Solutions exercices révision intra

Exercice 1

```java
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 () {
      //...
   }
}
```

```java
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;
   }
}
```

```java
package revisionIntra.surdefRedef;
import java.util.ArrayList;
public class C extends B {
   public void m3 () {
      //...
   }
   //nouvelle def : pas une redefinition car
   m4 non visible dans B
   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;
   }
}
```

Redéfinition
Surdéfinition
Nouvelle définition

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 : `i, b, s`
2. La classe C : `i, b`
3. La classe D : `b`
4. La classe E : `i, b, s`

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`.**Réponse** :

Instructions	**Console**
`Vehicule v1 = new Voiture();`	`Creation Voiture`
`info(v1);`	`voiture vehicule jaune / Volvo`

Mélanie Lord
lord.melanie@uqam.ca

`Avion a1 = new Avion("bleu");`	`Creation Avion`
	`avion rouge`
	`bleu / 2 moteurs en extra`

`v1 = a1;` `info(v1);`	`avion rouge vehicule rouge`

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 fonction 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).

```
//On suppose mot non null.
public static boolean estPalindrome(String mot) {
   boolean estPal = true;
   int j;
   Pile<Character> p = new PileArrayList<Character>();
   for (int i = 0 ; i < mot.length() ; i++) {
       p.empiler(mot.charAt(i));
   }

   j = 0;
   while (!p.estVide() && estPal) {
       if (p.depiler() != mot.charAt(j)) {
            estPal = false;
       }
       j++;
   }
   return estPal;
}
```

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.

```
public static void supprimer(File<Integer> f, int valeur) {
    int element;
    for (int i = f.taille(); i > 0; i--) {
        element = f.defiler();
        if (element != valeur) {
             f.enfiler(element);
        }
    }
}
```

Exercice 5

```
/**
 * Inverse les elements de la file f donnee.
 *
 * 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) {
   char c;
   if (f == null) {
      throw new NullPointerException();

   } else if (!f.estVide()){
      c = f.defiler();
      inverser(f);

      f.enfiler(c);  //au retour de la recursion, les elements sont
                     //enfiles dans l'ordre inverse.
   }
   //cas de base : f est vide
}
```