TP n°8
Code Safe

Dans ce TP, nous allons chercher à rendre notre code beaucoup plus “safe”. Il y a plusieurs choses dans notre code qui ne le protègent pas.

Nous allons avoir l’approche suivante. Être capable de mettre en évidence ce qui pose problème, dans quel cas ce problème peut avoir lieu. Trouver une solution pour résoudre ce problème.

Exercice 1 : Protection des classes de données.

Exemple de problème avec la classe Hote :

```
public class Hote extends Personne {

  private int delaiReponse;

  public Hote(String nom, String prenom, int age, int delaiReponse) {
    super(nom, prenom, age);
    this.delaiReponse = delaiReponse;
  }

  @Override
  public void afficher() {
    super.afficher();
    System.out.print(" qui s'engage à répondre dans les " + delaiReponse + "
heures");
  }

  ...
}
```

Création d’un objet de type *Hote* et affichage :

```
Hote hote = new Hote("Bardu", "Peter", 30, 12);
...
hote.afficher();
```

Résultat ?

```
Bardu Peter (30 ans) qui s’engage à répondre dans les 12 heures
```

Impossible d’en être certain car la classe *Hote* n’est pas protégée pour cela.

Les classes de “données” sont *Personne, Voyageur, Hote, Logement, Maison, Appartement*.
Lors de la création de ces classes il a fallu faire un choix. On dit souvent, “tout private, pas de getters, pas de setters”.

C’est un choix pour ne pas faire les choses automatiquement.
Le **vrai choix** que l’on doit se poser c’est :

- Est-ce que j’autorise une instance de cette classe à évoluer ?
- Ou pas ?

Dans notre cas, ces classes ne doivent pas évoluer. C’est-à-dire, lorsqu’une instance de *Maison* est créée, elle ne doit pas changer d’état, son adresse, sa superficie, … resteront toujours les mêmes et les valeurs pour ses champs ne doivent pas pouvoir être modifiées (pas de nouvelles affectations possibles).
Maison comme les autres citées, de doivent pas avoir la possibilité d’être modifiée.

1° - Trouvez un moyen de résoudre ce problème pour chacune des classes citées.

2° - Peut-on ajouter des getters/setters ? Est-ce utile de le faire ? Est-ce “safe” ?

Exercice 2 : Protection des classes Sejour, SejourCourt, SejourLong.

Pour la construction de ces classes nous souhaitons qu’elles puissent être modifiées.
On y retrouvera dans la classe *Sejour* des méthodes de type get et de type set (à faire).

Problème numéro 1 :

```
Sejour sejour = new SejourLong(dateArrivee, nbNuits, logement, nbVoyageurs);
dateArrivee.setYear(98);
sejour.afficher();
```

Ici nous venons de modifier l’objet de type *Date* passé en paramètre.
Résultat : Ce n’est pas terrible...

1° - Reproduisez ce code pour constater par vous même le résultat.

2° - Trouvez un moyen de corriger cette erreur.

Important : Mettez des commentaires dans votre code pour expliquer vos choix d’implémentation.

Problème numéro 2 :

```
Sejour sejour = new SejourLong(dateArrivee, nbNuits, logement, nbVoyageurs);
Date date = sejour.getDateArrivee();
date.setYear(98);
sejour.afficher();
```

Ici nous venons de modifier l'objet de type Date une fois récupéré par la méthode de type get.
Résultat : Ce n'est pas terrible...

1° - Reproduisez ce code pour constater par vous même le résultat.

2° - Trouvez un moyen de corriger cette erreur.

Important : Mettez des commentaires dans votre code pour expliquer vos choix d'implémentation.

3° - Faites une conclusion sur la classe Date, sur la façon dont elle a été créée ? Ce qu'elle implémente comme interface ? Ce qu'elle a comme méthode ?

4° - Faites également une conclusion, dans votre cas précis, sur la façon d'utiliser ce type de classe.
Doit-on toujours procéder ainsi ?

5° - Pourquoi n'a t-on pas fait la même chose pour le *Logement* ?

Problème numéro 3 :

```
Sejour sejour = new SejourLong(dateArrivee, nbNuits, logement, nbVoyageurs);
sejour.setLogement(logement2);
sejour.afficher();
```

Résultat : Ce n'est pas terrible…

Sejour est une classe qui peut évoluer, c'est un choix que l'on a souhaité faire. Par conséquent, on y retrouve des méthodes de type set.

1° - Identifiez les méthodes de type set qui posent problèmes et apportez les modifications.

Important : On peut lever une exception de type *IllegalArgumentException*. Ici la JavaDoc devient vraiment très importante.