Université de Montpellier / Polytech IG 4 / CPO2

Exercices sur les annotations et l'introspection

QUESTION 1 *Une opération de refactoring est une opération visant à améliorer le code sans changer son fonctionnement : renommer un attribut ou une méthode, séparer une méthode trop longue en plusieurs parties, etc. Définissez une annotation qui sera appliquée à un élément du code source pour décrire quelle opération de refactoring pourrait lui être appliquée. Une opération de refactoring sera annotée par :*

- *le nom de la personne qui sera en charge de l'effectuer,*
- *le nom du refactoring,*
- *un commentaire,*
- *une indication de sa présence éventuelle dans eclipse,*
- *le nombre de cas d'application dans l'élément ciblé.*

Cette annotation devra :

- *s'appliquer seulement aux classes, aux attributs et aux méthodes,*
- *pouvoir être répétée (plusieurs annotations de refactoring peuvent être posées sur un élément, voir la documentation Oracle*[1]*),*
- *être accessible à l'exécution.*

On donne cette énumération pour décrire les refactorings qui seront considérés :

```
public enum RefactoringName {
 rename, extractClass, extractMethod,
 encapsulatedField, inline, moveMethod
}
```

QUESTION 2 *Appliquer cette annotation aux bons éléments du code suivant pour indiquer qu'il est nécessaire :*

- *de renommer les attributs (dont les noms devraient commencer par des majuscules),*
- *d'encapsuler les attributs (en les rendant privés),*
- *d'extraire une classe `Adresse`.*

```
public class Personne {

public Personne (String nom,int age,int numrue,String nomrue,
    String ville,int codepostale)
{
   this.NomComplet = nom ;
   this.Age = age ;
   this.NumRue = numrue ;
   this.NomRue = nomrue ;
   this.Ville = ville ;
```

1. https ://docs.oracle.com/javase/tutorial/java/annotations/repeating.html

```
   this.CodePostale = codepostale ;
}
public String NomComplet ;
public int Age ;

public int NumRue ;
public String NomRue ;
public String Ville ;
public int CodePostale ;

public void AfficherAdresse()
{
   System.out.println("Numero de rue : "+NumRue);
   System.out.println("Nom de la rue : "+NomRue);
   System.out.println("Nom de la ville : "+Ville);
   System.out.println("CodePostale : "+CodePostale);
}
}
```

QUESTION 3 *Ecrivez un programme qui affiche les annotations de refactoring portant sur les éléments d'une classe (sur la classe, sur ses attributs et sur ses méthodes).*

QUESTION 4 *Dans cette question, nous utiliserons l'introspection pour effectuer une opération de refactoring nommée "extraction d'une interface". Nous prenons comme exemple les classes suivantes pour l'illustrer.*

```
package introspection;

import java.io.Serializable;
import java.util.Scanner;

public class Appartement implements Cloneable, Serializable{
   private String adresse; private int anneeConstruction;
    private int nbPieces; private double superficie;
    public static final double taxeFonciereAuM2 = 5;
    public Appartement() {}
    public double loyer(){return valLocBase() * coeffModerateur();}
    public double valLocBase() {return superficie*5*(1.0 + this.nbPieces / 10.0);}
    public double coeffModerateur() {return 1;}
    public String feuilleLoyer(){
        return "adresse = "+this.adresse + " val loc base = "+ this.valLocBase()
            + " coeff moderateur = "+ this.coeffModerateur()+ " loyer = " + this.loyer();
    }
    private void saisie(Scanner sc){
        System.out.println("Entrer l'adresse"); adresse = sc.next();
        System.out.println("Entrer l'annee de construction"); anneeConstruction = sc.nextInt();
        System.out.println("Entrer le nombre de pieces"); nbPieces = sc.nextInt();
        System.out.println("Entrer la superficie"); superficie = sc.nextDouble();
    }
    public static double getTaxefonciereaum2() {
        return taxeFonciereAuM2;
    }
}
```

```
public class AppartementResidence extends Appartement{
    private String services = "aucun service";
    AppartementResidence(){}
    public String getServices() {return services;}
    public void setServices(String services) {this.services = services;}
    public String feuilleLoyer(){return super.feuilleLoyer()+"\nen résidence";}
}
```

L'opération de refactoring consistera à créer le nom de l'interface en préfixant le nom de la classe par le caractère `I` *et à extraire l'interface suivante composée de ces déclarations :*

- *implémentation de l'interface qui représentera la superclasse directe (pour* `IAppartement` *c'est* `IObject`*, voir ci-dessous),*
- *implémentation des interfaces directes de la classe (pour* `IAppartement`*, c'est* `Cloneable, Serializable`*, voir ci-dessous),*
- *un bloc composé des signatures des méthodes d'instance publiques déclarées dans cette classe et qui n'apparaissent pas dans l'interface représentant sa super-classe.*

Pour les classes précédentes, cela donnerait :

```
public interface IAppartement extends IObject, Cloneable, Serializable{
  double loyer();
  double valLocBase();
  double coeffModerateur();
  String feuilleLoyer();
}

public interface IAppartementResidence extends IAppartement{
  String getServices();
  void setServices(String p);
}
```

Vous aurez besoin en particulier des méthodes suivantes du package `reflect` *ou de la classe* `Class` *(en plus de celles vues en cours) :*

- *dans la classe* `Class` *:*
  - `Class<?> getSuperclass()`
  - `Class<?>[] getInterfaces()`
  - *méthode* `Method[] getDeclaredMethods()`
    Returns an array of Method objects reflecting all the methods declared by the class represented by this Class object.
  - *méthode* `String getSimpleName()`
    Returns the simple name of the underlying class as given in the source code.
- *dans les classes* `Field, Method` *:*
  - *méthode* `int getModifiers()`
    Returns the Java language modifiers for the member or constructor represented by this Member, as an integer.
- *dans la classe* `Modifier`
  - *méthode* `static boolean isPublic(int mod)`
    Returns true if the integer argument includes the public modifier, false otherwise.
  - *méthode* `static boolean isStatic(int mod)`
    Returns true if the integer argument includes the static modifier, false otherwise.