INFO-H-200 : Programmation orientée objet
TP 4 - Héritage et polymorphisme - Exercices

Professeur : Hugues Bersini
http://cs.ulb.ac.be/public/teaching/infoh200

Année académique 2015-2016

Exercice 4.1

Que donne la compilation de ces fichiers ? Si cela compile, que donne l'exécution du programme Java suivant ?

```
// Fichier A.java
public class A {
   public A() {
   }
}

// Fichier B.java
public class B extends A {
   public B() {
      super();
   }
   public String toString() {
      return(" Hello " + super.toString());
   }
}

// Fichier testAB.java
public class TestAB {
   public TestAB() {
      A a = new B();
      System.out.println(a);
   }
   public static void main(String[] args) {
      TestAB tAB = new TestAB();
   }
}
```

Exercice 4.2

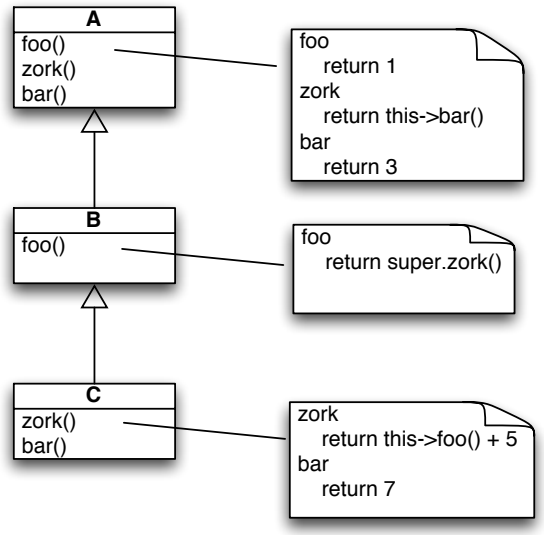
Supprimez dans le code qui suit les lignes qui provoquent une erreur et indiquez si l'erreur se produit à la compilation ou à l'exécution. Quel est le résultat de l'exécution qui s'affiche à l'écran après suppression des instructions à problème ?

```
// Fichier A.java
class A {
   public void a() {
      System.out.println("a de A") ;
   }
   public void b() {
      System.out.println("b de A") ;
   }
}

// Fichier B.java
class B extends A {
   public void b() {
      System.out.println("b de B") ;
   }
   public void c() {
      System.out.println("c de B") ;
   }
}

// Fichier Correction2
public class Correction2 {
   public static void main(String[] args) {
      A a1=new A();
      A b1=new B();
      B a2=new A();
      B b2=new B();
      a1.a() ;
      b1.a() ;
      a2.a() ;
      b2.a() ;
```

```
        a1.b() ;
        b1.b() ;
        a2.b() ;
        b2.b() ;
        a1.c() ;
        b1.c() ;
        a2.c() ;
        b2.c() ;
        ((B)a1).c() ;
        ((B)b1).c() ;
        ((B)a2).c() ;
        ((B)b2).c() ;
    }
}
```

Exercice 4.3

Etant donné le diagramme de classes ci-dessous, que renvoient les 6 instructions ci-dessous ?




1. ```
   A a = new A();
   a.zork();
   ```

2. ```
   B b = new B();
   b.zork();
   ```

3. ```
   C c = new C();
   c.zork();
   ```

4. ```
   B b = new C();
   b.zork();
   ```

5. ```
   A a = new C();
   a.zork();
   ```

6. ```
   A a = new B();
   a.zork();
   ```

Exercice 4.4

Implémenter les classes A, B et C telles que

- A est super classe de B qui est super classe de C.
- Le constructeur de la classe *x* imprime "constructeur de la classe *x*".
- Redéfinir la méthode `protected void finalize()` dans chacune des classes pour qu'elle imprime "destructeur de la classe *x*".
- Dans la fonction `main` construire un objet de la classe C, mettre sa référence à `null` et forcer le Garbage Collector avec la commande `System.gc()`.
- Que faire pour que le destructeur de chaque super classe soit également appelé ?

Exercice 4.5

Le but de cet exercice est de créer une application imprimant différentes formes géométriques dans la console. Une forme a un point d'origine, un caractère d'affichage (sa couleur), peut être déplacée dans les 4 directions cardinales et permet de savoir si elle occupe un point donné dans l'espace. Les formes se spécialisent en droite, rectangle, carré et cercle.
Une illustration est un ensemble de formes et peut être affichée dans la console (que l'on peut voir comme une grille de 50x50 points). L'illustration contiendra donc une liste de formes et pour se dessiner parcourra chaque point de chaque ligne en demandant à chaque forme si elle occupe ou non ce point de façon à afficher un blanc ou le caractère d'affichage de la forme.
Pour calculer la distance entre deux points, utilisez la distance euclidienne. Pour calculer la distance entre un point (x, y) et la droite passant par les points (x_1, y_2) et (x_2, y_2), vous pouvez utilisez la formule suivante.

$$\frac{(y_1 - y_2)x + (x_2 - x_1)y + x_1 * y_2 - x_2 * y_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$

1. Analyser, comprendre et exécuter le code fourni sur la page web des TPs.
2. Ajouter une classe `Cercle` qui hérite de `Forme` et essayer de l'utiliser.
3. Ajouter une classe `Carré` qui hérite de `Rectangle` et essayer de l'utiliser.
4. Ajouter une classe `Droite` qui hérite de `Forme` et essayer de l'utiliser.
5. Lancer des `RuntimeException` si quelque chose se passe mal (rayon négatif, ...).

Exercice 4.6

Reprendre le code *Illustrator* du TP précédent et effectuer les modifications suivantes :

— Adapter la classe `Form` pour la rendre abstraite. Rendez abstraites les méthodes nécessaires.
— Ajouter une interface `Drawable` déclarant la méthode `isOn(Point p)` et `getColor()`, que va implémenter `Form`. Ajouter une classe `AsciiNumber` implémentant `Drawable` et représentant un chiffre de 0 à 9 se dessinant sur une aire de 3 sur 4. Faites toutes les modifications nécessaires permettant d'avoir dans vos Illustrations des `AsciiNumber` et des `Form`.

Exercice 4.7 (Bonus)

— Ajouter une classe `Application` qui reçoit et parse des commandes (via `System.in`, la lecture au clavier en console), celles-ci permettent de créer des formes, de les supprimer et de les déplacer. Par exemple : `create circle 10 10 c 15` crée un cercle de centre $(10, 10)$, représenté par un `'c'` et de rayon 15.
— Ajouter un mécanisme pour que votre `Application` retienne toutes les commandes reçues et puisse les enregistrer dans un fichier.
— Ajouter un mécanisme permettant à votre `Application` d'appliquer les commandes lues dans un fichier. Ajouter éventuellement une fonction `wait(millisecond)` permettant ainsi de créer des *animations* console à partir d'un fichier.

INFO-H-200 - Programmation orientée objet
TP 4

Corrections

Exercice 4.1

```java
// Fichier A.java
class A {
  public A() {
  }
}

// Fichier B.java
class B extends A {
  public B() {
    super();
  }

  public String toString() {
    /*
     * Appel a la méthode de la classe parent: A . N'étant pas définie dans A ce sera la méthode toString définie dans la closse Object qui sera executée.
     * Cette méthode dans Object renvoie un String donnant le nom de la classe de l'objet suivi d'un '@' et d'un numéro identifiant l'objet.
     * Cette méthode renverra donc par exemple la chaine: "Hello ex1.B@38503429" si l'objet coourant est de classe B.
     */
    return(" Hello " + super.toString());
  }
}

// Fichier TestAB.java
public class TestAB {
  public TestAB() {
    A a = new B();
    /*
    * Appelle la méthode toString de l'objet a l'objet a étant une instance de la classe B c'est la méthode de la classe B qui sera exécutée
    */
    System.out.println(a);
  }

  public static void main(String[] args) {
    TestAB tAB = new TestAB();
  }
}
```

Ce programme affiche : Hello B@199a0c7c

1. La première instruction qui s'exécute est TestAB tAB = new TestAB() ; (23) qui crée un objet de type TestAB en utilisant le constructeur par défaut.
2. Ce constructeur par défaut (18) crée un nouvel objet de type B référencé par la variable a de type A (19). Cette opération est permise car B hérite de A et donc "B est un A".
3. L'instruction System.out.println(a) ; (20) est appelée. Quand println est appelé sur un objet, il affiche le résultat de l'envoi du message toString sur cet objet. La méthode toString de la classe B est donc appelée (11). Celle ci renvoie la chaine " Hello " concaténée au résultat de la méthode toString supérieure la plus proche dans la hiérarchie. Ici, la méthode appelée est celle de la classe Object qui affiche le type de l'objet (B) ainsi que son adresse (199a0c7c).

Exercice 4.2

```java
// Fichier A.java
class A {
  public void a() {
    System.out.println("a de A");
  }

  public void b() {
    System.out.println("b de A");
  }
}

// Fichier B.java
class B extends A {
  public void b() {
    System.out.println("b de B");
  }

  public void c() {
    System.out.println("c de B");
  }
}

// Fichier Correction2.java
public class Correction2 {
```

```
  public static void main(String[] args) {
    A a1 = new A();
    A b1 = new B();
    //  B a2 = new A(); // un objet de classe A ne peut être assigné à une variable de sous-classe B
    B b2 = new B();
    a1.a();
    b1.a();
    //  a2.a(); // a2 est supprimé (ligne 3)
    b2.a();
    a1.b();
    b1.b();
    //  a2.b(); // a2 est supprimé (ligne 3)
    b2.b();
    //  a1.c(); // a1 est une variable de type A, il n'y a pas de méthode c dans la classe A
    //  b1.c(); // b1 est une variable de type A, il n'y a pas de méthode c dans la classe A
    //  a2.c(); // a2 est supprimé (ligne 3)
    b2.c();
    //  ((B) a1).c(); // erreur à l'exécution: a1 contient à ce moment là une variable de type A, pas de type B
    ((B) b1).c(); // pas d'erreur: b1 contient bien à ce moment une variable de type B
    //  ((B) a2).c(); // a2 est supprimé (ligne 3)
    ((B) b2).c(); // pa de problème b2 est de type B par définition!
  }
}
```

Exercice 4.3

```
// Fichier A.java
class A {
  public int foo(){
    return 1;
  }

  public int zork(){
    return this.bar();
  }

  public int bar(){
    return 3;
  }
}

// Fichier B.java
class B extends A {
  public int foo(){
    return super.zork();
  }
}

// Fichier C.java
class C extends B {
  public int zork() {
    return this.foo()+5;
  }
  public int bar() {
    return 7;
  }
}

// Fichier TestZork.java
public class TestZork {
  public static void main(String[] args) {
    A a = new A();
    System.out.println(a.zork()); // 3

    B b = new B();
    System.out.println(b.zork()); // 3

    C c = new C();
    System.out.println(c.zork()); // 12

    b = new C();
    System.out.println(b.zork()); // 12

    a = new C();
    System.out.println(a.zork()); // 12

    a = new B();
    System.out.println(a.zork()); // 3
  }
}
```

Exercice 4.4

```java
// Fichier B.java
class B extends A{

  B() {                                          // Appelle tout d'abord constructeur de A
    System.out.println("constructeur B");
  }

  protected void finalize() throws Throwable {    // Appel explicite nécessaire si on veut que tous les destructeurs de la chaîne soient exécutés
    System.out.println("destructeur B");
    super.finalize();
  }
}

// Fichier C.java
class C extends B {

  C() {                                          // Appelle tout d'abord constructeur de B
    System.out.println("constructeur C");
  }

  protected void finalize() throws Throwable{     // Appel explicite nécessaire si on veut que tous les destructeurs de la chaîne soient exécutés
    System.out.println("destructeur C");
    super.finalize();
  }
}

// Fichier A.java
public class A {

  A (){                                          // Appelle tout d'abord constructeur de Object
    System.out.println("constructeur A");
  }

  protected void finalize() throws Throwable {    // Appel explicite nécessaire si on veut que tous les destructeurs de la chaîne soient exécutés
    System.out.println("destructeur A");
    super.finalize();
  }

  /*
   * Imprime:
   * constructeur A
   * constructeur B
   * constructeur C
   * Appel System.gc()
   * destructeur C
   * destructeur B
   * destructeur A
   */
  public static void main(String[] args) {
    C c = new C();
    c = null; // plus de référence sur l'objet de C
    System.out.println("Appel System.gc()");
    System.gc();
  }
}
```

Exercice 4.5

Voir les fichiers fournis !

Exercice 4.6

Voir les fichiers fournis !