TP9

Heritage

Personnages de Star Wars

On veut *modéliser* des affrontements entre des personnages de la Guerre des Etoiles.

```
public class ExecutableSW{
  public static void main(String[] args) {
    Personnage p1=new Ewok("Wicket",false,20,"bleu");
    System.out.println(p1);
    // Doit afficher :
    // Je suis un Ewok, je m'appelle Wicket, je suis un garçon,
    // j'ai 20 points de vie et je porte un chapeau bleu

    Personnage p2=new Wookie("Chewbaka",false,56,4);
    System.out.println(p2);
    // Doit afficher :
    // Je suis un Wookie, je m'appelle Chewbaka, je suis un garçon,
    // j'ai 56 points de vie et j'ai 4 enfants

    Personnage p3=new Jedi("Alora",true,45);
    System.out.println(p3);
    // Doit afficher :
    // Je suis un Jedi, je m'appelle Alora, je suis une fille,
    // j'ai 45 points de vie

    p2.attaque(p3,6); // p2 enlève 6 points de vie à p3
    System.out.println(p3);
    // Doit afficher :
    // Je suis un Jedi, je m'appelle Alora, je suis une fille,
    // j'ai 39 points de vie
    }
}
```

1. Réaliser un diagramme de classes permettant de gérer la hiérarchie des personnages.
2. Ecrire les classes `Personnage` et `Wookie` qui permettent de répondre à la classe `ExecutableSW`.

Refactorisation

On vous demande de factoriser au mieux ce code (éviter les codes redondants, introduire éventuellement des classes intermédiaires, ...).

```
public interface Outils {
  public void fonctionne();
}

public class Marteau implements Outils {
  private String marque ;
  public Marteau (String m) { this.marque = m; }
  public void taper() { System.out.println(" taper ");}
  public void fonctionne() {
    System.out.print("mon marteau "+ marque +" permet de ");
    taper();
    System .out. println();
  }
}

public class Ciseau implements Outils {
  private String marque ;
  public Ciseau (String m) { this.marque = m;}
  public void couper() { System.out.println (" couper ");}
  public void fonctionne() {
    System.out.print("mes ciseaux "+ marque +" permettent de ");
    couper();
    System .out. println();
  }
}

public class Secateur implements Outils {
  private String marque ;
  public Secateur ( String m) { this.marque = m;}
  public void couper() { System.out.println(" couper ");}
  public void fonctionne() {
    System.out.print("mon sécateur "+ marque +" permet de ");
    couper();
    System.out.println();
  }
}

public class ExecutableOutils {
  public static void main( String [] args) {
    Set<Outils> boiteAOutils = new HashSet<Outils >();
    boiteAOutils.add(new Marteau (" Expert "));
    boiteAOutils.add(new Marteau (" PowerGrip "));
    boiteAOutils.add(new Ciseau ("HPC"));
    boiteAOutils.add(new Ciseau (" SoftLine "));
    boiteAOutils.add(new Secateur (" Laguiole "));
    for(Outils o : boiteAOutils){
      o. fonctionne();
    }
  }
}
```

Employés de l'entreprise LAEROL

Dans l'entreprise **LAEROL**, on distingue plusieurs types d'employés :

- Les **Vendeurs**. Ils sont caractérisés par un nom, le chiffre d'affaire qu'ils réalisent par mois. Leur salaire mensuel est égal à 20 % du chiffre d'affaire qu'ils réalisent mensuellement, plus 400 euros.
- Les **Représentants**. Ils sont caractérisés par un nom, le nombre de clients démarchés par mois. Leur salaire mensuel vaut le nombre clients démarchés mensuellement multipliées par 50, plus 800 euros.
- Les **Techniciens**. Ils sont caractérisés par un nom, leur nombre d'heures de travail par mois. Leur salaire vaut leur nombre d'heures de travail mensuel multipliées par 15.
- Les **Manutentionnaires**. Ils sont caractérisés par un nom, leur nombre d'heures de travail par mois. Leur salaire vaut leur nombre d'heures de travail mensuel multipliées par 11.

1. Réaliser un diagramme de classes permettant de gérer la hiérarchie des employés.
2. Ecrire les classes `Employe` et `Représentant` qui permettent de répondre à l'exécutable suivant :

```
public class ExecutableLAEROL {
    public static void main(String[] args) {
        Employe tyrion=new Vendeur("Tyrion", 5000);
        tyrion.salaire(); // Doit afficher
        // Le vendeur Tyrion gagne 1400.0 euros
        Employe jon=new Representant("Jon", 20);
       jon.salaire(); // Doit afficher
        // Le représentant Jon gagne 1800.0 euros
        Employe joffrey=new Technicien("Joffrey", 75);
        joffrey.salaire(); // Doit afficher
        // Le technicien Joffrey gagne 1125 euros
         Employe hodor=new Manutentionnaire("Hodor", 45)
         hodor.salaire(); // Doit afficher
         // Le manut. Hodor gagne 495.0 euros
    }
}
```

Exceptions

Rappels de TD sur les Exceptions

On vous donne la classe Tableau suivante :

```
import java.util.List;
import java.util.ArrayList;
import java.lang.Math;
public class Tableau {
   private List<Integer> tab;
   public Tableau () {
       tab = new ArrayList<Integer>();
   }
   public void remplir() {
       int nb = (int)(Math.random()* 10);
       for(int i = 0; i< nb; ++i)
           tab.add((int)(Math.random()* 50));

   }
   public String toString() {return tab.toString();}
}
```

1. Dans un exécutable, utilisez la méthode min de Collections pour extraire le minimum du tableau. Traitez l'exception NoSuchElementException de façon à afficher "Il n'y a pas de minimum: le tableau est vide !" dans le cas où Collections.min lève l'exception NoSuchElementException.
2. Ajoutez une méthode get qui prend en entrée un indice (entier) et renvoie l'élément à cet indice si il existe et lève NoSuchElementException sinon. Testez cette méthode et traitez le cas d'exception dans votre exécutable.
3. Ajoutez une méthode getMax à votre classe. Vous l'implémenterez en faisant appel à la méthode max de Collections. Que se passe-t-il en cas d'exception ?

Exception personnalisée

Dans cet exercice, on va reproduire nous même le comportement de la méthode min de Collections en écrivant notre propre exception 'PasDeTelElementException'.

1. Écrivez une classe PasDeTelElementException qui étend Exception.
2. Définissez une méthode **getMin()** déterminant la valeur du minimum du tableau. Vous devrez lever une exception PasDeTelElementException si le tableau est vide.
3. Associez à cette classe une classe exécutable permettant de déterminer la valeur du minimum d'une liste en utilisant la classe Tableau et en gérant le cas où celui-ci est vide.

L'Entier plus petit plus grand qu'un entier donné

On suppose l'existence du tableau suivant : **[99, 38, 10, 49, 27, 74, 81, 60, 87]**. On désire rechercher dans ce tableau le plus petit entier plus grand qu'un entier donné et l'on veut que cet entier appartienne au tableau.

- Ainsi 90 n'appartient pas au tableau (traitement d'erreur).
- Mais 74 appartient au tableau, le plus petit entier plus grand que 74 est 81
- De meme 99 appartient au tableau mais il n'existe pas de plus petit élément plus grand que 99.

Définissez une classe Bibliotheque possédant une méthode **plusPetitPlusGrand** prenant un tableau d'entiers (ArrayList<Integer>) et un entier et répondant aux critères ci-dessus.

Animaux et zoo

1. On vous demande de définir une class **Animal** possédant deux attributs : un nom et un booléen indiquant s'il est ou non blessé.
2. Définissez une classe **Zoo** ayant comme attributs un nom et un tableau d'animaux.
3. Définissez une méthode *accueillir* permettant d'accueillir des animaux dans ce zoo.
4. Définissez une méthode **soigner** prenant un Animal en argument et permettant de soigner cet animal. Attention vouxs envisagerez les cas où l'animal n'est pas dans le zoo et celui où l'animal n'est pas blessé.
5. Définissez une classe **ExecZoo** contenant l'exécutable de vos classes.