UEF4.3. Programmation Orientée Objet

Nom:	Prénoms:	Groupe:

Contrôle intermédiaire (durée: 2h)

Exercice 1 (6 points) Qu'affiche le programme suivant ?

```
class A {
double x;
A(int k) {x=k;}
double plus(double a) { return x+a; }
public void imprimer() { System.out.println(" x = "+ x);}
}
class B extends A {
int y;
B (int k, int l) { super(k); y=l; }
double plus(double a) { return x+2*a;}
}
class C extends B {
int z ;
int x;
C (int k, int l, int m) {super(k, l); z= m; x=2*m;}
int plus(int a) { return x+3*a;}
public void imprimer() {
super.imprimer();
System.out.println(" z = "+ z);}
}
public class Exo1{
public static void main (String args[]) {
int i =2; double j=0.5;
A a = new A(3);
a.imprimer();
System.out.println(" plus("+ i +") = "+ a.plus(i) );
a = new B(3,4);
a.imprimer();
System.out.println(" plus("+ i +") = "+ a.plus(i) );
a = new C(3,4,5);
a.imprimer();
System.out.println(" plus("+ i +") = "+ a.plus(i) );
System.out.println(" plus("+ j +") = "+ a.plus(j) );
B b = new B(3, 4);
b.imprimer();
System.out.println(" plus("+ i +") = "+ b.plus(i) );
b = new C(3, 4, 5);
b.imprimer();
System.out.println(" plus("+ i +") = "+ b.plus(i) );
System.out.println(" plus("+ j +") = "+ a.plus(j) );
C c = new C (3,4,5);
c.imprimer();
System.out.println(" plus("+ i +") = "+ c.plus(i) );
System.out.println(" plus("+ j +") = "+ c.plus(j) );
}}
```

Réponse

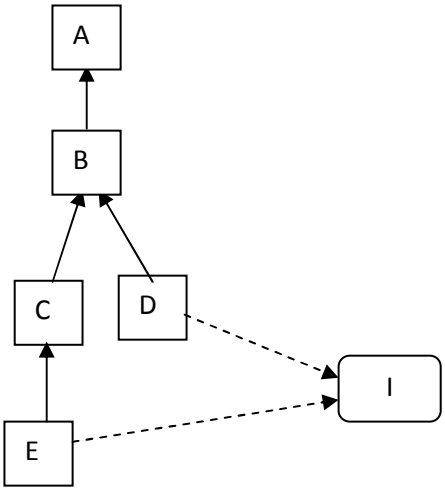
```
x = 3.0
plus(2) = 5.0
x = 3.0
plus(2) = 7.0
x = 3.0
z = 5
plus(2) = 7.0
plus(0.5) = 4.0
x = 3.0
plus(2) = 7.0
x = 3.0
z = 5
plus(2) = 7.0
plus(0.5) = 4.0
x = 3.0
z = 5
plus(2) = 16
plus(0.5) = 4.0
```

0,5 par réponse surlignée (0,5*6)

0,25 pour les autres (0,25 * 12)

Exercice 2 (4,75points)

Soit la hiérarchie de classes illustrée dans la figure ci-contre:



A, B, C, D, E sont des lasses et I est une interface implémentée par les classes D et E.

Dans la suite d'instructions suivante, indiquez pour chacune d'elles:

1. Si elle est correcte ou si elle provoque une erreur à la compilation.
2. Si on peut résoudre le problème posé à la compilation et comment.
3. Si elle provoque une erreur à l'exécution.

0,25 par case colorée. Si la correction proposée est erronée, la case exécution n'est pas prise en considération.

Instruction	**Compilation** (OK ou RREUR)	**Correction possible ?** (OUI ou NON)	**Correction**	**Exécution** (OK ou ERREUR)
`A a1 = new A();`	OK			
`I a2 = new E();`	OK			
`I a3 = new D();`	OK			
`I a4 = new C();`	ERREUR	NON		
`C a5 = new E();`	Ok			
`a5 = a1;`	ERREUR	OUI	`a5 = (C)a1;`	ERREUR
`a1 = a5;`	OK			
`B a6 = new B();`	OK			
`a5 = a6;`	ERREUR	OUI	`a5 = (C) a6;`	ERREUR
`E a7;`	OK			
`a7 = a6;`	ERREUR	OUI	`a7 = (E) a6;`	ERREUR
`a7 = a5;`	ERREUR	OUI	`a7 = (E) a5;`	OK
`a3 = a7;`	OK			

Exercice 3 (4 points)

1. On désire écrire un programme java qui permet à l'utilisateur de saisir, pendant l'exécution, des noms de pays et leurs superficies et les affiche triés selon l'ordre alphabétique du nom. Si plusieurs pays commencent par la même lettre alors ils sont affichés par ordre décroissant de leurs superficies.
Exemple: si l'utilisateur entre les 10 pays les plus vastes du monde ils seront affichés comme suit:

```
Australie    7692060
Argentine    2780400
Algérie      2381740
Brésil       8547400
Canada       9970610
Chine        9598050
Inde         3287260
Kazakhstan   2724900
Russie      17075400
USA          8629090
```

Pour cela utilisez la classe `Pays` définie par les attributs `nom` (de type `String`) et superficie (de type `long`) et la classe Exo3 contenant la méthode main. Dans cette dernière déclarez un tableau de taille 10, remplissez-le et triez-le selon l'ordre défini puis affichez ses éléments.
Indication: Pour connaitre l'initiale du nom d'un pays, utilisez la méthode suivante:

`public char charAt(int index)`
Returns the `char` value at the specified index. An index ranges from `0` to `length() - 1`. The first `char` value of the sequence is at index `0`, the next at index `1`, and so on, as for array indexing.

Solution

```
import java.util.Arrays;
import java.util.Scanner;

class Pays implements Comparable<Pays>{
      String nom;
      int superficie;
public Pays(String nom, int sup){
      this.nom = nom;
      superficie = sup;}

public String toString(){
return(nom + " \t" + superficie);}

public int compareTo(Pays p){
if (nom.charAt(0)==p.nom.charAt(0))
return (p.superficie-superficie);
else
return nom.compareTo(p.nom); }
}
```

```
public class Exo3 {
public static void main(String args[]){
Pays liste[]= new Pays[10];
Scanner sc = new Scanner(System.in);
String nom; long sup;

System.out.println("Entrez 10 noms de
pays et leurs superficies");
for (int i = 0; i<liste.length; i++){
nom = sc.next();
sup = sc.nextLong();
liste[i] = new Pays(nom, sup);     }

Arrays.sort(liste);

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

Exercice 4 (5,25 points)

On veut développer une application qui gère les sorties d'un utilisateur. Chaque sortie est définie par un nom, un lieu, une date et une heure (ex: "Jardin d'essais", "El Hama", 9/4/2016 à 10h00)

1. En vous aidant des classes du paquetage `java.time` utilisées en TP1, Ecrivez le programme java de cette application qui permet de gérer 10 sorties au maximum. Votre programme doit comporter deux classes: `Sortie` et `GestionSorties` et permettre à l'utilisateur de:
- ajouter une sortie
- afficher les sorties programmées
- afficher les sorties comprises entre deux dates données.

2. Pour améliorer l'application, on désire assurer que l'utilisateur ne programmera pas deux sorties à la même date et à la même heure. Expliquez ce que vous devez ajouter ou modifier dans le programme pour répondre à cette contrainte.

Solution : Question 1 (3,25 points)

```
import java.time.*;

class Sortie {
      String nom;
      String lieu;
      LocalDateTime dateEtHeure;

public Sortie (String n, String l, int
y, int m, int d, int h, int mn){
      nom = n;
      lieu = l;
      dateEtHeure =
LocalDateTime.of(y,m,d,h,mn);
}

public String toString(){
return (dateEtHeure.toString()+ " : "+
nom + " à" + lieu );
}
}
```

```
class GestionSortie {
Sortie sorties[];
int nbSorties;
public GestionSortie(){
   sorties = new Sortie[10];
   nbSorties = 0;
   }

public void ajouterSortie(String n,
String l, int y, int m, int d, int h, int
mn ){
sorties[nbSorties] = new
Sortie(n,l,y,m,d,h,mn);
nbSorties++;
}

public void afficherPeriode
(LocalDateTime d1, LocalDateTime d2){
int i = 0;
while (i<nbSorties &&
sorties[i].dateEtHeure.isAfter(d1) &&
sorties[i].dateEtHeure.isBefore(d2))
{System.out.println(sorties[i].toString());
i++;
}
}
```

Question 2 (2 points)

Pour garantir qu'un utilisateur ne programmera pas deux sorties à la même date et à la même heure il faut :

- Importer le paquetage `Java.util.*;`

- Remplacer l'attribut tableau de sorties par un objet de type HashSet d'objets de type Sortie

```
Set<Sortie> sorties = new HashSet<Sortie>();
```

-Redéfinir la méthode public int hashCode () de la classe object: elle doit retourner le code de hachage de l'attribut dateEtHeure en faisant appel à la méthode hashCode de la classe LocalDateTime

```
public int hashCode(){
          return dateEtHeure.hashCode();
}
```

- Redéfinir la méthode public boolean equals(Object o) de la classe Object : elle doit retourner la true si deux objets de type sortie ont la même valeur pour l'attribut dateEtHeure en faisant appel à la méthode equals de la classe LocaldateTime

```
public boolean equals(Object dt){
          return dateEtHeure.equals(((Sortie)dt).dateEtHeure);
     }
```

-modifier la méthode ajouter en utilisant la méthode add se HashSet
-modifier la méthode afficher en utilisant un itérateur pour parcourir le HashSet

0,25 par réponse surlignée (0,25 * 7)

0,25 pour les deux dernières réponses réunies