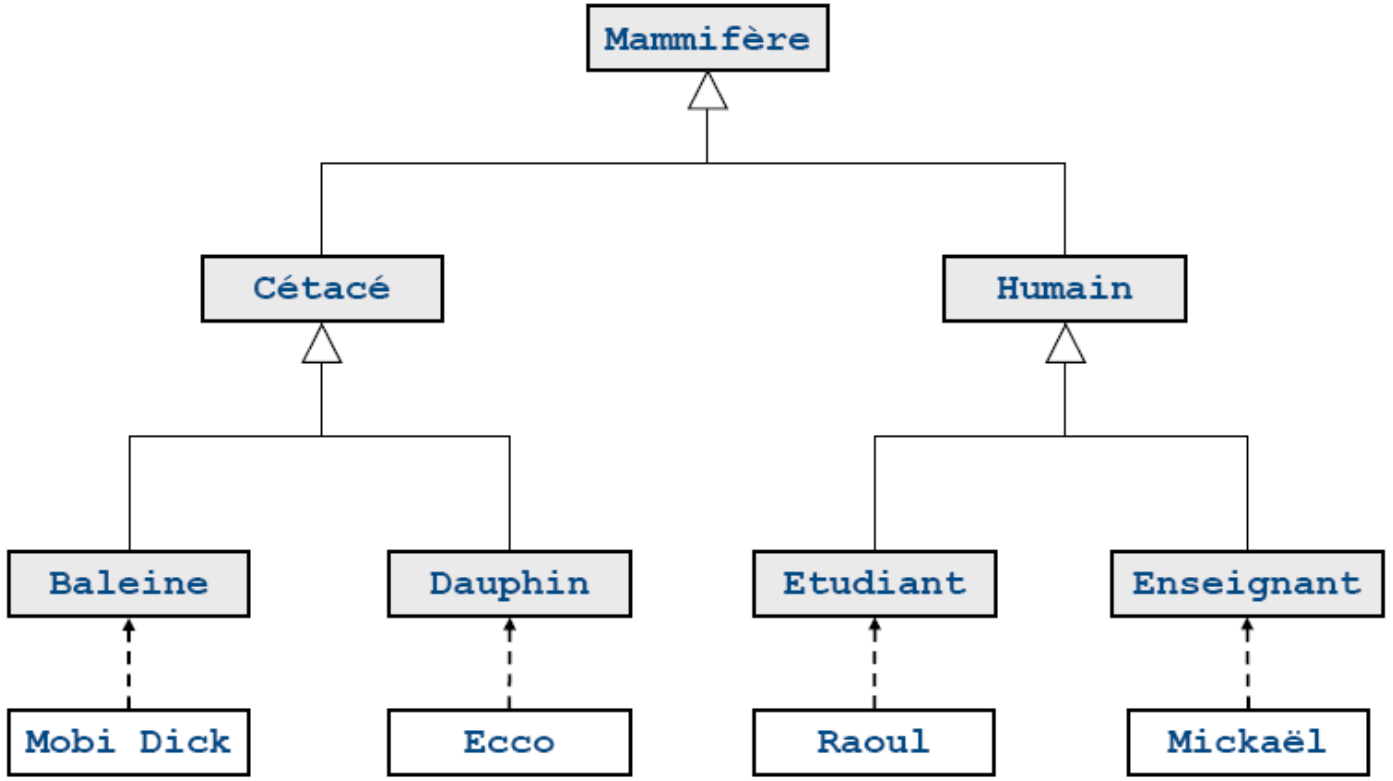
Remarque

VehiculePrioritaire est une Voiture.

PARTIE 3

Exemple d'héritage

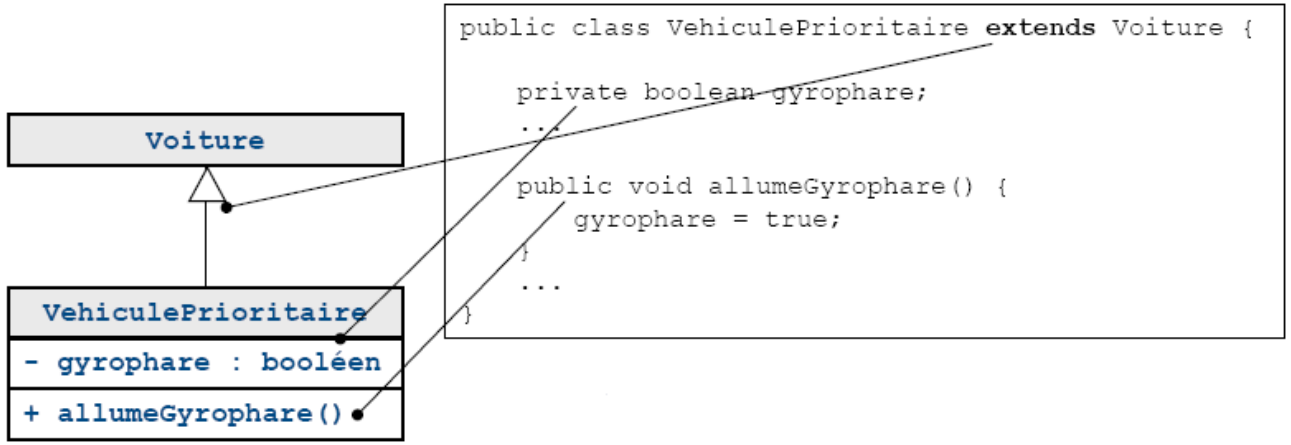
Exemple : espèces.



Héritage et Java

Héritage simple

- Une classe ne peut hériter que d'une seule autre classe ;
- Dans certains autres langages (ex : C++) possibilité d'héritage multiple ;
- Utilisation du mot-clé **extends après le nom de la classe.**

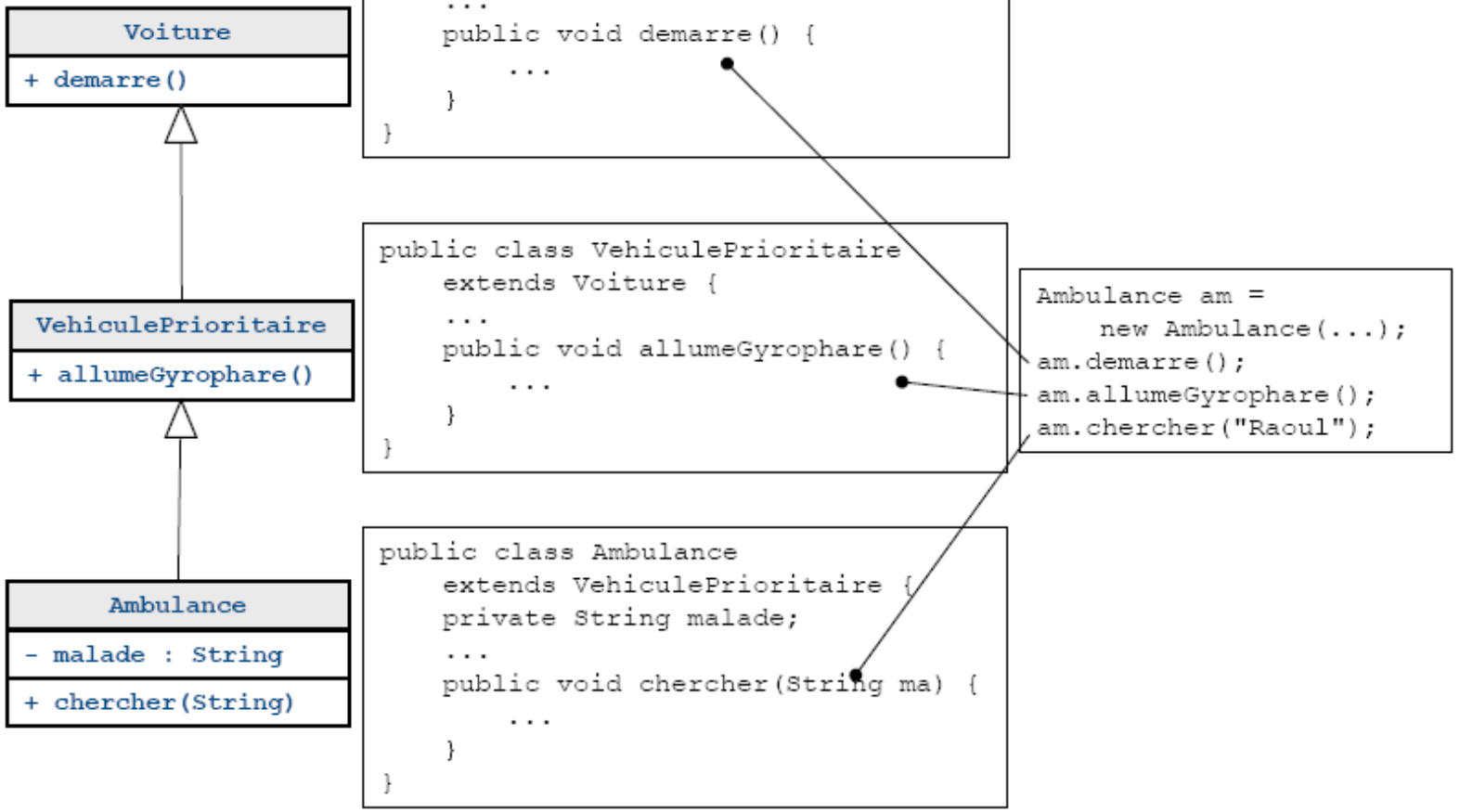


Remarque

- N'essayez pas d'hériter de plusieurs classes (extends Voiture, Sante, …) ça ne fonctionne pas

PARTIE 3

Héritage à plusieurs niveaux



PARTIE 3

Surcharge et redéfinition

L'héritage

- Une sous-classe peut ajouter des nouveaux attributs et/ou méthodes à ceux qu'elle hérite (surcharge en fait partie) ;
- Une sous-classe peut redéfinir (redéfinition) les méthodes à ceux dont elle hérite et fournir des implémentations spécifiques pour celles-ci.

Rappel de la surcharge : possibilité de définir des méthodes possédant le même nom mais dont les arguments (paramètres et valeur de retour) diffèrent.

Remarque

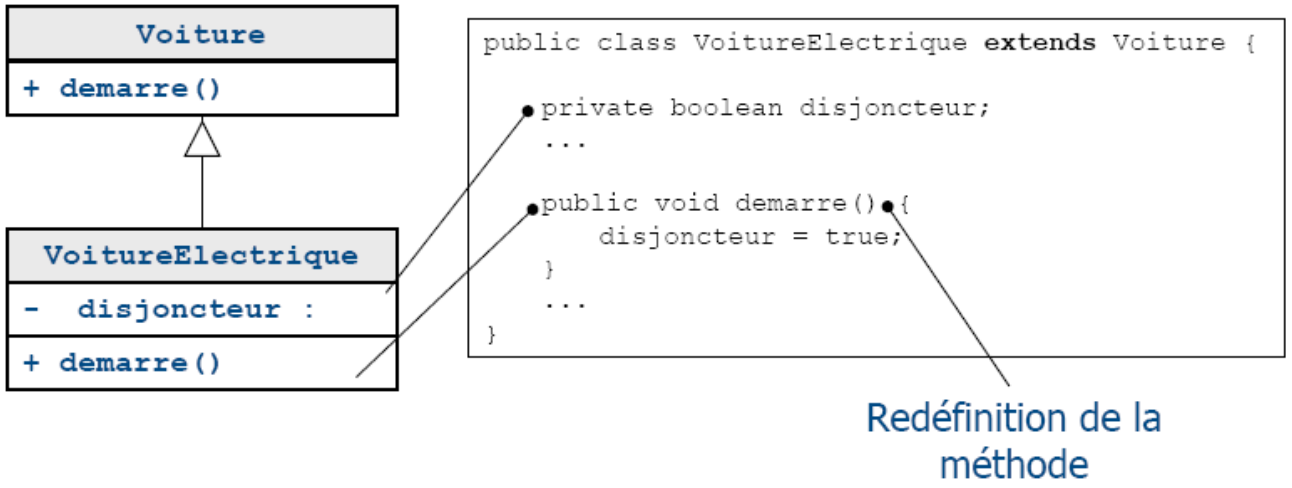
- Des méthodes surchargées peuvent avoir des types de retour différents à condition qu'elles aient des arguments différents.

Redéfinition (overriding) : lorsque la sous-classe définit une méthode dont le nom, les paramètres et le type de retour sont identiques.

Surcharge et redéfinition

Une voiture électrique est une voiture dont l'opération de démarrage est différente :

- Une voiture électrique répond aux même messages que la Voiture ;
- On démarre une voiture électrique en activant un disjoncteur.



Surcharge et redéfinition

```
public class Voiture {
    ...
    public void demarre() {
        ...
    }
}
```

Redéfinition

Surcharge

```
public class VoitureElectrique
    extends Voiture {
    ...
    public void demarre() {
        ...
    }
}
```

VoitureElectrique possède « au plus » une méthode de moins que *VehiculePrioritaire*

```
public class VehiculePrioritaire
    extends Voiture {
    ...
    public void demarre(int code) {
        ...
    }
}
```

VehiculePrioritaire possède « au plus » une méthode de plus que *VoitureElectrique*

Remarque

Ne pas confondre surcharge et redéfinition.
Dans le cas de la surcharge la sous-classe ajoute des méthodes tandis que la redéfinition « spécialise » des méthodes existantes.

PARTIE 3

Redéfinition avec réutilisation

Intérêt

- La redéfinition d'une méthode cache le code de la méthode héritée ;
- Réutiliser le code de la méthode héritée par le mot-clé **super; super permet ainsi la désignation explicite de l'instance d'une** classe dont le type est celui de la classe mère.
- Accès aux attributs et méthodes redéfinies par la classe courante mais que l'on désire utiliser :

```
super.nomSuperClasseMethodeAppelee(...);
```

Exemple de la Voiture : les limites à résoudre :

- L'appel à la méthode démarre de VoitureElectrique ne modifie que l'attribut disjoncteur.

Redéfinition avec réutilisation

```
public class Voiture {

    private boolean estDemarree;
    ...

    public void demarre() {
        estDemarree = true;
    }
}
```

Mise à jour de l'attribut *estDemarree*

```
public class VoitureElectrique extends Voiture {

    private boolean disjoncteur;
    ...

    public void demarre() {
        disjoncteur = true;
        super.demarre();
    }
    ...
}
```

```
public class TestMaVoiture {

    public static void main (String[] argv) {
        // Déclaration puis création
        VoitureElectrique laRochelle =
            new VoitureElectrique(...);
        laRochelle.demarre();
    }
}
```

Envoi d'un message par appel de *demarre*

Remarque

- La position de super n'a ici aucune importance.

PARTIE 3

Usage des constructeurs : suite

Possibilité comme les méthodes de réutiliser le code des constructeurs de la super-classe.

Appel explicite d'un constructeur de la classe mère à l'intérieur d'un constructeur de la classe fille.

- Utilise le mot-clé **super :**

super(paramètres du constructeur);

Appel implicite d'un constructeur de la classe mère est effectué quand il n'existe pas d'appel explicite. Java insère implicitement l'appel **super().**

Remarque

L'appel au constructeur de la superclasse doit se faire absolument en première instruction.

Usage des constructeurs : suite

Exemple : chaînage des constructeurs.

```
public class A {
    public A() {
        System.out.println("Classe A");
    }
}
```

```
public class B extends A {
    public B(String message) {
        super(); // Appel implicite
        System.out.println("Classe B");
        System.out.println(message);
    }
}
```

```
public class C extends B {
    public C(String debut) {
        super("Message issu C" + debut);
        System.out.println("Classe C");
        System.out.println("Fin");
    }
}
```

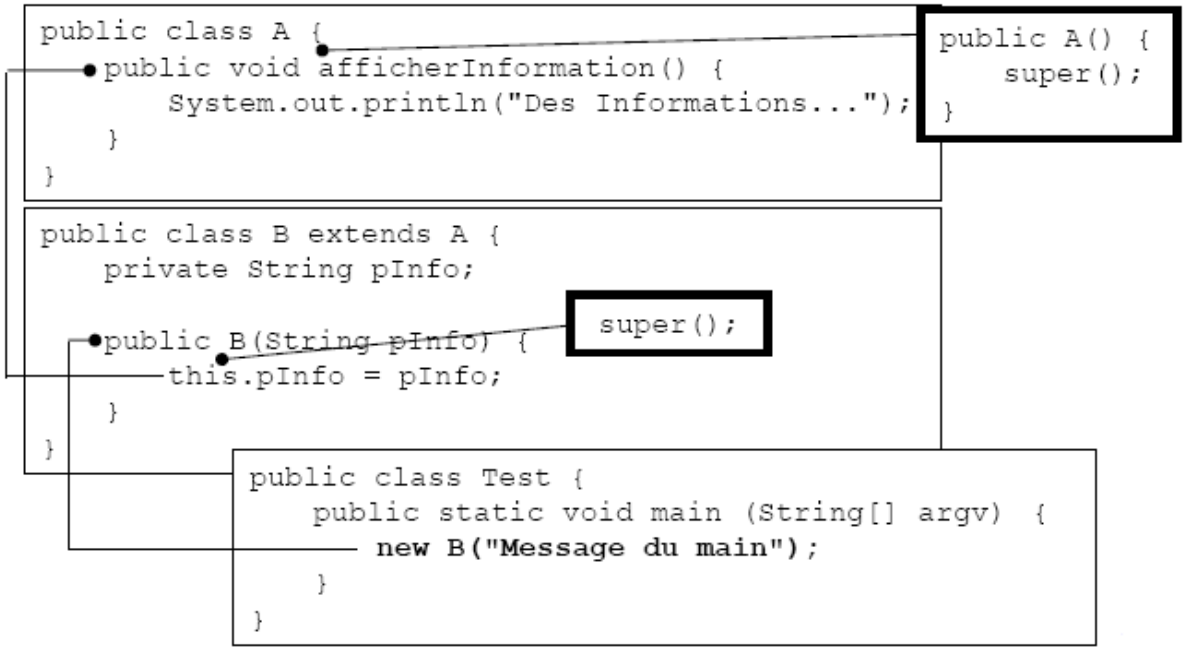
```
public class Test {
    public static void main (String[] argv) {
        new C(" et Message du main");
    }
}
```

```
Console [<arrêté> C:\Pr....exe (23/07/04 15:15)]
Classe A
Classe B
Message issu C et Message du main
Classe C
Fin
```

Usage des constructeurs : suite

Rappel : si une classe ne définit pas explicitement de constructeur, elle possède alors un constructeur par défaut

- Sans paramètre ;
- Qui ne fait rien ;
- Inutile si un autre constructeur est défini explicitement.



Usage des constructeurs : suite

```
public class Voiture {
    ...
    public Voiture(int p) {
        this(p, new Galerie());
    }

    public Voiture(int p, Galerie g) {
        puissance = p;
        moteur = new Moteur(puissance);
        galerie = g;
        ...
    }
    ...
}
```

Constructeurs explicites désactivation du constructeur par défaut

```
public class VoiturePrioritaire
    extends Voiture {

    private boolean gyrophare;

    public VoiturePrioritaire(int p, Galerie g) {
        this.gyrophare = false;
    }
}
```

```
super();
```

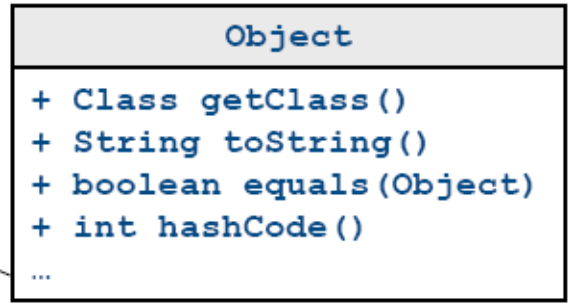
Erreur : il n'existe pas dans Voiture de constructeur sans paramètre

La classe Object : le mystère résolu

La classe **Object est la classe de plus haut niveau dans la** hiérarchie d'héritage :

- Toute classe autre que **Object possède une super-classe ;**
- Toute classe hérite directement ou indirectement de la classe **Object ;**
- Une classe qui ne définit pas de clause **extends hérite de la classe Object.**

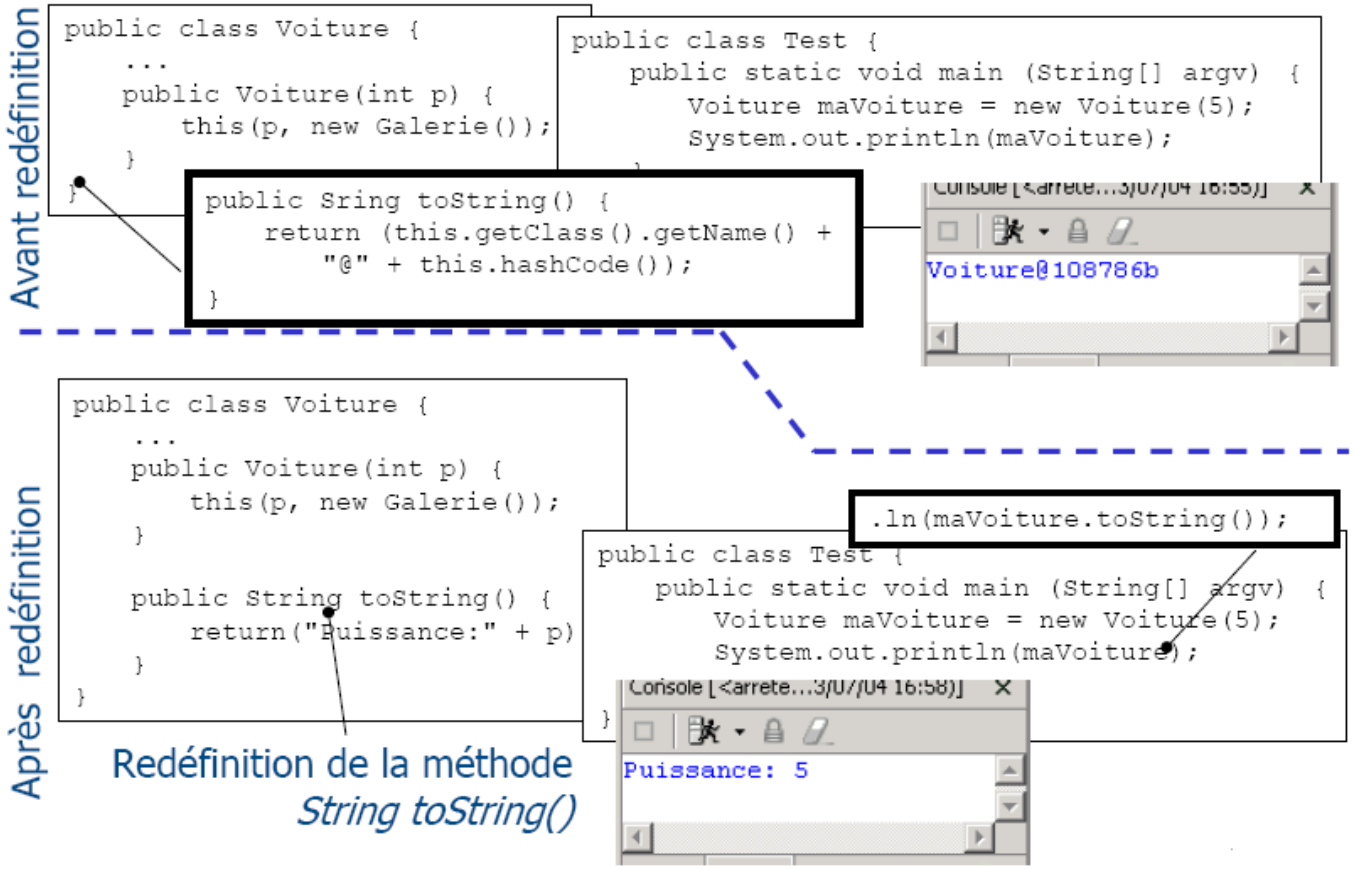
```
public class Voiture extends Object {
    ...

    public Voiture(int p, Galerie g) {
        puissance = p;
        moteur = new Moteur(puissance);
        galerie = g;
        ...
    }
    ...
}
```



Il n'est pas nécessaire d'écrire explicitement extends Object.

PARTIE 3

La classe Object : le mystère résolu



PARTIE 3

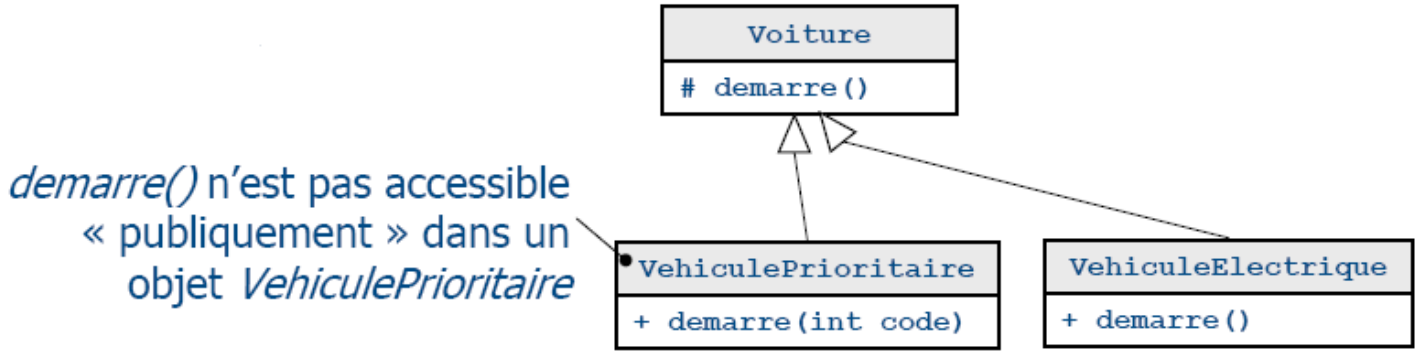
Droits d'accès aux attributs et méthodes

Exemple de la Voiture : les limites à résoudre

- demarre() est disponible dans la classe VehiculePrioritaire C'est-à-dire que l'on peut démarrer sans donner le code !
- Solution : protéger la méthode demarre() de la classe Voiture.

Réalisation

- Utilisation du mot-clé **protected devant la définition des méthodes** et/ou attributs ;
- Les membres sont accessibles dans la classe où ils sont définis, dans toutes ses sous-classes.



Droits d'accès aux attributs et méthodes

Exemple : accès aux méthodes

```
public class Voiture {

    private boolean estDemarree;
    ...

    protected void demarre() {
        estDemarree = true;
    }
}
```

```
public class VoiturePrioritaire
    extends Voiture {

    private int codeVoiture;

    public void demarre(int code) {
        if (codeVoiture == code) {
            super.demarre();
        };
    }
}
```

```
public class TestMaVoiture {

    public static void main (String[] argv) {
        // Déclaration puis création de maVoiture
        VehiculeElectrique laRochelle = new VehiculeElectrique(...);
        larochelle.demarre(); // Appel le demarre de VehiculeElectrique

        VehiculePrioritaire pompier = new VehiculePrioritaire(...);
        pompier.demarre(1234); // Appel le demarre VoiturePrioritaire
        pompier.demarre(); // Erreur puisque demarre n'est pas public
    }}
```

Méthodes et classes finales

Définition

- Utilisation du mot-clé **final.**
- Méthode : interdire une éventuelle redéfinition d'une méthode :

```
public final void demarre();
```

- Classe : interdire toute spécialisation ou héritage de la classe concernée :

```
public final class VoitureElectrique extends Voiture {
        ...
}
```

Remarque

- La classe String par exemple est finale.

Classes abstraites - définitions

Les classes abstraites servent à définir des concepts incomplets qui seront complétés dans les sous-classes.

- Une méthode abstraite est une méthode dont l'implémentation n'est pas fournie. Cette méthode devra être redéfinie dans une des sous-classes :

```
abstract void affiche();
```

- Une classe qui contient au moins une méthode abstraite doit être déclarée abstract.
- Une classe abstraite ne peut pas être instanciée.
- Une classe dérivée (qui hérite) d'une classe abstraite ne redéfinissant pas toutes les méthodes abstraites sera elle-même abstraite.

Principe

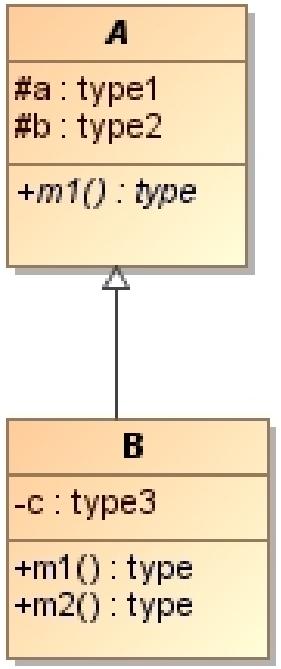


Diagramme de classe

```
public abstract class A {
        protected type1 a;
        protected type2 b;

        public A(type1 a, type2 b) {
                this.a = a;
                this.b = b;
        }

        Public abstract type m1();
}


public class B extends A {
        private type3 c;

        public B(type1 a, type2 b, type3 c) {
                super(a, b);
                this.c = c;
        }

        public type m1() {
                return …;
        }

        public type m2(){...}
}
```

Remarque

- Une classe abstraite et une méthode abstraite sont représentées en italique dans un diagramme de classe.

PARTIE 3

Exemple

```
// classe abstraite
abstract class Animal {
        // Méthode abstraite
        public abstract void animalSound();
        // Méthode concrète
        public void sleep() {
                System.out.println("Zzz");
        }
}
```

```
// Sous classe
class Pig extends Animal {
        public void animalSound() {
                // Redefinition de la méthode
                System.out.println("The pig says: wee wee");
        }
}
```

```
class Main {
        public static void main(String[] args) {
                Pig myPig = new Pig();
                myPig.animalSound();
                myPig.sleep();
        }
}
```

Polymorphisme - définition

Le polymorphisme est la capacité pour une entité de prendre plusieurs formes.

En programmation par objets, cela caractérise une entité qui fait référence au moment de l'exécution à des occurrences de différentes classes.

Principe

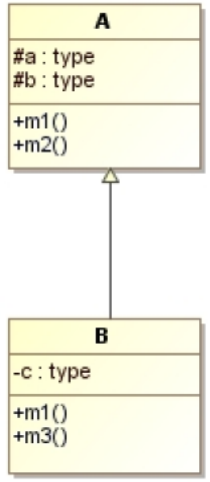


Diagramme de classe

```
A o = new A() ;
o.m1() ;
o.m2();
```

B hérite de la classe A.

```
B o = new B();
o.m1(); ok
o.m2(); ok
o.m3(); ok
```

```
A o = new B();
o.m1(); ok
o.m2(); ok
o.m3(); Erreur de compilation
```

o est un objet de type A donc on peut appliquer sur o juste les méthodes de la classe A. Dans **l'exécution** si la méthode est redéfinie dans la classe B on exécute celle de B sinon on exécute celle de A.

PARTIE 3

Synthèse

- En Java, dans la plupart des situations où il y a des relations d'héritage, la détermination de la méthode à invoquer n'est pas effectuée lors de la compilation. C'est seulement à l'exécution que la machine virtuelle déterminera la méthode à invoquer selon le type effectif de l'objet référencé à ce moment-là. Ce mécanisme s'appelle "Recherche dynamique de méthode" (Late Binding ou Dynamic Binding). Ce mécanisme de recherche dynamique (durant l'exécution de l'application) sert de base à la mise en œuvre de la propriété appelée **polymorphisme**.

- On pourrait définir le polymorphisme comme la propriété permettant à un programme de réagir de manière différenciée à l'envoi d'un même message (invocation de méthode) en fonction des objets qui reçoivent ce message.

Exemple

```
class Animal {
    public void animalSound() {
           System.out
           .println("The animal makes a sound");
    }
}
class Pig extends Animal {
    public void animalSound() {
           System.out
           .println("The pig says: wee wee");
    }
}
class Dog extends Animal {
    public void animalSound() {
           System.out
           .println("The dog says: bow wow");
    }
}
```

```
class Main {
    public static void main(String[] args) {
           Animal myAnimal = new Animal();
           Animal myPig = new Pig();
           Animal myDog = new Dog();
           myAnimal.animalSound();
           myPig.animalSound();
           myDog.animalSound();
    }
}
```

Remarque

Pourquoi et quand utiliser "Héritage" et "Polymorphisme" ?
- C'est utile pour la réutilisabilité du code : réutilisez les attributs et les méthodes d'une classe existante lorsque vous créez une nouvelle classe.

Interface - définition

- Une interface est constituée d'un ensemble de déclarations de méthodes sans implantation.
- On n'y trouve uniquement, pour chaque méthode, que la définition de son profil, c'est-à-dire son en-tête suivi de « ; ».
- Les interfaces sont définies par le mot clé interface :

```
public interface NomInterface {
    public  void affiche() ;
    //
}
```

- Si les interfaces permettent de déclarer des variables de référence portant leur type, elles ne sont pas, par contre, instanciables. En particulier, une interface ne possède pas de constructeur.

Définition

- Une classe peut implanter une ou plusieurs interface(s) : la définition de la classe comporte alors le mot clé **implements** suivi d'une liste de noms d'interfaces (les noms des interfaces implantées y sont séparés par une virgule) :

```
public class Classe implements interface1, interface 2 {

}
```

- **Synthèse**

Une interface définit un ensemble de méthodes mais ne donne pas leurs implémentations. Chaque classe qui implémente cette interface accepte d'implémenter toutes les méthodes définies dans l'interface, par conséquent accepter un certain comportement.

Remarque

- Une classe Java Hérite d'une seule classe (Héritage Simple entre les classes) ;
- Une classe Java peut Implémenter plusieurs interfaces ;
- Une interface peut Hériter de plusieurs interfaces (Héritage Multiple entre les interfaces).

Exemple

```
interface Animal {
    public void animalSound();
    public void sleep();
}
```

```
class Pig implements Animal {
    public void animalSound() {
      System.out.println("The pig says: wee wee");
    }
    public void sleep() {
      System.out.println("Zzz");
    }
}
```

```
class Main {
        public static void main(String[] args) {
                Pig myPig = new Pig();
                myPig.animalSound();
                myPig.sleep();
        }
}
```

Enumérations - définition

Une énumération est un type de données particulier, dans lequel une variable ne peut prendre qu'un nombre restreint de valeurs. Ces valeurs sont des constantes nommées, par exemple ÉTÉ, HIVER, PRINTEMPS et AUTOMNE.

Déclaration d'une énumération

La façon la plus simple de déclarer une énumération consiste à l'écrire lorsque l'on crée une variable, comme dans le code qui suit.

```
1.  enum Saisons  {ETE, HIVER, PRINTEMPS, AUTOMNE} ;

2.  Saisons saison = Saisons.HIVER ;
```

La première ligne de ce code déclare l'énumération.

La deuxième ligne instancie une variable de type Saisons (exactement comme si Saisons était d'une classe normale), et lui donne une valeur en suivant la syntaxe indiquée.

Déclaration d'une énumération

Déclarer une énumération de cette façon est possible, mais ne permet pas d'utiliser l'énumération **Saisons** ailleurs que dans la classe où elle est définie. Il est donc possible de déclarer une énumération dans un fichier séparé, comme une classe, en remplaçant simplement le mot-clé class par le mot-clé enum. La façon la plus simple de déclarer une énumération consiste à l'écrire lorsque l'on crée une variable, comme dans le code qui suit :

```
public enum Saison {    // dans le fichier Saisons.java

 ETE, HIVER, PRINTEMPS, AUTOMNE

}
```

Classe énumération

Une énumération est en fait une classe, d'où cette appellation de classe énumération. Cette classe étend la classe Enum, qui elle-même étend la classe Object, comme toutes les classes en Java.

- **Méthode toString()**

La méthode toString() de cette classe énumération est surchargée : elle retourne une chaîne de caractères qui porte le nom de la constante considérée. Dans notre exemple :

```
System.out.println(" Season : " + Saison.ETE) ;
```

```
-> ÉTÉ
```

- **Méthode valueOf()**

La méthode toString() a une méthode symétrique : valueOf(String), qui retourne la valeur énumérée à partir de sa chaîne de caractères (2 façons).

```
Season season= Season.valueOf("AUTOMNE") ;
```

```
-> season prend la valeur Season.AUTOMNE
```

```
Season season = Enum.valueOf(Season.class, "AUTOMNE") ;
```

```
-> season prend la valeur Season.AUTOMNE
```

Méthode values()

La méthode statique values() retourne un tableau de toutes les valeurs énumérées disponibles.

```
Season [] seasons = Season.values() ;
```

- **Méthode ordinal()**

La méthode ordinal() permet de retrouver le numéro d'ordre d'un élément énuméré, dans la liste de tous les éléments d'une énumération. Le premier numéro d'ordre est 0 :

```
Season season = Season.HIVER ;
System.out.println("Season : " + season + " [" + season.ordinal() + "]") ;
> Season : HIVER[1]
```

- **Méthode compareTo()**

La méthode compareTo() compare les numéros d'ordre de deux éléments énumérés. Si le premier élément est placé avant le deuxième dans la liste, alors l'entier retourné est négatif, sinon il est positif.

```
System.out.println(Season.ETE.compareTo(Season.PRINTEMPS)) ;
> -2
```

Constructeurs privés

Il est possible de définir des constructeurs privés pour chacune des valeurs énumérées.

Supposons que nous souhaitions associer à chacune de nos valeurs énumérées ETE, HIVER, PRINTEMPS, AUTOMNE, une abréviation : ETE, HIV, PRI, AUT.

```
public enum Season {  // dans le fichier Season.java
     ETE("ETE"), HIVER("HIV"), PRINTEMPS("PRI"), AUTOMNE("AUT") ;
     private String abreviation ;
     private Season(String abreviation) {
         this.abreviation = abreviation ;
    }
     public String getAbreviation() {
         return  this.abreviation ;
    }
}
```

```
Season season = Season.HIVER ;
System.out.println ("Season : " + season + " [" + season.getAbreviation() + "]") ;
> Season : HIVER [HIV]
```

Énumération dans une instruction Switch

Les énumérations sont souvent utilisées dans Switch, avec les instructions pour vérifier les valeurs correspondantes :

```
enum Level {
        LOW, MEDIUM, HIGH
}
public class Main {
        public static void main(String[] args) {
                Level myVar = Level.MEDIUM;
                switch(myVar) {
                        case LOW: System.out.println("Low level"); break;
                        case MEDIUM: System.out.println("Medium level"); break;
                        case HIGH: System.out.println("High level"); break;
                }
        }
}
```

La sortie sera : Medium level

Classes internes

En Java, il est également possible d'imbriquer des classes (une classe dans une classe). Le but des classes imbriquées est de regrouper des classes qui vont ensemble, ce qui rend le code plus lisible et maintenable.

Pour accéder à la classe interne, créez un objet de la classe externe, puis créez un objet de la classe interne :

Exemple :

```
class OuterClass {
          int x = 10;
          class InnerClass {
                    int y = 5;
          }
}
public class Main {
          public static void main(String[] args) {
                    OuterClass myOuter = new OuterClass();
                    OuterClass.InnerClass myInner = myOuter.new InnerClass();
                    System.out.println(myInner.y + myOuter.x);
          }
}
// Outputs 15 (5 + 10)
```

Classes internes privées

Contrairement à une classe "normale", une classe interne peut être private ou protected. Ainsi, les objets extérieurs n'accèdent pas à la classe interne.

Exemple :

```
class OuterClass {
          int x = 10;
          private class InnerClass {
                    int y = 5;
          }
}
public class Main {
          public static void main(String[] args) {
                    OuterClass myOuter = new OuterClass();
                    OuterClass.InnerClass myInner = myOuter.new InnerClass();
                    System.out.println(myInner.y + myOuter.x);
          }
}
```

```
Main.java:13: error: OuterClass.InnerClass has private access in OuterClass
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
                ^
```

Classes internes statiques

Une classe interne peut également être static, ainsi sera accessible sans créer d'objet de la classe externe :

Exemple :

```
class OuterClass {
          int x = 10;
          static class InnerClass {
                    int y = 5;
          }
}
public class Main {
          public static void main(String[] args) {
                    OuterClass.InnerClass myInner = new OuterClass.InnerClass();
                    System.out.println(myInner.y);
          }
}

//Outputs 5
```

Accéder à la classe externe à partir de la classe interne

L'un des avantages des classes internes est qu'elles peuvent accéder aux attributs et aux méthodes de la classe externe :

Exemple :

```
class OuterClass {
          int x = 10; class InnerClass {
                    public int myInnerMethod() {
                              return x;
                    }
          }
}
public class Main {
          public static void main(String[] args) {
                    OuterClass myOuter = new OuterClass();
                    OuterClass.InnerClass myInner = myOuter.new InnerClass();
                    System.out.println(myInner.myInnerMethod());
          }
}
// Outputs 10
```

OFPPT
La Voie de l'Avenir

WEBFORCE
BE THE CHANGE

CHAPITRE n° 1

Coder une application en JAVA

Contexte

Un programme peut échouer pour de nombreuses raisons:

- Entrée incorrecte ;
- Erreur de programmation ;
- …

L'arrêt violent d'un programme suite à l'apparition d'une erreur peut parfois être évité.

Deux aspects de la gestion des erreurs:

- Détection et rétablissement.

Contexte

Exemple : la méthode part de la classe Integer lorsque l'objet String ne peut être transformé en entier.

Cette méthode ne permet pas de proposer une solution à cette situation. Ainsi, elle génère une exception :

```
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int  n = Integer.parseInt(sc.next());
}
```

```
java
Exception in thread "main" java.lang.NumberFormatException: For input string: "java"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
        at java.lang.Integer.parseInt(Integer.java:447)
        at java.lang.Integer.parseInt(Integer.java:497)
        at Test.main(Test.java:21)
```

Définition

La gestion des exceptions en Java offre un mécanisme flexible pour passer le contrôle depuis le point de détection de l'erreur à un gestionnaire de récupération de l'erreur.

Lorsqu'une méthode détecte une erreur, elle lève une exception. Celle-ci est équivalente à un signal ou à une interruption logicielle.

Une exception est un **objet** Java (instance de la classe Exception) envoyé à la JVM.

Principe (1)

- Toute méthode qui a un problème lève (**throw**) une exception.

- Tout code qui appelle cette méthode doit attraper (**catch**) cette exception. On peut alors prendre des mesures pour rétablir la situation.

- La plupart des exceptions demandent à être "attrapées" faute de quoi le programme s'arrête en cours d'exécution

Principe (2)

- La recherche se fait en local, puis remonte la liste des appelants - acquittement de l'exception et reprise du code.

Structures spécialisées :

try {...}

catch {...}

Localisation du traitement

- Les traitements des erreurs et conditions « anormales » s'effectuent dans une zone du programme spéciale (bloc « catch »).
- Plutôt que de compliquer le code du traitement normal, on traite les conditions anormales à part.
- Le traitement « normal » apparaît ainsi plus simple et plus lisible.

Exemple

```
try {
                x = System.in.read();
}
catch (IOException e) {
                System.out.println("Erreur : " +  e.getMessage());
}
```

Traitement normal

Bloc try-catch

Pour un seule bloc try nous pouvons avoir plusieurs bloc catch, chaque bloc catch attrape une exception spécifique.

```
try {
            ...
} catch (UneException e) {
            // traitement de l'exception e
} catch (UneAutreException e2) {
            // traitement de l'exception e2
}
```

Terminologie

- Une instruction ou une méthode peut lever ou lancer (throw) une exception ;
- Une anomalie de fonctionnement provoque la création d'une exception.

Terminologie

- Une méthode peut attraper ou traiter (catch) une exception.

- Une méthode peut laisser se propager une exception :

    - En n’attrapant pas l'erreur, elle remonte alors vers la méthode appelante qui peut elle-même l'attraper ou la laisser remonter.

throw

throw expression

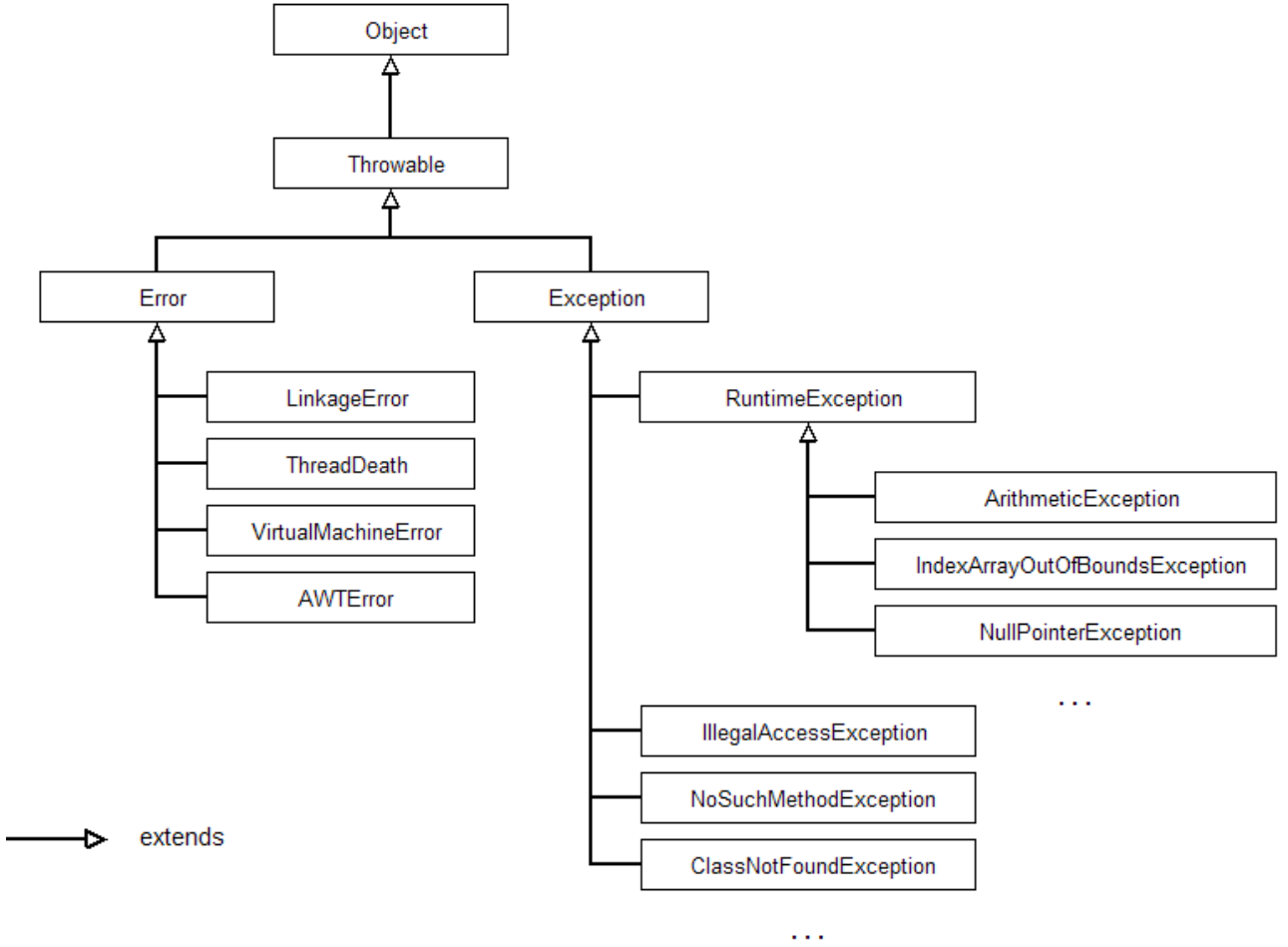
- L'expression doit retourner un objet du type exception décrivant l'exception ou l'erreur qui s'est produite.
- Lorsque l'interpréteur rencontre une instruction throw, il stoppe l'exécution du programme et cherche un gestionnaire d'exception capable de saisir l'exception (un bloc try/catch).

Exemple

```
public static long factoriel(int x) throws IllegalArgumentException {
        if(x < 0) throw new IllegalArgumentException("x < 0");
        long fact = 1;
        for (int i = 1; i <= x; i++){
                fact = fact * i;
        }
        return fact;
}

public static void main(String[] args) {
        try {
                System.out.println(factoriel(-3));
        } catch (IllegalArgumentException e) {
                System.out.println(e.getMessage());
        }
}
```

PARTIE 3

Hiérarchie des exceptions



Types d'exceptions

- **Throwable** possède 2 sous classes standard: **java.lang.Error** et **java.lang.Exception ;**

- Les exceptions qui sont des sous classes de **Error** sont généralement irrécupérables (ex: la VM n'a plus de mémoire) ;

- Les exceptions sous classes de **Exception** indiquent des situations moins sévères (ex: EOFException signalant la fin d'un fichier ou ArrayIndexOutOfBoundsException) indiquant l'accès en dehors des limites d'un tableau.

Données et méthodes d'un objet exception

- En tant qu'objet, une exception peut contenir des données et des méthodes.
- Throwable possède un champ String stockant un message d'erreur décrivant la situation.
- On peut la récupérer à l'aide de la méthode getMessage().

Exceptions contrôlées et non contrôlées (1)

- Non contrôlé pour le type Error ;
- Contrôlé pour le type Exception sauf si RuntimeException ;
- Les exceptions contrôlées ne se produisent que dans des circonstances spécifiques et bien définies ;
- Si on écrit une méthode qui lève une exception contrôlée, on doit utiliser une clause **throws** afin de déclarer l'exception au sein de la signature de la méthode.

Exceptions contrôlées et non contrôlées (2)

On parle d'exception contrôlée par le compilateur, Java effectue un contrôle afin de s'assurer qu'il y a une déclaration dans les signatures des méthodes. Il provoquera une erreur de compilation si ce n'est pas le cas.

Exception contrôlée (1)

```
public int m() throws MonException {

            …

 }
```

- Si une méthode m1() fait appel à m() :
    - Soit m1() entoure l'appel de m() avec un bloc try catch(MonException) ;
    - Soit m1() déclare qu'elle peut lancer le même type d'exception que m() (ou bien une super classe de l'exception).

Exception contrôlée (2)

Une méthode comportant un throws ne lance pas nécessairement cette exception.

- Intérêt: la méthode risque d'être redéfinie par du code qui renverra cette exception.

Une méthode peut renvoyer plusieurs types d'exceptions :

- Public int m2() throws MonException, TonException ;
- Les exceptions peuvent être contrôlées ou non.

try/catch/finally

- C'est le mécanisme de gestion des exceptions ;
- La clause try établit un bloc de code sujet au traitement des exceptions ;
- Il peut être suivit de plusieurs clauses catch chacune d'elles représentant un bloc d'instructions conçues pour gérer un type d'exception spécifique ;
- La clause finally est facultative et suit les blocs catch ;
- Il y a au moins une clause catch/finally pour un try.

catch

- Chaque clause catch est déclarée avec un seul argument spécifiant le type d'exceptions ;
- Lorsqu'une exception est levée, l'interpréteur Java cherche une clause catch possédant un argument du même type ou une super classe de ce type ;
- Le code contenu dans cette clause doit prendre les mesures s'imposant pour sortir de la condition exceptionnelle.

finally

- Ce bloc contient les instructions qui sont toujours exécutées après avoir quitté la clause try, indépendamment du type de sortie (normalement, exception gérée par un catch, exception non saisie, return, break, continue).

- Seul l'appel de System.exit() empêchera l'exécution du bloc finally.

Messages d'erreurs

- Les messages d'erreurs affichées par la JVM correspondent à la sortie de la méthode printStackTrace ;
- Une bonne compréhension de ces messages fait gagner du temps pour corriger le code ;
- Le message commence par le message d'erreur associé à l'erreur ;
- Ensuite on voit la pile des méthodes traversées depuis la méthode main pour arriver à l'erreur.

Messages d'erreurs

- La méthode la plus récemment exécutée est en premier, la méthode main en dernier.

- A chaque étape, on obtient le numéro de la ligne en cause :

```
Exception in thread "main" java.lang.IllegalArgumentException: x < 0
at Test.factoriel(Test.java:13)
at Test.main(Test.java:23)
```

CHAPITRE n° 1

Coder une application en JAVA

1. Introduction à JAVA
2. Introduction des notions de base en JAVA
3. Programmation OO en JAVA
4. Gestion des exceptions en JAVA
5. **Gestion des entrées et sorties**
6. Manipulation des collections

Contexte

- Un programme a souvent besoin de lire des données depuis une source externe ou d'écrire des données vers une destination externe ;
- La source et la destination de ces échanges peuvent être de natures multiples : fichier, socket réseau, clavier, écran... ;
- De même, la nature des données échangées peut être diverse : texte, images, son....

Contexte

- En java, le mécanisme de flux (‘stream’) permet de représenter tous ces processus d’envoi/réception de données.
- Les flux traitent toujours les données de façon séquentielle :
    - À l’ouverture du flux, la position courante est définie (ex: au début d’un fichier ouvert en lecture) ;
    - Dès qu’une donnée est lue ou écrite, la position courante avance ; le flux n’a aucune mémoire et donne toujours accès à une seule donnée à un instant donné.

Les flux

- Java met à disposition dans le package java.io.*, deux grandes catégories de flux :
    - Les flux de caractères (textes, html, xml…) et les flux d'octets (nombres entiers/réels, images, sons…) ;
    - Les flux d'entrée (lecture) et les flux de sortie (écriture).

Les classes de flux

- Les noms des classes de flux en java se décomposent en un préfixe suivi d'un suffixe.

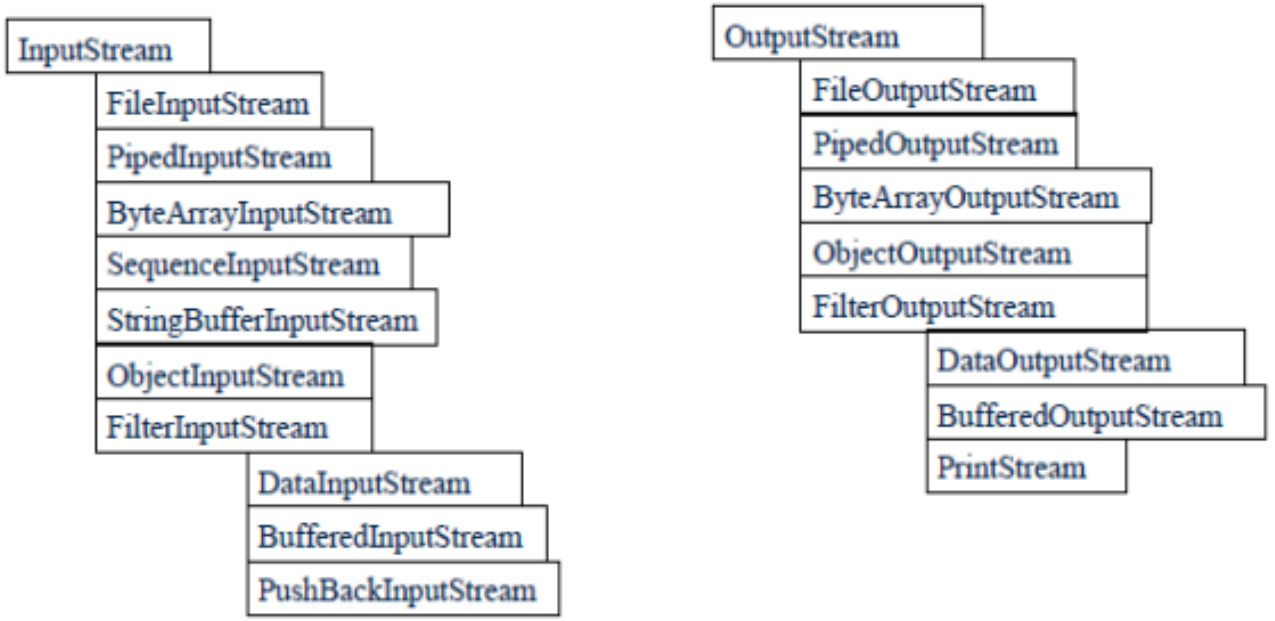
- Suffixes (classes abstraites de flux) :

	Flux d'octets	Flux de caractères
Flux entré	InputStream	Reader
Flux sortie	OutputStream	Writer

- Préfixes: indiquent avec quoi le flux communique (File, Object, String) et le type de traitements ou filtre associé (Buffered, Data…).

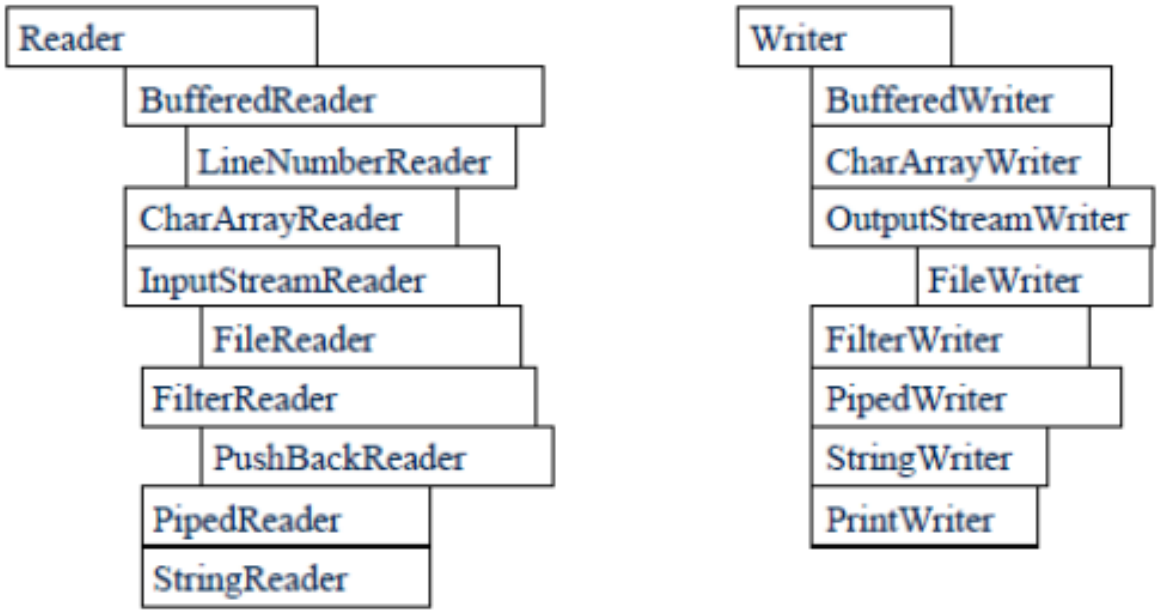
Classes de flux d'octets

La figure suivante présente la hiérarchie des classes de flux d'octets :



Les classes de flux de caractères

La figure suivante présente la hiérarchie des classes de flux de caractères :



Principales méthodes

Principales méthodes de lecture:

- int read() : prochain élément du flux (ou -1 si « fin de flux »)
- int read(byte[] buf) pour InputStream
- int read(char[] buf) pour Reader
- long skip(long nb) : saute nb éléments
- void close()

Principales méthodes d'écriture:

- void write(int c)
- void write (byte[] / char[] buf)
- void write(String s) : pour writer
- void flush()
- void close()

Les flux « concrets » (1/2)

Lecture/écriture (séquentielle) dans fichiers :

- FileInputStream/FileOutputStream
- FileReader/FileWriter
- Nom du fichier spécifié dans le constructeur, par exemple: new FileReader(filename)

Lecture/écriture dans tableau en mémoire :

- ByteArrayInput(/Output)Stream
- CharArrayReader/CharArrayWriter
- Tableau donné dans le constructeur : new CharArrayReader(tab), récupéré par toByteArray() (respectivement toCharArray()).

Les flux « concrets » (2/2)

Lecture/écriture dans une chaîne :

- StringReader/StringWriter
- Chaîne donnée dans le constructeur (new StringReader(ch)), ou récupérée par toString()

Enchaînements de flux « pipes » :

- PipedInputStream/PipedOutputstream
- PipedReader/PipedWriter

Conversions pour flux binaire

Pour lire/écrire autre chose que des octets ou caractères, on utilise des classes avec d'autres méthodes, qui font la conversion avec les flux « concrets » de bas niveau :

- **Flux données binaires (types primitifs) :**
    - DataInputStream/DataOutputStream ;
    - Ex: New DataOutputStream(outStream), ...
    - Méthodes de lecture : readFloat(), readInt(),... et écriture : writeFloat(x), writeInt(i),...
- **Flux données binaires (objets persistants) :**
    - ObjectInputStream / ObjectOutputStream
    - Mêmes méthodes que DataXXXStream + Object readObject() (resp. writeObject(o))
    - Objets doivent être de classe implémentant l'interface Serializable

Autres conversions de flux

- **Flux « avec tampon » (pour éviter des accès disque ou réseau à chaque lecture/écriture) :**
  - BufferedInputStream / BufferedOutputStream
  - BufferedReader / BufferredWriter
- **Flux pour écriture formatée en mode texte (comme pratiqué sur System.out) :**
  - Classe PrintWriter avec comme méthodes utiles : print(), println(), printf() ...
- **Flux compressés :**
  - package java.util.zip (classes ZipInputStream, ZipOutputStream,...)
- **Conversion flux octets / flux caractères :**
  - InputStreamReader/OutputStreamWriter

En pratique : écriture d'un fichier « texte»

1) **Importer les classes nécessaires :**

   - import java.io.*;

2) **Ouvrir un flux de caractères vers un fichier :**

   - FileWriter out = new FileWriter("nomFichier");

3) **Ecrire les caractères et/ou chaînes de caractères sur ce flux :**

   - out.write(ch); // ch peut être : un char ou une String

4) **Fermeture du flux :**

   - out.close();

Remarque

- Remarque : chaque étape (sauf l'import) est susceptible de lancer une IOException donc mettre les instructions à risque dans un bloc try/catch ou bien, utiliser une clause throws.

En pratique : lecture d'un fichier « texte»

1) **Importer les classes nécessaires :**

   - import java.io.*;

2) **Ouvrir un flux de caractères depuis un fichier :**

   - FileReader in = new FileReader("nomFichier");

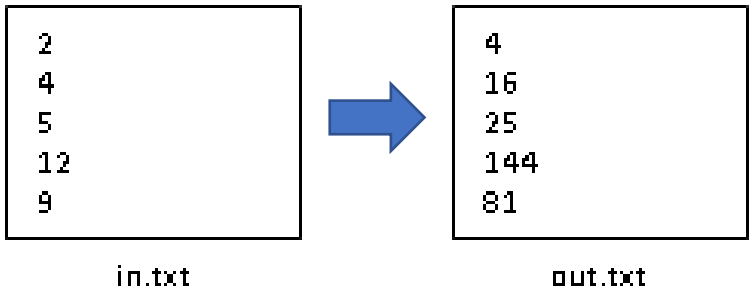
3) **Lire caractère par caractère sur ce flux :**

   - ch = in.read(); // ch vaut -1 en fin de fichier

4) **Fermeture du flux :**

   - in.close();

Exemple :




```
import java.io.*;
public class Copy {
        public static void main(String[] args) {
                try {
                        BufferedReader in = new BufferedReader(new FileReader("file/in.txt"));
                        PrintWriter out = new PrintWriter(new FileWriter("file/out.txt"));
                        String str;
                        while(((str = in.readLine()) != null)){
                                out.println((int)Math.pow(Integer.parseInt(str), 2));
                        }
                        out.close();
                        in.close();
                } catch (FileNotFoundException e) {
                        System.out.println("Fichier introuvable");
                } catch (IOException e) {
                        e.getMessage();
                }
        }
}
```

En pratique : lecture d'un fichier « texte»

Remarques :

- Chaque étape (sauf l'import) peut lancer une IOException: bloc try/catch (ou throws).
- Vous pouvez utiliser la classe BufferedReader qui dispose d'une méthode readLine qui permet de lire chacune des lignes du fichier (avec FileReader seule, on ne pourrait accéder qu'à des caractères).

```
BufferedReader in = new BufferedReader (new FileReader ("nomFichier")) ;

ligne = in.readLine() ; // ligne vaut null en fin de fichiera
```

En pratique : écriture d'un fichier « binaire »

1) **Importer : import java.io.*;**

2) **Ouvrir un flux d'octets vers un fichier :**

- FileOutputStream out = new FileOutputStream("nomFich");

3) **Brancher un flux d'écriture d'objets et de données sur le flux d'octets :**

- DataOutputStream datOut = new DataOutputStream(out);

4) **Ecrire les nombres et/ou objets sur ce flux :**

- datOut.writeInt(i); // i entier
- datOut.writeFloat(x); // x flottant
- ……

5) **Fermeture des flux : datOut.close();v**

En pratique : lecture d'un fichier « binaire »

1) **Importer : import java.io.*;**

2) **Ouvrir un flux d'octets depuis un fichier :**

- FileInputStream in = new FileInputStream("nomFich");

3) **Brancher un flux de lecture d'objets et de données sur le flux d'octets :**

- DataInputStream datIn = new DataInputStream(in);

4) **Lire les nombres et/ou objets sur ce flux :**

- int i = datIn.readInt();
- float x = datIn.readFloat();
- double y = datIn.readDouble();
- …

5) **Fermeture des flux : datIn.close();**

Conseil pratique

Pour tous vos fichiers utilisez systématiquement les flux d'entrées et de sorties bufférisés (BufferedWriter et BufferedReader par exemple, pour vos fichiers de textes).

- Dans le cas d'un flux non bufférisé le programme lit ou écrit par exemple sur le disque dur, les données au fur et à mesure, alors que les accès disque sont excessivement coûteux en temps.

Lecture/écriture simultanées et non-séquentielles sur fichier

- **Possible via classe RandomAccessFile**
- **constructeur : RandomAccessFile(filename,mode)**
    - mode valant "r" (readonly) ou "rw" (read/write)
- **écrire sur le fichier :**
    - raf.writeInt(i); // i entier
    - raf.writeFloat(x); // x flottant
    - raf.writeUTF(s); // s String
    - …
- **lire sur le fichier :**
    - int i = raf.readInt();
    - String s = raf.readUTF();
- **taille :** int len = raf.length()
- **déplacement dans le fichier :** raf.seek(pos);

Classes utilitaires pour entrées-sorties

- **File** : pour manipuler fichiers/répertoires (accès à la taille, listage du contenu, suppression, renommage, …).
    - constructeurs :
        - File(String name), File(String path, String name), ...
    - méthodes :
        - boolean exists(), long length(), File getParent() boolean isDirectory(), String[] list() void delete(), void mkdir(), boolean renameTo(File f), ...

Classes utilitaires pour entrées-sorties

- **StreamTokenizer** : pour découpage de flux de caractères en « jetons (ou tokens) » de type mot, nombre, blanc ou caractère :
    - constructeur : StreamTokenizer(Reader r) ;
    - lecture jeton suivant : int nextToken() renvoie et met dans le champ ttype le type du jeton parmi TT_WORD, TT_NUMBER, TT_EOL, TT_EOF, et met le nombre dans le champ nval, ou la chaîne dans le champ sval le cas échéant.

CHAPITRE n° 1

Coder une application en JAVA

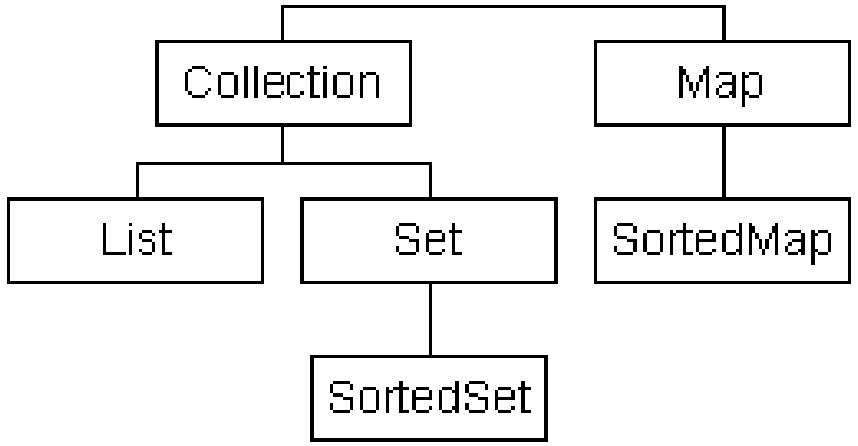
1. Introduction à JAVA
2. Introduction des notions de base en JAVA
3. Programmation OO en JAVA
4. Gestion des exceptions en JAVA
5. Gestion des entrées et sorties
6. **Manipulation des collections**

Définition

- Les collections sont des objets qui permettent de gérer des ensembles d'objets. Ces ensembles de données peuvent être définis avec plusieurs caractéristiques : la possibilité de gérer des doublons, de gérer un ordre de tri, etc. ...
- Chaque objet contenu dans une collection est appelé un élément.
- L'API Collections propose un ensemble d'interfaces et de classes dont le but est de stocker de multiples objets. Elle propose quatre grandes familles de collections, chacune définie par une interface de base :
    - **List** : collection d'éléments ordonnés qui accepte les doublons ;
    - **Set** : collection d'éléments non ordonnés par défaut qui n'accepte pas les doublons ;
    - **Map** : collection sous la forme d'une association de paires clé/valeur ;
    - **Queue** et **Deque** : collections qui stockent des éléments dans un certain ordre avant qu'ils ne soient extraits pour traitement.

Les interfaces des collections

- Le Framework de java 2 définit 6 interfaces en relation directe avec les collections qui sont regroupées dans deux arborescences :

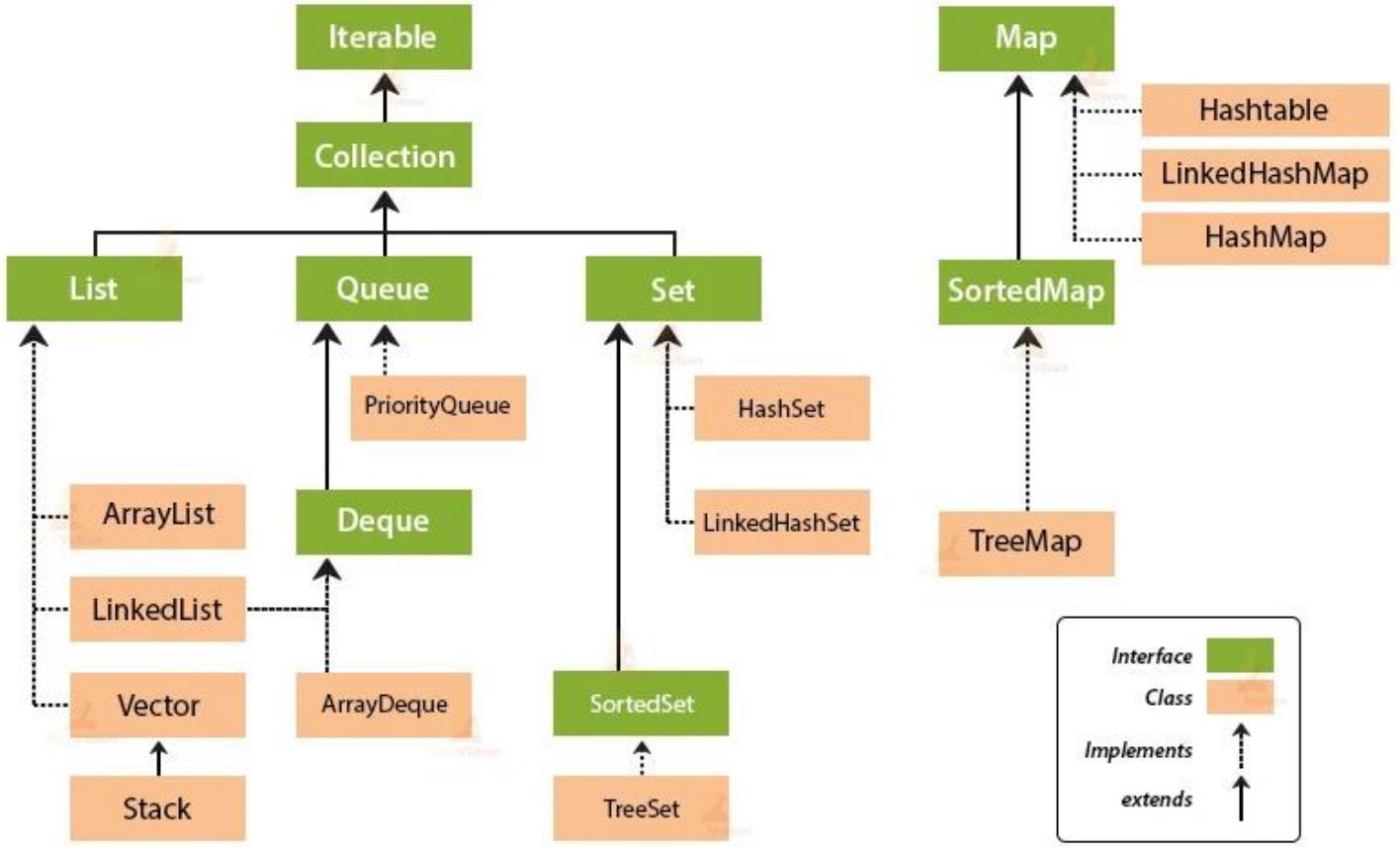


L'API Collections possède deux grandes familles chacune définies par une interface :

- **java.util.Collection** : pour gérer un groupe d'objets.
- **java.util.Map** : pour gérer des éléments de type paires de clé/valeur.

PARTIE 3

La hiérarchie des collections



Les interfaces des collections

	Set collection d'éléments uniques	List collection avec doublons	Map collection sous la forme clé/valeur
Tableau redimensionnable		ArrayList, Vector	
Arbre	TreeSet		TreeMap
Liste chaînée		LinkedList	
Collection utilisant une table de hachage	HashSet		HashMap, HashTable

L'interface Collection

Cette interface définit plusieurs méthodes :

Méthode	Rôle
boolean add(Object)	ajoute l'élément fourni en paramètre à la collection. La valeur de retour indique si la collection a été mise à jour
boolean addAll(Collection)	ajoute à la collection tous les éléments de la collection fournie en paramètre
void clear()	supprime tous les éléments de la collection
boolean contains(Object)	indique si la collection contient au moins un élément identique à celui fourni en paramètre
boolean containsAll(Collection)	indique si tous les éléments de la collection fournie en paramètre sont contenus dans la collection
boolean isEmpty()	indique si la collection est vide
Iterator iterator()	renvoie un objet qui permet de parcourir l'ensemble des éléments de la collection
boolean remove(Object)	supprime l'élément fourni en paramètre de la collection. La valeur de retour indique si la collection a été mise à jour
boolean removeAll(Collection)	supprime tous les éléments de la collection qui sont contenus dans la collection fournie en paramètre
int size()	renvoie le nombre d'éléments contenu dans la collection
Object[] toArray()	renvoie d'un tableau d'objets qui contient tous les éléments de la collection

L'interface Iterator

Cette interface définit des méthodes pour des objets capables de parcourir les données d'une collection.

Méthode	Rôle
boolean hasNext()	indique s'il reste au moins un élément à parcourir dans la collection
Object next()	renvoie le prochain élément dans la collection
void remove()	supprime le dernier élément parcouru

Exemple

```
Iterator iterator = collection.iterator();
while (iterator.hasNext()) {
        System.out.println("objet = "+iterator.next());
}
```

Remarque :

La méthode remove() permet de supprimer l'élément renvoyé par le dernier appel à la méthode next(). Il est ainsi impossible d'appeler la méthode remove() sans un appel correspondant à next() : on ne peut pas appeler deux fois de suite la méthode remove().

```
Iterator iterator = collection.iterator();
if (iterator.hasNext()) {
        iterator.next();
        itérator.remove();
}
```

Si aucun appel à la méthode next() ne correspond à celui de la méthode remove(), une exception de type IllegalStateException est levée.

List

Une liste est une collection ordonnée d'éléments qui autorise d'avoir des doublons. Etant ordonné, un élément d'une liste peut être accédé à partir de son index.

- Cette interface étend l'interface Collection ;
- Les collections qui implémentent cette interface autorisent les doublons dans les éléments de la liste. Ils autorisent aussi l'insertion d'éléments null ;
- L'interface List propose plusieurs méthodes pour un accès à partir d'un index aux éléments de la liste. La gestion de cet index commence à zéro ;
- Pour les listes, une interface particulière est définie pour assurer le parcours dans les deux sens de la liste et assurer des mises à jour : l'interface **ListIterator.**

List

Méthode	Rôle
Iterator iterator()	renvoie un objet capable de parcourir la liste
Object set (int, Object)	remplace l'élément contenu à la position précisée par l'objet fourni en paramètre
void add(int, Object)	ajouter l'élément fourni en paramètre à la position précisée
Object get(int)	renvoie l'élément à la position précisée
int indexOf(Object)	renvoie l'index du premier élément fourni en paramètre dans la liste ou -1 si l'élément n'est pas dans la liste
ListIterator listIterator()	renvoie un objet pour parcourir la liste et la mettre à jour
List subList(int,int)	renvoie un extrait de la liste contenant les éléments entre les deux index fournis (le premier index est inclus et le second est exclu). Les éléments contenus dans la liste de retour sont des références sur la liste originale. Des mises à jour de ces éléments impactent la liste originale.
int lastIndexOf(Object)	renvoie l'index du dernier élément fourni en paramètre dans la liste ou -1 si l'élément n'est pas dans la liste
Object set(int, Object)	remplace l'élément à la position indiquée avec l'objet fourni

List

Le Framework propose des classes qui implémentent l'interface List : **LinkedList, Vector** et **ArrayList**.

ArrayList

Cette classe représente un **tableau** d'objets dont la taille est **dynamique**.

Méthode	Rôle
boolean add(Object)	ajoute un élément à la fin du tableau
boolean addAll(Collection)	ajoute tous les éléments de la collection fournie en paramètre à la fin du tableau
boolean addAll(int, Collection)	ajoute tous les éléments de la collection fournie en paramètre dans la collection à partir de la position précisée
void clear()	supprime tous les éléments du tableau
void ensureCapacity(int)	permet d'augmenter la capacité du tableau pour s'assurer qu'il puisse contenir le nombre d'éléments passé en paramètre
Object get(index)	renvoie l'élément du tableau dont la position est précisée
int indexOf(Object)	renvoie la position de la première occurrence de l'élément fourni en paramètre
boolean isEmpty()	indique si le tableau est vide
int lastIndexOf(Object)	renvoie la position de la dernière occurrence de l'élément fourni en paramètre
Object remove(int)	supprime dans le tableau l'élément fourni en paramètre
void removeRange(int,int)	supprime tous les éléments du tableau de la première position fourni incluse jusqu'à la dernière position fournie exclue
Object set(int, Object)	remplace l'élément à la position indiquée par celui fourni en paramètre
int size()	renvoie le nombre d'élément du tableau
void trimToSize()	ajuste la capacité du tableau sur sa taille actuelle

Vector

```
public static void main(String[] args) {

        Vector v = new Vector();

        v.add("said");

        v.add("rachid");

        v.add(14);

        v.add(14.02f);

        v.add("said");


        for(int i = 0; i < v.size(); i++)

                System.out.println(v.get(i));


}
```

Vector de String

```
public static void main(String[] args) {

    Vector <String>v = new Vector<String>();

    v.add("said");

    v.add("rachid");

    v.add("said");

    for(String s : v)

        System.out.println(s);

    for(int i = 0; i < v.size(); i++)

        System.out.println(v.get(i));

}
```

ArrayList

```
public static void main(String[] args) {

        ArrayList liste = new ArrayList();

        liste.add("sendes");

        liste.add("rachid");

        liste.add(14);

        liste.add(14.02);

        liste.add(new Date());


        Iterator it = liste.iterator();


        while(it.hasNext())

                System.out.println(it.next());

}
```

Liste de String

```
public static void main(String[] args) {
        ArrayList<String> liste = new ArrayList<String>();
        liste.add("Amal");
        liste.add("Younes");
        liste.add("Yassine");
        liste.add("Taha");

        Iterator<String> it = liste.iterator();
        while (it.hasNext())
            System.out.println(it.next());

        for (String s : liste)
            System.out.println(s);

        for (int i = 0; i < liste.size(); i++)
            System.out.println(liste.get(i));
}
```

Exercice

Que sera le résultat de ce programme ?

```
public static void main(String[] args) {
        ArrayList<String> liste = new ArrayList<String>();
        liste.add("Amal");
        liste.add("Younes");
        liste.add("Yassine");
        liste.add("Taha");

        liste.set(1, "Halima");
        liste.add(1, "Safa");

        for(String s : liste)
            System.out.println(s);
}
```

Exercice

Expliquez la différence entre les deux déclarations :

List <String> liste = **new ArrayList<String>();**

ArrayList <String> liste = **new ArrayList<String>();**

Le premier objet liste possède juste les méthodes de l'interface List, par contre le 2ème objet possède les méthodes de List et les méthodes propres à ArrayList, ainsi le 2ème objet possède plus de méthodes. Ici, On parle de **polymorphisme d'interface**.

Tri d'une liste

```
ArrayList<String> list = new ArrayList<String>();
list.add("said");
list.add("wafa");
list.add("ali");
list.add("amal");

list.add(1, "hind");
list.set(3, "karim");

Collections.sort(list);

System.out.println(list);
```

Tri d'une liste des étudiants

```
public class Etudiant implements Comparable<Etudiant> {

    private String nom;
    private String prenom;

    public Etudiant(String nom, String prenom) {
        super();
        this.nom = nom;
        this.prenom = prenom;
    }
    public String toString() {
            return this.nom + " " + this.prenom;
    }
    public int compareTo(Etudiant e) {
        if (!this.nom.equals(e.nom))
            return this.nom.compareTo(e.nom);
        else
            return this.prenom.compareTo(e.prenom);
    }
}
```

```
public static void main(String[] args) {
        List<Etudiant> e = new ArrayList<Etudiant>();
        e.add(new Etudiant("Safi", "kamal"));
        e.add(new Etudiant("Alami", "said"));
        e.add(new Etudiant("Alaoui", "wafa"));
        e.add(new Etudiant("Rami", "ali"));
        e.add(new Etudiant("Safi", "Amal"));

        for(Etudiant ee : e)
            System.out.println(ee);

        Collections.sort(e);

        System.out.println("Après le tri");
        for(Etudiant ee : e)
            System.out.println(ee);
}
```

List : LinkedList

```
public class ListExample {
    public static void main(String args[]) {
        LinkedList<String> queue = new LinkedList<String>();
        queue.addFirst("Bernadine");
        queue.addFirst("Elizabeth");
        queue.addFirst("Gene");
        queue.addFirst("Elizabeth");
        queue.addFirst("Clara");

        System.out.println(queue);

        queue.removeLast();
        queue.removeLast();

        System.out.println(queue);
    }
}
```

```
[Clara, Elizabeth, Gene, Elizabeth, Bernadine]
[Clara, Elizabeth, Gene]
```

Set

C'est une interface identique à celle de Collection. Trois implémentations possibles :

- **TreeSet**: les éléments sont rangés de manière triée ;
- **HashSet**: les éléments sont rangés suivant une méthode de hachage ;
- **LinkedHashSet**: Comme HashSet mais les éléments sont accessibles en ordre d'insertion (L'ordre d'affiche est le même que l'ordre d'insertion).

Fonction de hachage

- Une fonction de hachage permet de transformer une clé en une valeur de hachage (un index), donnant ainsi la position d'un élément dans le tableau.

- Si la clé n'est pas un entier naturel, il faut trouver un moyen de la considérer comme tel. Par exemple, si la clé est de type *chaine de caractères*, on peut calculer la somme des positions dans l'alphabet de chaque lettre pour obtenir un entier naturel.

Fonction de hachage : Exemple

Exemple simple de fonction de hachage sur des chaînes de longueur l à valeurs dans l'intervalle [0, N - 1] :

```
public static final int B=256;
public static final int N=311;
public static int h(String ch){
        int v = 0;
        int i;
        for (i=0; i<ch.length(); i++) {
                v = (v*B + ch.charAt(i)) % N;
        }
        return v;
}
```

Exercice

Que sera le résultat du programme suivant :

```
public static void main(String[] args) {
        Set <String> set = new TreeSet<String>();
        set.add("said");
        set.add("sendes");
        set.add("reda");
        set.add("ilham");
        set.add("said");
        set.add("amal");
        set.add("zineb");
        for(String s : set)
            System.out.println(s);
}
```

Exercice

Que sera le résultat du programme suivant :

```
public static void main(String[] args) {
        Set set = new TreeSet();
        set.add("zineb");
        set.add("salma");
        set.add(14);
        set.add("amine");
        for(Object o : set)
            System.out.println(o);
}
```

Exercice

Que sera le résultat du programme suivant :

```
import java.util.*;
public class SetExample {
   public static void main(String args[]) {
      Set<String> set = new HashSet<String>(); // Une table de Hachage
      set.add("Bernadine");
      set.add("Elizabeth");
      set.add("Gene");
      set.add("Elizabeth");
      set.add("Clara");
      System.out.println(set);
      Set<String> setTrie = new TreeSet<String>(set); // Un Set trié
      System.out.println(setTrie);
   }
}
```

Exercice

Que sera le résultat du programme suivant :

```
public static void main(String[] args) {
        Set <String> set = new LinkedHashSet<String>();
        set.add("said");
        set.add("sendes");
        set.add("reda");
        set.add("ilham");
        set.add("said");
        set.add("amal");
        set.add("zineb");
        for(String s : set)
                System.out.println(s);
}
```

Map

- Un groupe de paires contenant une clé et une valeur associée à cette clé. Cette interface n'hérite ni de Set ni de Collection. La raison est que Collection traite des données simples alors que Map des données composées (clé,valeur).

- SortedMap est un Map trié.

HashTable

Un **HashTable** est une implémentation de **Map** qui associe une clé à une valeur. N'importe quel objet, mis à part null peut y être ajouté.

```
Map monHashtable = new Hashtable() ;

monHashtable.put(new Integer(1),"Janvier");
monHashtable.put(new Integer(2),"Fevrier");
monHashtable.put(new Integer(3),"Mars");
monHashtable.put(new Integer(4),"Avril");
monHashtable.put(new Integer(5),"Mai");
monHashtable.put(new Integer(6),"Juin");
monHashtable.put(new Integer(7),"Juillet");
monHashtable.put(new Integer(8),"Aout");
monHashtable.put(new Integer(9),"Septembre");
monHashtable.put(new Integer(10),"Octobre");
monHashtable.put(new Integer(11),"Novembre");
monHashtable.put(new Integer(12),"Décembre");
System.out.println(monHashtable);
```

```
{12=Décembre, 11=Novembre, 10=Octobre, 9=Septembre, 8=Aout, 7=Juillet, 6=Juin, 5=Mai, 4=Avril, 3=Mars, 2=Fevrier, 1=Janvier}
```

TreeMap

```
TreeMap<Integer, String> map = new TreeMap<Integer, String>();
 map.put(new Integer(2), "Donnée 2");
 map.put(new Integer(1), "Donnée 1");
 map.put(new Integer(3), "Donnée 3");
 System.out.println(map.get(2));
 Set <Integer> set = map.keySet();
 for(Integer c : set){
                System.out.println(map.get(c));
 }
```

Donnée 2

Donnée 1

Donnée 2

Donnée 3

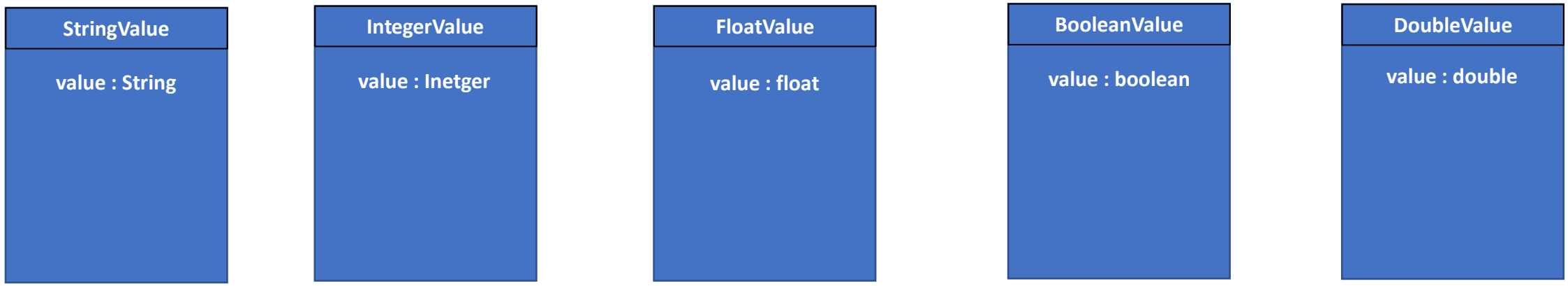
Définition

- La généricité équivalant au template en C++ est probablement la fonctionnalité la plus demandée dans Java depuis son origine. Intégrée à Java 5.0, la généricité est utilisée par les classes de collection pour laisser le choix au programmeur de spécifier une classe différente de « Object » comme classe des éléments stockés. La classe des éléments est spécifiée entre les symboles < et > qui suivent la classe de collection.

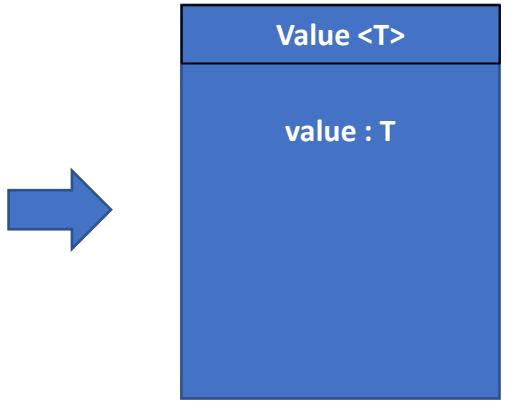
- Par exemple, ArrayList <Integer> représente une collection de classe dans laquelle seuls des objets de classe Integer pourront être ajoutés. La généricité simplifie alors la consultation des éléments d'une collection en évitant de faire appel à l'opérateur de cast.

Généricité - problématique

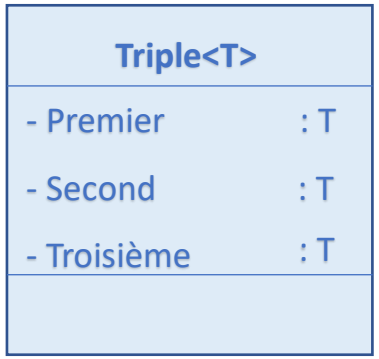
Supposons qu'on souhaite développer un type pour gérer une valeur qui change de type (int, float, String, etc.), ainsi, il faut créer autant de classe que de types visés.



Les types génériques apportent une solution simple et efficace à une telle problématique, en développant une seule classe générique.



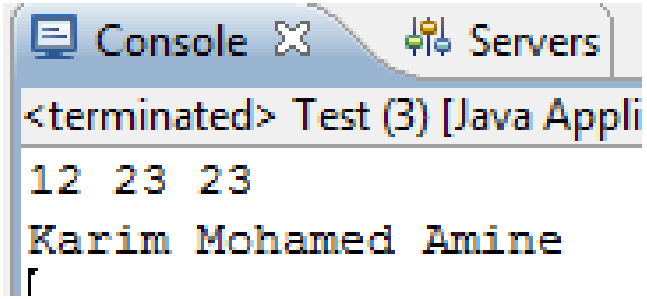
Exemple 1 : Classe générique




```
public class Triple<T> {
        private T premier;
        private T second;
        private T troisieme;

        public Triple(T premier, T second, T troisieme) {
                this.premier = premier;
                this.second = second;
                this.troisieme = troisieme;
        }
        public void affiche(){
                System.out.println(this.premier+" "+
        this.second+" "+this.troisieme );
        }
}
```

Exemple 1 : Classe générique

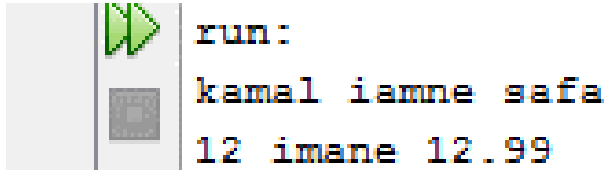
```
public class Test {
        public static void main(String[] args) {
                Triple <Integer> c1 = new Triple <Integer>(12, 23, 23);
                c1.affiche();
                Triple <String> c2 = new Triple <String>("Karim",  "Mohamed", "Amine");
                c2.affiche();
        }
}
```



Exemple 2 : Classe générique

```
public class Triple <T1, T2, T3> {
    private T1 a;
    private T2 b;
    private T3 c;

    public Triple(T1 a, T2 b, T3 c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }
    public void affiche(){
     System.out.println(this.a +" "+this.b+" "+this.c);
    }
}
```

```
Triple <String, String, String> t1 =
        new Triple<String, String, String>(
        "kamal", "iamne","safa");

    t1.affiche();
    Triple <Integer, String, Double> t2 =
        new Triple<Integer, String, Double>
        (12,"imane", 12.99);
    t2.affiche();
```



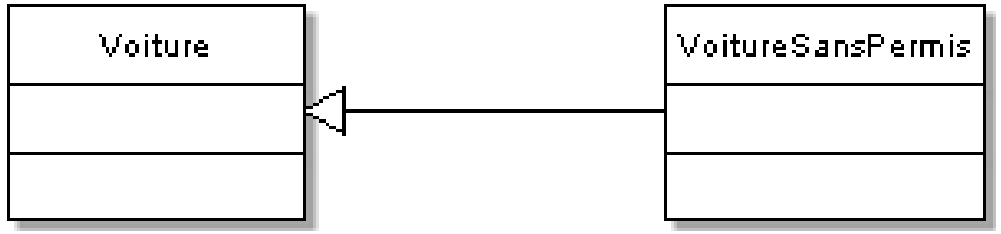
Généricité et collection

- Vous pouvez aussi utiliser la généricité sur les objets servant à gérer des collections ;
- C'est même l'un des points les plus utiles de la généricité ;
- En effet, lorsque vous listiez le contenu d'un **ArrayList** par exemple, vous n'étiez JAMAIS sûrs à 100 % de savoir sur quel type de référence vous alliez tomber.

Généricité et collection

```
import java.util.ArrayList;


public class Test {
	public static void main(String[] args) {
		System.out.println("Liste de String");
		System.out.println("----------------------------");
		ArrayList<String> listeString= new ArrayList<String>();
		listeString.add("Une chaîne");
		listeString.add("Une Autre");
		listeString.add("Encore une autre");
		listeString.add("Allez, une dernière");
		for(String str : listeString)
			System.out.println(str);

		System.out.println("\nListe de float");
		System.out.println("----------------------");
		ArrayList<Float> listeFloat = new ArrayList<Float>();
		listeFloat.add(12.25f);
		listeFloat.add(15.25f);
		listeFloat.add(2.25f);
		listeFloat.add(128764.25f);
		 for(float f : listeFloat)
				System.out.println(f);
	}
}
```

Héritage et généricité



//Un ArrayList n'acceptant que des instances de Voiture ou de ses sous-classes

```
ArrayList<? extends Voiture> listVoitureSP = new ArrayList<VoitureSansPermis>();
```

Héritage et généricité

Exercice : Créer une méthode qui autorise un objet de type **List** de n'importe quel super classe de la classe **Voiture**, **Voiture** y compris.

```
static void affiche(List<? super Voiture> list) {
        for(Object v : list)
                System.out.print(v.toString());

}
```

OFPPT
La Voie de l'Avenir

WEBFORCE
BE THE CHANGE

CHAPITRE n° 2

Réaliser des interfaces graphiques simples

Ce que vous allez apprendre dans ce chapitre :

- Création d'une Activity
- Réalisation d'une interface graphique simple avec du XML
- Communication basique entre les écrans

8 heures

OFPPT
La Voie de l'Avenir

WEBFORCE
BE THE CHANGE

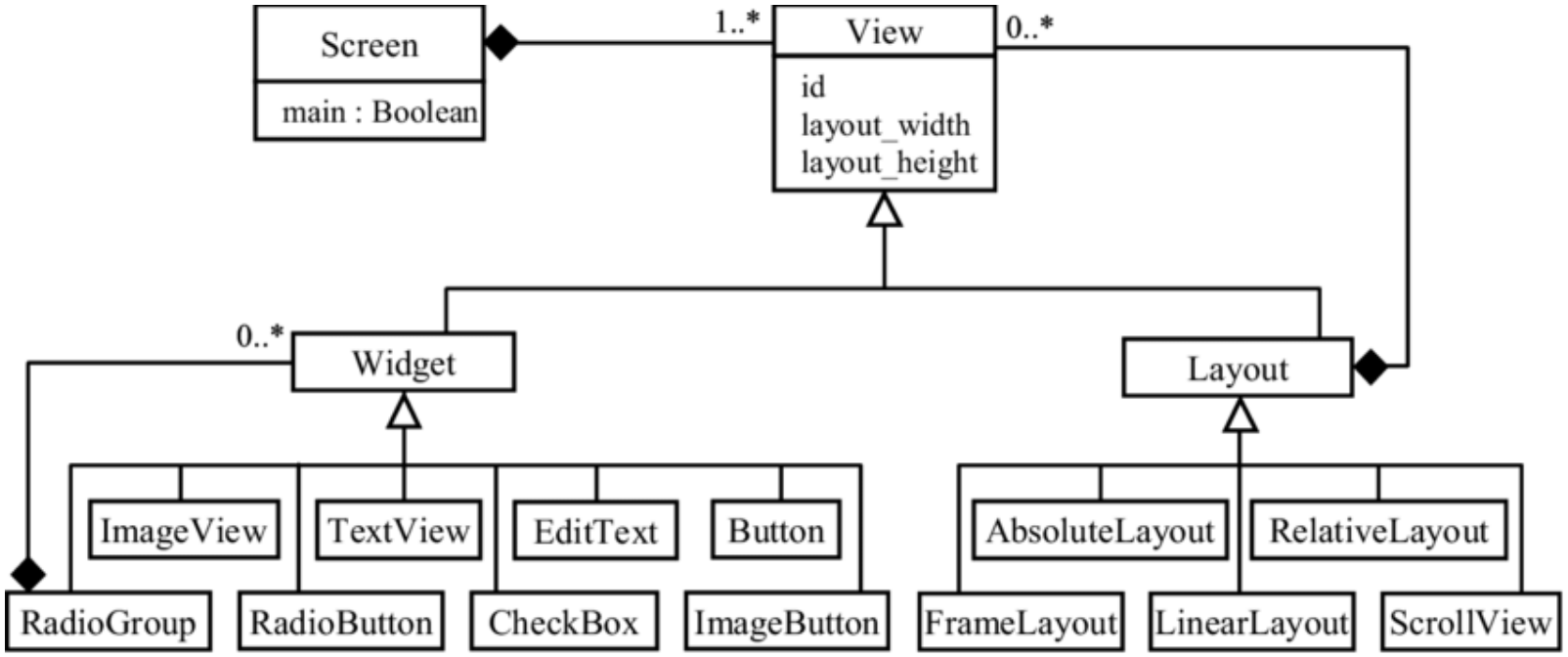
CHAPITRE n° 2

Réaliser des interfaces graphiques simples

1. **Création d'une Activity**
2. Réalisation d'une interface graphique simple avec du XML
3. Communication basique entre les écrans

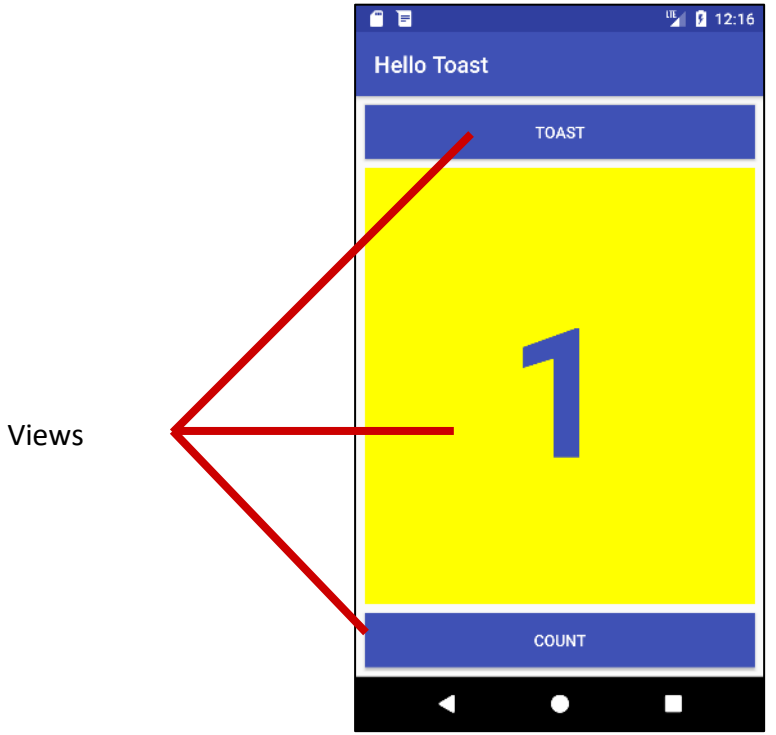
Méta-modèle d'une application Android

- Une application mobile est composée de plusieurs écrans (Screen), chaque écran est composé de plusieurs View.
- Une view peut être un widght ou un Layout.



View

Si vous regardez votre appareil mobile, chaque élément de l'interface utilisateur que vous voyez est une **View**.



PARTIE 3

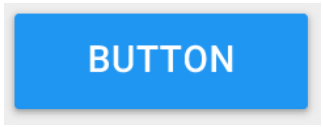
View

Les sous-classes View sont des éléments de base de l'interface utilisateur.

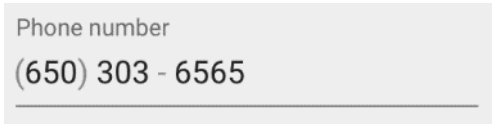
- Affichage de texte (classe TextView), édition de texte (classe EditText) ;
- Boutons (classe Button), menus, autres contrôles ;
- Possibilité de défilement (ScrollView, RecyclerView) ;
- Afficher des images (ImageView) ;
- Regroupement de vues (ConstraintLayout et LinearLayout).

Exemples de sous-classes de View

Button

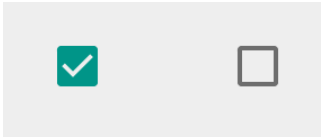


EditText



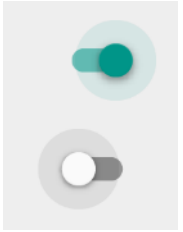
Slider

CheckBox



RadioButton

Switch



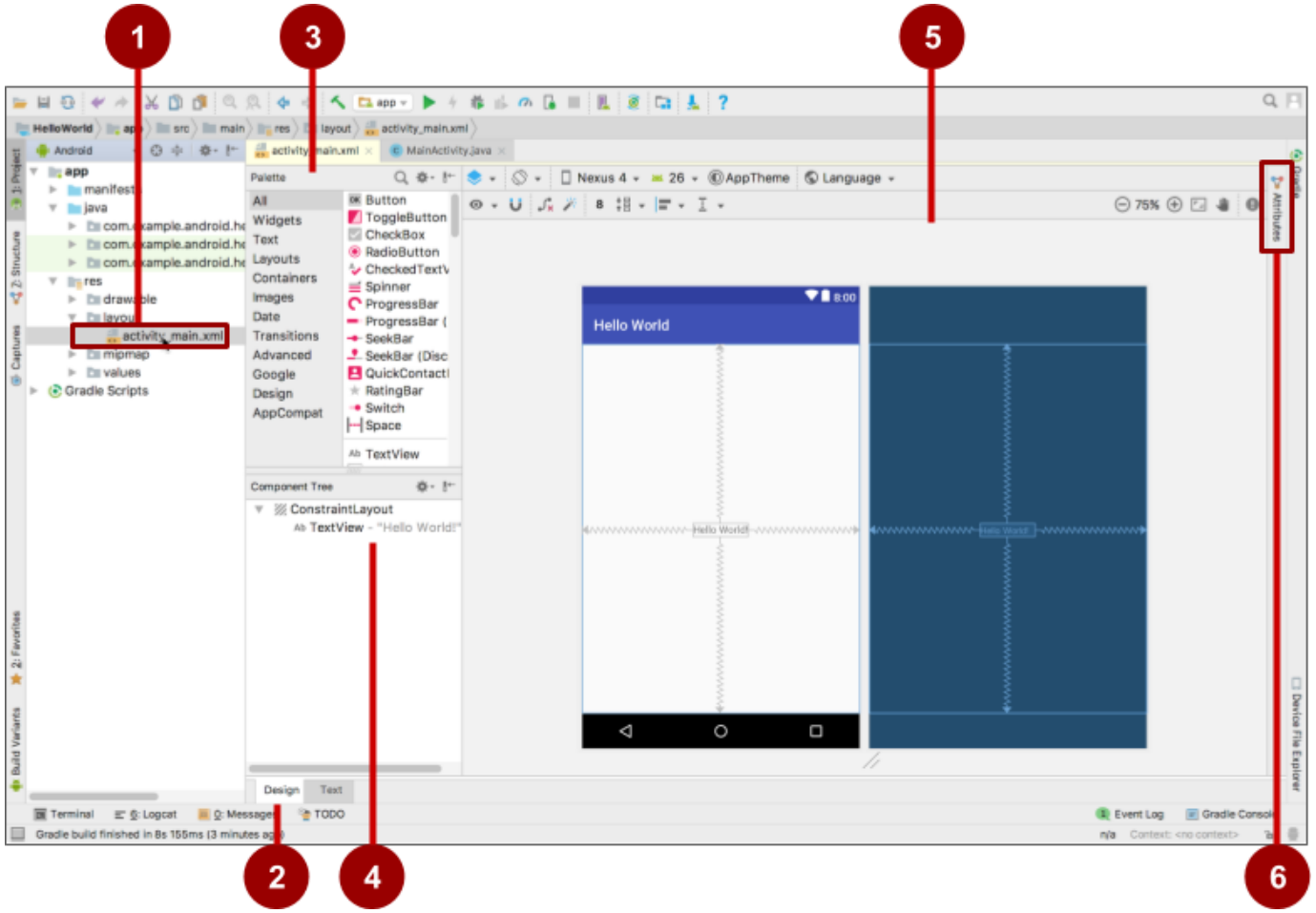
Attributs de View

- Couleur, dimensions, positionnement ;
- Peut avoir le focus (c'est-à-dire être sélectionné pour recevoir les entrées de l'utilisateur) ;
- Peut être interactif (répondre aux clics de l'utilisateur) ;
- Peut être visible ou non ;
- Relations avec d'autres vues.

Créer des vues (Views) et des mises en page (Layouts)

- Éditeur de mise en page d'Android Studio : représentation visuelle du XML ;
- Éditeur XML ;
- Code Java.

Éditeur de mise en page d'Android Studio



1. Fichier de mise en page XML ;
2. Onglets Design et Texte ;
3. Volet Palette ;
4. Arbre des composants ;
5. Volets Design et Blueprint ;
6. Onglet Attributs.

PARTIE 3

View définie en XML

Exemple d'un TextView en XML :

```
<TextView
        android:id="@+id/show_count"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/myBackgroundColor"
        android:text="@string/count_initial_value"
        android:textColor="@color/colorPrimary"
        android:textSize="@dimen/count_text_size"
        android:textStyle="bold"
/>
```

View définie en XML

android:<property_name>="<property_value>"

Exemple: android:layout_width="match_parent"

android:<property_name>="@<resource_type>/resource_id"

Exemple: android:text="@string/button_label_next"

android:<property_name>="@+id/view_id"

Exemple: android:id="@+id/show_count"

View en JAVA

context

Dans l'Activity:

TextView myText = new TextView(**this**);

myText.setText("Afficher le text!");

Context est une interface permettant d'obtenir des informations globales sur **l'environnement** d'une application.

Obtenez le Context :

Context context = getApplicationContext() ;

Une activité est son propre contexte : TextView myText = new TextView(this).

Etats d'une Activity

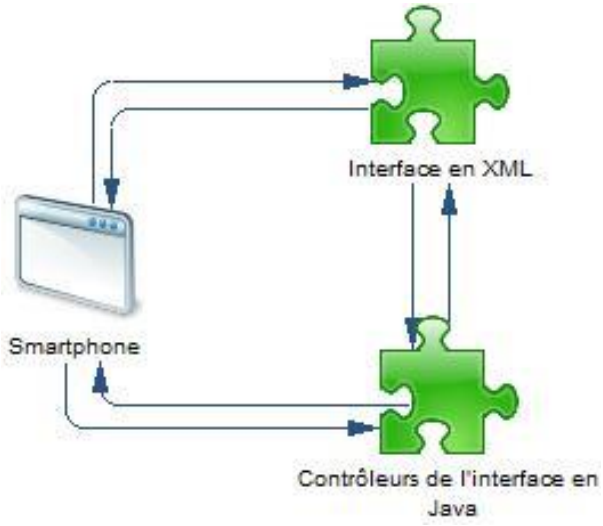
Une activité, vis-à-vis du code, est séparée en 2 morceaux: un fichier XML, et un fichier JAVA.

Le fichier XML :

- Interface graphique de l'activity ;
- Conception de l'IHM : design et ergonomie ;
- Déclaration des éléments graphiques : View, Buttons, Layouts, …etc ;
- Déclaration des identifiants des éléments graphiques.

Le fichier Java

- Le contrôle des éléments déclarés dans le fichier XML ;
- Chaque activity est liée à un seul et unique fichier XML qui définit son interface graphique.



CHAPITRE n° 2

Réaliser des interfaces graphiques simples

1. Création d'une Activity
2. **Réalisation d'une interface graphique simple avec du XML**
3. Communication basique entre les écrans

Les interfaces graphiques sous Android

Des fichiers XML avec des descriptions de mise en page sont utilisés pour définir l'interface utilisateur pour les activités. Les fichiers sont placés dans le dossier : /res/layout

- **View : classe mère de tous les composants graphiques**
- **Implantations élémentaires de View :**
  - **TextView :** affiche une chaîne
  - **EditText** : permet la saisie d'une chaîne (propriété inputType pour le type d'entrée attendu)
  - **Button** : bouton cliquable, variante de type interrupteur avec ToggleButton
  - **CheckBox** : case à cocher
  - **RadioButton** : bouton radio regroupable dans un RadioGroup
  - **CheckedTextView** : chaîne cochable (implante Checkable)
  - **ProgressBar** : barre de progression variante avec étoiles de notation avec ratingBar
  - **SeekBar** : barre de réglage
  - **SearchView** : champ de recherche avec proposition de suggestions
  - **ImageView** : affichage d'une ressource image
  - **ImageButton** : bouton avec image
  - **VideoView** : affichage contrôlable de vidéo

Les interfaces graphiques sous Android

- Déclaration des views
  - Déclaration TextView en XML :

```xml
<TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/textView"
  android:textSize="8sp"
  android:textColor="#112233" />
```

  - Déclaration TextView en JAVA :

```java
TextView textView = new TextView(this);
textView.setText(R.string.textView);
textView.setTextSize(8);
textView.setTextColor(0x112233);
```

  - Rendu de la vue :

Rendu d'un TextView

Remarque

- FILL_PARENT (nommé MATCH_PARENT) dans API niveau 8 et plus).

Les interfaces graphiques sous Android

- Déclaration des views
  - Déclaration du Checkbox en XML :

Remarque

- FILL_PARENT (nommé MATCH_PARENT dans API niveau 8 et plus)

```xml
<CheckBox
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/checkBox"
  android:checked="true" />
```

- Déclaration du Checkbox en JAVA :

```java
CheckBox checkBox = new CheckBox(this);
checkBox.setText(R.string.checkBox);
checkBox.setChecked(true)
if(checkBox.isChecked())
  // Faire quelque chose si le bouton est coché
```

- Rendu de la vue :

PARTIE 3

ViewGroup contient des Views "enfants"

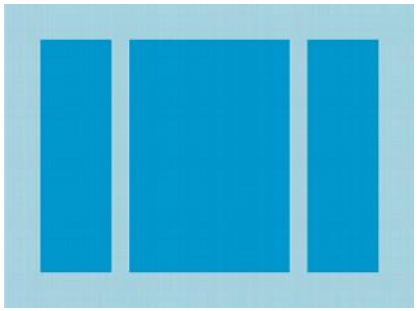
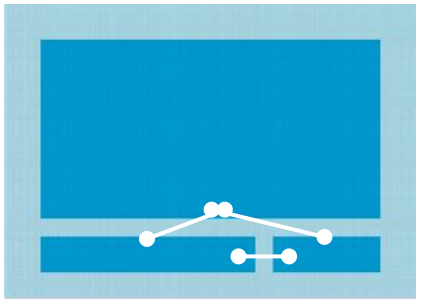
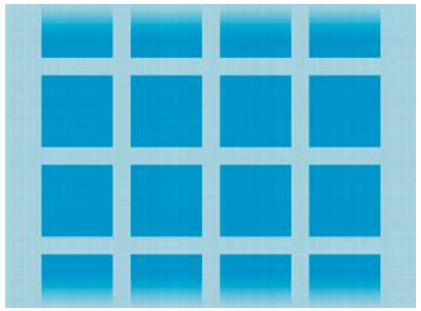
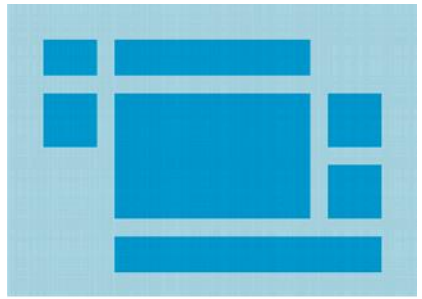
- **ConstraintLayout** : Positionne les éléments de l'interface utilisateur en utilisant des connexions de contrainte avec d'autres éléments et avec les bords de la disposition ;
- **ScrollView** : Contient un élément et permet le défilement ;
- **RecyclerView** : Contient une liste d'éléments et permet le défilement en ajoutant et en supprimant des éléments de manière dynamique.

ViewGroup contient des Views "enfants"

Layouts :

- Sont des types spécifiques de ViewGroups (sous-classes de ViewGroup) ;
- Contiennent des vues enfants ;
- Peuvent se trouver dans une ligne, une colonne, une grille, un tableau, …

Classes Layout communes

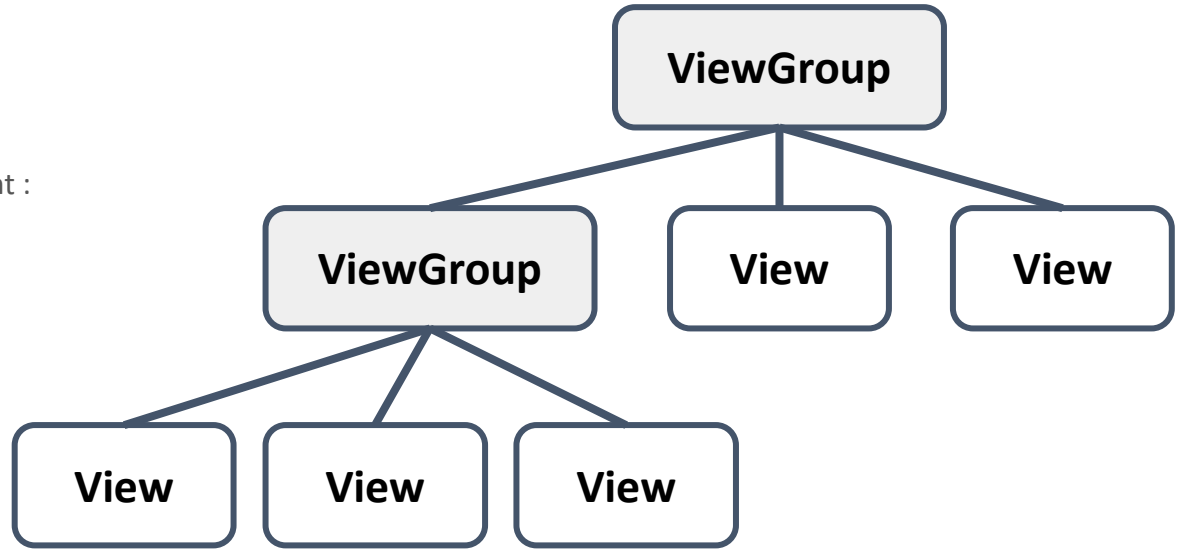
`LinearLayout` `ConstraintLayout` `GridLayout` `TableLayout`

- **ConstraintLayout** : Relier les vues avec des contraintes ;
- **LinearLayout** : Rangée horizontale ou verticale ;
- **RelativeLayout** (disposition relative) : Vues enfants relatives les unes aux autres ;
- **TableLayout** : Lignes et colonnes ;
- **FrameLayout** : Affiche un enfant d'une pile d'enfants.

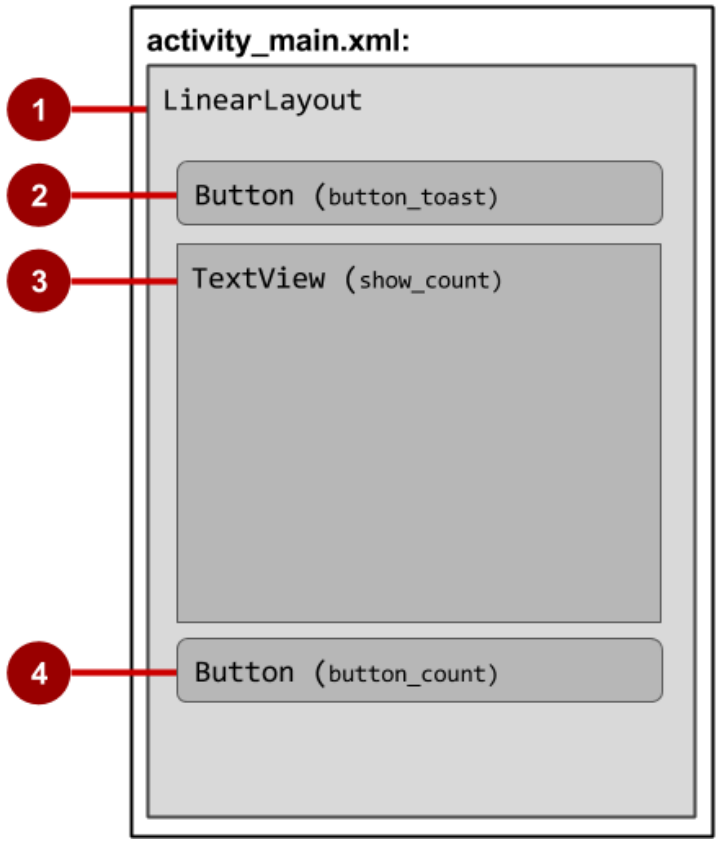
Hiérarchie des classes et hiérarchie de Layout

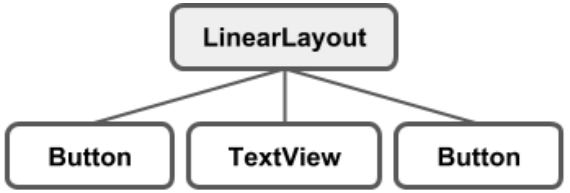
- La hiérarchie des classes des views est l'héritage standard des classes orientées objet :
  - Par exemple, un bouton est un TextView est une View est un objet.
  - Relation superclasse-sous-classe.
- La hiérarchie de Layout est la manière dont les vues sont disposées visuellement :
  - Par exemple, LinearLayout peut contenir des boutons disposés en ligne.
  - Relation parent-enfant, une colonne, une grille, un tableau, …

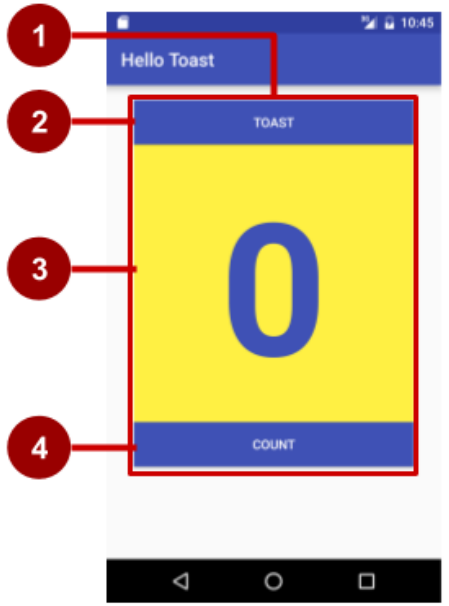
La racine est toujours un ViewGroup :



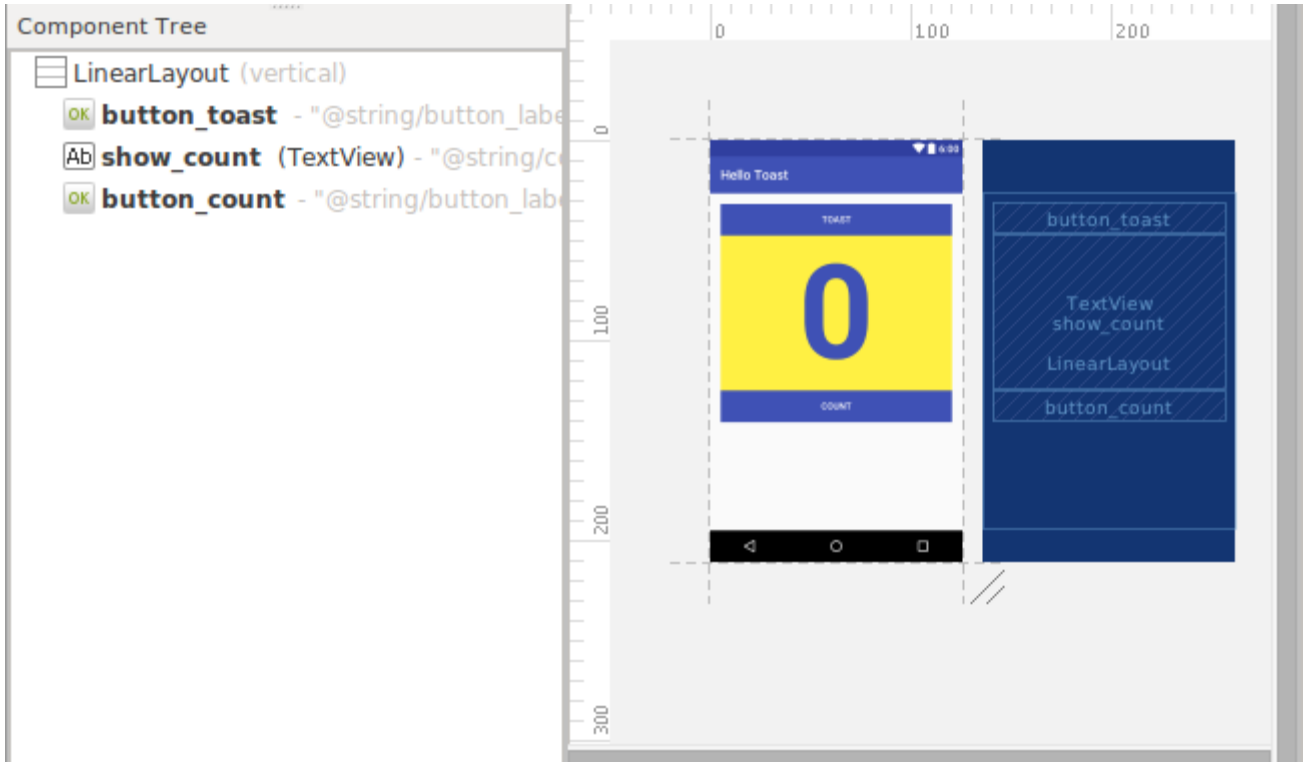
Hiérarchie des Views et disposition des écrans







Hiérarchie des Views et disposition des écrans



Layout créé en XML

```
<LinearLayout
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
    <Button
       ... />
    <TextView
       ... />
    <Button
       ... />
</LinearLayout>
```

Layout créé en JAVA

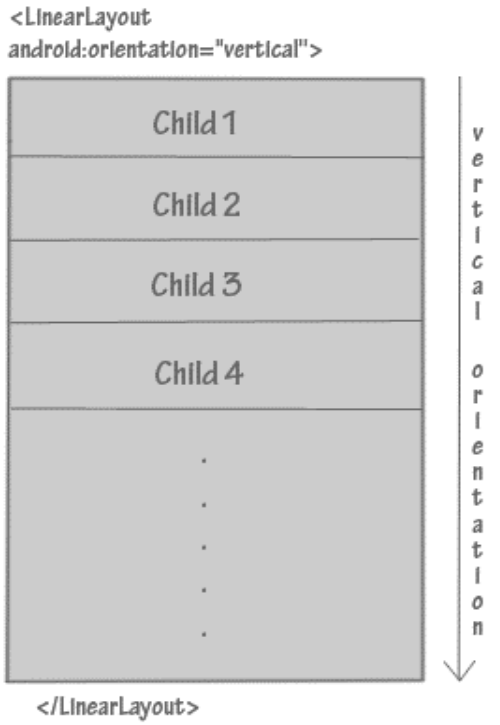
```
LinearLayout linearL = new LinearLayout(this);

linearL.setOrientation(LinearLayout.VERTICAL);


TextView myText = new TextView(this);

myText.setText("Afficher le text!");


linearL.addView(myText);

setContentView(linearL);

LinearLayout.LayoutParams layoutParams =

  new Linear.LayoutParams(

     LayoutParams.MATCH_PARENT,

     LayoutParams.MATCH_CONTENT);


myView.setLayoutParams(layoutParams);
```

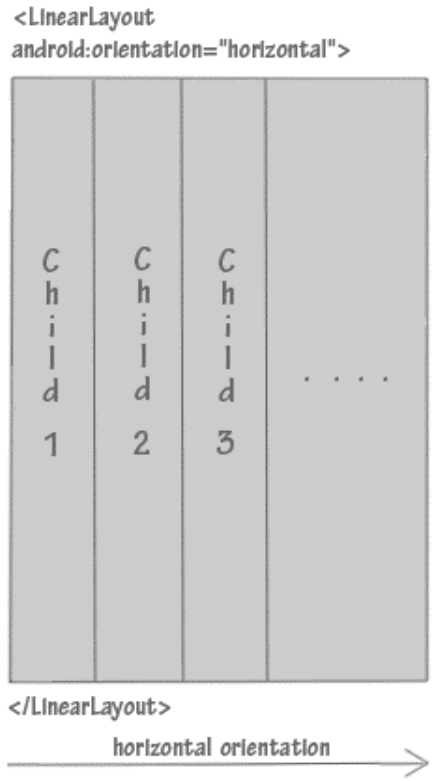
LinearLayout

Ce layout se charge de mettre les vues sur une même ligne, selon une certaine orientation. L'attribut pour préciser cette orientation est « Android: orientation ».

On peut lui donner deux valeurs :

- verticale pour que les composants soient placés de haut en bas (en colonne) ;
- Horizontale pour que les composants soient placés de gauche à droite (en ligne).

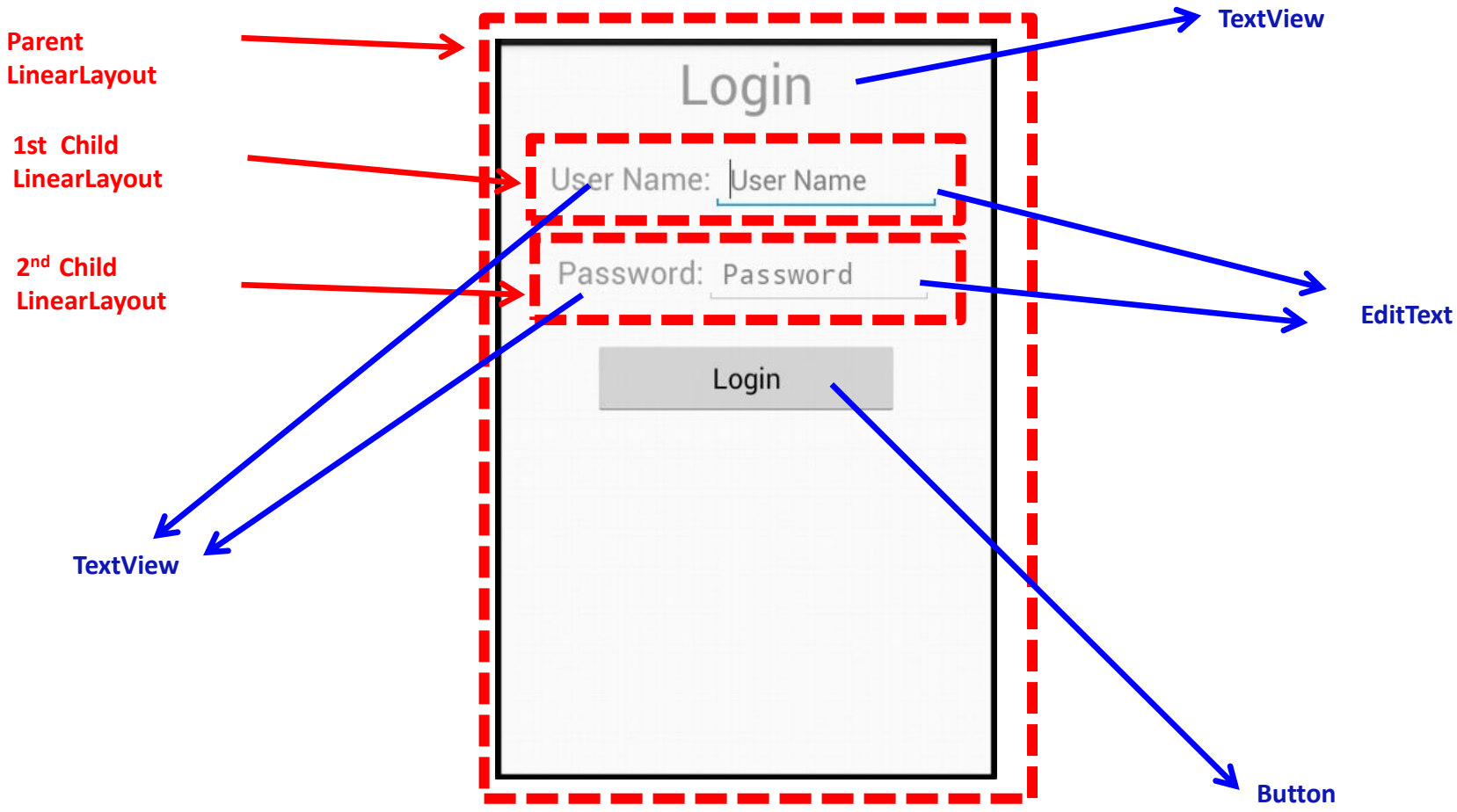




LinearLayout

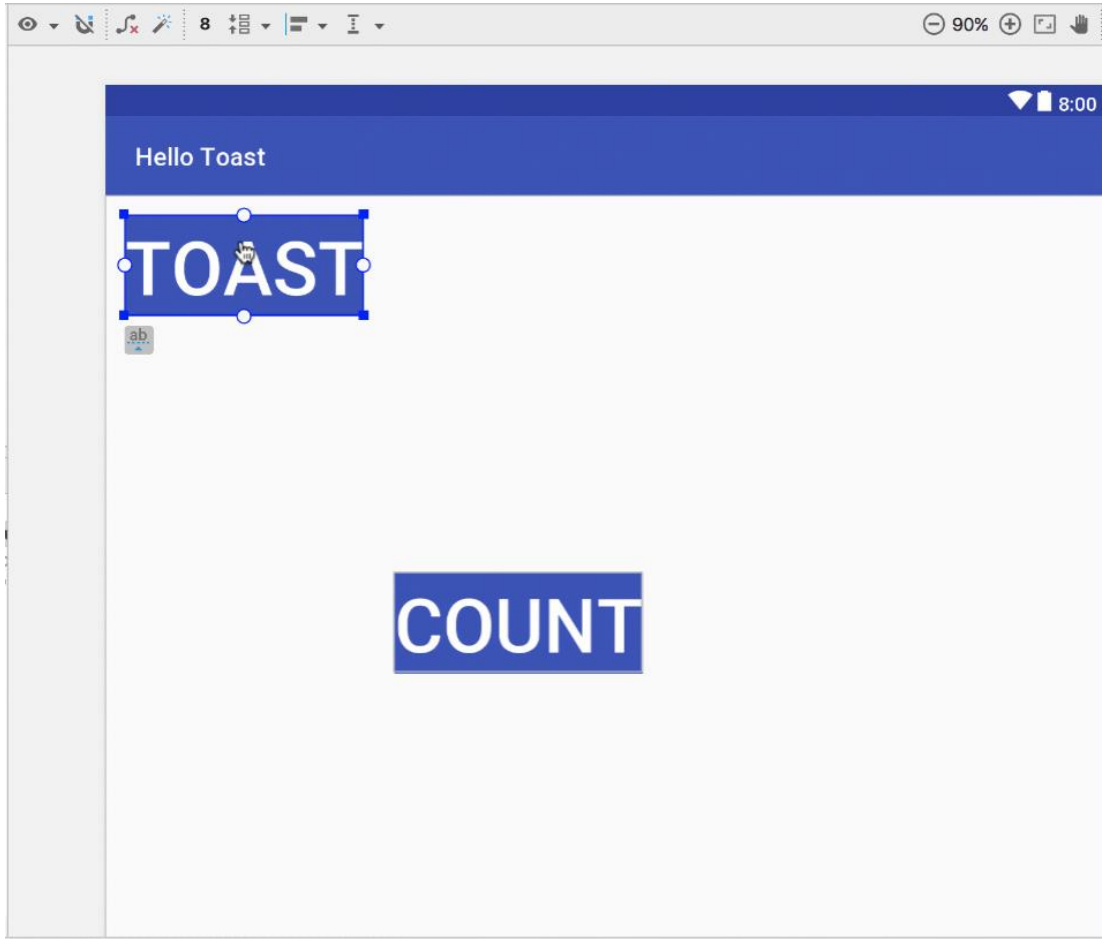
Démo 1 : Réaliser l'interface suivante :

LinearLayout



L'éditeur Layout avec ConstraintLayout

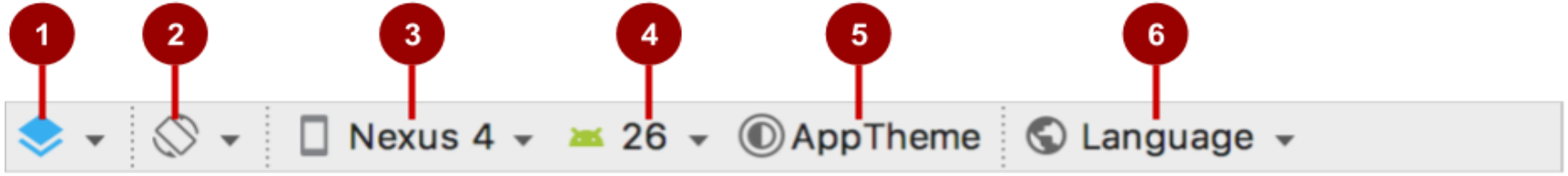
- Connecter les éléments de l'interface utilisateur à la disposition parentale ;
- Redimensionner et positionner des éléments ;
- Aligner des éléments sur d'autres ;
- Ajuster les marges et les dimensions ;
- Modifier les attributs.



ConstraintLayout

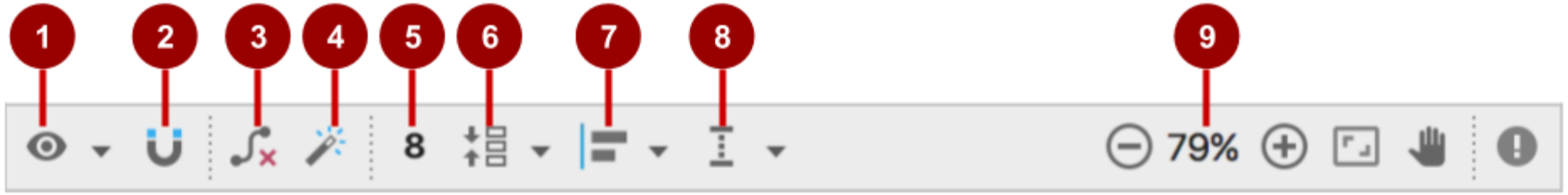
- Layout par défaut pour les nouveaux projets Android Studio ;
- ViewGroup qui offre une grande flexibilité pour la conception de la mise en page ;
- Fournit des contraintes pour déterminer la position et l'alignement des éléments de l'interface utilisateur ;
- La contrainte est une connexion à une autre vue, un layout parente ou une ligne directrice invisible.

Barre d'outils principale de l'éditeur de mise en page



1. Sélectionnez Surface de conception : Volets Design et Blueprint ;
2. Orientation dans l'éditeur : Portrait et Paysage ;
3. Dispositif dans l'éditeur : Choisissez le dispositif pour l'aperçu ;
4. Version de l'API dans l'éditeur : Choisissez l'API pour l'aperçu ;
5. Thème dans l'éditeur : Choisissez le thème pour l'aperçu ;
6. Locale dans l'éditeur : Choisissez la langue/locale pour l'aperçu.

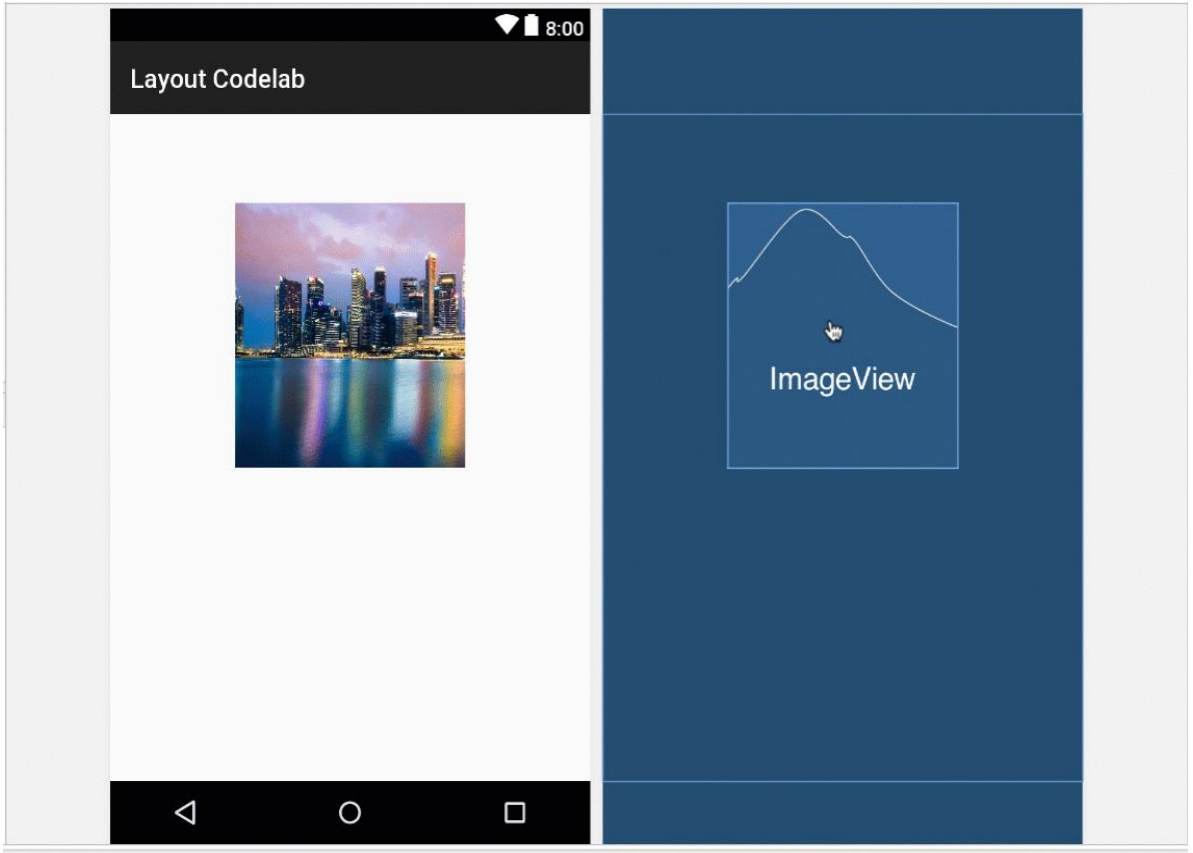
Barre d'outils ConstraintLayout dans l'éditeur de mise en page



1. Afficher : Afficher les contraintes et Afficher les marges ;
2. Autoconnect : Activer ou désactiver ;
3. Effacer toutes les contraintes : Effacer toutes les contraintes de la mise en page (Layout) ;
4. Inférer les contraintes : Créer des contraintes par inférence ;
5. Marges par défaut : Définir les marges par défaut ;
6. Emballer : Emballer ou développer les éléments sélectionnés ;
7. Aligner : Aligner les éléments sélectionnés ;
8. Lignes directrices : Ajouter des lignes directrices verticales ou horizontales ;
9. Contrôles de zoom : Zoom avant ou arrière.

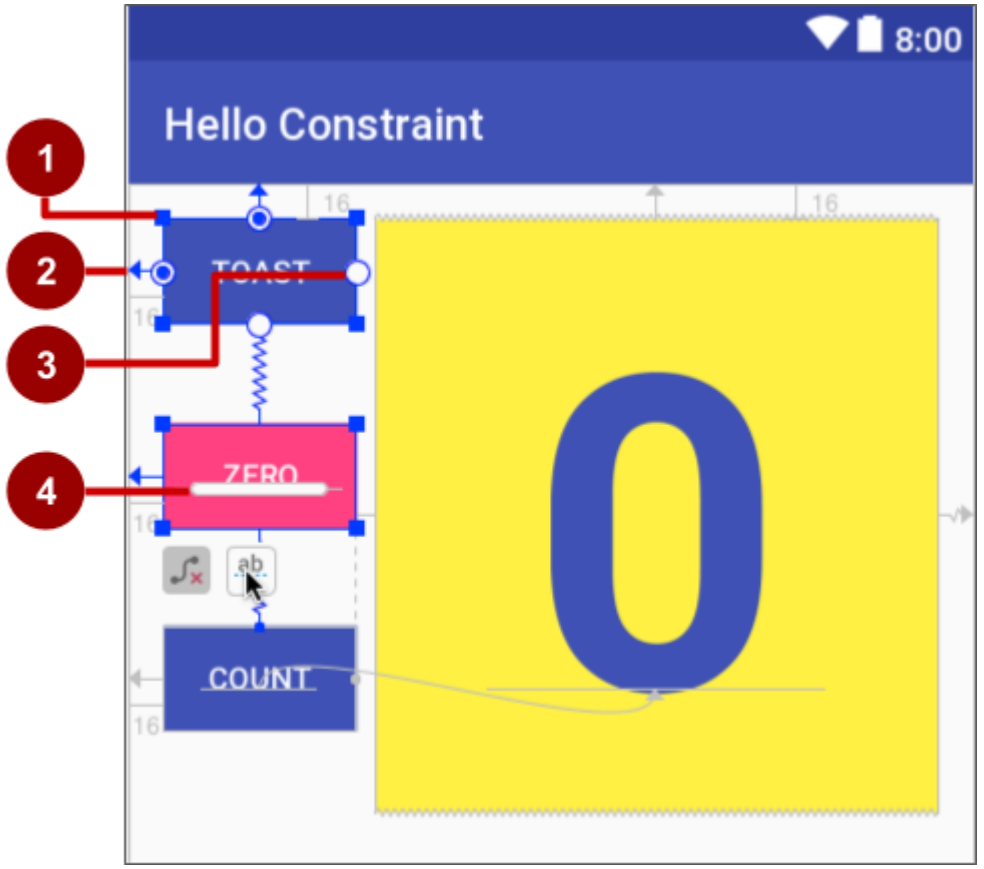
Autoconnect

- Activer l'autoconnect dans la barre d'outils si elle est désactivée ;
- Faire glisser l'élément vers n'importe quelle partie de layout ;
- Autoconnect génère des contraintes par rapport au modèle parent.



Les poignées de ConstraintLayout

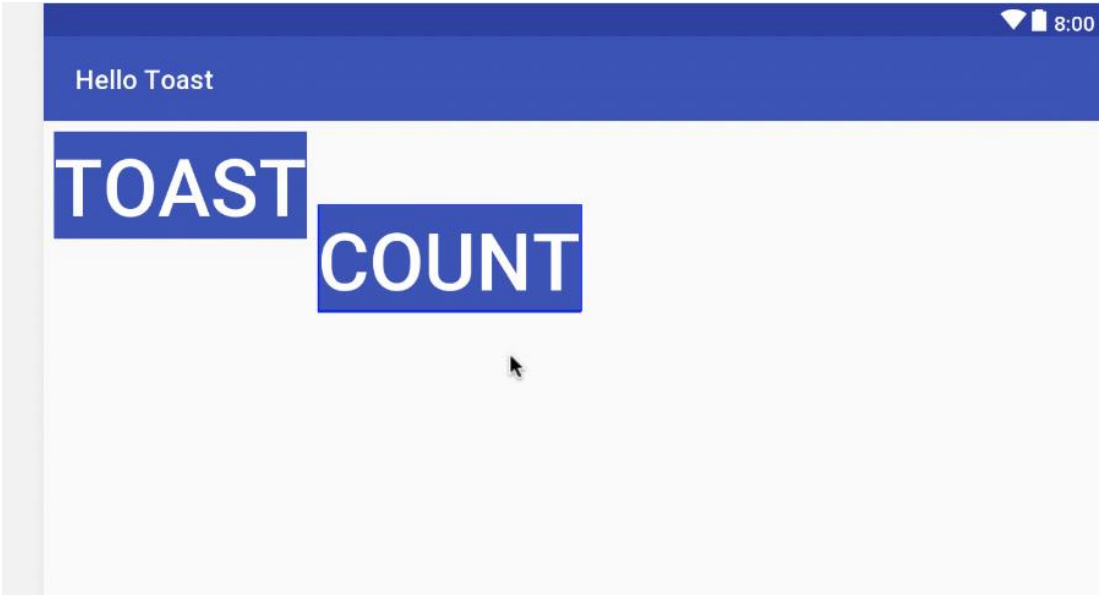
1. Poignée de redimensionnement ;
2. Ligne de contrainte et poignée ;
3. Poignée de contrainte ;
4. Poignée de ligne de base.



Aligner les éléments par ligne de base

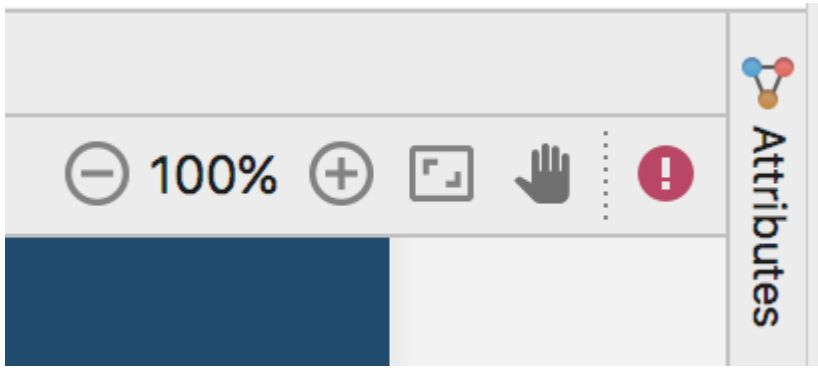
1. Cliquez sur le bouton de contrainte de ligne de base 

2. Faites glisser de la ligne de base vers la ligne de base d'un autre élément.



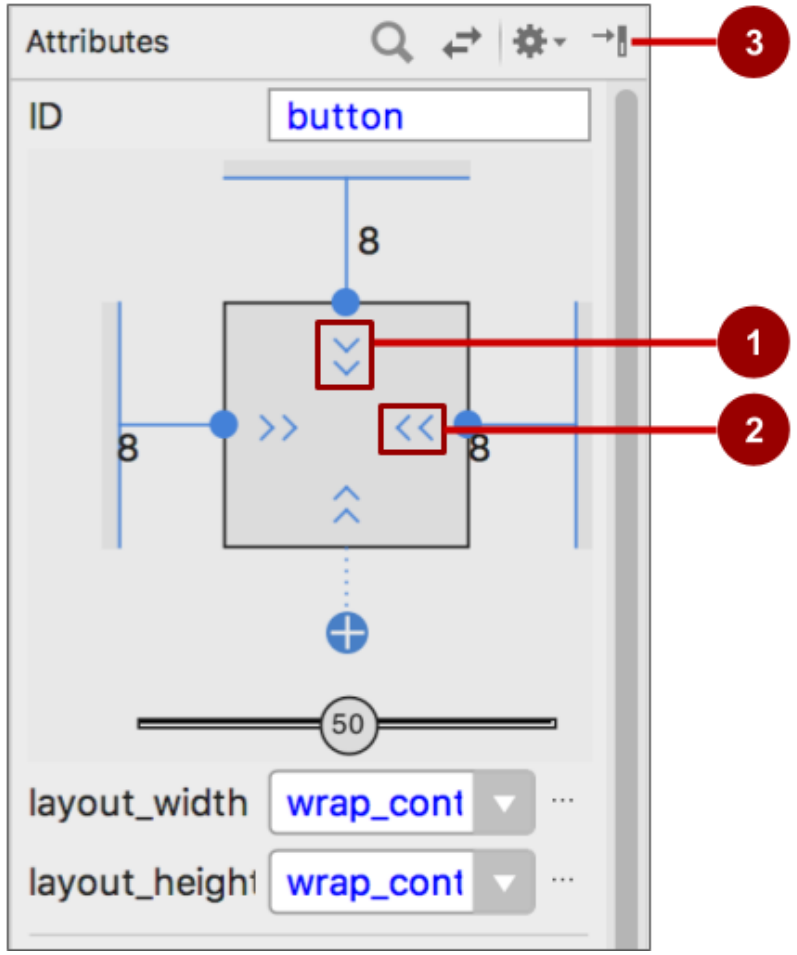
Volet Attributs

- Cliquez sur l'onglet Attributs
- Le volet Attributs comprend :
    - Des contrôles de marge pour le positionnement ;
    - Des attributs tels que layout_width.



Volet attributs de l'inspecteur de views

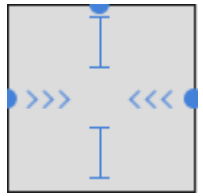
1. Le contrôle de la taille de la vue (view) verticale spécifie le layout_height ;
2. Le contrôle de la taille de la vue (view) horizontale spécifie la largeur de la vue ;
3. Bouton de fermeture du volet des attributs.



Layout_width et layout_height

layout_width et layout_height changent avec les contrôles de taille :

- match_constraint : Développe l'élément pour remplir son parent ;
- wrap_content : Réduit l'élément pour contenir le contenu ;
- Nombre fixe de dp (pixels indépendants de la densité).

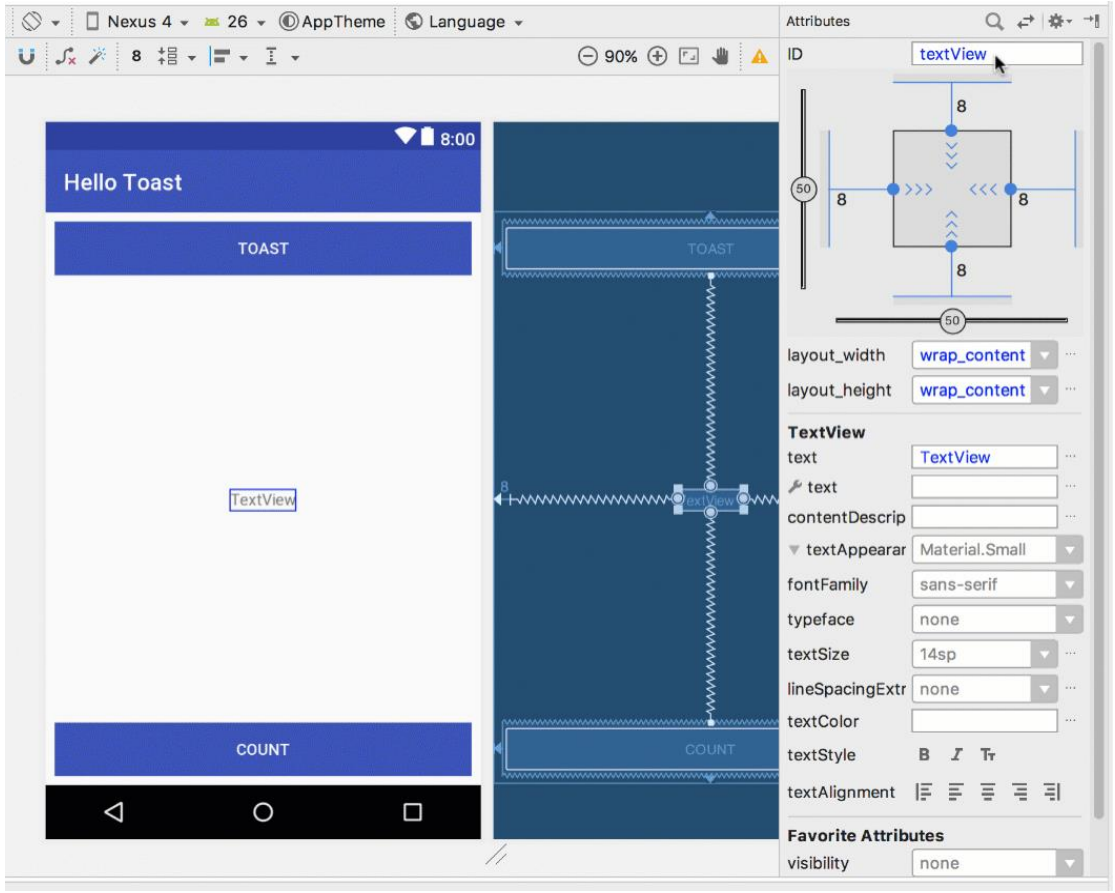


Définir les attributs

Pour afficher et modifier tous les attributs d'un élément :

- Cliquez sur l'onglet **Attributs ;**
- Sélectionnez l'élément dans le dessin, le plan ou l'arbre des composants ;
- Modifiez les attributs les plus utilisés ;
- Cliquez  en haut ou sur Afficher d'autres attributs en bas pour voir et modifier d'autres attributs.

Exemple de définition des attributs : TextView



PARTIE 3

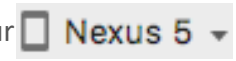
Prévisualisation des mises en page

Prévisualisez la mise en page avec une orientation horizontale/verticale :

1. Cliquez sur le bouton Orientation dans l'éditeur 

2. Choisissez Passer en Paysage ou Passer en Portrait.

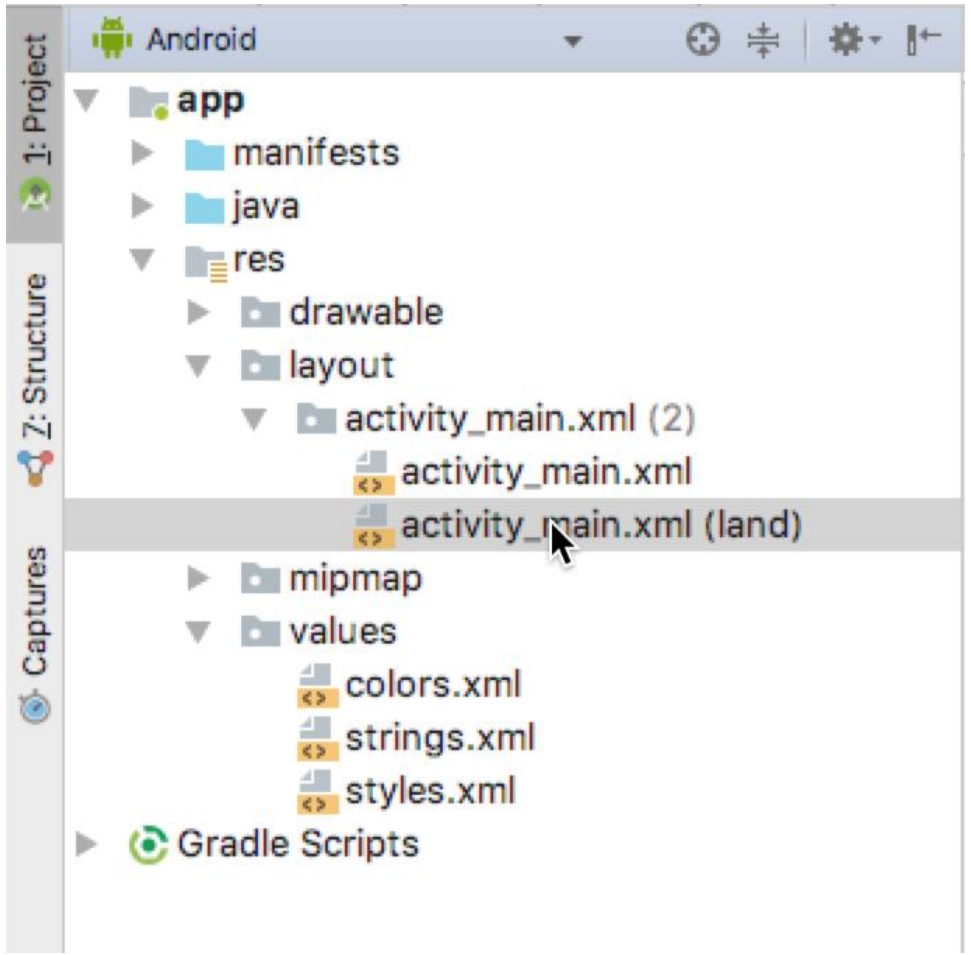
Prévisualisez la mise en page avec différents appareils :

1. Cliquez sur Dispositif dans le bouton Editeur 

2. Choisissez le dispositif.

Créer une variante de mise en page pour le paysage

1. Cliquez sur le bouton Orientation dans l'éditeur 
2. Choisissez Créer une variante de paysage ;
3. Variante de mise en page créée : activity_main.xml (land) ;
4. Modifiez la variante de présentation selon vos besoins.



Créer une variante de mise en page pour les tablettes

1. Cliquez sur Orientation dans l'éditeur de mise en page 

2. Choisissez **Créer une variante de mise en page x-large ;**

3. Variante de mise en page créée : activity_main.xml (xlarge) ;

4. Modifiez la variante de mise en page selon vos besoins.

Événements

Quelque chose qui se passe

- Dans l'interface utilisateur : Cliquez, tapez, glissez ;
- Dispositif : Détecter l'activité comme marcher, conduire, incliner ;
- Les événements sont " détectés " par le système Android.

Gestionnaires d'événements

Méthodes qui agissent en réponse à un clic :

- Une méthode, appelée gestionnaire d'événement, est déclenchée par un événement spécifique et agit en réponse à cet événement.

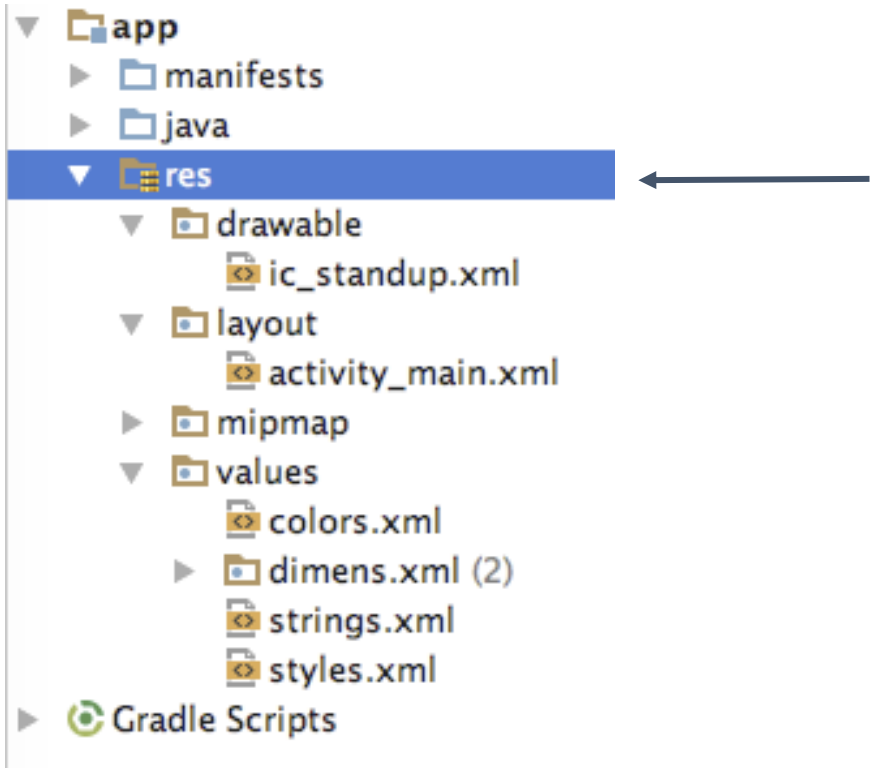
Alternative : Définir le gestionnaire de clics en Java

```
final Button button = (Button) findViewById(R.id.button_id);

button.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {

        String msg = "Hello Toast!";

        Toast toast = Toast.makeText(this, msg, duration);

        toast.show();

        }

     });
```

Ressources

- Séparez les données statiques du code dans vos layouts ;
- Chaînes de caractères, dimensions, images, texte de menu, couleurs, styles...
- Utile pour la localisation.

Ressources dans le projet



ressources et fichiers de ressources stockés dans le dossier res

Se référer aux ressources dans le code

- Layout :

  R.layout.activity_main

  setContentView(R.layout.activity_main);

- View :

  R.id.recyclerview

  rv = (RecyclerView) findViewById(R.id.recyclerview);

- String :

  In Java: R.string.title

  In XML: android:text="@string/title"

Dimensions

- Density-independent Pixels (dp) : pour les views ;
- Pixels indépendants de l'échelle (sp) : pour le texte.

N'utilisez pas d'unités dépendant du périphérique ou de la densité :

- Pixels réels (px)
- Mesure réelle (in, mm)
- Points - typographie 1/72 pouce (p

CHAPITRE n° 2

Réaliser des interfaces graphiques simples

1. Création d'une Activity
2. Réalisation d'une interface graphique simple avec du XML
3. **Communication basique entre les écrans**

Navigation entre les activités

- **Les Intents**

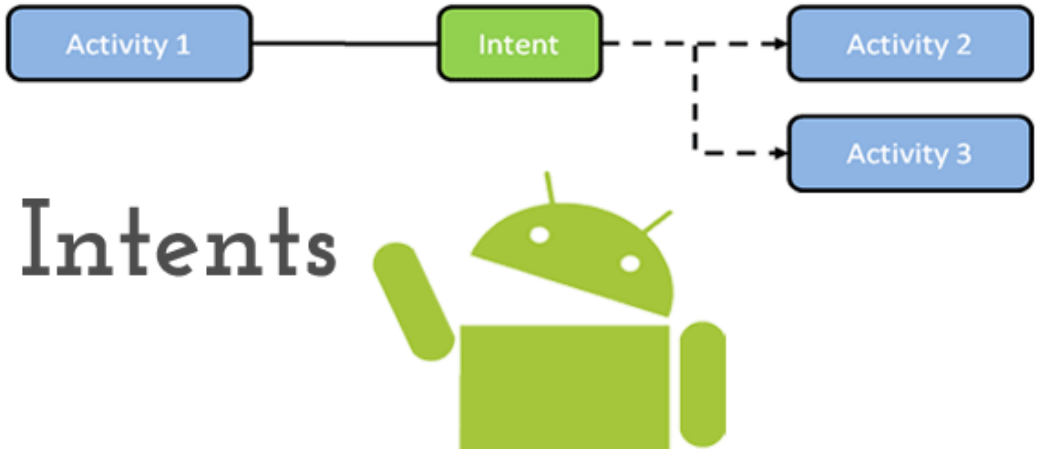
Les objets de type Intent permettant de faire communiquer vos activités entre elles, mais pas seulement : une activité communique des informations également à des services de cette manière.

Vous utiliserez donc un intent pour :

- Une activité de votre application lance une seconde activité de votre programme (ou d'un autre programme Android) ;
- Une activité de votre application lance un service de votre programme (ou d'un autre programme) ;
- Un broadcast receiver reçoit un intent de l'OS lui-même ou d'une autre application (type l'utilisateur vient de changer la langue ou le time zone).

```
Intent intent = new Intent(Mactivity_A.this, Mactivity_B.class);
startActivity(intent);
```

Navigation entre les activités



Sur nos boutons déclenchant une action, il faut créer un intent pointant vers l'activité voulue et lancer l'activité.

on crée l'intent pointant vers l'activité voulue :

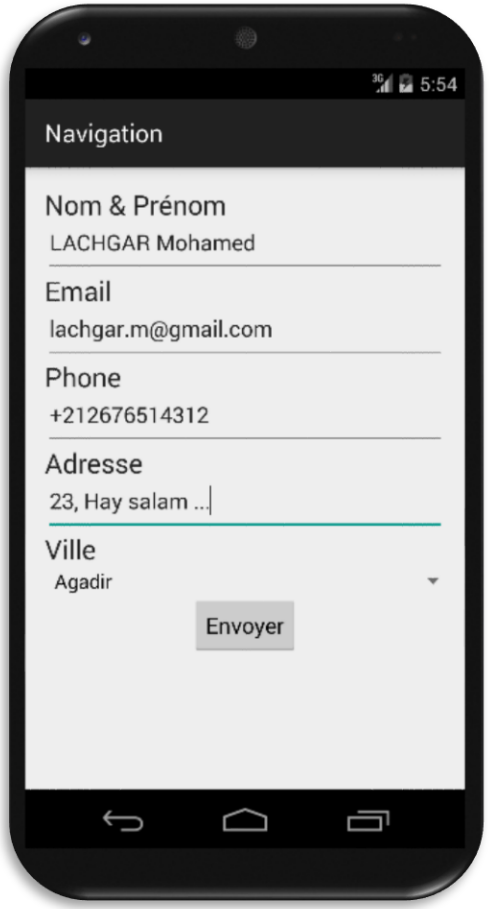
Intent intent = new Intent(this, Activity2.class);

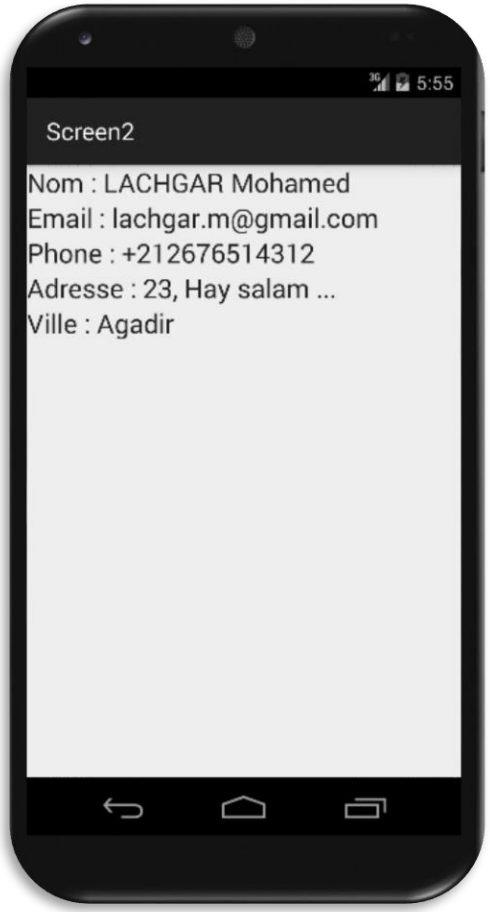
on fournit à l'intent les paramétres afin que l'activité recevant l'intent ait ces informations :

intent.putExtra(EXTRA_LOGIN, strUserName);

on démarre l'activité : **startActivity(intent);**

Navigation entre les activités





Navigation entre les activités

```
@Override
  public void onClick(View v) {
     final Context context = this;
     Intent intent = new Intent(context, Screen2.class);
     HashMap<String, String> data = new HashMap<>();
     data.put("nom", nom.getText().toString());
     data.put("email", email.getText().toString());
     data.put("phone", phone.getText().toString());
     data.put("adresse", adresse.getText().toString());
     data.put("ville", ville.getSelectedItem().toString());
     intent.putExtra("data", data);
     startActivity(intent);
  }
```

Navigation

Screen 2

```
Intent i = getIntent();
Serializable s = getIntent().getSerializableExtra("data");
if (s != null) {
      HashMap data = (HashMap) s;
      nom.setText("Nom : " + data.get("nom"));
      email.setText("Email : " + data.get("email"));
      phone.setText("Phone : " + data.get("phone"));
      adresse.setText("Adresse : " + data.get("adresse"));
      ville.setText("Ville : " + data.get("ville"));
} else {
}
```

CHAPITRE n° 3

Maîtriser Git

Ce que vous allez apprendre dans ce chapitre :

- Manipulation des repository
- Utilisation des commandes Git

8 heures

CHAPITRE n° 3

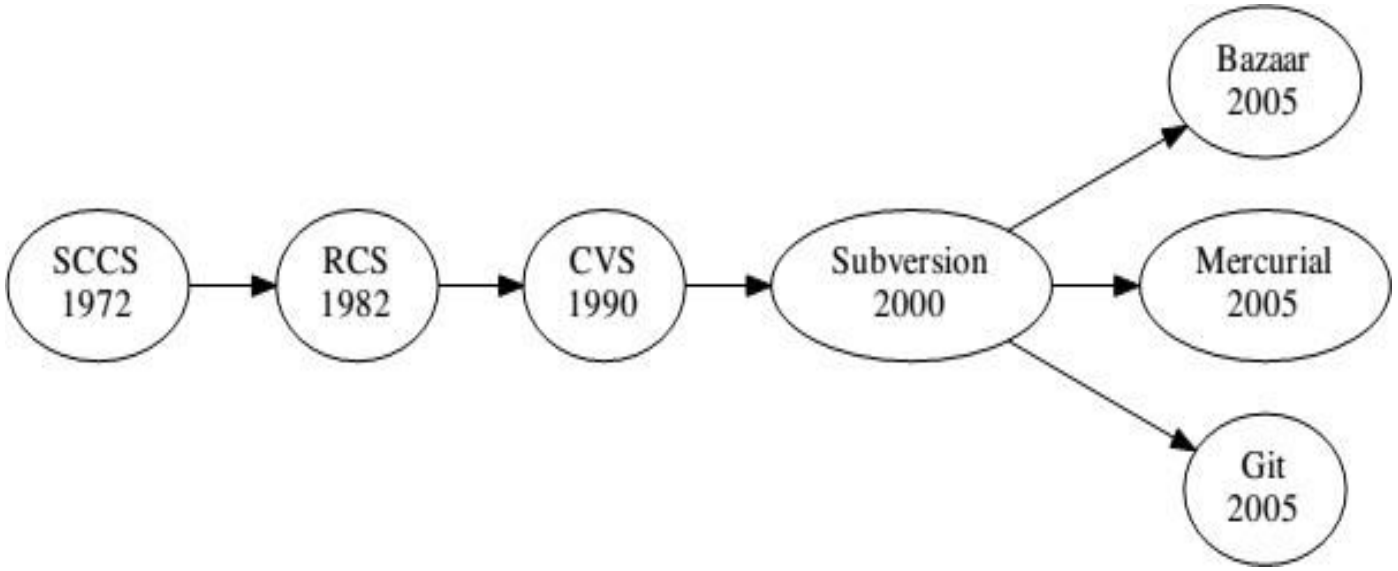
Maîtriser Git

1. **Introduction des différents outils de gestion de versioning**
2. Manipulation des repository
3. Utilisation des commandes Git

Introduction

Git :

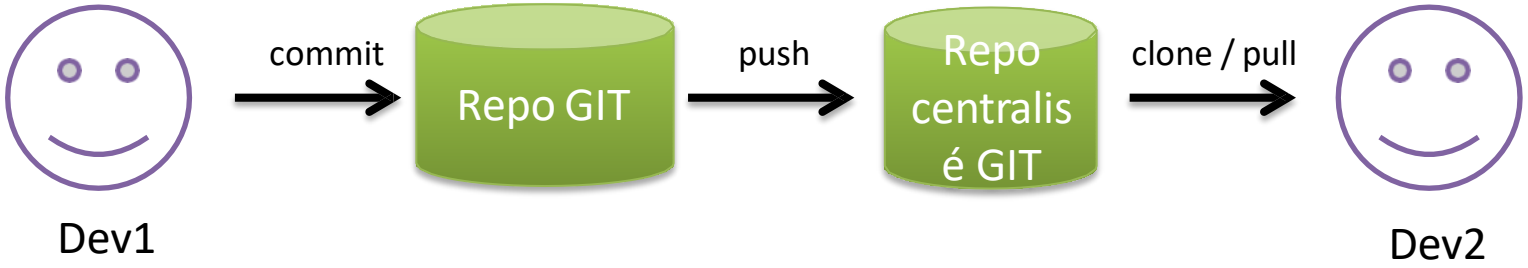
- Historique des modifications ;
- Résolution des conflits par la fusion :
- Possibilité d'avoir les branche.



Introduction

- Deux type de gestion de version:
  - Centralized Revision Control System (CVCS) - Système centralisé de contrôle des révisions ;
  - Distributed Version Control System (DVCS) - Système de contrôle de version distribué.

SVN : système de versioning centralisé (perte d'informations)



Git : système de versioning distribué (Historique à chaque utilisateur)

Centralized Revision Control System (CVCS)

- Architecture client serveur ;
- Un seul repository ;
- Un serveur central contient toutes les données ;
- Beaucoup de requêtes entre le client et le serveur (assez lent) ;
- Dépendance du serveur (check out) ;
- **Exemples** : CVS, SVN…

Distributed Version Control System (DVCS)

- Toutes les données sont sur notre machine ;
- Les opérations sont très rapides ;
- Connexion internet seulement pour les pull et push ;
- **Exemples** : Git, mercurial…

Contrôle de version

Un système de contrôle de versions vise à :

- Garder un historique des différentes versions des fichiers d'un projet ;
- Permettre le retour à une version antérieure quelconque ;
- Garder un historique des modifications avec leur nature, leur date, leur auteur ;
- Permettre un accès souple à ces fichiers, en local ou via un réseau ;
- Permettre à des utilisateurs distincts et souvent distants de travailler ensemble sur les mêmes fichiers.

CHAPITRE n° 3

Maîtriser Git

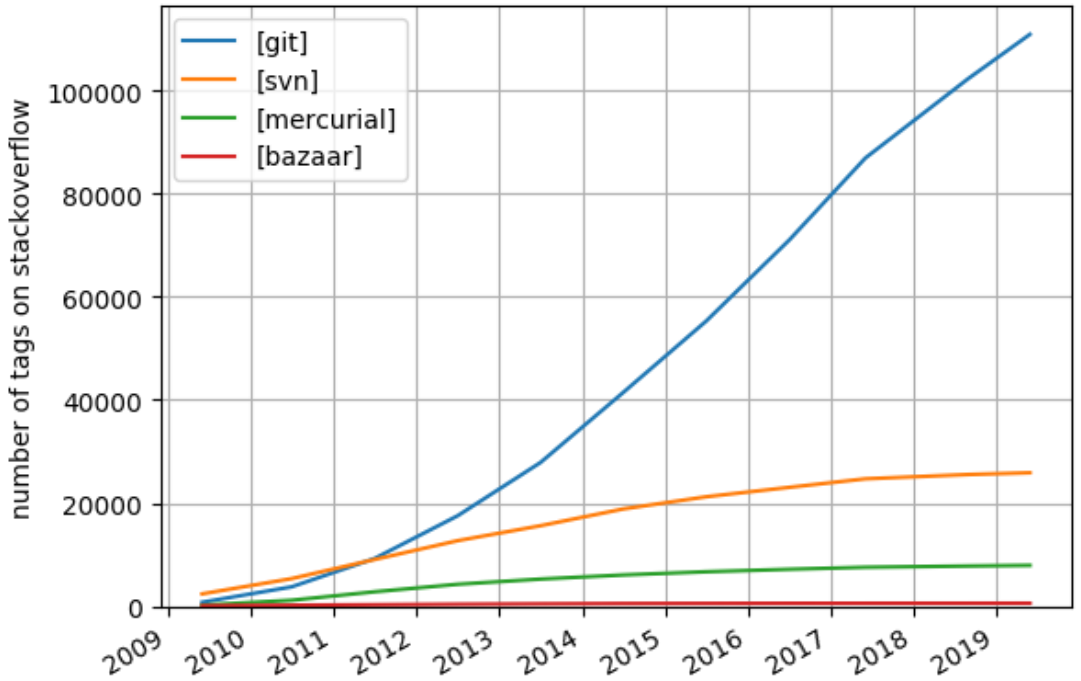
1. Introduction des différents outils de gestion de versioning
2. **Manipulation des repository**
3. Utilisation des commandes Git

Historique GIT

- Créé en avril 2005 par Linus Torvalds ;
- Objectif : gérer le workflow d'intégration des patches du noyau Linux ;
- Remplacer BitKeeper ;
- En Mai 2013, 36 % de professionnels utilise Git.

Historique GIT

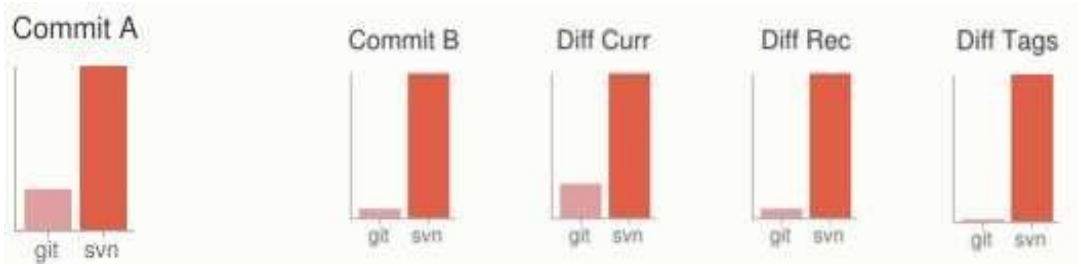
- Créé en avril 2005 par Linus Torvalds ;
- Objectif : gérer le workflow d'intégration des patches du noyau Linux ;
- Remplacer BitKeeper ;
- En Mai 2013, 36 % de professionnels utilise Git.



Pourquoi GIT ?

Pourquoi utiliser Git face à d'autres solutions ?

- Git a de nombreux avantages face à de plus anciens systèmes comme CVS et Subversion (SVN).
- Git est extrêmement simple d'utilisation, notamment couplé à une interface graphique comme GitHub.
  - Git est rapide :



  - Travailler hors-ligne : Git ne requiert pas de serveur distant pour fonctionner.
  - Et bien entendu, Git est gratuit et open source.

Les entreprises et les projets qui utilisent Git

Terminologie

- Le dépôt (repository) abrite tous les projets gérés sur un serveur. Il archive toutes les versions des fichiers stockés ;
- Au sein d'un dépôt se trouvent un ou plusieurs projets. Les projets contiennent des fichiers organisés en arborescence ;
- Les copies locales ou copies de travail correspondent aux espaces de travail (workspaces) des développeurs ;
- Chaque modification faite au dépôt constitue une révision ou version. Le numéro de révision commence à 1 et augmente de 1 à chaque opération.

CHAPITRE n° 3

Maîtriser Git

1. Introduction des différents outils de gestion de versioning
2. Manipulation des repository
3. **Utilisation des commandes Git**

Git configuration

- **Config :**
  - `git config --global user.name "Mohamed LACHGAR"`
  - `git config --global user.email "lachgar.m@gmail.com`

- **Initialisation :**
  - `mkdir formation`
  - `cd formation`
  - `git init`

- **Cloner le dépôt :**
  - `git clone "url du dépôt"`
  - `#On peut spécier un nom du dépôt`
  - `git clone url myrepository`

Git configuration (gitignore)

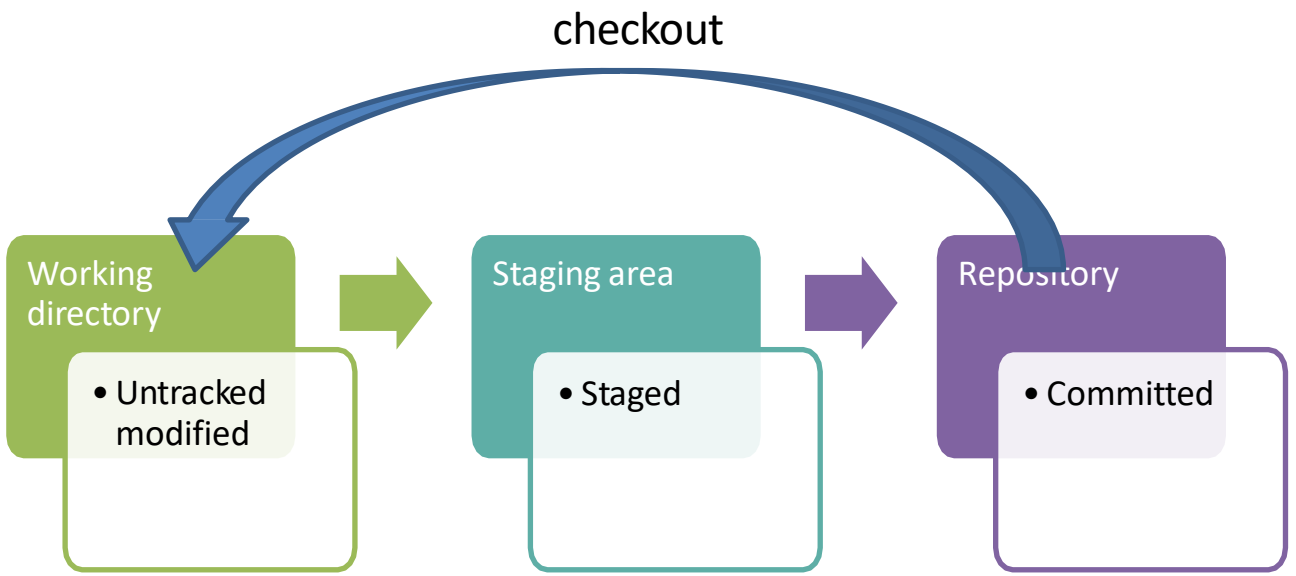
- gitignore :
  - données sensibles ou inutiles

- Créer/modifier le .gitignore à la racine du projet :
  - Exemples de fichiers à ignorer:
    - # /hybris/
    - /hybris/data
    - /hybris/log
    - /hybris/roles
    - /hybris/temp

- lib.so : le fichier lib.so sera ignoré ;
- *.class : tous les fichiers en .class seront ignorés ;
- conf/ : tous les fichiers du dossier conf seront ignorés ;
- conf/**/*. yml : tous les fichiers . yml de tous les sous-dossiers de conf/ seront ignorés.

Git workflow

- Status du fichier



Git workflow

- Staging area

```
#affiche le statut de la working directory et de la staging area
git status
# ajoute un fichier à la staging area
git add
# : unstage un nouveau fichier
git rm --cached
git checkout --: retire un fichier de la staging area
```

Git workflow

- **Commit**
  - Un commit est pointeur sur branch ;
  - Chaque commit est unique :

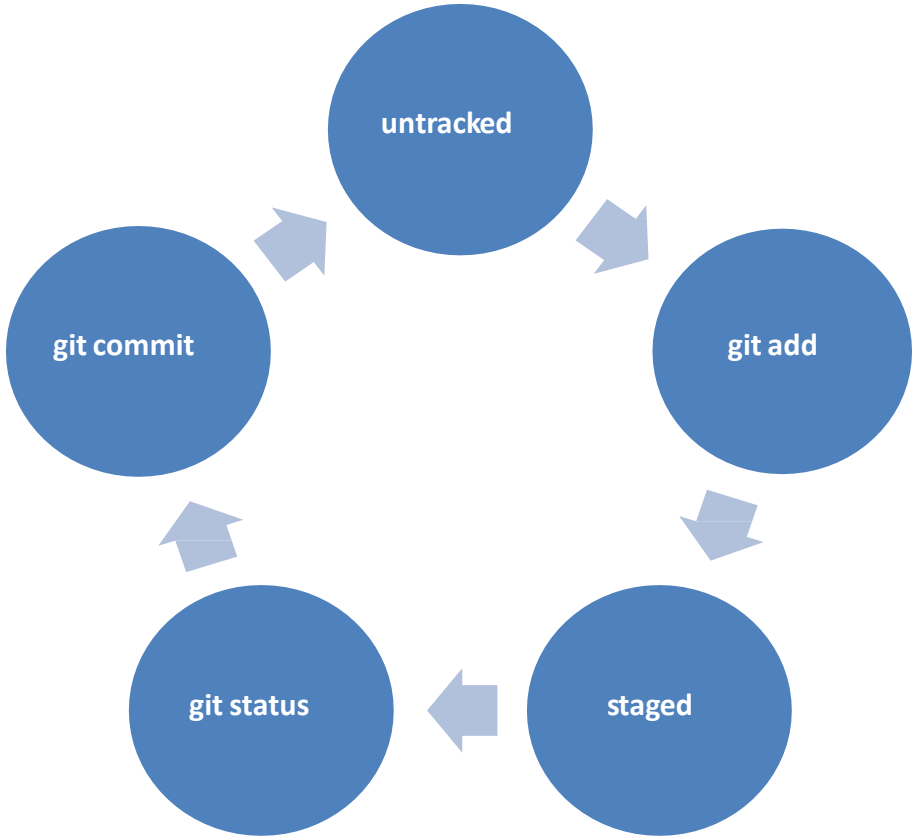
```
git commit -m "mon commentaire de commit"
# génère un commit avec les modifications contenues dans
#la staging area

git commit -a -m "mon commentaire de commit"
# ajoute tous les fichiers modifiés (pas les ajouts /
#suppressions) à la staging area et commite)
```

- **Revert**

```
# Revert a pushed commit
git revert <commit_hash>
```

Git workflow

- Status fichier résumé



Routines Git

- **Git status**

La **commande git status** affiche la liste des fichiers modifiés ainsi que les fichiers qui doivent encore être ajoutés ou validés.

Usage :

git status

- **Git add**

La **commande git add** peut être utilisée pour ajouter des fichiers à l'index. Par exemple, la commande suivante ajoutera un fichier nommé temp.txt dans le répertoire local de l'index :

git add temp.txt

Routines Git

- **Historique: git log**

L' **exécution de** la **commande git log** génère le log d'une branche.

Un exemple de sortie :

commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw Author: Mohamed LACHGAR <lachgar.m@gmail.com>

Date: Mon Oct 1 12:56:29 2019 -060

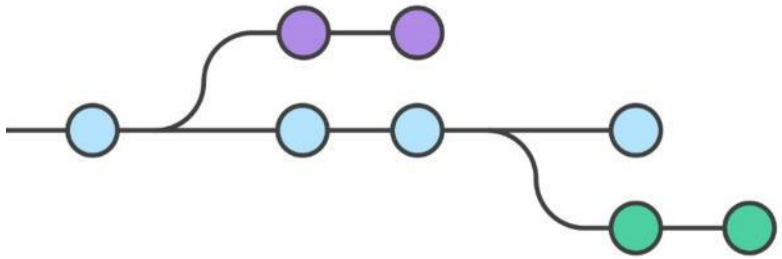
- **Git push**

Git push est une autre commandes GIT de base. Un simple push envoie les modifications locales apportées à la branche principale associée :

git push origin master #branche master

git push origin nouvelleBranche # nouvelle branche

Gestion des branches



```
git push origin master #branche master
git push origin nouvelleBranche # nouvelle branche
```

- **Mise à jour du branch**

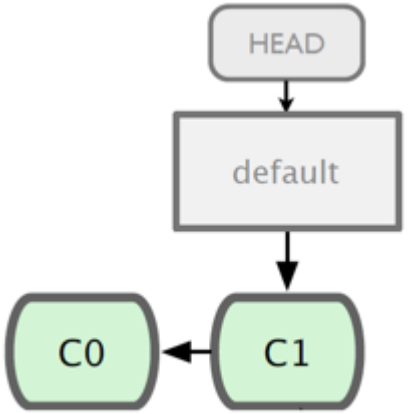
```
# J'ai developpé et commité sur ma branche locale
# Mais l'origine de ma branche (le master
# par exemple) a  également évolué !
```

```
git checkout master
git pull
git checkout nouvelleBranche
git rebase master nouvelleBranche
```

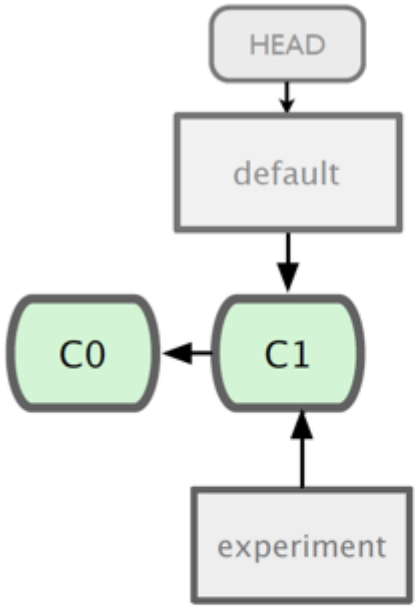
Gestion des branches

```
git checkout master
# Je dev sur le master (alors que j'aurai du créer une branche)
…
git commit -am "Je commit sur le master"
# Je m'en aperçoit
git branch newFeature
# Maintenant je dois remettre le master « clean »
# Si je veux quand même conserver les fichiers
git reset --soft HEAD~1

# Si je ne veux pas conserver les fichiers
git reset --hard HEAD~1
```

Gestion des branches

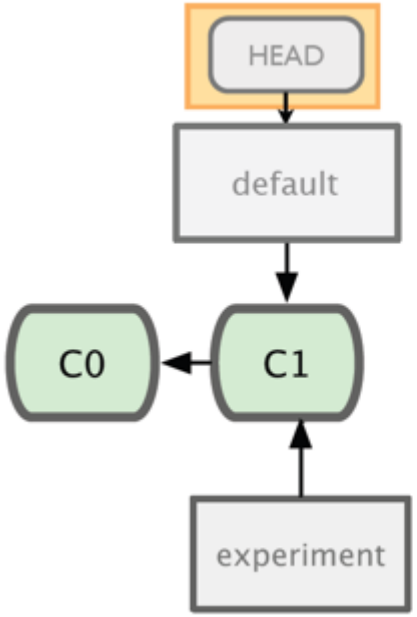


Gestion des branches



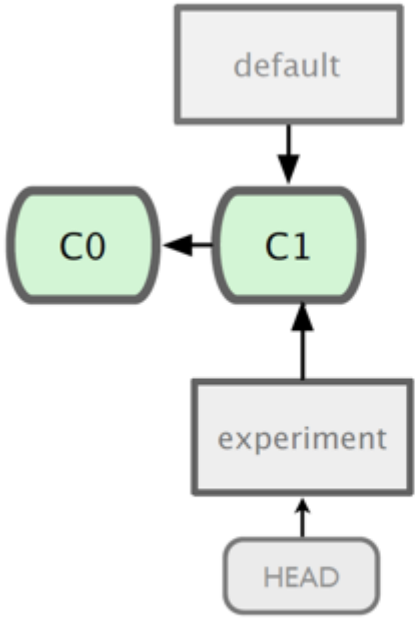
git branch experiment

Gestion des branches



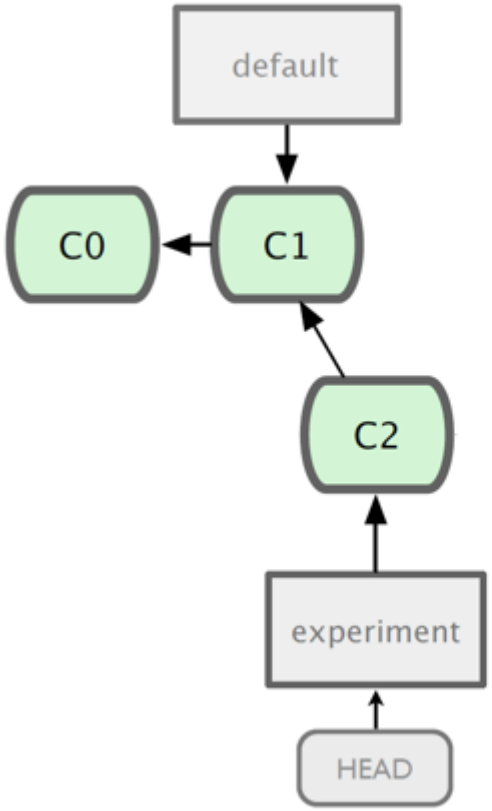

```
$ git branch
* default
  experiment
```

Gestion des branches



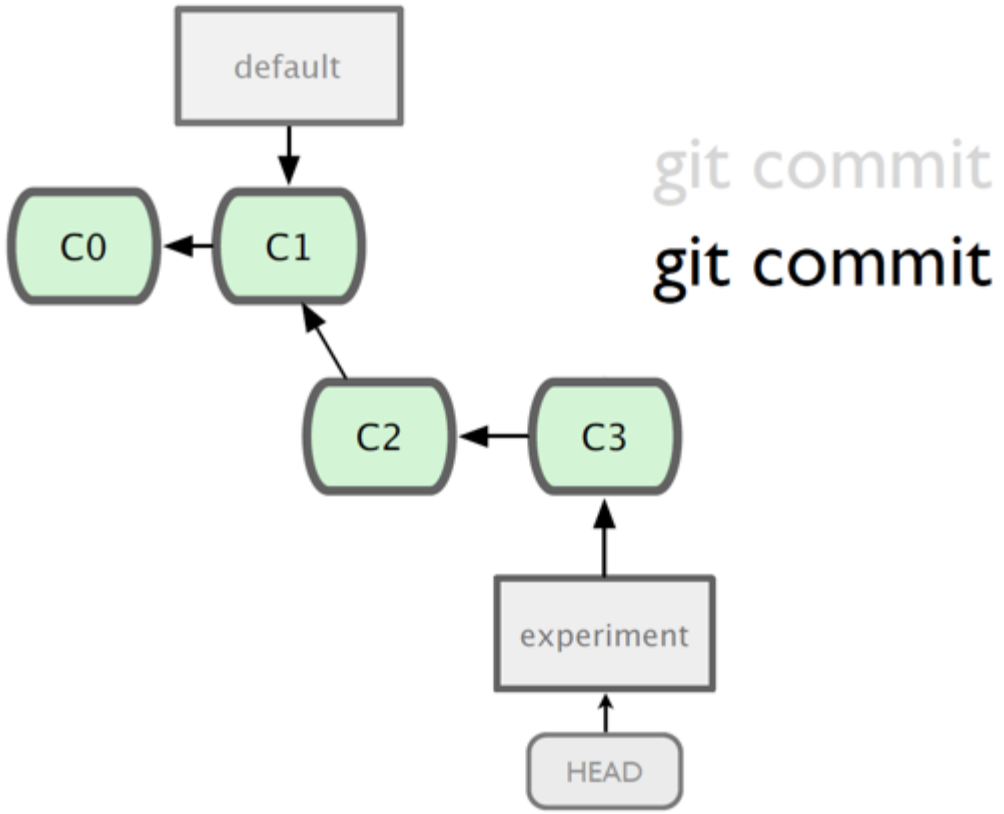
git checkout experiment

Gestion des branches

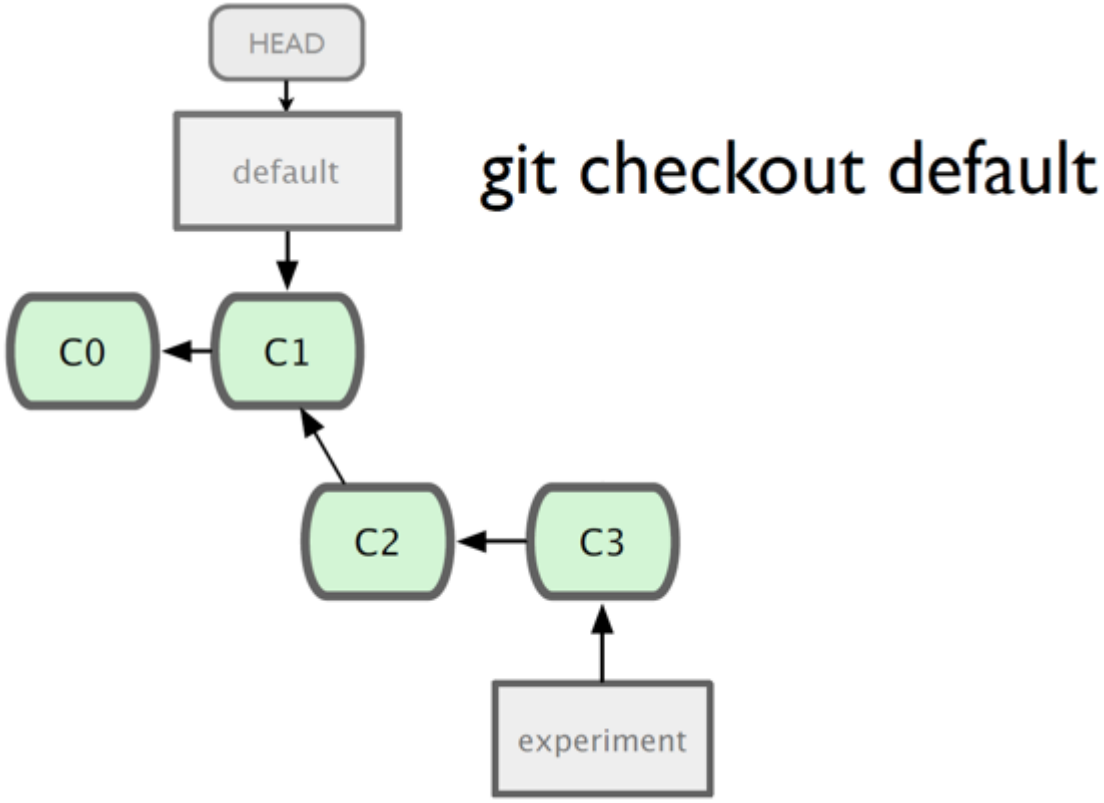


git commit

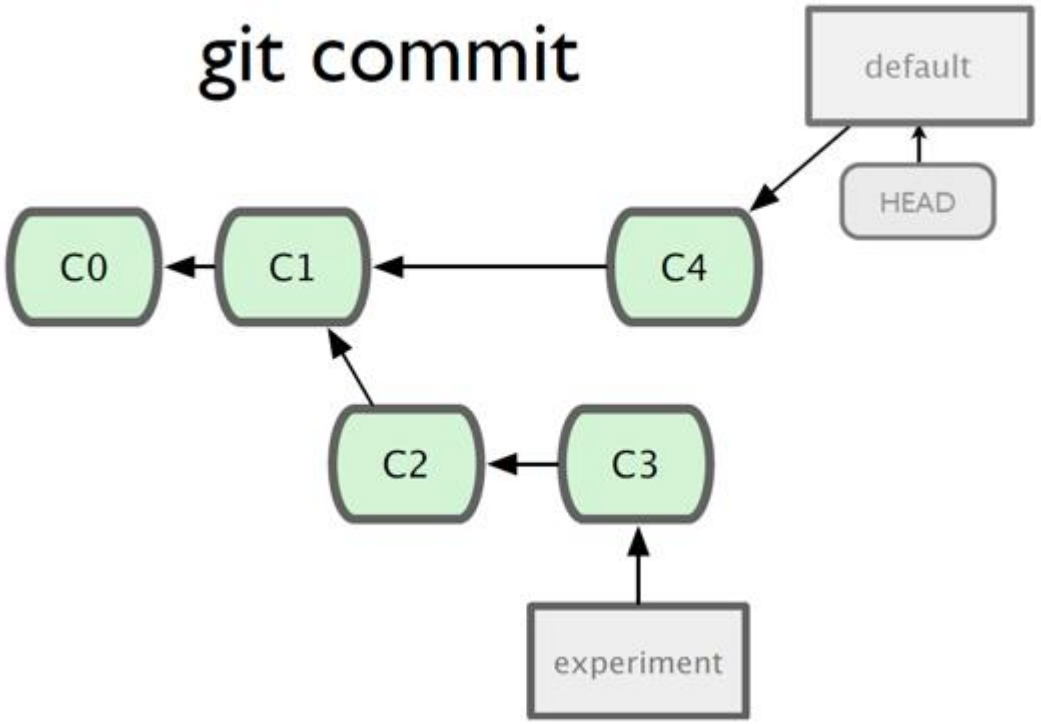
Gestion des branches



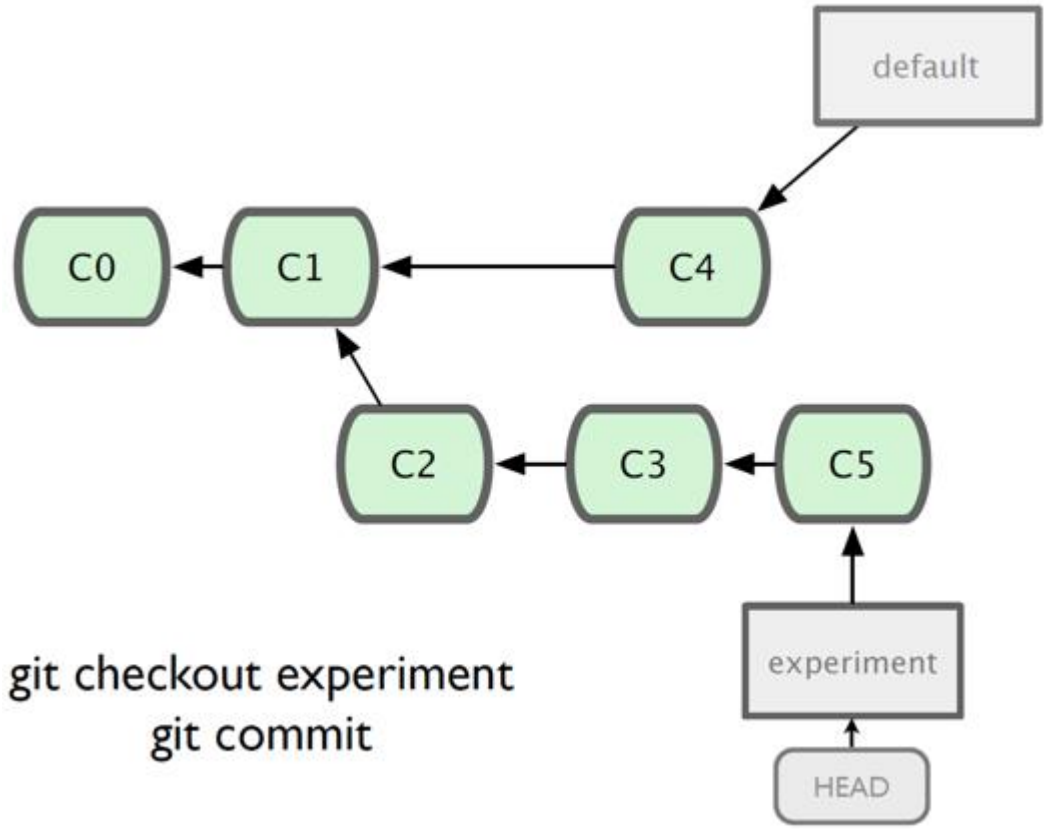
Gestion des branches



Gestion des branches



Gestion des branches



Gestion des branches

- **Lister les branches:**
  - git branch
- **Récupérer les branches distantes:**
  - git fetch
    - ➢ Permet à un utilisateur d'extraire tous les fichiers du dépôt distant qui ne sont pas actuellement dans le répertoire de travail local ;
    - ➢ Exemple d'utilisation: **git fetch origin**
- **Lister les branches distantes:**
  - git branch –a
- **Supprimer une branche**
  - git branch –d <nom-branche>

Gestion des branches

- **Gestion des merges :**
  - git merge branch1

  La **commande git merge** est utilisée pour fusionner une branche dans la branche active.

- **Gestion des conflits**
  - Git crée un conflit lorsqu'il ne sait pas quelle version d'un fichier il doit conserver. Cela arrive quand un fichier a été modifié au même endroit par 2 sources distinctes !
  - **git status** permet de comprendre la nature du conflit : c'est votre point de départ pour le résoudre.
  - À vous de choisir la solution qui vous convient le mieux pour résoudre votre conflit :
    - conserver la version distante ;
    - conserver la version locale ;
    - faire une version adaptée à la circonstance.