UEF 7.3. Patrons De Conception

TD/TP 2

Les Patrons de Création

Exercice 1 :

Les clients d'une banque sont classés en deux catégories: ceux qui ont droit au crédit et ceux qui ne l'ont pas. Lors de la demande d'une carte de paiement, les premiers (de la première catégorie) reçoivent une carte de crédit (à débit différé sur leur compte) alors que les seconds reçoivent une carte de débit (à débit immédiat sur leur compte).

1. Quel patron de conception permet la création de la carte de paiement pour un client ? Modélisez son utilisation avec un diagramme de classes.

Exercice 2 :

Tout programme doit pouvoir rapporter les erreurs ou encore afficher des messages pour le débugger. Soit l'interface suivante :

```
public interface Trace {

// turn on and off debugging
public void setDebug( boolean debug );

// write out a debug message
public void debug( String message );

// write out an error message
public void error( String message );

}
```

1. Ecrire une classe **`SystemTrace`** qui implémente l'interface **`Trace`**. Les instances de cette classe affiche les résultats dans la fenêtre de ligne de commande.
2. Ecrire une classe **`TestTrace`** qui utilise la classe **`SystemTrace`**
3. Ecrire une classe **`FileTrace`** qui implémente l'interface **`Trace`** mais affiche les résultats dans le fichier c:/trace.log.
4. Modifiez le code de la classe **`TestTrace`** pour conserver la trace dans un fichier.
5. Cette dernière action nécessite de modifier le code à plusieurs endroits. Afin de rassembler la création des systèmes de trace en un seul endroit, écrivez la classe **`TraceFactory`** possédant la méthode de fabrique **`getTrace`**.
6. Ecrivez les sous classes **`SystemTraceFactory`** et **`FileTraceFactory`** qui créent respectivement une instance de la classe **`SystemTrace`** et **`FileTrace`**.
7. Modifiez la classe **`TestTrace`** pour laisser le choix à l'utilisateur de conserver la trace dans le fichier ou en console. Dans le cas Si l'ouverture du fichier lance une exception, la trace est redirigé directement vers la console.

8. Quel est l'avantage d'appliquer cette solution par rapport à la précédente.
9. Transformez **`TraceFactory`** en Singleton.

Ref. : https://www.javaworld.com/article/2077386/learn-java/factory-methods.html

Exercice 3 : (tiré du CI 2017/2018)

Soit le code suivant :

```
public abstract class AnimalU {
 public static Map<String, Animal> instances =
                new HashMap<String, Animal>();
 //..
 public abstract Animal instantitateAnimal();
}

public interface Animal {
// Methodes
}

public class Chat implements Animal {
// Methodes
}
public class Chien implements Animal {
// Methodes
}
public class ChatU extends AnimalU{
 public Animal instantitateAnimal() {
  if (instances.containsKey("chat")) {
   return (Animal)instances.get("chat");
  }
  else  {
    Chat c= new Chat();
    AnimalU.instances.put("chat",c) ;
    return c;}
 }
}
public class ChienU extends AnimalU{
 public Animal instantitateAnimal(){
  if (instances.containsKey("chien")) {
   return (Animal) instances.get("chien") ;
  }
  else {
    Chien c= new Chien();
    AnimalU.instances.put("chien",c) ;
    return c;}
 }
 }
}
```

1. Donnez le digramme de classe correspondant à ce code.
2. Quel patron est utilisé pour la création des animaux ?
3. Pourquoi a-t-on utilisé l'attribut instances ?
4. Proposez une nouvelle solution sans cet attribut et avec un patron de conception.
5. Donnez le nouveau code.

Exercice 4 :

On désire implémenter un moteur de jeu générique dans lequel deux éléments peuvent varier : les adversaires (on prévoit des obstacles Puzzle et NastyVillain) et les joueurs (on prévoit initialement deux types de joueurs Kitty et KungFuGuy). Cependant les divers jeux que l'on veut créer visent des publics différents : Kitty vs. Puzzle d'un côté, et KungFuGuy vs. NastyVillain de l'autre : pas question de faire jouer une instance de Kitty contre une de NastyVillain.

1. Utilisez le patron de la fabrique abstraite pour permettre la création et le lancement de différents jeux à partir moteur générique.
2. Comment ajouter un jeu Fairies vs. Gnomes dans votre moteur de jeu ?

Exercice 5 : (tiré du CI 2015/2016)

Lors d'un entretient dans une entreprise, on vous remets les parties de code des classes java présenté ci-dessous :

```
interface I{
P1 f () ;
}
class C1 implements I{
public P1 f () {  return new A1P1()}
}
class C2 implements I{
public P1 f () {  return new A2P1()}
}
class AP{
private P1 x1, x2 ;
private P2 y1,y2 ;
private A a1, a2 ;
…
public void g(boolean b){
…
if (b) {
x1=new c1.f() ;  y1=new A1P2() ; a1=new A1(x1,y1) ;
…
}
else {
x2=new c2.f() ;  y2=new A2P2() ; a2=new A2(x2,y2) ;
…
}
…
}
}
```

1. Le responsable informatique veut tester vos connaissances et vous demande :

a. Quel est le pattern appliqué par les concepteurs ?
b. Dans ce contexte, un(des) meilleur(s) pattern(s) est(sont) possible(s). Lequel(s) et pourquoi ?
c. Faire le diagramme de classe UML de votre nouvelle solution.
Ayant réussit à ce premier test, il vous rajoute certaines spécifications : la classe AP fonctionne dans une système concurrentiel, mais il ne peut y avoir qu'une seule instance.

2. a. En utilisant les patrons de conception, quelle solution proposez vous ?
   b. Donnez la partie de code permettant de mettre en œuvre votre solution.

Exercice 6 :

Soit le code suivant :

```
public class Example {
   public static void main(String[] args) {
        Cook cook = new Cook();
        Meal meal;
        BurgerMealBuilder burgerBuilder = new BurgerMealBuilder();
        HealthyMealBuilder healthyBuilder = new HealthyMealBuilder();
        cook.setMealBuilder (burgerBuilder);
        cook.constructMeal();
        meal = cook.getMeal();
        System.out.println("Order up! A " + meal);
        cook.setMealBuilder (healthyBuilder);
        cook.constructMeal();
        meal = cook.getMeal();
        System.out.println("Order up! A " + meal);
        } }
```

1. Faites le diagramme de classe UML correspondant à ce code.
2. Quel patron de création implémente-il ?
3. Donnez le code des autres classes.

Exercice 7 :

« Plane Action Hero » est un jeu d'arcade en 2D qui simule l'avancement d'un avion qui tire incessamment des balles ou des missiles, qui ramasse des bonus, qui doit éviter des pièges et qui combat des ennemis.

Un avion est caractérisé par un état (de 0 à 100, s'il a atteint 0, le joueur perd), d'une vitesse et d'une force de frappe.

Le joueur affronte les ennemis suivants :

- Des hélicoptères
- Une variété d'avions de chasse
- De grands avions sophistiqués

Chaque avion de chasse ennemi est composé des éléments suivants :

- Une carcasse dont la vitesse varie de 1 à 20
- Un lance missile dont la force est de 1 à 10
- Un bouclier dont la force varie de 0 à 10

Le joueur doit ramasser les bonus suivants :

- Un missile
- Un renforce-bouclier qui lui donne un bouclier pour quelques secondes
- Un bonus aléatoire qui augmente l'état, la vitesse et/ou la force de frappe.
- Un cloneur qui crée un clone de l'avion pendant quelques secondes

Le joueur doit éviter les pièges suivants :

- Une bombe qui affecte l'état de l'avion
- Un ralentisseur qui réduit la vitesse
- Un poison qui fait que si l'avion tire sur un ennemi, il se duplique

En utilisant les patrons de création, proposez une solution orientée objet pour la création de ce jeu. Précisez les patrons utilisés et donner le diagramme de classe.