Exercices de révisions

Exercice 1

```
package revisionIntra.surdefRedef;
public abstract class A {
   protected void m1 (String s, int i) {
      //...
   }

   protected void m1 (int s, String i) {
      //...
   }

   void m2 (int s, String i) {
      //...
   }

   double m2 (int s) {
      double d = 0;
      //...
      return d;
   }

   public String toString() {
      String s = "";
      //...
      return "";
   }

   public void m3 () {
      //...
   }
}
```

```
package revisionIntra.surdefRedef;
import java.util.ArrayList;
public class B extends A {
   public char m1 (boolean b) {
      char c = 'c';
      //...
      return c;
   }

   public double m2 (int j) {
      double d = 0;
      //...
      return d;
   }

   public String toString(String ch) {
      String s = "";
      //...
      return s;
   }

   private int m4 (ArrayList l) {
      int i = 0;
      //...
      return i;
   }

   public boolean equals (Object object) {
      boolean b = true;
      //...
      return b;
   }
}
```

```
package revisionIntra.surdefRedef;
import java.util.ArrayList;
public class C extends B {
   public void m3 () {
      //...
   }

   public int m4 (ArrayList l) {
      int i = 0;
      //...
      return i;
   }

   protected void m1 (int s, String i) {
      //...
   }

   public boolean equals (C object) {
      boolean b = true;
      //...
      return b;
   }
}
```

Dans les classes A, B et C, dites quelles sont les méthodes qui sont des surdéfintions ou surcharge et celles qui sont des redéfinitions ou masquage.

Classe A
Surdéfintions :
Redéfinitions :

Classe B:
Surdéfinitions :
Redéfinitions :

Classe C :
Surdéfinitions :
Redéfinitions :

Mélanie Lord
lord.melanie@uqam.ca

Exercice 2

Supposons les cinq classes suivantes :

```java
package revisionIntra.visibilite1;

public abstract class A {
   protected static int i = 0;
   public static boolean b = true;
   static String s = "titan";
   private static char c = 'c';

   //...
}
```

```java
package revisionIntra.visibilite1;

public class B extends A {
   //...
}
```

```java
package revisionIntra.visibilite2;

import revisionIntra.visibilite1.A;

public class C extends A {
   //...
}
```

```java
package revisionIntra.visibilite2;

import revisionIntra.visibilite1.A;

public class D {
   //...
}
```

```java
package revisionIntra.visibilite1;

public class E {
   //...
}
```

Quelles variables déclarées dans la classe A sont visibles dans :

1. La classe B : _____
2. La classe C : _____
3. La classe D : _____
4. La classe E : _____

Mélanie Lord
lord.melanie@uqam.ca

Exercice 3

Supposons les classes suivantes :

```
public abstract class Vehicule {
  protected String couleur;

  public Vehicule (String c) {
    this();
    System.out.println(quoi ());
  }

  public Vehicule () {
    System.out.println("Creation " +
        getClass().getSimpleName());
    couleur = "rouge";
  }

  public abstract String quoi();

  public String toString () {
    return "vehicule " + couleur;
  }
}
```

```
public class Voiture extends Vehicule {
  private String marque = "Volvo";

  public Voiture(String couleur) {
    super(couleur);
  }

  public Voiture() {
    couleur = "jaune";
  }

  public String quoi () {
    return "voiture";
  }

  public String toString() {
    return super.toString() + " / " +
           marque;
  }
}
```

```
public class Avion extends Vehicule {
  private int nbrMoteurs = 4;

  public Avion(String couleur) {
    super(couleur);
    System.out.println(couleur + " / "
      + getNbrMoteursExtra()
      + " moteurs en extra");
  }

  public String quoi() {
    return "avion " + couleur;
  }

  public int getNbrMoteursExtra() {
    return nbrMoteurs - 2;
  }
}
```

```
public class TestVehicule {

  public static void info(Vehicule v) {
    System.out.println(v.quoi() + " "
      + v.toString());
  }

  public static void main(String[] args) {

    Vehicule v1 = new Voiture();
    info(v1);
    Avion a1 = new Avion("bleu");
    v1 = a1;
    info(v1);

  }
}
```

Dites ce qui sera affiché à l'écran (la console) après l'exécution de la méthode `main` de la classe `TestVehicule`.

Exercice 4

a) Un palindrome est une suite de caractères qui se lit d'une façon identique tant dans un sens que dans l'autre. À titre d'exemple, les mots **radar** et **laval** sont des palindromes. À l'aide des services sur les types Pile et File, vous devez écrire une méthode Java qui déterminera si la chaîne reçue en paramètre est un palindrome ou non. Vous devez, dans la conception de votre solution, utiliser les structures suivantes : une pile **et** une file (respectivement les interfaces de Pile.java et File.java).

```
public static boolean estPalindrome(String mot) {
  //À faire
}
```

b) Écrivez une méthode `supprimer` qui prend 2 paramètres : une file et une valeur à supprimer. Cette méthode doit supprimer de la file tous les éléments ayant la valeur donnée en paramètre. Comme variables locales, **n'utilisez que des variables de type primitif**.

Par exemple, si le contenu de la file *avant* l'appel de la méthode est :

```
début   +2   -1   -3   -1   0   +2   +8   -3   fin,
```

et si la valeur fournie est -1, le contenu de la file *après* l'appel de la méthode sera

```
début   +2   -3   0   +2   +8   -3   fin.
```

Exercice 5

Définition du TDA Map :

Un map est une structure de données qui associe des clés à des valeurs. C'est donc un ensemble de paires (ou associations) clé -> valeur, où l'on dit que valeur est associée à clé.

- Les associations clé/valeur contenues dans un map ne sont pas ordonnées (n'ont pas de positions précises, ne sont pas indicées).
- Un Map ne peut pas contenir deux fois la même clé (doublons de clés interdits).
- À chaque clé, on ne peut associer qu'une seule valeur.
- À deux clés différentes, on peut associer la même valeur (il peut y avoir des doublons de valeurs, mais pas de clés).
- Une clé dans un Map ne peut pas être null.
- Une valeur dans un Map ne peut pas être null.
- Les clés sont toutes du même type K et les valeurs sont toutes du même type V.

Voici l'interface qui définit les opérations sur un TDA Map :

```
public interface TDAMap <K, V> {
   /**
    * Retourne le nombre d'associations cle/valeur dans ce TDAMap.
    * @return le nombre d'associations cle/valeur dans ce TDAMap.
    */
   public int taille ();

   /**
    * Retourne vrai si ce TDAMap est vide, faux sinon.
    * @return vrai si ce TDAMap est vide, faux sinon.
    */
   public boolean estVide();

   /**
    * Supprime toutes les associations cle/valeur de ce TDAMap. Apres l'appel
    * de cette methode, estVide() retourne vrai.
    */
   public void vider();

   /**
    * Retourne la valeur associee a la cle donnee dans ce TDAMap. Si la cle
    * donnee n'existe pas, retourne la valeur null.
    * @param cle la cle de la valeur qu'on veut obtenir.
    * @return la valeur associee a la cle donnee dans ce TDAMap.
    * @throws NullPointerException si cle est null.
    */
   public V obtenir (K cle);

   /**
    * Ajoute a ce TDAMap l'association cle -> valeur donnee en parametre,
    * si cle n'existe pas deja dans ce TDAMap.
    * @param cle la cle a laquelle correspond valeur
    * @param valeur la valeur qui correspond a cle
    * @return vrai si l'association cle/valeur a ete ajoutee, faux sinon.
    * @throws NullPointerException si cle ou valeur sont null.
    */
   public boolean ajouter(K cle, V valeur);

   /**
    * Retourne vrai si ce TDAMap contient la cle donnee, faux sinon.
    * @param cle la cle dont on teste l'existence.
    * @return vrai si ce TDAMap contient la cle donnee, faux sinon.
    * @throws NullPointerException si cle est null.
    */
   public boolean contientCle (K cle);
```

```
    /**
     * Retourne vrai si ce TDAMap contient au moins une valeur egale a la valeur donnee, faux sinon.
     * @param valeur la valeur dont on teste l'existence.
     * @return vrai si ce TDAMap contient la valeur donnee, faux sinon.
     * @throws NullPointerException si valeur est null.
     */
    public boolean contientValeur (V valeur);

    /**
     * Supprime de ce TDAMap l'association cle/valeur pour la cle donnee, si elle
     * existe, et retourne la valeur associee a la cle donnee. Si la cle donnee
     * n'existe pas, retourne null.
     * @param cle la cle de l'association cle/valeur qu'on veut retirer de ce TDAMap.
     * @return la valeur associee a la cle donnee, si cette association existait
     *         avant la suppression, null sinon.
     * @throws NullPointerException si cle est null.
     */
    public V supprimer (K cle);

    /**
     * Retourne une chaine de caracteres representant ce TDAMap.
     * La chaine doit etre forme comme suit :
     *    - une association cle/valeur par ligne
     *    - chaque association cle/valeur indiquee comme suit : cle --> valeur
     * @return
     */
    public String toString();
}
```

Il s'agit de concevoir la classe `MapArrayList<K,V>` qui implémente l'interface `TDAMap<K,V>`, en utilisant une `ArrayList<Paire<K,V>>` comme structure de données (la classe `Paire` est donnée ci-dessous).

Note sur l'implémentation :

- La classe `Paire`, ici, sert à représenter une association clé/valeur : l'item 1 de la paire représente la clé et l'item 2 de la paire représente la valeur associée à cette clé. Un `MapArrayList` est constitué d'un ensemble de paires (clé/valeur), qu'on stockera dans une ArrayList pour les besoins de cette implémentation.
- Le type `K` est le type (générique) des clés et le type `V` est le type (générique) des valeurs dans ce TDAMap.
- Votre classe doit contenir un constructeur sans argument qui construit un `MapArrayList` vide (sans aucun élément).

Classe Paire

```
package revisionIntra.tda_genericite;

import java.util.ArrayList;

public class Paire<I, J> {

    private I item1;
    private J item2;

    public Paire(I item1, J item2) {
        this.item1 = item1;
        this.item2 = item2;
    }

    public I getItem1() {
        return item1;
    }

    public J getItem2() {
        return item2;
    }

    public void setItem1(I item1) {
        this.item1 = item1;
    }
```

```
public void setItem2(J item2) {
      this.item2 = item2;
   }

   //on suppose item1 et item2 non null ici
   public String toString() {
      return "(" + item1.toString() + ", " + item2.toString() + ")";
   }

   //on suppose item1 et item2 non null ici
   public boolean equals(Object autrePair) {
      return autrePair != null
               && this.getClass().equals(autrePair.getClass())
               && this.item1.equals(((Paire) autrePair).item1)
               && this.item2.equals(((Paire) autrePair).item2);
   }
}
```

Exercice 6

Implémentez la méthode récursive suivante :

```
/**
 * Inverse les elements de la file f donnee. On suppose ici que le service
 * taille() n'existe pas dans File.
 *
 * Vous ne pouvez utiliser que des variables locales de type primitif.
 *
 * @param f la file dont on veut inverser les elements.
 * @throws NullPointerException si f est null.
 */
public static void inverser (File<Character> f) {
   //À FAIRE
}
```