Université de Carthage
Ecole Supérieure de la Statistique et de l'Analyse de l'Information

Enseignantes : Mme Aïcha El Golli

Niveau d'études : 2ème année

Année Universitaire 2022/2023

Examen Session Principale
Module : POO -JavaI

Durée de l'épreuve : 1h30

Remarques : *Les Documents, calculatrices,* ***téléphone portable sont interdits****.*

Veuillez rendre une copie propre et claire. Si la syntaxe d'une instruction est fausse alors la note est 0. La qualité de l'écriture et de la présentation sera prise en compte dans la note finale.

Questions de réflexion (6 points)

1) Sous Eclipse nous travaillons dans un dossier projet (project folder) qui, en plus de vos fichiers sources et vos fichiers compilés, inclus toutes les ressources nécessaires à votre projet de développement ainsi que deux fichiers de configuration (classpath et .project) utilisés par l'IDE. Ces deux fichiers peuvent être supprimés. **(Vrai/faux) ?** si oui pourquoi ? **faux 1pt**
2) Ayant la déclaration suivante : String[] t= new String[]{"hello", null, ""} ; que retournent les instructions suivantes :
   a. t[1].equals(t[2]) ? **une erreur 1pt**
   b. t[2].equals(t[1]) ? **false 0,5pt**
   c. t[0].charAt(1) ? **e 0,5pt**
3) Soit une interface Java I, et deux classes C1 et C2 qui l'implémentent. Lesquelles des déclarations suivantes sont justes ou fausses ? **Pourquoi** ?

1. I x = new I() ; **faux, on n'instancie pas une interface 1pt**
2. C1 y = new C1() ; **juste, déclaration suivie d'instanciation par une classe qui implémente l'interface 0,5pt**
3. I[] z = {new C1(), new C2()} **; idem, même si deux classes différentes. Elles implémentent la même interface 0,5pt**
4. C1 w = new C2() **; faux, on déclare une classe et on instancie avec une autre (types incompatibles) 1pt**

Exercice 1 (4 Points)

Considérons le programme suivant :

```
class ClassA{
public static int x=4;
public int y;
public ClassA(){this(6);}
public ClassA(int x){ClassA.x++;y=x;}
public static void f(ClassA a1){x++;}
public void f(int y){this.y=y;}
public String g(){return "POO en Java";}
public String h(){return this.g();}
}
class ClassB extends ClassA{
public ClassB(){ }
public String g(){return "POO en C++";}
public String h(){return super.h();}
}
public class TestExamen{
public static void main(String[] args){
ClassA a1 =new ClassA();
ClassA a2 =new ClassB();
ClassA.f(a1); a2.f(17);
System.out.println(
          ClassA.x +" "+a1.y +" "+a2.y);
System.out.println(a1.g());
System.out.println(a2.h());
ClassA a3=a1; ClassA a4=a2;
System.out.println(
          ClassA.x +" "+a3.y+" "+a4.y);
}}
```

a. Que va-t-on obtenir à l'écran lors de l'exécution de ce programme ?

7 6 17	**0,5pt**
POO en Java	**0,25pt**
POO en C++	**0,25pt**
7 6 17	**0,5pt**

b. Même question en ajoutant l'instruction "super(6);" au début du constructeur B()

La même chose **1pt (0,25pt chaque ligne)**

c. Indiquez si les instructions suivantes sont correctes ou pas (ce qui se passe à la compilation et à l'exécution) et, pour les instructions qui vous semblent correctes, indiquez ce qui serait affiché, s'il y a affichage.

	Compilation	Exécution
1) ClassA o=new ClassB(); 2) System.out.println(o.h()); 3) ClassA a=(ClassA) o;	Ok upcast**0,25pt** Ok**0,25pt** ok**0,25pt**	Ok**0,25pt** **0,25pt** POO en C++ **0,25pt** ok

Exercice 2 (10 Points)

Le tri à sélection est un algorithme classique permettant de trier un tableau d'entiers. Il peut s'écrire de la façon suivante en Java :

```
public static void triSelection(int[] tab) {
    for (int i = 0; i < tab.length - 1; i++)
     {  int index = i;
        for (int j = i + 1; j < tab.length; j++)
        {   if (tab[j] < tab[index])
              index = j;
         }
        int min = tab[index];
        tab[index] = tab[i];
        tab[i] = min;
     }                                    }
```

Cette implémentation du tri à sélection permet de trier un tableau d'entiers. Maintenant on veut pouvoir utiliser tout autre type de données (muni d'une relation d'ordre) sans avoir à réécrire l'algorithme à chaque fois. Pour cela on va supposer définie comme suit, l'interface Sortable :

```
public interface Sortable {
// échange les éléments en positions i et j
void swap(int i, int j);
// retourne vrai si l'élément de position i est plus grand que l'élément de position j
boolean bigger(int i, int j);
// nombre d'éléments à trier
int size();}
```

Les objets des classes implémentant cette interface devront représenter ainsi des tableaux d'éléments, comparables entre eux, que l'on souhaite trier.

1. Réécrire la méthode triSelection(Sortable t) qui met en œuvre le tri à sélection pour les objets Sortables. Finir la méthode suivante : static void triSelection(Sortable t){…} **4pt**

```
public static void triSelection(Sortable t)
{ for (int i = 0; i < t.size() - 1; i++)          1pt
  {
    int index = i;                                0,5pt
    for (int j = i + 1; j < t.size(); j++)        1pt
    {
        if (t.bigger(index,j))  index = j;        0,5pt  0,5pt
    }
    t.swap(index, i);                             0,5pt
  }}
```

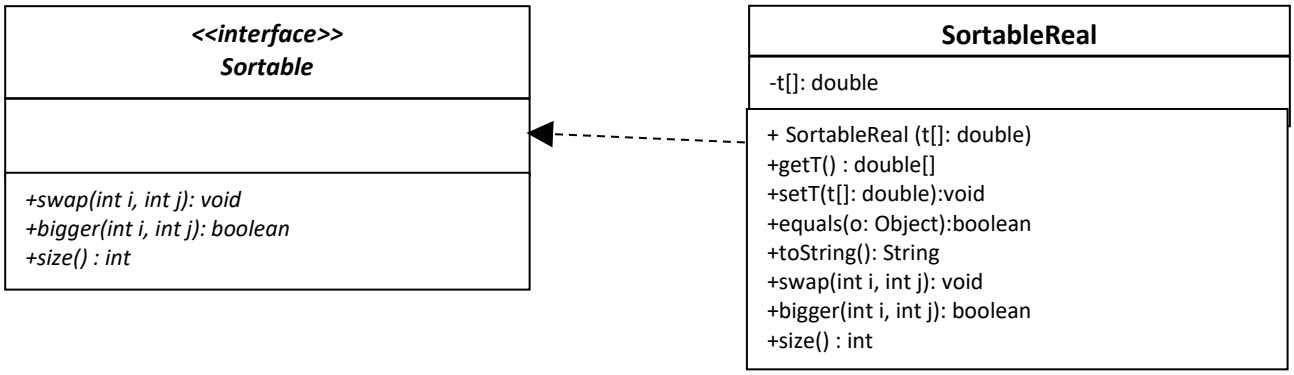
2. Ecrivez une classe SortableReal qui implémente l'interface Sortable et permet à la méthode triSelection(Sortable t) de trier des doubles (selon leur ordre naturel). **4pt**

a. Ecrire un constructeur qui prend en paramètre un tableau de double plein.

b. Ecrire un getter et un setter

c. Ecrire la redéfinition de la méthode toString() qui permet de définir une représentation textuelle d'un objet SortableReal en retournant les éléments de son tableau de double séparés pas des espaces.

d. Redéfinir la méthode equals de Object qui teste l'égalité entre deux Objets SortableReal et retourne true quand les contenus sont les même, i.e. leurs tableaux ont la même taille et des cases aux mêmes contenus.



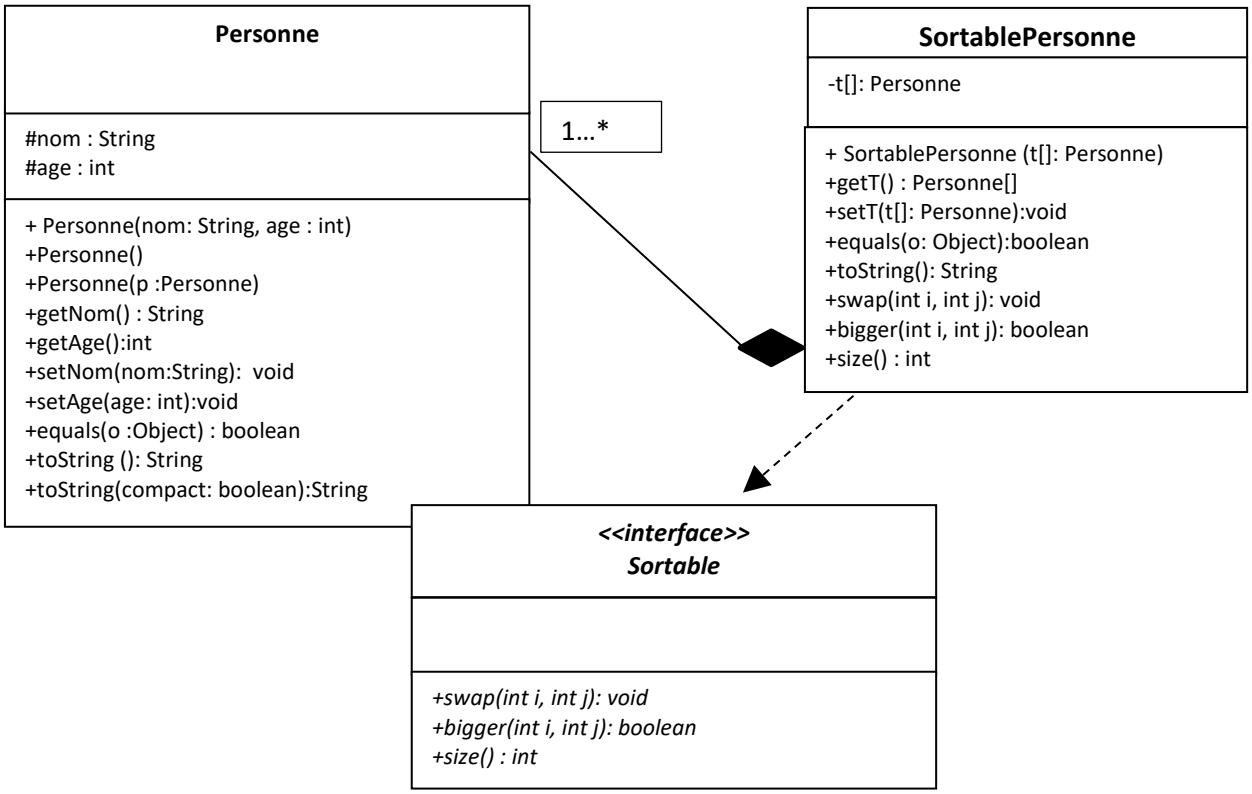

```
public class SortableReal implements Sortable { //0,5pt
private double []t; //0,25pt
public SortableReal(double []n) {//0,25pt
t= n;}
public double[] getT() {//0,25pt
return t;}
public void setT(double[] t) {//0,25pt
this.t = t;}
public String toString() {//0,5pt
String s="";
for(int i=0; i<t.length;i++)
s+=t[i]+" ";
return s;}
public boolean equals(Object o) //1pt
{
if(o==this) return true;
if(!(o instanceof SortableReal)) return false;
SortableReal et= (SortableReal) o;
if(et.size()!=this.size()) return false;

for(int i=0;i<et.size();i++)
      if(et.t[i]!=this.t[i]) return false;
return true;}
//////////////////////BONUS//////////////////////
@Override
public void swap(int i, int j) {//0,25pt
double tmp = t[j];
t[j] = t[i];
t[i] = tmp;}
@Override
public boolean bigger(int i, int j) {//0,5pt
return (t[i]>t[j]);}
@Override
public int size() {//0,25pt
return t.length;}
}
```

3. Si nous allons écrire une classe SortablePersonne qui implémente l'interface Sortable et représentant des personnes (un tableau t de personnes) et permettant à la méthode triSelection(Sortable t) de trier des personnes selon leurs âges. **2pt**

a. Ecrivez **l'implémentation du constructeur** de la classe SortablePersonne, sachant qu'un objet SortablePersonne est composé (au sens **composition**) de plusieurs objets Personne.

b. Ecrivez **l'implémentation de la méthode bigger(int i, int j),** sachant que la classe Personne est muni d'un attribut age et son getter getAge().




```
public class SortablePersonne implements Sortable {
private Personne[] t;
public SortablePersonne(Personne []n) {{//1pt
t= new Personne[n.length];
for(int i = 0; i<n.length; i++)
t[i]= new Personne(n[i]);}
@Override
public boolean bigger(int i, int j) {{//1pt
return (t[i].getAge()>t[j].getAge()); }
```

Bon travail