UNIVERSITÉ DE SHERBROOKE

Département d'informatique

IFT232 — Méthodes de conception orientées objet

Plan de cours

Enseignant :

Courriel :
Site du cours :
Disponibilité :

Professeur responsable :

Horaire :

Description officielle de l'activité pédagogique[1]

Objectifs	Spécifier, concevoir et tester des composants logiciels; tester l'intégration des composants; mesurer la qualité de la conception; appliquer le processus de conception au sein d'un cadre documenté et normalisé.
Contenu	Types abstraits algébriques. Critères de conception. Instanciation, classes et métaclasses. Encapsulation. Héritage, polymorphisme et délégation. Réflexivité et introspection. Conception orientée objets: composition en classes, schémas de conception (*design patterns*) et cadres d'application (*frameworks*). Mesure de qualité de la conception. Refactorisation. Tests unitaires. Programmation par composants. Introduction à la programmation agile. Techniques de revue de conception et de code. Documentation de la conception.
Crédits	3
Organisation	3 heures d'exposé magistral par semaine 1 heure d'exercices par semaine 5 heures de travail personnel par semaine
Préalable	GIF600 ou IFT159
Concomitante	Aucune
Antérieure	Aucune
Particularités	Aucune

[1] http://www.usherbrooke.ca/fiches-cours/ift232

1 Présentation

Cette section présente les objectifs et le contenu détaillé du cours. Cette section représente la description officielle du cours telle qu'adoptée par les comités de programme du département d'informatique.

1.1 Mise en contexte

La programmation orientée objet (POO) est un paradigme informatique considéré comme une des innovations les plus importantes en génie logiciel[2]. À travers SIMULA, les origines de la POO remontent aussi loin qu'en 1967. La POO a donné par la suite naissance à d'autres innovations significatives en informatique comme le modèle MVC (1980), les schémas de conception (1991) et la refactorisation (1993). La POO repose sur 5 concepts fortement intégrés qu'elle articule en un tout cohérent : objet (encapsulation), classe (instanciation et réutilisation), message (communication), héritage (réutilisation) et méthode (comportement). Comme tout paradigme, la POO a développé sa propre culture, sa propre façon de concevoir le monde : on n'écrit pas un programme objet comme on écrit un programme fonctionnel ou procédural. Cette culture prend toute sa mesure dans les méthodologies de développement agile.

1.2 Objectifs spécifiques

L'objectif du cours IFT232 est de développer un point de vue intégré et vertical de la conception orientée objets : de la conception jusqu'au programme final, tout en assurant sa maintenance et son évolution. A la fin du cours, un étudiant ou une étudiante sera capable, en suivant une méthodologie agile, d'analyser en termes de POO un problème de taille moyenne, de concevoir une solution adaptée, de la programmer en objets et de la faire évoluer. Il ou elle devra être en outre capable de la valider et d'en juger la qualité à l'aide de critères bien définis. Il est à noter que le but du cours n'est pas d'apprendre un langage, mais bien le processus menant à résoudre un problème en utilisant les concepts fondamentaux de la conception et de la programmation par objets.

À la fin de cette activité pédagogique, l'étudiante ou l'étudiant sera capable :

1. Maîtriser un langage de spécification.
2. Maîtriser les concepts de base de la programmation par objets.
3. Maîtriser le processus de refactorisation d'un programme.
4. Maîtriser les schémas de conception.
5. Maîtriser les processus de tests.
6. Connaître diverses approches de la conception orientée objet.
7. Comprendre et appliquer les concepts principaux d'une méthode de développement agile.
8. Être capable d'évaluer la qualité de conception d'un programme.

[2] David A. Wheller, « The Most Important Software Innovation », révision du 22 juillet 2011, http://www.dwheeler.com/innovation/innovation.html

1.3 Contenu détaillé

Thème	Contenu	Heures	Objectifs
1	**Programmation par objets** • Les principes de base de la programmation par objets : transmission de messages, instanciation, classes, prototypes, encapsulation, variables d'instance, variables de classe, héritage, polymorphisme, réflexivité, métaclasses. • Application aux langages de programmation.	8	2
2	**Spécifications** • Types abstraits algébriques • Programmation par contrats • Spécification de programmes; préconditions, postconditions, invariants.	3	5, 6
3	**Tests**: • Tests unitaires.	3	5
4	**Méthodes de conception** • Méthodes de conception agiles. • Revue de conception • Revue de code	5	6, 7
5	**Documentation de la conception** • UML.	3	1
6	**Refactorisation** • Critères de conception et qualité de la conception. • Techniques de refactorisation.	3	3, 4, 8
7	**Programmation par composants** • Composants, événements, introspection, • Exemples d'application.	4	4, 6
8	**Schéma de conception** • Principes généraux • State, Singleton, Observer, Adapter, Command, Composite, Abstract Factory, Factory cyMethod, Template Method…	12	4, 8
9	**Cadre de travail:** • Principes et applications • Études de cas.	3	4, 6, 8
10	**Projet de session** • Remise, présentation, démonstration.	4	1-8

2 Organisation

Cette section présente la méthode pédagogique utilisée, le calendrier officiel du cours, la méthode d'évaluation, ainsi que l'échéancier des travaux.

2.1 Méthode pédagogique

Une semaine comprend 4 heures de présence en classe. Sur toute la session, il y aura en moyenne trois heures de cours dit théoriques (45 h) et une heure d'exercices (15 h). Des séances d'exercices de 2 heures alterneront avec les blocs de cours théoriques. Lorsque le projet sera commencé, les heures d'exercices seront consacrées au projet pour permettre aux équipes d'organiser leur travail et de se rencontrer plus facilement. Le cours a une approche pratique et met l'accent sur l'implémentation des solutions conçues. Tout le matériel pédagogique sera placé sur Moodle.

2.2 Calendrier du cours

	Semaine du	**Thème**	**Références**	**Exercices, évaluations, projet**
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				

2.3 Évaluation

Devoirs (3) :
Tests (3) :
Projet :
Examen final:

2.3.1 Qualité du français et de la présentation

Conformément aux articles 36, 37 et 38 du règlement facultaire d'évaluation des apprentissages[3] l'enseignant peut retourner à l'étudiante ou à l'étudiant tout travail non conforme aux exigences quant à la qualité de la langue et aux normes de présentation.

2.3.2 Plagiat

Un document dont le texte et la structure se rapportent à des textes intégraux tirés d'un livre, d'une publication scientifique ou même d'un site Internet, doit être référencé adéquatement. Lors de la correction de tout travail individuel ou de groupe une attention spéciale sera portée au plagiat, défini dans le Règlement des études comme « le fait, dans une activité pédagogique évaluée, de faire passer indûment pour siens des passages ou des idées tirés de l'œuvre d'autrui. ». Le cas échéant, le plagiat est un délit qui contrevient à l'article 8.1.2 du Règlement des

[3] http://www.usherbrooke.ca/sciences/intranet/informations-academiques/reglement-d-evaluation/

études[4] : « tout acte ou manœuvre visant à tromper quant au rendement scolaire ou quant à la réussite d'une exigence relative à une activité pédagogique. » À titre de sanction disciplinaire, les mesures suivantes peuvent être imposées : a) l'obligation de reprendre un travail, un examen ou une activité pédagogique et b) l'attribution de la note E ou de la note 0 pour un travail, un examen ou une activité évaluée. Tout travail suspecté de plagiat sera référé au Secrétaire de la Faculté des sciences.

2.4 Directives particulières pour les travaux pratiques

Les travaux pratiques sont effectués par équipe de deux étudiant(e)s. La qualité du français et de la présentation sera aussi considérée dans le résultat du travail. Les travaux pratiques sont réalisés en Java. L'environnement de programmation intégré Éclipse (http://www.eclipse.org/) est disponible au département d'informatique. L'environnement de tests utilisé est JUnit (http://www.junit.org/). Les travaux pratiques peuvent être réalisés sous tout autre environnement, mais le code et les tests devront fonctionner sous Eclipse et Java SE 6 et JUnit 4 au moment de la remise du travail. Les énoncés des travaux seront rendus disponibles sur Moodle. La remise du travail (code, tests et autres documents) sera effectuée dans Moodle. La remise du travail s'effectue au jour et à l'heure exigés. Le non respect de la date de remise entraîne une pénalité de 25% de la note par jour de retard. Cela signifie qu'il faut toujours viser à terminer son travail de programmation au moins 24 heures avant la date de remise pour tenir compte des pannes possibles. Ceci est un conseil qui vaut son pesant de points.

2.5 Directives particulières pour le projet

Le projet se fait par équipe de 6 à 10 personnes. Une équipe doit idéalement comporter un nombre pair de membres. Chaque équipe définit son propre projet. Les projets peuvent être réalisés dans le langage à objets et l'environnement de développement de votre choix. A la mi-session, le professeur doit approuver la composition de l'équipe, la définition du projet et le langage utilisé. Pour la remise du projet sont demandés une présentation accompagnée d'une démonstration, le code source et les tests unitaires, un rapport pour le groupe, un rapport individuel. La remise de tous les documents et fichiers liés au projet se fait dans Moodle.

3 Matériel nécessaire pour le cours

Les manuels et articles principaux sur lesquels est basé le cours sont [2], [3], [4], [6], [7], [8]. Ils ne sont pas obligatoires. Le langage utilisé dans le cours est Java. Les projets peuvent être réalisés dans le langage à objets le plus approprié à la nature du projet, e.g. existence d'une librairie donnée.

4 Références

1. J. Bloch, *Effective Java Programming* Language Guide, 2e édition, Addison-Wesley, 2008.
2. M. Fowler et K. Scott, *UML distilled, A Brief Guide to the Standard Object Oriented Modeling Language*, Addison-Wesley, 2003.
3. K. Beck, *Extreme Programming Explained*, 2e édition, Addison-Wesley, 2004
4. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.
5. E. Freeman, E. Freeman, K. Sierra, B. Bates, *Head First Design Patterns***,** O'Reilly, 2004.
6. Sun Microsystems, *JavaBean*s, v 1.01, 1997.
7. E. Gamma and K. Beck, *JUnitTest Infected: Programmers Love Writing Tests*, Java Report, 1998, v.3, n. 7.
8. M. Fowler, et al., *Refactoring, Improving the design of existing code*, Addison-Wiley, 1999.
9. R. Mitchell, J. McKim, B. Meyers, *Design by Contracts*, by Example, Addison-Wesley, 2001.
10. E. Gamma, K. Beck, *JUnit, a Cook's Tour*, http://junit.sourceforge.net/doc/cookstour/cookstour.htm

[4] http://www.usherbrooke.ca/programmes/etude