LOG2410 – TP5 Hiver 2020

Conception à base de patrons II

1 - Objectifs

Ce laboratoire permettra aux étudiants de se familiariser avec l'implémentation des patrons de conception « Factory Method », « Singleton », « Mediator » et « Facade ». Cette implémentation est effectuée dans l'environnement de développement Visual Studio à l'aide du langage C++. Afin de simplifier le travail à effectuer, le cadriciel qui est fourni est complet, et le but du travail pratique consiste à identifier les patrons implémentés, et à en comprendre la structure dans le contexte du cadriciel afin de répondre à certaines questions portant sur leur utilisation concrète dans le code. Afin de vérifier votre compréhension de chaque patron, on vous demande de développer au moins un test pour chacun. Les tests qu'on vous demande de développer doivent vérifier un aspect important de l'utilisation du patron mais ne doivent pas nécessairement couvrir toute la fonctionnalité offerte par le patron. *À noter que les tests demandés peuvent s'inspirer directement du code fourni.*

2 - Patron Factory Method (25 points)

1) Identifiez L'intention et les avantages du patron Factory Method.
2) Tracer un diagramme de classes avec Enterprise Architect (ou avec un autre outil) identifiant les classes qui participent au patron Factory Method, et ajouter des notes en UML pour indiquer les rôles joués par les classes dans le contexte du patron. Exportez le tout sous la forme d'une figure.
3) Développer au moins un test qui pourrait être ajouté à la méthode de test TP5Tests::TestFactory() afin de démontrer votre compréhension du patron et des classes impliquées. Vous devez fournir le code C++ de la méthode.
4) Justifiez en quoi le test que vous proposez démontre l'utilisation du patron et confirme l'implémentation correcte du patron en fonction de comportement attendu d'une Factory Method.
5) Expliquez concrètement, en donnant un exemple, de quelle façon un développeur du module PolyCharge-Pricing pourrait tirer avantage de la présence des classes associées au patron Factory Method pour étendre le module. Vous n'avez pas à développer de code mais simplement à fournir une explication.

3 - Patron Singleton (25 points)

1) Identifiez L'intention et les avantages du patron Singleton.
2) Tracer un diagramme de classes avec Enterprise Architect (ou avec un autre outil) contenant les différentes instances du patron Singleton, et ajouter des notes en UML pour identifier les indices qui permettent de savoir qu'il s'agit bien de Singleton. Exportez le tout sous la forme d'une figure.
3) Développer au moins un test qui pourrait être ajouté à la méthode de test TP5Tests::TestSingleton() afin de démontrer votre compréhension du patron et des classes impliquées. Vous devez fournir le code C++ de la méthode.
4) Justifiez en quoi le test que vous proposez démontre l'utilisation du patron et confirme l'implémentation correcte du patron en fonction de comportement attendu d'un Singleton.
5) Expliquez concrètement, en donnant un exemple, de quelle façon un développeur qui utilise le module PolyCharge-Pricing peut tirer avantage de la présence des classes Singleton dans son utilisation du module. Vous n'avez pas à développer de code mais simplement à fournir une explication.

4 - Patron Mediator (25 points)

1) Identifiez L'intention et les avantages du patron Mediator.
2) Tracer un diagramme de classes avec Enterprise Architect (ou avec un autre outil) contenant les différentes instances et classes participant au patron Mediator, et ajouter des notes en UML pour indiquer ce qui permet de savoir qu'il s'agit bien d'un Mediator. Exportez le tout sous la forme d'une figure.
3) Développer au moins un test qui pourrait être ajouté à la méthode de test TP5Tests::TestMediator() afin de démontrer votre compréhension du patron et des classes impliquées. Vous devez fournir le code C++ de la méthode.
4) Justifiez en quoi le test que vous proposez démontre l'utilisation du patron et confirme l'implémentation correcte du patron en fonction de comportement attendu d'un Mediator.
5) Expliquez concrètement, en donnant un exemple, de quelle façon un développeur du module PolyCharge-Pricing peut tirer avantage de la présence d'instances du patron Mediator dans le module. Vous n'avez pas à développer de code mais simplement à fournir une explication.

5 - Patron Facade (25 points)

1) Identifiez L'intention et les avantages du patron Facade.
2) Tracer un diagramme de classes avec Enterprise Architect (ou avec un autre outil) illustrant la structure des classes participant au patron Facade, et ajouter des notes en UML pour indiquer ce qui permet de savoir qu'il s'agit bien d'une Facade. Exportez le tout sous la forme d'une figure.
3) Développer au moins un test qui pourrait être ajouté à la méthode de test TP5Tests::TestFacade() afin de démontrer votre compréhension du patron et des classes impliquées. Vous devez fournir le code C++ de la méthode.
4) Justifiez en quoi le test que vous proposez démontre l'utilisation du patron et confirme l'implémentation correcte du patron en fonction de comportement attendu d'une Facade.
5) Expliquez concrètement, en donnant un exemple, de quelle façon un développeur qui utilise le module PolyCharge-Pricing peut tirer avantage de la présence du patron Facade dans le module. Vous n'avez pas à développer de code mais simplement à fournir une explication.

6 - À remettre

Le TP5 est à remettre sur le site Moodle du cours au plus tard le **lundi 20 avril avant 11h55**. Vous devez remettre une archive `LOG2410_TP5_matricule1_matricule2.zip` qui contient les éléments suivants :

a) Le fichier `ReponsesAuxQuestions.pdf` avec la réponse aux questions posées, incluant les diagrammes de classe.
b) Le fichier TP5Tests.cpp implémentant les tests que vous proposez pour vérifier le bon fonctionnement de chacun des patrons.