Mis-a-niveau-JAVA

Hossein Khani

Dauphine
UNIVERSITÉ PARIS

Exercice

Soit la température T. Ecrire une classe qui affiche:

- froid si T < 8
- frais si 8 < T < 17
- bon si 17< T < 25 chaud sinon

A vous:

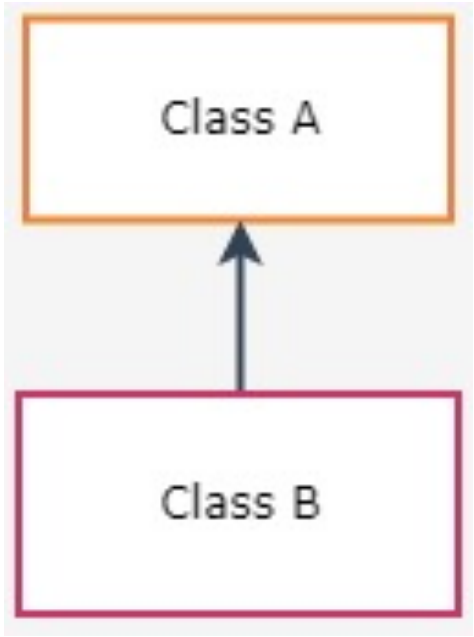
- Ecrivez une classe Rationnel qui définit les nombres rationnels. La classe a les attributs privés suivants :
- numerateur : Le numérateur;
- denominateur : Le dénominateur.

- La classe Rationnel doit disposer des constructeurs suivants :
- Rationnel();
- Rationnel(int numerateur, int denominateur);
- Rationnel(Rationnel r);

- Rationnel additionner(Rationnel r) : ajout du Rationnel en parameter.

Héritage:

- L’héritage permet d’écrire qu’une classe **B** se comporte comme une classe **A** mais avec quelques différences.



- Classe **A** s’appelle la classe **mére**.
- Classe **B** s’appelle la classe **filles**.

- Classe **B** ne comporte que ce qui change par rapport au code de **A** .

Exemple

Machine.JAVA

```java
public class Machine {

    public void start() {
        System.out.println("Machine Start.");
    }

    public void stop() {
        System.out.println("Machine Stop.");
    }

}
```

Car.JAVA

```java
public class Car extends Machine {

}
```

Main.JAVA

```java
public class Main {

    public static void main(String[] args) {
        Machine machine1 = new Machine();
        machine1.start();
        machine1.stop();
        Car car1 = new Car();
        car1.start();
        car1.stop();

    }

}
```

Output

```
Machine Start.
Machine Stop.
Machine Start.
Machine Stop.
```

Faite Attention:

En *Java*, on hérite d'une **seule** et **unique** classe.

Override:

- les méthodes “protected” et “public” de la classe mère sont accessibles par la classe enfant en deux forme:

- soit le comportement est le même : on peut/doit omettre la ré-écriture de la méthode

- soit le comportement est différent : on peut ré-écrire la méthode

Exemple: Override

```java
public class Machine {

    public void start() {
        System.out.println("Machine Start.");
    }

    public void stop() {
        System.out.println("Machine Stop.");
    }

}
```

```java
public class Car extends Machine {

    public void start() {
        System.out.println("Car Start.");
    }

    public void stop() {
        System.out.println("Car Stop.");
    }
}
```

```java
public class Car extends Machine {

    @Override
    public void start() {
        // TODO Auto-generated method stub
        super.start();
    }

    @Override
    public void stop() {
        // TODO Auto-generated method stub
        super.stop();
    }

}
```

"Eclipse shourcut"

Constructeur et Héritage

- Les constructeurs par défaut: Les Constructeurs sans paramètre.

VS

- Les constructeurs paramétrés.

- L'instanciation de l'objet fille appelle automatiquement le constructeur par défaut de la classe mère.

- Pour appeler les constructeurs paramétrés de la classe mère on utilise le mot clés super().

Exercice (Cont.)

- Créez une classe TextBox qui n'est rien de plus qu'une Box à laquelle on ajoute du texte. Implémentez un constructeur qui prend en argument quatre **int** représentant les coordonnées des deux coins du TextBox et une chaîne de caractères qui représente le texte.

- Implémentez une méthode **public** String toString()

Opérateur instanceof

- Syntaxe:

objet instanceof **nomClasse**

```
class Person {
    public String name;


    public Person(String name) {
        System.out.println("Constructor running!");
        this.name = name;
    }
}

public class Main {

    public static void main(String[] args) {

        Person person1 = new Person("Hossein");
        System.out.println(person1 instanceof Person);

    }
}
```

- Le résultat est un booléen :
  - ✓ true si x (ici person1) est de la classe (Person),
  - ✓ false sinon

Exercice (Cont.)

Redéfinissez la méthode equals de la classe Object. Vérifiez que vous codez bien une redéfiniion en utilisant l'annotation *@Override*. Vous pouvez tester votre méthode avec le code suivant.

```
BoxM b = new BoxM(0,10,10,0);
TextBox tb = new TextBox(0,10,10,0,"hello");
System.out.println(b.equals(tb) + " ?? " + tb.equals(b));
BoxM fb = new TextBox(0,10,10,0,"hello");
System.out.println(fb.equals(tb) + " ?? " + tb.equals(fb));
```

Visibilité

- Quatre modificateurs de visibilité pour les membres d'une classe.

- **Public :**
  - ✓ Visible par tout le monde.

- **private :**
  - ✓ Visible que dans la classe.

- **Sans modificateur :**
  - ✓ Visible par les classes du même package.

- **Protected :**
  - ✓ Visible par les classes héritées et celle du même package,

Faites Attention:

- ✓ Les classes en java peuvent être public ou sans modificateur.

Membres public et private

- Les members “public” sont toujours accessible par une classe fille,

- Les members “private” restent inaccessible par une classe fille (ils ont hérité, mais pas accessible. Pour y accéder il faut qu’on utilise **getter()**),

- Les members “protected” sont accessible par une classe fille.

Exercice

- Considérons cinq classes A, B, C, D et E tel que A, B et C sont dans le même package package1 et les classes D et E sont mis dans un autre package package2. Les classes B et D héritent de la classe A.

```
1 package package1;
2 public class A {
3   private int champPrive;
4   int champSansModificateur;
5   protected int champProtege;
6   public int champPublique;
7 }
```

- Complétez le tableau ci-dessous en cochant les cases pour lesquelles les variables d'instance de la classe A sont visibles.

	Classe A	Classe B	Classe C	Classe D	Classe E
champPrive					
champSansModificateur					
champProtege					
champPublique					

- Si la classe A n'était pas déclarée public, est ce que cela change la visibilité des variables?