NOM :

PRÉNOM :

PLACE :

Correction examen de Programmation Objet

—Licence Informatique—

Vous avez droit à une unique feuille A4 comme document.
Tous les exercices sont indépendants. Même si l'on ne sait pas répondre à une question, on peut utiliser la réponse dans la suite de l'exercice. Une grande importance sera accordée à la qualité de la rédaction (lisibilité, indentation,...).

Si vous avez besoin de place, il y a une page libre à la fin du document. Merci de **ne pas écrire sur les pages d'annexes** que vous devez **détacher avant de rendre votre copie**.

Le barème est indicatif et pourra changer à la correction.

Durée : 2h00.

▸ Exercice 1. (Fonction Générique – 3 pt.)

1. Écrire une fonction `isSorted` qui prend en paramètre une liste dont les éléments appartiennent à une classe `C` et sont comparables et qui renvoie `true` ou `false` selon que la liste est triée dans l'ordre croissant ou non.

```
public static <T extends Comparable<T>> boolean isSorted(List<T> l) {
    for (int i = 1; i< l.size(); i++) {
        if (l.get(i-1).compareTo(l.get(i)) > 0) return false;
    }
    return true;
}
```

PLACE :

2. Donner un exemple d'utilisation de `isSorted` : vérifier que la liste $(2, 4, 6, 5, 7)$ n'est pas triée.

```
List<Integer> l = new ArrayList<>();
l.add(2); l.add(4); l.add(6); l.add(5); l.add(7);
System.out.println(isSorted(l));
```

▸ **Exercice 2. (Classes Génériques – 4 pt.)** Un `Ranker` est un objet qui enregistre les objets d'un certain type en leur donnant un numéro d'ordre. Les numéros d'ordre sont consécutifs et commencent à 0. Si l'on essaye d'enregistrer une deuxième fois un objet, on lui conserve son ancien numéro d'ordre. On veut donc avoir l'interface suivante :

— `r = Ranker()` : crée un `Ranker` sans aucun objet enregistré ;
— `r.rank(x)` : enregistre `x` et renvoi son numéro ;
— `r.size()` : le nombre d'objets enregistrés ;
— `r.unrank(i)` : renvoi l'objet de rang `i`.

1. Écrire une interface `Ranker` pour *ranker* des objets de type `T`.

```
public interface Ranker<T> {
    int rank(T t);
    T unrank(int i);
    int size();
}
```

2. Écrire l'entête, les attributs et le constructeur d'une implementation `RankerList` de l'interface `Ranker` stockant les objets dans une liste.

```
public class RankerList<T> implements Ranker<T>{
    private ArrayList<T> tab;
    RankerList() { tab = new ArrayList<>(); }
```

PLACE :

3. Implémenter les méthodes de la classe `RankerList` précédente.

```
    @Override
    public int rank(T t) {
        int pos = tab.indexOf(t);
        if (pos != -1) return pos;
        tab.add(t);
        return tab.size() - 1;
    }
    @Override
    public T unrank(int i) { return tab.get(i); }
    @Override
    public int size() { return tab.size(); }
```

4. Cette implémentation n'est pas très efficace car pour trouver un objet, on doit parcourir la liste. Donner une meilleure implémentation où l'on stocke le rang d'un objet dans une `HashMap`. On veut réutiliser au maximum ce qui a déjà été écrit.

```
public class RankerMap<T> extends RankerList<T> implements Ranker<T> {
    private HashMap<T, Integer> map;

    RankerMap() { super(); map = new HashMap<>(); }
    @Override
    public int rank(T t) {
        Integer pos = map.get(t);
        if (pos != null) return pos;
        map.put(t, size());
        return super.rank(t);
    }
}
```

PLACE :

▸ **Exercice 3. (Interface Graphique / Odomètre – 5 pt.)**

Un Odomètre est un outil qui mesure la distance parcouru. On veux ajouter en bas d'une fenêtre d'une application java, un tel outil qui affiche en une ligne comme

```
Position : x=423, y=583   Distance=32843.3245
```

Les deux premières valeurs sont la position de la souris, et la troisième est la distance (en pixels) parcourue par la souris dans la fenêtre. Pour calculer la distance, on fait la somme des distances entre tous les points obtenus consécutivement en écoutant les mouvements de la souris.

On se reportera en **??** pour des extraits des documentations des classes utiles.

1. Écrire les commandes permettant d'utiliser le minimum de classes nécessaires.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
```

On veut Écrire une classe `Odo` qui prend en paramètre une fenêtre (`JFrame`), qui écoute les mouvements de la souris et les affiches en bas. Attention ! On devra mesurer tous les mouvements qu'un bouton soit enfoncé ou non.

2. Écrire l'entête de la classe `Odo`.

```
public class Odo extends JPanel implements MouseMotionListener {
```

3. Écrire les attributs de la classe `Odo` ainsi que le constructeur.

```
    private JLabel infoPosition;
    private int x = 0, y = 0;
    private float d = 0;

    public Odo(JFrame fr) {
        fr.addMouseMotionListener(this);
        fr.add(this, BorderLayout.SOUTH);

        add(new JLabel("Position :"));
        infoPosition = new JLabel("out");
        add(infoPosition);
    }
```

PLACE :

4. Écrire les méthodes de la classe `Odo` qui permettent d'écouter les mouvements de la souris et de mettre à jour l'affichage.

```
public void mouseMoved(MouseEvent e) {
    int oldx = x, oldy = y; x = e.getX(); y = e.getY();
    d += Math.sqrt((x-oldx)*(x-oldx) + (y-oldy)*(y-oldy));
    infoPosition.setText("x=" + e.getX() + ", y=" + e.getY() +
                         "   Distance=" + d);
}
public void mouseDragged(MouseEvent e) { mouseMoved(e); }
```

5. Écrire un extrait de code qui ouvre une nouvelle fenêtre de dimension 800×600, s'assure que la mise en page de la fenêtre est correcte et crée un odomètre.

```
JFrame frame = new JFrame("Odometre");
frame.setLayout(new BorderLayout());

Odo odo = new Odo(frame);
frame.setSize(800, 600);
frame.setVisible(true);
```

PLACE :

▸ **Exercice 4. (Personnage – 8 pt.)** Dans cet exercice, on veux modéliser un personnage d'un jeu d'aventure. Le personnage a un certain nombre de pièce d'équipement (outils, protections, armes, clés, sac à dos), qu'il va mettre et utiliser.

Le personnage va manipuler des pièces d'équipement. Une pièce d'équipement peut subir les opérations suivantes :

— il est possible de `charger` et `decharger` avec une autre pièce d'équipement. Par exemple on peut charger quelque chose dans un sac, charger un outil avec une pile ou une arme avec un chargeur.

— on peut aussi `utiliser` une pièce d'équipement.

Les opérations `charger`, `decharger` et `utiliser` retourne un booléen qui répond si l'opération s'est bien déroulée.

1. Écrire l'interface Java `Equipement` correspondante.

```
public interface Equipement {
    public boolean charger(Equipement e);
    public boolean decharger(Equipement e);
    public boolean utiliser();
}
```

Un chargeur (pile, arme) est une pièce d'équipement particulière qui implémente en outre la méthode `getCharge` qui retourne un entier contenant le nombre de charge. Le nombre de charge est fixé à la création et l'on ne peut charger ou décharger un chargeur. Une utilisation d'un chargeur diminue le nombre de charge de 1.

2. Écrire l'entête de la classe `Chargeur`.

```
public class Chargeur implements Equipement {
```

3. Écrire les attributs et le constructeur de la classe `Chargeur`.

```
    int charge;
    public Chargeur(int charge) {
        this.charge = charge;
    }
```

4. Écrire toutes les autres méthodes de la classe `Chargeur`.

```
    @Override
    public boolean charger(Equipement e) { return false; };
    @Override
    public boolean decharger(Equipement e) { return false; };
    public int getCharge() { return charge; }
    @Override
    public boolean utiliser() {
        if (charge <= 0) return false;
        charge--;
        return true;
    }
```

On veut maintenant écrire une classe `EquipementCharge` pour modéliser tous les équipements qui peuvent accepter un chargeur. Il nous faut tenir compte que chaque équipement spécifique ne peut accepter qu'un certain type de chargeur. Pour ceci, on va se baser sur le nom. Il faut modifier l'architecture précédente pour que chaque équipement (pas seulement les chargeurs) ait un nom qui est donné à la création de l'équipement et que l'on consulte grâce à une méthode `getName`.

5. Comment doit-on modifier `Equipement` pour ceci ?

```
// On remplace l'interface par une classe abstraite:
public abstract class Equipement {
    private String name;
    public Equipement(String name) {
        this.name = name;
    }
    public String getName() { return name; }
    abstract public boolean charger(Equipement e);
    abstract public boolean decharger(Equipement e);
    abstract public boolean utiliser();
}
```

PLACE :

6. Que faut-il modifier dans `Chargeur` ?

```
public class Chargeur extends Equipement {
    public Chargeur(String name, int charge) {
        super(name);
```

7. Comment créer un équipement modélisant une “Pile AA” avec une charge de `24` ?

```
        Equipement pile = new Chargeur("pile AA", 24);
```

On revient à l'écriture de la classe `EquipementCharge`. Outre le nom de l'objet, on va passer en paramètre une chaîne de caractère décrivant le chargeur. Par exemple, on pourra créer une lampe contenant une pile comme précédemment avec :

```
new EquipementCharge("lampe torche", "pile AA");
```

8. Comment peut-on coder le fait qu'il n'y a pas de chargeur dans un `EquipementCharge` ?

```
On utilise un attribut chargeur que l'on met à null quand
il n'y a pas de chargeur.
```

9. Écrire les attributs et le constructeur

```
    private String chargeurName;
    private Chargeur chargeur;
    public EquipementCharge(String name, String chargeurName) {
        super(name);
        this.chargeurName = chargeurName;
        this.chargeur = null;
    }
```

PLACE :

10. Écrire la fonction `charger`. La charge n'est possible que si l'on a un chargeur avec le bon nom.

```
public boolean charger(Equipement e) {
    if (e instanceof Chargeur && e.getName().equals(chargeurName)) {
        this.chargeur = (Chargeur) e;
        return true;
    }
    return false;
};
```

11. Écrire la fonction `decharger`. La decharge n'est possible que si l'équipement est chargé.

```
public boolean decharger(Equipement e) {
    boolean ok = e != null;
    e = null;
    return ok;
};
```

12. Écrire la fonction `utiliser`. On ne peut utiliser l'équipement que si il est chargé avec une charge non nulle. Si c'est le cas, le chargeur est aussi utilisé.

```
public boolean utiliser() {
    if (chargeur == null) return false;
    if (chargeur.getCharge() <= 0) return false;
    chargeur.utiliser();
    return true;
}
```

PLACE :