Section	**B3**
Date	**25/03/2019**
Enseignant	**M. MALDONADO**
Matière	**DESIGN PATTERNS**

DESIGN PATTERNS CONSTRUCTEURS

Il s'agit dans ce TP d'illustrer les Design Patterns de construction abordés en cours.
Le choix porte sur les design patterns suivants :

- Singleton (exo Monopoly Banque)
- Factory (exo Monopoly Propriétés)
- Abstract Factory (exo Widgets)
- Builder (exo Pizza)
- Prototype (exo Biscuit)

Vous devrez choisir 4 Design Patterns sur les 5 proposés dans cette liste.

MONOPOLY BANQUE *(Singleton)*

On peut distinguer dans le jeu du monopoly deux catégories d'acteurs du jeu :

- Les joueurs (humains)
- Et la banque

Ces deux types de joueurs disposent de cash (opérations encaisser/payer) et d'un portefeuille de propriétés (opérations acheter / vendre).
En revanche, les joueurs sont positionnés sur une case et lancent les dés à chaque tour (ce que ne fait pas la banque), et le jeu peut comporter plusieurs joueurs, mais il n'y a qu'une seule banque.

1- Ecrire une classe JoueurHumain et une classe Banque, héritant d'une classe Joueur.
La classe Banque implémentera le design pattern Singleton.

2- Ecrire un main dans lequel :
a) Vous instancierez deux fois la classe Banque (cash 100 000), deux fois la classe Joueur (cash 50 000)
b) Vous ferez payer par la première banque 20 000 au premier joueur, par la seconde banque 10 000 au second
c) Vous afficherez les cash des 4 participants

MONOPOLY PROPRIETES *(Factory)*

Toujours dans le jeu du monopoly, il existe 3 types de propriétés (données : titre, prixAchat) :

- Les terrains
- Les gares
- Et les compagnies d'eau et d'électricité

Ces types de propriétés se distinguent par la manière de calculer le loyer.

1- Implémenter par le design pattern Factory la création des propriétés du jeu du monopoly :
- Classe abstraite ProprieteFactory
- Sous classes TerrainFactory, GareFactory, □□Factory
La méthode calculerLoyer sera spécialisée à chaque fois par un affichage :
« Je calcule le loyer d'un *<terrain/gare/compagnie d'Eau et Electricité>*

2- Ecrire un main qui crée une liste de 4 propriétés
   - Deux terrains
   - Une gare
   - Une compagnie d'Eau et Electricité et les affiche.

WIDGETS *(Abstract Factory)*

Dans le cadre du développement d'outils graphiques, on peut être amené à envisager plusieurs versions d'une bibliothèque d'objets graphiques (widgets) :

- Version Windows (*WPF* par exemple)
- Version Unix (*MOTIF* par exemple)
- Version Mac (*COCOA* par exemple)

Les composants à créer seront :

- Des boutons (classe *Button*)
- Des zones de textes (classe *TextBox*)

La seule méthode à implémenter sera draw() qui affichera :
« Je suis un <type d'objet> de la bibliothèque <nom de la bibliothèque> »

1- Implémenter par le design pattern Abstract Factory la création des factories générant les objets graphiques associés.
2- Ecrire un main créant un bouton et une zone de texte en version Windows, puis en version Unix

VOITURE *(Builder)*

La fabrication d'une voiture est un processus complexe (design pattern Builder) nécessitant trois étapes :

1- Montage du châssis
2- Montage de la carrosserie
3- Montage du moteur

Ces trois informations définissent les caractéristiques d'une voiture (champs de type string).

L'enchaînement de ces trois étapes est pris en charge par la méthode creerVoiture() de la classe VoitureBuilder (classe abstraite). En outre cette classe dispose d'une donnée interne de type Voiture et d'un accesseur getVoiture().

Cette classe est spécialisée en deux sous-classes :

- VoitureClioBuilder : châssis court, carrosserie 3 portes, moteur essence.
- VoitureEspaceBuilder : châssis long, carrosserie 5 portes, moteur diesel

La classe Ouvrier dispose :

- D'une méthode setVoitureBuilder() permettant de définir le monteur de voiture à utiliser.
- D'une méthode assemblerVoioture() qui enchaîne les étapes de création d'une voiture
- Et d'un accesseur getVoiture() permettant de récupérer le résultat

1- Ecrire la classe Voiture et ses accesseurs associés.
2- Ecrire les classes VoitureBuilder, VoitureClioBuilder et VoitureEspaceBuilder
3- Ecrire la classe Ouvrier
4- Ecrire un main qui crée un ouvrier, et lui associe
   - le VoitureClioBuilder permettant de commander une clio.
   - le VoitureEspaceBuilder permettant de commander une espace.

BISCUIT *(Propotype)*

Un biscuit est caractérisé par une méthode manger(). Dans cet exercice, il existe deux types de biscuit :

- Biscuit palmito caractérisé par :
  - Un nombre d'oreilles (entier),
  - La méthode clone(), qui reprend le nombre d'oreilles
  - La méthode manger() affichera : *Palmito, ça se croque par ses <n> oreilles !*
- Biscuit pepito, caractérisé par :
  - Un type de chocolat (au lait, noir, blanc)
  - La méthode clone(), qui reprend le type de chocolat
  - La méthode manger() affichera : *Aïe Pepito au chocolat <type de chocolat> !*

On souhaite créer un paquet de biscuits par clonage d'un biscuit initial (design pattern Prototype).

1- Ecrire une classe abstraite Biscuit implémentant l'interface système cloneable (java, PHP) ou ICloneable (C#)
Spécialiser cette classe en deux sous-classes : BiscuitPalmito et BiscuitPepito

2- Ecrire une classe MachineABiscuit dont le constructeur prend en paramètre un Biscuit. Cette classe proposera une méthode FairePaquetBiscuit() qui clonera 24 fois le biscuit de départ et retournera la liste de biscuits obtenus.

3- Ecrire un main qui crée une machine à biscuit et produit :
- un paquet de 24 palmitos
- un paquet de 24 pepito
et les affiche