Corrigés 6

Exercices sur l'héritage et les paquetages

1 Exercice sur l'héritage : personnes

On définit comme suit une classe de personnes :

```
public class LaDate {
    private int jour;
    private int mois;
    private int annee;
    public LaDate (int j, int m, int a){
        this.jour=j;
        this.mois = m;
        this.annee = a;
    }
    public int getJour(){
      return jour;
    }
    public int getMois(){
      return mois;
    }
    public int getAnnee(){
      return annee;
    }
    public String toString() {
        return jour+"/"+mois+"/"+annee;
    }
}
public class LaPersonne {
    protected LaDate naissance;
    protected String nom;
    public LaPersonne(String name, LaDate naiss) {
        this. nom = name;
        this.naissance = naiss;
    }
    public LaDate getNaissance(){
      return naissance;
    }
    public String getNom(){
      return nom;
```

```
    }
    public String toString() {
        return nom+" ("+naissance + ")";
    }
}
```

Pour certaines personnes, on connaît en plus le lieu de naissance, mais pas pour toutes.

Ecrire une sous-classe des personnes dont on connaît le lieu de naissance. Redéfinir la méthode `toString` pour que le lieu de naissance soit dans la chaîne renvoyée.

Ecrivez un petit programme qui utilise la nouvelle sous-classe.

```
public class LaPersonneLieuNaissance extends LaPersonne {
    protected String lieuNaissance;
    public LaPersonneLieuNaissance(String name, LaDate naiss, String lieu) {
        super(name,naiss);
        lieuNaissance = lieu;
    }
    public String getLieuNaissance(){
        return lieuNaissance;
    }
    public String toString() {
        return nom+" (né(e) le "+naissance + " à " + lieuNaissance + ")";
    }
}
public class ExoHeritage1{
    public static void main(String[] args){
        LaDate date = new LaDate(10,10,1970);
        LaPersonneLieuNaissance lpln;
        lpln = new LaPersonneLieuNaissance("Jeannot",date,"Toulon");
        System.out.println(lpln);
    }
}
```

2 Exercice sur l'héritage : employés

Une entreprise a un certain nombre d'employés. Un employé est connu par son nom, son matricule (qui l'identifie de façon unique) et son indice salairial. Le salaire est calculé en multipliant cet indice par une certaine valeur qui peut changer en cas d'augmentation générale des salaires, mais qui est la même pour tous les employés.

Question 1

Écrivez la classe des employés avec les informations utiles et des méthodes pour obtenir les caractéristiques d'un employé et pour calculer son salaire, ainsi que l'inévitable méthode `toString`.

```
class Employe{
    protected String nom;
    protected int matricule;
    protected int indiceSalarial;
    protected static int valeur = 12;
    public Employe(String n, int m, int i){
```

```
        nom = n;
        matricule = m;
        indiceSalarial = i;
    }
    public String getNom(){
        return nom;
    }
    public int getMatricule(){
        return matricule;
    }
    public int getIndiceSalarial(){
        return indiceSalarial;
    }
    public static int getValeurIndice(){
        return valeur;
    }
    public void setIndiceSalarial(int i){
        indiceSalarial = i;
    }
    public static void setValeurIndice(int v){
        valeur = v;
    }
    public int calculerSalaire(){
        return indiceSalarial *valeur;
    }
    public String toString(){
        return nom + "␣" + matricule + "␣" + indiceSalarial;
    }
}
```

A noter que l'énoncé demande que la valeur de l'indice soit une variable statique (c'est la même valeur pour tous les employés). De ce fait, les méthodes pour accéder et modifier cette variable sont également statiques.

Noter également qu'il n'y a pas de méthodes pour changer le nom et le matricule qui ne changent pas, mais qu'il y en a pour changer l'indice et sa valeur car le salaire d'un employé change au fil du temps (on espère qu'il augmente, mais ce n'est pas sûr !).

Question 2

Les commerciaux ont un salaire composé d'un fixe et d'un intéressement proportionnel à leurs ventes. Ecrivez une sous-classe des commerciaux qui contient l'information sur leurs ventes du dernier mois, une méthode pour mettre à jour cette information et redéfinissez la méthode de calcul de leurs salaires.

```
class Commercial extends Employe{
    protected int ventesDuMois;
    protected int pourcentage;
    public Commercial(String n, int m, int i, int p){
        super(n,m,i);
        pourcentage = p;
    }
    public int getVentes(){
```

```
        return ventesDuMois;
    }
    public int getPourcentage(){
        return pourcentage;
    }
    public void enregistreVentes(int i){
        ventesDuMois = i;
    }
    public int calculerSalaire(){
        return (indiceSalarial *valeur)+(ventesDuMois*pourcentage/100);
    }
    public String toString(){
        return nom + " " + matricule + " indice salarial: " + indiceSalarial
            + " pourcentage: " + pourcentage + " ventes du mois: " +
            ventesDuMois;
    }
}
```

Question 3

Il s'agit d'écrire une classe appelée `Personnel` représentant le personnel d'une entreprise. Ce personnel comporte plusieurs employés dont certains sont des commerciaux et d'autres pas. Cette classe aura les caractéristiques suivantes :

- tous les employés seront enregistrés dans un unique attribut de type `ArrayList`.
- à la construction de l'objet, il n'y aura aucun employé enregistré.
- la classe contient deux méthodes pour ajouter et retirer un employé.
- la classe contient une méthode qui calcule le montant total des salaires du personnel (la somme des salaires des employés).
- la classe contient une méthode `toString`

En ce qui concerne les types des méthodes, ils sont évidents (voir le code), sauf pour la méthode retirer. La question qui se pose est : comment repérer l'objet à retirer de l'ArrayList. Nous avons choisi ici de l'identifier par son matricule. Une autre solution acceptable est de donner l'objet lui-même.

```
import java.util.ArrayList;
public class Personnel {
    private ArrayList<Employe> liste;
    public Personnel() {
        liste = new ArrayList<Employe>();
    }
    public void ajouter(Employe emp) {
        if (emp!=null && !liste.contains(emp))
            liste.add(emp);
    }
    public void retirer(int mat) {
        int idx = 0;
        while(idx<liste.size() && liste.get(idx).getMatricule()!= mat)
            idx++;
        if (idx<liste.size())
            liste.remove(idx);
        else
            throw new RuntimeException("Employé non trouvé "+mat);
```

```
    }
    public int totalSalaires() {
        int res = 0;
        for (int i=0; i<liste.size(); i++) {
            res = res + liste.get(i).calculerSalaire();
        }
        return res;
    }
    public String toString() {
        String res = "";
        for (int i=0; i<liste.size(); i++) {
            res =res + liste.get(i).toString() + "\n";
        }
        return res;
    }
}
```

Question 4

Écrivez une nouvelle classe avec une méthode `main` qui contiennent les éléments suivants :

— la création de deux objets représentant des employés non commerciaux.
— la création de deux objets représentant des commerciaux.
— la création d'un objet instance de la classe `Personnel` qui contiennent les quatre employés créés.
— le calcul et l'affichage du salaire total du personnel.

```
public class Main {
    public static void main(String[] args) {
        Employe e1, e2;
        Commercial c1, c2;
        Personnel pers;
        e1 = new Employe("Jean␣Martin",101,75);
        e2 = new Employe("Paul␣Dubois",107,105);
        c1 = new Commercial("Raymond␣Durand",121,75,8);
        c2 = new Commercial("Gérard␣Leroy",123,75,11);
        c1.enregistreVentes(1500);
        c2.enregistreVentes(1300);
        pers = new Personnel();
        pers.ajouter(e1);
        pers.ajouter(e2);
        pers.ajouter(c1);
        pers.ajouter(c2);
        System.out.println(pers);
        System.out.println("------------");
        System.out.println("salaire␣de␣" + e1.getNom() + "␣:␣" +
                           e1.calculerSalaire());
        System.out.println("salaire␣de␣" + e2.getNom() + "␣:␣" +
                           e2.calculerSalaire());
        System.out.println("salaire␣de␣" + c1.getNom() + "␣:␣" +
                           c1.calculerSalaire());
        System.out.println("salaire␣de␣" + c2.getNom() + "␣:␣" +
                           c2.calculerSalaire());
```

```
        System.out.println("------------");
        System.out.println("total salaires : " + pers.totalSalaires());
    }
}
```

3 Exercice sur les paquetages

On donne le code de 4 classes appartenant à deux paquetages différents. Chacune des classes contient 4 méthodes avec chacune un mode de visibilité différent (public, private, protected ou rien). Dans l'une des classes, il y a une méthode main avec un appel à chaque méthode de chaque classe.

Dîtes lesquels de ces appels sont incorrects et pourquoi. Vous pouvez utiliser les numéros de ligne pour les identifier.

```
package packageA;
public class Class1A{
    public void methPublique(){
        System.out.println("methPublique de Class1A");
    }
    private void methPrivate(){
        System.out.println("methPrivate de Class1A");
    }
    protected void methProtected(){
        System.out.println("methProtected de Class1A");
    }
    void methRienDuTout(){
        System.out.println("methRienDuTout de Class1A");
    }
}
```

```
package packageA;
public class Class2A{
    public void methPublique(){
        System.out.println("methPublique de Class2A");
    }
    private void methPrivate(){
        System.out.println("methPrivate de Class2A");
    }
    protected void methProtected(){
        System.out.println("methProtected de Class2A");
    }
    void methRienDuTout(){
        System.out.println("methRienDuTout de Class2A");
    }
}
```

```
1 package packageB;
2 import packageA.*;
3 public class Class1B extends Class1A{
4     public void methPublique(){
5         System.out.println("methPublique de Class1B");
```

```
6       }
7       private void methPrivate(){
8           System.out.println("methPrivate de Class1B");
9       }
10      protected void methProtected(){
11          System.out.println("methProtected de Class1B");
12      }
13      void methRienDuTout(){
14          System.out.println("methRienDuTout de Class1B");
15      }
16      public static void main(String[] args){
17          Class1A objet1A = new Class1A();
18          Class2A objet2A = new Class2A();
19          Class1B objet1B = new Class1B();
20          Class2B objet2B = new Class2B();
21          objet1A.methPublique();
22          objet1A.methPrivate();
23          objet1A.methProtected();
24          objet1A.methRienDuTout();
25          objet2A.methPublique();
26          objet2A.methPrivate();
27          objet2A.methProtected();
28          objet2A.methRienDuTout();
29          objet1B.methPublique();
30          objet1B.methPrivate();
31          objet1B.methProtected();
32          objet1B.methRienDuTout();
33          objet2B.methPublique();
34          objet2B.methPrivate();
35          objet2B.methProtected();
36          objet2B.methRienDuTout();
37      }
38  }
```

```
package packageB;
public class Class2B{
    public void methPublique(){
        System.out.println("methPublique de Class2B");
    }
    private void methPrivate(){
        System.out.println("methPrivate de Class2B");
    }
    protected void methProtected(){
        System.out.println("methProtected de Class2B");
    }
    void methRienDuTout(){
        System.out.println("methRienDuTout de Class2B");
    }
}
```

Lorsqu'on compile ce programme, il y a 7 erreurs pour 16 appels de méthodes.

Reprenons un par un les appels.

— `objet1A.methPublique();` correct : une méthode public est utilisable partout.

— `objet1A.methPrivate();` erreur : la méthode private de la classe Class1A n'est pas utilisable dans une autre classe comme Class1B.
— `objet1A.methProtected();` correct : la méthode protected de Class1A est utilisable dans la classe Class1B parce que cette dernière est une sous-classe de Class1A. **Le compilateur considère cet appel comme une erreur, contrairement aux explications couramment données sur le mode protected**.
— `objet1A.methRienDuTout();` erreur : une méthode sans précision d'accessibilité de la classe Class1A n'est utilisable que dans le paquetage `packageA`. Or le main est dans une classe du paquetage `packageB`.
— `objet2A.methPublique();` correct : une méthode public est utilisable partout.
— `objet2A.methPrivate();` erreur : la méthode private de la classe Class2A n'est pas utilisable dans une autre classe comme Class1B.
— `objet2A.methProtected();` erreur : la méthode protected de Class2A n'est pas utilisable dans Class1B qui n'est pas dans le même package et qui n'est pas une sous-classe de Class2A.
— `objet2A.methRienDuTout();` erreur : une méthode sans précision d'accessibilité de la classe Class2A n'est utilisable que dans le paquetage `packageA`. Or le main est dans une classe du paquetage `packageB`.
— `objet1B.methPublique();` correct : une méthode public est utilisable partout.
— `objet1B.methPrivate();` correct : l'appel à la méthode private de la classe Class1B est permis dans le main qui est dans cette même classe Class1B.
— `objet1B.methProtected();` correct : l'appel à la méthode protected de la classe Class1B est permis dans le main qui est dans cette même classe Class1B.
— `objet1B.methRienDuTout();` correct : l'appel à la méthode sans précision d'accessibilité de la classe Class1B est permis dans le main qui est dans cette même classe Class1B, qui par définition est dans le même package qu'elle-même.
— `objet2B.methPublique();` correct : une méthode public est utilisable partout.
— `objet2B.methPrivate();` erreur : la méthode private de la classe Class2B n'est pas utilisable dans une autre classe comme Class1B.
— `objet2B.methProtected();` correct : l'appel à la méthode protected de la classe Class2B est permis dans le main parce que celui-ci appartient au même paquetage que la classe Class2B.
— `objet2B.methRienDuTout();` correct : l'appel à la méthode sans précision d'accessibilité de la classe Class2B est permis dans le main qui est dans cette même classe Class1B parce que ces deux classes sont dans le même paquetage.