TP 5: Design Patterns (2)

Exercice 1. Seul

Soit le programme Java suivant :

```
public class Q1 {
        public URL toURL(String path) {
                URL url;
                try { url = new URL(path); }
                catch(MalformedURLException e) {
                        url = null;
                }
                return url;
        }
}
```

Lors de la conversion du String en URL, des erreurs peuvent survenir (MalformedURLException). On voudrait collecter ces erreurs dans un collecteur global à tout le programme.

Question #1: Créer un tel collecteur et modifier le code ci-dessus pour collecter les erreurs.

Question #2: Modifier la classe Math pour qu'elle soit un singleton. Comment appeler la méthode toRadians maintenant ? Argumenter sur le pour et le contre des deux implémentations.

Exercice 2. Patron de Crêpe

On veut modéliser un logiciel dédié à la création de galettes. Il existe une pléthore de recettes de galette comme la galette saucisse ou la galette caramel beurre-salé. Les galettes sont créées par un cuisinier. Le montage d'une galette se déroule en 3 étapes et consiste à faire la pâte, à y ajouter la garniture, puis à emballer le tout.

Question #3: Donner le code Java d'une opération monterGalette() en utilisant un pattern vu en cours. Faîtes un diagramme de classes UML de l'application. Identifier les « rôles » de chaque classe dans le design pattern.

Exercice 3. En visite

Dans cet exercice nous allons modéliser des expressions arithmétiques sous la forme d'un arbre binaire. Seules l'addition et la soustraction devront être considérées. Un Arbre possède un Noeud racine et un nom. Un Noeud est soit un NoeudPlus, soit un NoeudMoins, soit un NoeudValeur. NoeudPlus et NoeudMoins possède chacun un noeud droit et un noeud gauche. NoeudValeur possède une valeur (un entier). Vous modéliserez Noeud sous la forme d'une interface.

Question #4: Implémenter les classes/interfaces Java nécessaires pour pouvoir encoder des arbres binaires tels que spécifiés ci-dessus.

Question #5: Ajoutez le patron de conception Visiteur à l'arbre : ajoutez les méthodes accept(VisiteurArbre) à ce diagramme de classes et définissez l'interface VisiteurArbre et ses méthodes comme vu en cours.
Donnez le code Java de chacune des méthodes accept ajoutées.
Pourquoi les méthodes accept sont-elles nécessaires ? (hint : « double dispatch »)

Question #6: Implémenter en Java un visiteur permettant d'afficher dans la console et en notation préfixée la formule arithmétique que représente l'arbre

Question #7: Implémenter en Java un visiteur permettant de calculer la formule arithmétique que représente l'arbre.

Question #8: Implémenter en Java un visiteur permettant d'afficher dans la console sous la forme de balises XML la formule arithmétique que représente l'arbre.