© *Jean-Baptiste APOUNG KAMGA <jean-baptiste.apoung@math.u-psud.fr>*

# *Fiche de TDM : Outils pour l'ajout de fonctionnalités à un code existant (Patrons de conception ou design patterns)*

Il est question dans cette fiche de

1. comprendre et de manipuler quelques modèles (patterns) de conception
2. identifier des patterns de conception, et des codes qui en ont besoin
3. modifier un code existant en faisant usage de ces modèles

**Thème -** 1 *Motifs de conception*

**Exercice-**1 **:** Récupération

**Q-**1-1 **:** Récupérer dans mon répertoire de cours *doc/apoung/M2_IM_GL/* le fichier **design_patterns.tar.gz** et le décompresser.

```
tar -zxvf dessign_patterns.tar.gz
```

Chaque fichier est structurer de la manière suivante :

1. un ensemble de classes permettant de définir le *design pattern*
2. un programme principal *main* permettant de l'évaluer.

En guise d'exemple voici le fichier **Strategy.cpp**

```
// The Strategy design pattern.
#include <iostream>
using namespace std;

class NameStrategy {
public:
  virtual void greet() = 0;
};

class SayHi : public NameStrategy {
public:
  void greet() {
    cout << "Hi! How's it going?" << endl;
  }
};

class Ignore : public NameStrategy {
public:
  void greet() {
    cout << "(Pretend I don't see you)" << endl;
  }
};

class Admission : public NameStrategy {
public:
  void greet() {
    cout << "I'm sorry. I forgot your name." << endl;
  }
};
```

```
// The "Context" controls the strategy:
class Context {
  NameStrategy& strategy;
public:
  Context(NameStrategy& strat) : strategy(strat) {}
  void greet() { strategy.greet(); }
};

int main() {
  SayHi sayhi;
  Ignore ignore;
  Admission admission;
  Context c1(sayhi), c2(ignore), c3(admission);
  c1.greet();
  c2.greet();
  c3.greet();
} ///:~
```

Les noms de fichiers pouvant être trompeurs, en regardant le contenu de chacun de ces fichiers réaliser la tâche suivante :

1. créer trois répertoires du nom des catégories de modèles
2. répartir les fichiers dans ces répertoires en respectant leur catégorie
3. dans chaque répertoire, créé un Makefile permettant de compiler les codes
4. exécuter les codes et en se servant du cours dire si les exemples illustrent bien la définition sous-jacente du *design pattern* considéré.

---

**Thème -** 2 *Applications*

---

**Exercice-**1 **:** Transformation de code Dans la démarche de transformation (restructuration) du projet *Miramesh-Splitter*) entrepris lors des Tps précédents :

1. identifier les portions et constructions nécessitant une restructuration à l'aide des *design patterns*
2. apporter ces restructurations au code et évaluer les modifications entreprises.(*on prendra soin de faire des sauvegardes des versions ultérieures dans un dépôt à l'aide d'un outil de gestion de version de son choix.*)

Transformez vos codes (notamment ceux du projet (TPs précédents)) en utilisant les outils d'analyse de dépendance ci-dessus. Utilisez les techniques de réductions de dépendances vues en cours pour corriger les anomalies observées.