LOG2410 – TP4 (12%)
Conception à base de patrons
Été 2019

1. Objectifs

Ce laboratoire permettra aux étudiants de se familiariser avec l'implémentation des patrons de conception « Proxy », « Composite », « Visiteur », « Commande » et « Observateur ». Cette implémentation est effectuée à l'aide du logiciel MS Visual Studio et le langage C++ sera utilisé tout au long du processus de développement. Un cadriciel est fourni, qui doit être complété afin d'obtenir le résultat final souhaité.

2. Mise en contexte

L'application PolyRoboVac permet de nettoyer les planchers et tapis de façon autonome. Elle permet au robot de cartographier une pièce avant de la nettoyer, en détectant automatiquement les obstacles comme les murs, escaliers, meubles, etc. Une fois cartographiée, la disposition d'une pièce peut être sauvegardée sur un serveur et téléchargée vers le robot au besoin. Pour chaque pièce cartographiée, un ou plusieurs programmes de nettoyage peuvent être définis et exécutés par le robot.

Une description complète de l'application est disponible sur Moodle à la section 2 (Spécifications du système).

Une classe de test est fournie (`Test_TP4`) qui vous permet de vérifier que vos implémentations sont correctes.

3. Patron Proxy (20 pts)

Afin d'éviter que les différentes opérations soient effectuées directement sur le robot, il est demandé d'implémenter une classe de protection pour assurer une certaine sécurité dans l'exécution de ces opérations.

Pour mettre en œuvre ce mécanisme de sécurité, trois classes vous sont fournies : `AbstractRobot`, `Robot` et `RobotProxy` conformément à la structure du patron Proxy.

Les deux premières classes sont complètes, vous devez compléter l'implémentation des méthodes de la troisième classe.

Implémentation

On vous demande de compléter le fichier suivant :

- `RobotProxy.cpp`

afin que les tests programmés dans la méthode `Test_TP4::executeProxyTest()` s'exécutent avec succès.

Questions à répondre

3.1) Identifiez les points suivants :
   a) L'intention du patron Proxy.
   b) La structure des classes réelles qui participent au patron ainsi que leurs rôles. Réalisez un diagramme de classes avec Enterprise Architect pour l'instance du patron Proxy. Ajoutez des notes en UML pour indiquer les rôles, et exportez le tout en `pdf`.

4. Patron Composite (20 pts)

Une maison est composée de plusieurs pièces contenant chacune plusieurs obstacles. Pour permettre une représentation générique de la maison et des pièces, une adaptation du patron Composite est utilisée.

La majorité des classes sont complètes, vous devez implémenter les méthodes de la classe `CompositeNode`.

Implémentation

On vous demande de compléter le fichier suivant :

- `CompositeNode.cpp`

afin que les tests programmés dans la méthode `Test_TP4::executeCompositeTest()` s'exécutent avec succès.

Questions à répondre

4.1) Identifiez les points suivants :
   a) L'intention du patron Composite.
   b) La structure des classes réelles qui participent au patron ainsi que leurs rôles. Réalisez un diagramme de classes avec Enterprise Architect pour l'instance du patron Composite. Ajoutez des notes en UML pour indiquer les rôles, et exportez le tout en `pdf`.

4.2) Dans l'implémentation actuelle du système PolyRoboVac, quelle adaptation a été faite du patron?

4.3) Proposez au moins un test supplémentaire sur les méthodes de la classe `CompositeNode`, et expliquez quel aspect de la classe est testé, qui ne l'était pas auparavant.

5. Patron Visiteur (20 pts)

On désire ajouter des opérations supplémentaires sans modifier la hiérarchie des classes du Composite.

La classe `AbstractVisitor` est complète, vous devez compléter l'implémentation des deux classes qui en dérivent.

Implémentation

On vous demande de compléter les fichiers suivants :

- `TotalSurfaceToCleanVisitor.cpp`
- `ObstaclesDetectionVisitor.cpp`

afin que les tests programmés dans la méthode `Test_TP4::executeVisiteurTest()` s'exécutent avec succès.

Questions à répondre

5.1) Identifiez les points suivants :
   a) L'intention du patron Visiteur.
   b) La structure des classes réelles qui participent au patron ainsi que leurs rôles. Réalisez un diagramme de classes avec Enterprise Architect pour l'instance du patron Visiteur. Ajoutez des notes en UML pour indiquer les rôles, et exportez le tout en `pdf`.

5.2) Dans l'implémentation actuelle du système PolyRoboVac, le temps de nettoyage n'est pas pris en compte. Selon vous, doit-on inclure cette fonctionnalité dans la classe `TotalSurfaceToCleanVisitor` ou créer une nouvelle classe. Justifiez votre réponse.

6. Patron Commande (20 pts)

Dans le système réel du robot, les programmes de nettoyage peuvent contenir plusieurs étapes, ces étapes peuvent être vu sous la forme de commandes à exécuter par le robot. Une version minimaliste de cette idée est implémentée à l'aide des classes `AbstractCommand`, `TotalSurfaceCalculatorCommand` et `ObstacleDetectionCommand`.

On vous demande de compléter l'implémentation des deux classes concrètes.

Implémentation

On vous demande de compléter les fichiers suivants :

- `TotalSurfaceCalculatorCommand.cpp`
- `ObstacleDetectionCommand.cpp`

afin que les tests programmés dans la méthode `Test_TP4::executeCommandeTest()` s'exécutent avec succès.

Questions à répondre

6.1) Identifiez les points suivants :
   a) L'intention du patron Commande.
   b) La structure des classes réelles qui participent au patron ainsi que leurs rôles. Réalisez un diagramme de classes avec Enterprise Architect pour l'instance du patron Commande. Ajoutez des notes en UML pour indiquer les rôles, et exportez le tout en `pdf`.

6.2) Dans la structure de classes actuelle de Commande, il n'est pas prévu de faire des annulations et des réexécutions (« Undo/Redo ») de commande, que faudrait-il ajouter pour que cela puisse être possible?

7. Patron Observateur (20 pts)

Durant l'exécution des commandes sur le robot, plusieurs situations d'erreur peuvent se produire qui exigent d'avertir le propriétaire. Un mécanisme d'avertissement doit donc être mis en place pour transmettre l'information sur l'état du robot. Trois composantes principales du système ont été identifiées qui peuvent produire des erreurs, soient : le système d'aspiration, le moteur et les senseurs.

Les classes abstraites `Observable` et `AbstractObserver` sont complètent. Vous devez compléter l'implémentation de toutes les classes qui en dérivent.

Implémentation

On vous demande de compléter les fichiers suivants :

- `Motor.cpp`
- `Sensor.cpp`
- `Vacuum.cpp`
- `Tablet.cpp`

afin que les tests programmés dans la méthode `Test_TP4::executeObservateurTest()` s'exécutent avec succès.

Questions à répondre

7.1) Identifiez les points suivants :
   a) L'intention du patron Observateur.
   b) La structure des classes réelles qui participent au patron ainsi que leurs rôles. Réalisez un diagramme de classes avec Enterprise Architect pour l'instance du patron Observateur. Ajoutez des notes en UML pour indiquer les rôles, et exportez le tout en `pdf`.

7.2) Dans la structure de classes actuelle d'Observateur, il n'est pas prévu de supporter les téléphones intelligents, que faudrait-il ajouter pour que cela puisse être possible?

8. Remise

Remettre un fichier compressé (.zip) nommé `LOG2410_MatriculeA_MatriculeB_TP4.zip` qui contient les éléments suivants :

- le fichier `ReponsesAuxQuestions.pdf` avec la réponse aux questions :
  - 3.1a)
  - 4.1a)
  - 4.2)
  - 4.3)
  - 5.1a)
  - 5.2)
  - 6.1a)
  - 6.2)
  - 7.1a)
  - 7.2)
- le fichier `DiagrammeDeClasses_Proxy.pdf` de la question 3.1b);
- le fichier `DiagrammeDeClasses_Composite.pdf` de la question 4.1b);
- le fichier `DiagrammeDeClasses_Visiteur.pdf` de la question 5.1b);
- le fichier `DiagrammeDeClasses_Commande.pdf` de la question 6.1b);
- le fichier `DiagrammeDeClasses_Observateur.pdf` de la question 7.1b);
- les fichiers `C++` que vous avez modifiés pour le patron Proxy, Composite, Visiteur, Commande et Observateur.

IMPORTANT : Seuls les fichiers mentionnés explicitement dans chacun des patrons peuvent être modifiés.