Direction Recherche et Ingénierie de la Formation

Examen de passage à la 2ème année

Session Juillet 2017 (Correction)

***Filière** : Techniques de Développement Informatique* — ***Épreuve** : Synthèse*
***Niveau**: TS* — ***Variante** : V1*
***Durée** : 5 heures* — ***Barème** : / 120pts*

❖ Partie I : Théorie (40 pts)

➢ Dossier 1: L'essentiel en technologies de l'information (14 pts)

▪ Exercice 1: Conversion numérique

NB: la calculatrice est strictement interdite.

Remplir le tableau suivant:

Décimal	Binaire	Octal	Hexadécimal
995	**1111100011**	**1743**	**3E3**
242	**11110010**	**362**	**F2**
222	**11011110**	**336**	**DE**
267	**100001011**	**413**	**10B**

▪ Exercice 2: Algèbre de Boole

Soit la fonction logique suivante:

$$F(g,h,k) = gh\bar{k} + \bar{g}\bar{h}k + g\bar{h}\bar{k} + g\bar{h}k$$

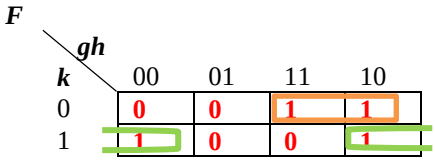
1- Simplifier analytiquement la fonction logique F.

$$F(g,h,k) = g\bar{k} + \bar{h}k$$

2- Construire la table de vérité.

g	h	k	F
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

3- Simplifier avec la méthode de Karnaugh la fonction logique ***F***



F	gh	00	01	11	10
k					
0		0	0	1	1
1		1	0	0	1

$$F(g,h,k) = g\bar{k} + \bar{h}k$$

➢ Dossier 2: Programmation structurée

Suppression du minimum d'un tableau

Il s'agit de supprimer le premier minimum rencontré dans un tableau déjà rempli par 10 réels.

Exemple:

```
Tableau: | 9 | -2.5 | 78 | 0 | -2.5 | 3 | 1.1 | 1 | 2 | 3 |
Nouveau tableau: | 9 | 78 | 0 | -2.5 | 3 | 1.1 | 1 | 2 | 3 |
```

1- Écrire une fonction qui retourne la position du premier minimum rencontré dans un tableau de taille quelconque passé en paramètre
2- Écrire une procédure qui supprime un élément d'un tableau de taille quelconque passé en paramètre, en passant la position de l'élément à supprimer par paramètre
3- Utiliser la fonction et la procédure pour supprimer le premier minimum rencontré dans un tableau déjà rempli par 10 réels.

```
Tableau b : Réel [ 10 ] <- [ 9 , -2.5 , 78 , 0 , -2.5 , 3 , 1.1 , 1 , 2 , 3 ]
Variable c : Entier
Début
     c <- PositionnerMini ( b , 10 )
     SupprimerA ( b , 10 , c )
Fin
Fonction PositionnerMini ( e : Réel [ 10 ] ; f : Entier ) : Entier
Variable g , h : Entier
Début
     g <- 1
     Pour h <- 2 à f
          Si e [ h ] < e [ g ] Alors
             g <- h
          FinSi
     FinPour
     Retourner g
Fin
Procédure SupprimerA ( R e : Réel [ 10 ] ; f , g : Entier )
Variable h : Entier
Début
     Pour h <- g à f - 1
          e [ h ] <- e [ h + 1 ]
     FinPour
Fin
```

➢ Dossier 3: Analyse et conception orientée objet

Gestion d'une équipe de football

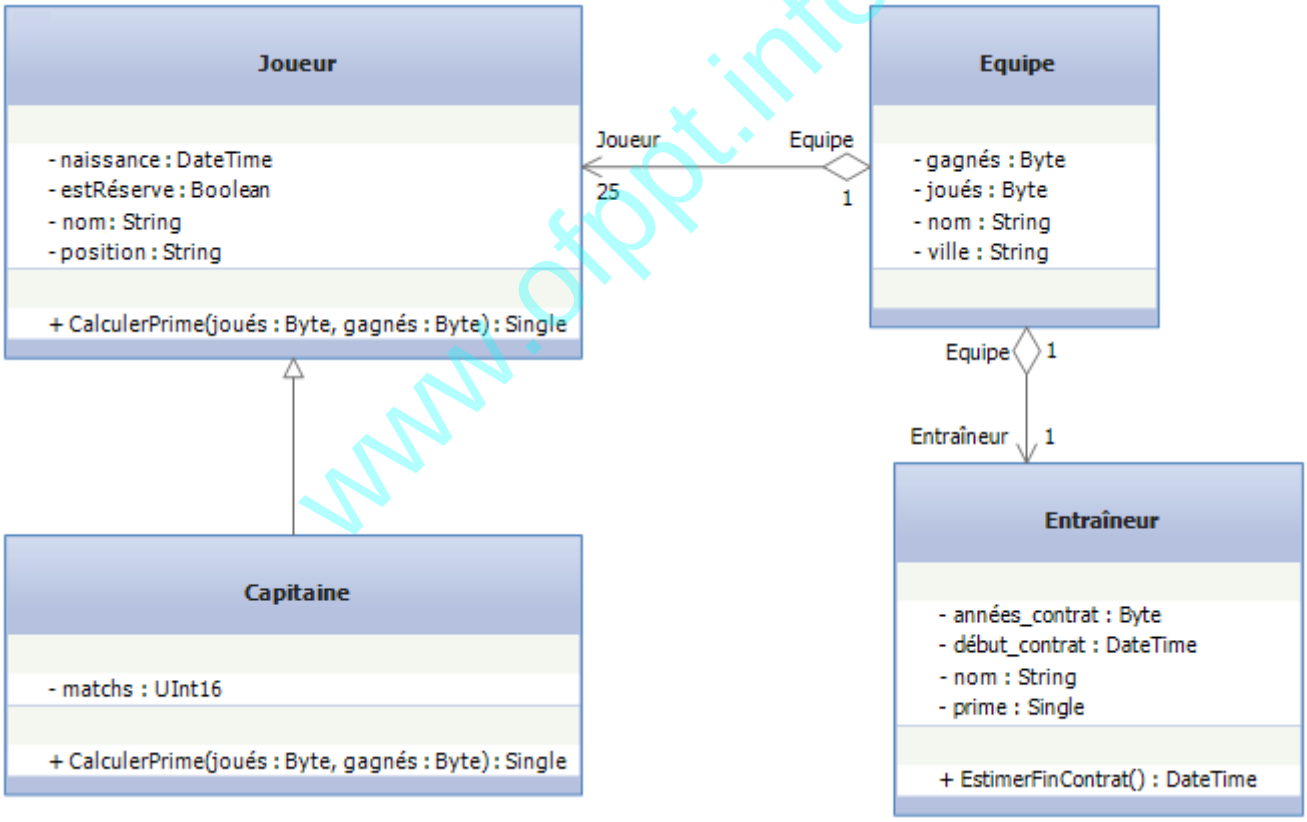
L'administration d'un club de football souhaite gérer son équipe. L'équipe de football a son nom, le nom de la ville qu'elle représente, le nombre de matchs joués, et le nombre de matchs gagnés pendant le championnat national.

L'équipe de football est formée d'un total de vingt-cinq joueurs (incluant ceux en réserve). Chaque joueur a son nom complet, sa date de naissance, sa position sur le terrain, et une indication sur le fait qu'il est en réserve ou non. Chaque joueur touche une prime mensuelle calculée automatiquement à partir du nombre de matchs joués, du nombre de matchs gagnés par l'équipe pendant le championnat national, et du fait qu'il est en réserve ou non.

Le capitaine de l'équipe est un joueur bien expérimenté. La caractéristique la plus importante pour ce joueur est le nombre de matchs qu'il a joué dans sa carrière. Le calcul de la prime mensuelle du capitaine de l'équipe tient en compte du nombre de matchs qu'il a joués aussi.

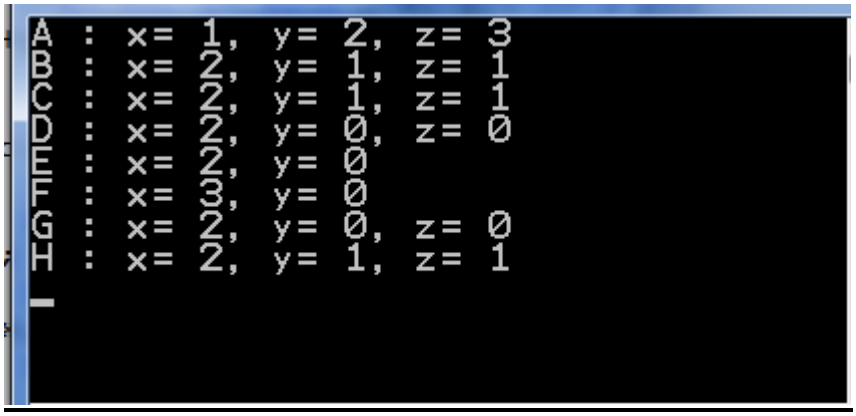
L'équipe est entraînée par un entraîneur. L'entraîneur a son nom complet, la date de début de son contrat avec l'équipe, le nombre d'années du contrat, et sa prime annuelle. La date de fin du contrat est estimée automatiquement.

Établir le diagramme de classes permettant de gérer les données de cette équipe. Toutes les variables d'instance doivent être privées, et toutes les méthodes doivent être publiques.



❖ Partie II: Pratique (80 pts)

➢ Dossier 1: Programmation structurée



➢ Dossier 2: Programmation événementielle et orientée objet (65 pts)

Développement d'une application pour la gestion d'une équipe de football

L'administration d'un club de football souhaite gérer son équipe. L'analyse de l'application mène au développement des classes suivantes (toutes les classes doivent être **publiques**, toutes les variables d'instance doivent être **privées**, et toutes les méthodes doivent être **publiques**):

1- Classe "Joueur":

a- Écrire la classe "**Joueur**" caractérisée par son nom complet, sa date de naissance, sa position sur le terrain, et s'il est en réserve (vrai) ou non (faux).

b- Ajouter tous les accesseurs

c- Ajouter à cette classe deux constructeurs: un constructeur par défaut, et un constructeur d'initialisation avec tous les paramètres.

d- Écrire une méthode polymorphe "**CalculerPrime(joués,gagnés)**" qui admet en paramètre les nombres des matchs joués et gagnés par son équipe pendant le championnat national, et qui retourne la prime mensuelle du joueur suivant la relation:

$$Prime = \begin{cases} 10000\ MAD\ x\ \dfrac{gagnés}{joués}, si\ le\ joueur\ est\ principal. \\ 50\%\ de\ la\ prime\ du\ joueur\ principal, si\ le\ joueur\ est\ en\ réserve. \end{cases}$$

```
using System;
namespace Foot
{
    public class Joueur
    {
        private string nom,position;
        private DateTime naissance;
        private bool estRéserve;
        public string Nom
        {
            get
            {
                return nom;
            }
            set
            {
                nom = value;
            }
        }
```

```
        public DateTime Naissance
        {
            get
            {
                return naissance;
            }
            set
            {
                naissance = value;
            }
        }
        public string Position
        {
            get
            {
                return position;
            }
            set
            {
                position = value;
            }
        }
        public bool EstRéserve
        {
            get
            {
                return estRéserve;
            }
            set
            {
                estRéserve = value;
            }
        }
        public Joueur(){}
        public Joueur(string nom, string position, DateTime naissance, bool estRéserve)
        {
            this.nom = nom;
            this.position = position;
            this.naissance = naissance;
            this.estRéserve = estRéserve;
        }
        public virtual float CalculerPrime(byte joués, byte gagnés)
        {
            float prime = 10000 * gagnés / joués;
            return prime = estRéserve ? prime : prime / 2;
        }
    }
}
```


2- Classe "Capitaine":

a- Écrire la classe "**Capitaine**" qui hérite de la classe "**Joueur**". Le capitaine est caractérisé par le nombre de matchs qu'il a joué dans sa carrière.

b- Ajouter à cette classe un constructeur d'initialisation avec tous les paramètres. Le nombre de matchs doit être au moins 100; sinon lever une exception.

c- Réécrire la méthode polymorphe "**CalculerPrime(joués,gagnés)**" sachant que:

$$Prime_{Capitaine} = Prime_{Joueur} + 50\ MAD\ x\ nombre\ de\ matchs\ de\ carrière.$$

```
using System;
namespace Foot
{
```

```
    public class Capitaine : Joueur
    {
        private ushort matchs;
        public Capitaine(string nom, string position, DateTime naissance, bool estRéserve, ushort matchs)
            : base(nom, position, naissance, estRéserve)
        {
            if (matchs < 100) throw new Exception("Expérience insuffisante!");
            this.matchs = matchs;
        }
        public override float CalculerPrime(byte joués, byte gagnés)
        {
            return CalculerPrime(joués, gagnés) + 50 * matchs;
        }
    }
}
```

3- Classe "Entraîneur":

a- Écrire la classe "**Entraîneur**" caractérisée par son nom complet, sa date de début de son contrat avec l'équipe, le nombre d'années du contrat, et sa prime annuelle.

b- Ajouter à cette classe un constructeur d'initialisation avec tous les paramètres. La prime annuelle doit être entre 200000 MAD et 300000 MAD; sinon lever une exception

c- Écrire une méthode "**EstimerFinContrat()**" qui retourne la date de fin du contrat.

```
using System;
namespace Foot
{
    public class Entraîneur
    {
        private string nom;
        private DateTime début_contrat;
        private byte années_contrat;
        private float prime;
        public Entraîneur(string nom, DateTime début_contrat, byte années_contrat, float prime)
        {
            this.nom = nom;
            this.début_contrat = début_contrat;
            this.années_contrat = années_contrat;
            if (prime < 200000 || prime > 300000) throw new Exception("Erreur prime!");
            this.prime = prime;
        }
        public DateTime EstimerFinContrat()
        {
            return début_contrat.AddYears(années_contrat);
        }
    }
}
```

4- Classe "Equipe":

a- Écrire la classe "**Equipe**" caractérisée par son nom, le nom de la ville qu'elle représente, le nombre de matchs joués, le nombre de matchs gagnés pendant le championnat national, la liste de ses joueurs, et son entraîneur.

b- Ajouter un accesseur à lecture seule pour la liste des joueurs

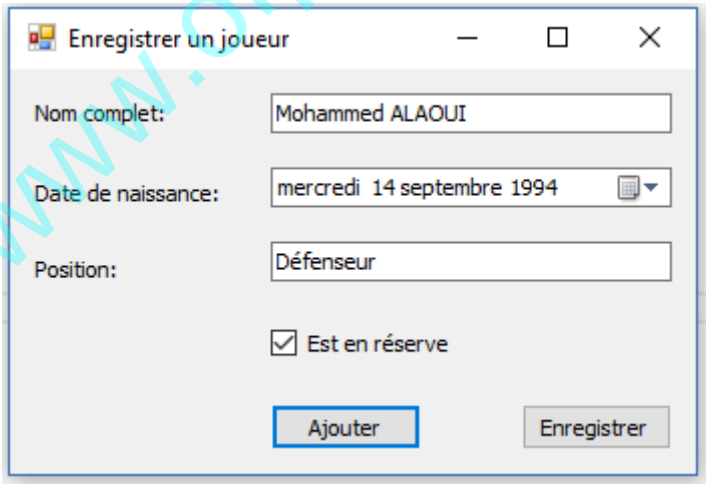
c- Ajouter à cette classe deux constructeurs: un constructeur sans paramètres, et un constructeur d'initialisation avec tous les paramètres sauf la liste des joueurs. Les deux constructeurs doivent instancier la liste des joueurs avec une capacité de 25.

```
using System;
using System.Collections.Generic;
namespace Foot
{
```

```
    public class Equipe
    {
        private string nom, ville;
        private byte joués, gagnés;
        private List<Joueur> joueurs;
        private Entraîneur entraîneur;
        public List<Joueur> Joueurs
        {
            get
            {
                return joueurs;
            }
        }
        public Equipe()
        {
            joueurs = new List<Joueur>(25);
        }
        public Equipe(string nom, string ville, byte joués, byte gagnés, Entraîneur entraîneur)
            : this()
        {
            this.nom = nom;
            this.ville = ville;
            this.joués = joués;
            this.gagnés = gagnés;
            this.entraîneur = entraîneur;
        }
    }
}
```

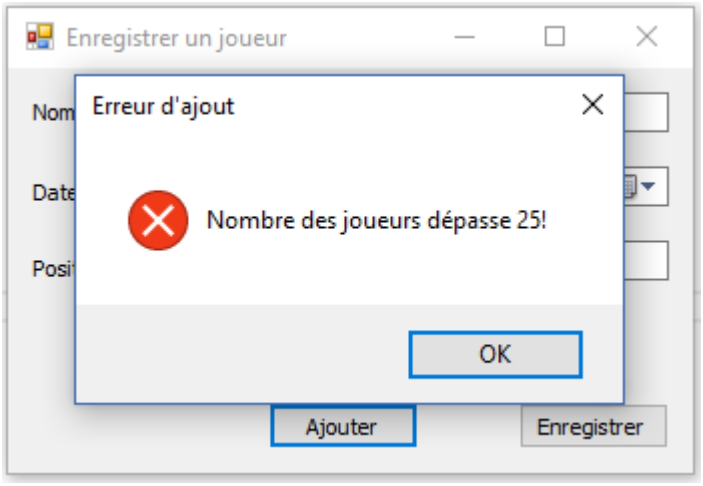
5- Formulaire d'enregistrement des joueurs:

NB: donner uniquement le code à mettre à l'intérieur des méthodes événementielles. L'entête de ces méthodes événementielles n'est pas demandé.

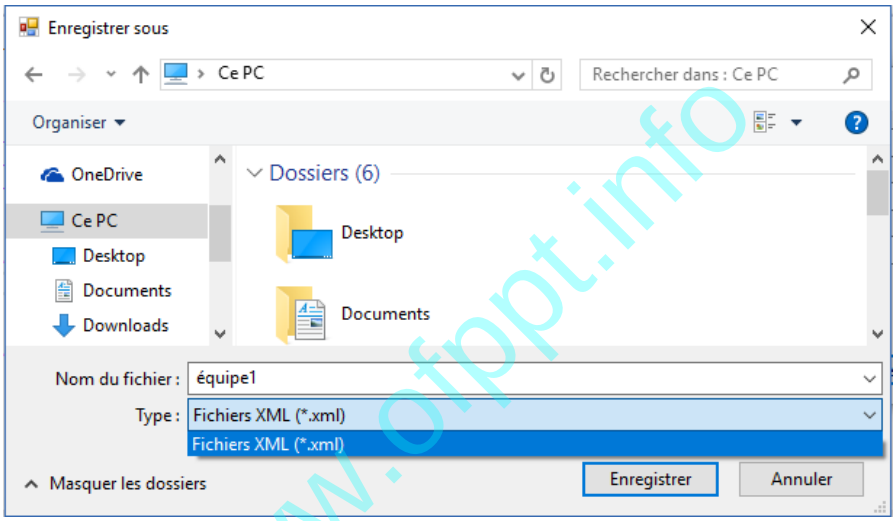


Soit l'équipe "équipe1" une instance de la classe "Equipe". Le formulaire précédent permet d'enregistrer l'équipe avec sa liste de joueurs.

a- Écrire le code du bouton "Ajouter" permettant d'ajouter un joueur à la liste des joueurs de "équipe1". Lever une exception si le nombre de joueurs dépasse 25. Capturer l'exception dans une boite de message d'erreur.



b- Écrire le code du bouton "Enregistrer" permettant d'enregistrer l'équipe dans un fichier XML, en utilisant une boîte de dialogue de sauvegarde de fichiers.




```
using System;
using System.Windows.Forms;
using System.IO;
using System.Xml.Serialization;
namespace Foot
{
    public partial class Form1 : Form
    {
        private Equipe équipe1;
        public Form1()
        {
            InitializeComponent();
            équipe1 = new Equipe();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                if (équipe1.Joueurs.Count == 25) throw new Exception("Nombre des joueurs dépasse 25!");
      équipe1.Joueurs.Add(new Joueur(textBox1.Text, textBox2.Text, dateTimePicker1.Value, checkBox1.Checked));
            }
            catch (Exception a)
            {
```

```
                MessageBox.Show(a.Message, "Erreur d'ajout", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
        private void button2_Click(object sender, EventArgs e)
        {
            SaveFileDialog saveFileDialog1 = new SaveFileDialog();
            saveFileDialog1.Filter = "Fichiers XML (*.xml)|*.xml";
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                    XmlSerializer format = new XmlSerializer(typeof(Equipe));
                    using (StreamWriter fluxXML = new StreamWriter(saveFileDialog1.FileName))
                        format.Serialize(fluxXML, équipe1);
            }
        }
    }
}
```