esprit Ecole Supérieure Privée d'Ingénierie et de Technologies	**Année Universitaire : 2015-2016** **DS**	
Module : Conception par objet et programmation Java		**Documents autorisés** : Non
Enseignants : Equipe Java		**Nombre de pages** : 4
Date : 30/10/2015	**Heure :**9h00	**Durée** : 1h00
Classes : 3 INFO A, 4 INFINI		

PARTIE 1 : (4.5)

Choisir la bonne réponse en justifiant votre choix. Toute réponse non justifiée ne sera pas comptabilisée.

Question 1 :

Soit le code suivant :

```
class One {
public One() { System.out.print(1); }
}
class Two extends One {
public Two() { System.out.print(2); }
 }
class Three extends Two {
public Three() { System.out.print(3); }
}
public class Numbers{
public static void main( String[] args) {
new Three(); }
}
```

Quel est le résultat lorsque ce code est exécuté ?

A. 1

B. 3

C. 123

D. 321

E. Aucun affichage

Réponse C : Un appel implicite au constructeur par défaut de la classe mère s'effectue avant d'exécuter le contenu du constructeur de la classe courante

Réponse C : Avant d'exécuter ses instructions le constructeur de la classe Three appel le constructeur par défaut de la classe mère via un super() implicite, et pareil pour la classe Two().

Question 2 :

Soit le code suivant :

```
1. class Person {
2. String name = "No name";
3. public Person(String nm) { name = nm; }
4. }
5. class Employee extends Person {
6. String emplD = "0000";
7. public Employee(String id) { empID = id; }
8 }
9. public class EmployeeTest {
10. public static void main(String[] args) {
11. Employee e = new Employee("4321");
12. System.out.println(e.empID);  }
14. }
```

Quel est le résultat ?

A. 4321

B. 0000

C. Une exception est levée lors de l'exécution.

D. Compilation échoue en raison d'une erreur dans la ligne 7.

Réponse D : Constructeur par défaut de la classe mère manquant OU appel au constructeur paramétré de la classe mère manquant

Question 3:

Soit le code suivant :

```
1. int []x= {1, 2,3,4, 5};
2.int y[] =x;
3. System.out.println(y[2]);
```

Quelle est la bonne réponse ?

A. Ligne 3 affiche la valeur 2.

B. Ligne 3 affiche la valeur 3.

C. Compilation échoue en raison d'une erreur dans la ligne 2.

D. Compilation échoue en raison d'une erreur dans la ligne 3.

Réponse B : Au niveau de la ligne 2 on a affecté les objets de x à y et vu que l'indexation d'un tableau commence de 0, l'index 2 affiche la valeur de la troisième case

PARTIE 2 : (15.5)

Après La suspension temporaire de Joseph "Sepp" Blatter l'actuel président de la Fifa. Et en prévision du pire vue qu'Issa Hayatou assura l'intérim, certaines fédérations de football ont jugé utile de se doter de leur propre application pour la gestion des équipes et des joueurs. Dans ce contexte et après une brève analyse il apparait que :

- Une **équipe** est caractérisée par un identifiant de type entier et un nom de type chaine de caractère.
- Un **membre** est caractérisé par un identifiant de type entier, un nom de type chaine de caractère.
- Un **joueur** est un membre spécialisé par son numéro de type entier
- Un **entraineur** est un membre spécialisé par le nombre de ses années d'expériences de type entier.
- Une **équipe** possède plusieurs membres qui sont au nombre maximal de 30.

Travail demandé :

Respecter les Règles (Conventions) du Nommage Java

Les Getters et les setters ne sont à implémenter que si leur demande est explicite.

I. Implémenter la classe Membre. Elle doit comprendre : (2.5)
   - Uniquement un constructeur paramétré. (0.5)
   - Les Getters et les Setters seulement pour l'attribut identifiant. (0.5)
   - La méthode String **toString**().(0.5)
   - La méthode **boolean equals(Membre m)**. sachant que deux membres sont égaux s'ils ont le même id et le même nom. (1)

```java
public class Membre {

    protected int id;
    protected String nom;

    public Membre(int id, String nom) {
        super();
        this.id = id;
        this.nom = nom;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    @Override
    public String toString() {
        return "Membre [id=" + id + ", nom=" + nom + "]";
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null)
            return false;
        if (obj instanceof Membre) {
            Membre membre = (Membre) obj;
            if (id == membre.id && nom.equals(membre.nom))
                return true;
        }
        return false;
    }
}
```

II. Implémenter les classes Joueur et Entraineur. Veiller à ce qu'elles contiennent un constructeur et la méthode toString().(2)

```java
public class Joueur extends Membre {

    int numero;

    public Joueur(int id, String nom, int numero) {
        super(id, nom);
        this.numero = numero;
    }

    @Override
    public String toString() {
        return "Joueur " + super.toString() + " [numero=" + numero + "]";
    }
}
```

```java
public class Entraineur extends Membre {

    private int nbrAnnees;

    public Entraineur(int id, String nom, int nbrAnnees) {
        super(id, nom);
        this.nbrAnnees = nbrAnnees;
    }

    @Override
    public String toString() {
        return "Entraineur " + super.toString() + " [nbrAnnees=" + nbrAnnees
                + "]";
    }
}
```

III. Implémenter la classe Equipe. Elle doit comprendre :

1. La méthode **int rechercher(Membre membre) :** cette méthode retourne l'indice (position) du membre s'il existe, -1 sinon. (1)

```java
public int rechercher(Membre membre) {
    for (int i = 0; i < nbrMembre; i++) {
        if (membres[i].equals(membre))
            return i;
    }
    return -1;
}
```

2. La méthode **void ajouter (Membre membre) :** cette méthode permet d'ajouter un nouveau membre à l'équipe, tout en prenant en considération qu'une équipe ne peut dépasser sa capacité maximale (30 membres) et qu'un membre ne peut être affecté deux fois à la même équipe. (1.5)

```
public void ajouter(Membre membre) {
    if (nbrMembre < MAX_SIZE && rechercher(membre) == -1) {
        membres[nbrMembre] = membre;
        nbrMembre++;
    }
}
```

3. La méthode **void supprimer(Membre membre) :** cette méthode permet de supprimer un membre de l'équipe. (2)

```
public void supprimer(Membre membre) {
    int position = rechercher(membre);
    if (position != -1) {
        for (int i = position; i < nbrMembre; i++) {
            membres[i] = membres[i + 1];
        }
        nbrMembre--;
    }
}
```

4. La méthode **String toString()** : cette méthode retourne une chaine de caractère contenant l'id, le nom de l'équipe et uniquement la liste de ses joueurs. (1.5)

```
public String toString() {
    String joueurs = "";
    for (int i = 0; i < nbrMembre; i++) {
        if (membres[i] instanceof Joueur)
            joueurs += "\n" + membres[i].toString();
    }
    return "Equipe [id=" + id + ", nom=" + nom + "]" + joueurs;
}
```

5. La méthode **void valider()** : cette méthode affiche le message suivant « Equipe valide » si l'équipe possède un et un seul entraineur parmi ses membres, et déclenche une Exception personnalisée de type **EquipeNonValideException** sinon. (1.5)

```java
public void valider() throws EquipeNonValideException {
    int nbEntraineur = 0;
    for (int i = 0; i < nbrMembre; i++) {
        if (membres[i] instanceof Entraineur)
            nbEntraineur++;
    }
    if (nbEntraineur == 1) {
        System.out.println("Equipe valide");
    } else {
        throw new EquipeNonValideException();
    }
}
```

6. La méthode **void afficherNumeros()** : cette méthode affiche uniquement les numéros des joueurs de l'équipe. (1.5)

```java
public void afficherNumeros() {
    for (int i = 0; i < nbrMembre; i++) {
        if (membres[i] instanceof Joueur)
            System.out.println(((Joueur) membres[i]).numero);
    }
}
```

7. La méthode **void transfererJoueur(Joueur joueur, Equipe e1, Equipe e2)** : cette méthode permet de transférer le joueur de l'équipe e1 vers l'équipe e2. Si le joueur n'est pas dans l'équipe e1, une Exception personnalisée **TransfertException** sera levée avec le message suivant « Transfert impossible » (2)

```java
public void transfererJoueur(Joueur joueur, Equipe e1, Equipe e2)
        throws TransfertException {
    if (e1.rechercher(joueur) == -1) {
        throw new TransfertException("Transfert impossible");
    } else {
        e1.supprimer(joueur);
        e2.ajouter(joueur);
    }
}
```

```java
public class EquipeNonValideException extends Exception{
}
```

```java
public class TransfertException extends Exception {
    public TransfertException(String message) {
        super(message);
    }
}
```