CÉGEP DE MATANE

TECHNIQUES DE L'INFORMATIQUE

(420.B0)

Automne 2026

PLAN D'ÉTUDES

Pondération : 2 - 3 - 2

Unités : 2,33

75 heures

420-345-MT

Programmation et architecture

Préalables : PA 420-285-MT Programmation orientée objet

Adopté en assemblée départementale
le 20 août 2026

Nadine Giasson St-Amand
O-129
#2127
stamandnadine@cgmatane.qc.ca

Description du cours

L'étudiant améliore l'architecture de programmes et utilise une terminologie interdisciplinaire afin de communiquer plus facilement avec ses pairs. Il développe des applications en exploitant les patrons d'architecture applicative et de conception. Il développe des applications en utilisant des méthodes agiles et en appliquant les principes de l'intégration continue. Il utilise les logiciels de versionnage et de partage de code source. Il écrit des programmes avec une attention particulière pour la maintenabilité du code source. Il s'initie à la maintenance de code patrimonial.

Objectifs de formation fondamentale du projet éducatif

- La communication et la maîtrise de la langue
- La pensée autonome
- Le sens des responsabilités

Objectifs et standards

OBJECTIF	STANDARD
ÉNONCÉ DE LA COMPÉTENCE	**CONTEXTE DE RÉALISATION**
00Q6 : Exploiter les principes de la programmation orientée objet	● À partir d'un problème. ● À l'aide de règles de nomenclature et de codage.
ÉLÉMENTS DE COMPÉTENCE	**CRITÈRES DE PERFORMANCE**
1 Analyser le problème.	1.1 Décomposition du problème en fonction des exigences de l'approche orientée objet. 1.2 Détermination correcte des données d'entrée, des données de sortie et de la nature des traitements. 1.3 Détermination juste des classes à modéliser. 1.4 Détermination correcte des algorithmes à produire.
2 Modéliser les classes.	2.1 Détermination correcte des attributs et des méthodes des classes. 2.2 Application judicieuse des principes d'encapsulation et d'héritage. 2.3 Représentation graphique correcte des classes et de leurs relations. 2.4 Respect des règles de nomenclature.

3	Produire les algorithmes pour les méthodes.	3.1	Détermination adéquate des opérations nécessaires pour chaque méthode.
		3.2	Détermination correcte d'une séquence logique des opérations.
		3.3	Vérification appropriée du fonctionnement des algorithmes.
		3.4	Représentation correcte des algorithmes.
4	Générer l'interface graphique.	4.1	Choix approprié des éléments graphiques pour l'affichage et la saisie.
		4.2	Positionnement correct des éléments graphiques.
		4.3	Paramétrage correct des éléments graphiques.
5	Programmer des classes.	5.1	Choix approprié des instructions, des types de données élémentaires et des structures de données.
		5.2	Organisation logique des instructions.
		5.3	Programmation correcte des messages à afficher à l'utilisatrice ou à l'utilisateur.
		5.4	Intégration correcte des classes dans le programme.
		5.5	Fonctionnement correct du programme.
		5.6	Respect de la syntaxe du langage.
		5.7	Respect des règles de codage.
6	Documenter la programmation.	6.1	Notation claire de commentaires dans le code informatique.
		6.2	Notation claire de la documentation d'aide à la programmation.
		6.3	Utilisation appropriée des générateurs de documentation.
7	Appliquer la procédure liée à la gestion des versions de programmes.	7.1	Configuration correcte du système de gestion de versions.
		7.2	Soumission méthodique du code modifié.
		7.3	Gestion judicieuse des branches et des conflits.

OBJECTIF	STANDARD
ÉNONCÉ DE LA COMPÉTENCE	**CONTEXTE DE RÉALISATION**
00SE : Interagir dans un contexte professionnel	● Pour des formes d'organisation du travail variées. ● À l'aide de normes, de méthodes et de bonnes pratiques en matière de développement d'applications et de gestion de réseaux. ● À l'aide de lois, de codes d'éthique et de politiques d'entreprise.
ÉLÉMENTS DE COMPÉTENCE	**CRITÈRES DE PERFORMANCE**

2	Travailler au sein d'une équipe multidisciplinaire.	2.1	Manifestation d'attitudes et de comportements de respect, d'ouverture d'esprit et de collaboration.
		2.2	Communication efficace avec les membres de l'équipe.
		2.3	Exécution correcte des tâches confiées.
		2.4	Respect des règles de fonctionnement en équipe.
		2.5	Respect de la culture d'entreprise.
		2.6	Respect des normes, des méthodes ou des bonnes pratiques en matière de développement d'applications et de gestion de réseaux.
		2.7	Respect des limites d'intervention professionnelle et de l'expertise des membres d'autres professions.
		2.8	Respect des échéances.

Activités d'apprentissage

- Appliquer les techniques de polymorphisme au traitement d'objets hétérogènes.
- Programmer un logiciel selon un patron d'architecture applicative en procédant au versionnage continu du code source.
- Programmer un logiciel qui exploite plusieurs patrons de conception.
- Identifier un couplage nuisible dans un code source.
- Maintenir un programme en suivant des méthodes agiles pour y intégrer un patron de conception.
- Expérimenter le processus de l'intégration continue.

Architecture

Principes d'architecture :

- divisions de responsabilités, cohésion et couplage, patterns & anti-patterns
- trucs mnémotechniques : KISS, SOLID, DRY, GRASP
- qualité d'une architecture : compromis, robustesse, portabilité, maintenabilité, optimisation, sécurité

Patrons d'architecture applicative (software architecture pattern) :

- modèle
- vue
- contrôleur
- MVC & DAO combinés
- observeur/observé
- autres architectures

Patrons de conception (design pattern) :

- singleton, commande, décorateur, gabarit (template), monteur (builder), etc

Programmation orientée objet

Approfondissement des concepts et techniques objets

- Types natifs, classes définies par le programmeur, classes importées
- Objets, référence, visibilité et cycle de vie
- Héritage, interface contractuelle, surcharge, polymorphismes
- Agrégation et composition, génériques, collections

Technologies d'interfaces graphiques, programmation des événements

Diagrammes : de classe, d'objets, de séquence, de collaboration

Processus

Processus agile :

- travail d'équipe, itérations & 'sprint review', kanban & backlog, mêlée, documentation 'lean'
- intégration continue : versionnement (commit), tickets (issues), livraisons (release)
- qualité : revue de code, réingénierie de code, tests, acceptance

Documentation :

- prototypes, cas d'utilisation, diagrammes, readme.md, guides, démonstrations
- documentation intégrée : code sémantique, journalisation, commit commenté relié au ticket, annotation

Calendrier des activités pédagogiques et d'évaluation

	Théorie	Activités	Évaluations
1	Architecture - Les Modèles	Exploiter les modèles et le polymorphisme - 5%	Laboratoires Mirador (25%)
2	Architecture - Les Vues	Afficher des données dans une vue encapsulée - 10%	
3	Architecture - Les DAO	Servir des données à l'application - 10%	
4	Principes d'architecture	Revue de code	
5	Architecture - Les Contrôleurs	Gérer des événements d'édition - 20%	Projet Éditeur (25%)
6	Observeur & observé	Versionner en équipe	
7	Type Interface et la Réutilisabilité	Enregistrer les objets dans un fichier - 5%	
8	Maintenabilité & interopérabilité	Identifier et corriger des couplages	
9	Design pattern Commande	Exploiter une pile de commandes - 5%	Laboratoires Patterns (20%)
10	Design pattern Template	Filtrer des listes de données - 5%	
11	Design pattern Monteur (Builder)	Dériver un composant graphique personnalisable - 5%	
12	Design pattern Décorateur	Superposer des traitements à des objets - 5%	
13	Génie logiciel et processus	Preuve de concept avec une API	Préparations BootCamp (15%)
14	Rétroingénierie et réingénierie	Adapter un code existant	
15	Qualité	Tester l'application	
	Semaine des examens	Journée BootCamp	Journée BootCamp (15%)

** L'ordre des semaines peut varier pour le bon fonctionnement des projets.*

Méthodes et moyens pédagogiques

L'apprentissage s'articule autour de projets intégrateurs qui servent à mettre en pratique les différentes techniques et notions du cours. Chaque projet intégrateur est divisé en étapes qui sont chacune d'une durée de une semaine. Les **méthodes** et **moyens** mis en place pour guider cet apprentissage sont :

- Un **Calendrier** des **Rencontres** hebdomadaires à présence obligatoire pour présenter la nouvelle matière et à réaliser les laboratoires
- Un **Espace virtuel** de liaison : soit un espace de discussion et un outil de vidéoconférence qui servent à faire des suivis et à fournir l'aide demandée entre les cours.
- Des documents mis à la disposition des participants classés par Modules dans un **Site**.

À chaque semaine :

- *En support aux apprentissages sont d'abord présentés des **diaporamas introductoires**, des **articles**, des **vidéos**, des **exemples versionnés**, des **échantillons de code**, des **procédures**.*
- *Ensuite, le participant est invité à réaliser plusieurs **exercices** formatifs sous forme de travail dirigé Ici le professeur joue le rôle de l'architecte.*
- *Enfin, le participant intègre la nouvelle matière ou technique en l'appliquant à son **laboratoire, devoir ou projet intégrateur**. Des Tickets de suivi Git guident l'étudiant dans les tâches. Ici le professeur joue le rôle du chargé de projet.*

Pour sa part, le participant devra adopter les pratiques suivantes :

=> L'étudiant devra fournir un effort personnel pour réaliser ses travaux de manière autonome. Il devra surtout prendre connaissance des notes de cours et expérimenter les échantillons et les exercices avant de poser des questions. Il devra suivre les étapes Recherche - Test - Question.

=> L'étudiant devra versionner son code et son travail d'une manière régulière, selon les instructions du guide de versionnement, et participer au processus d'**intégration continue**. La rétention de code n'est pas permise sur le poste de travail ou dans une branche.

=> L'étudiant devra rendre compte de son avancement dans les projets, selon le format de la mêlée agile, il devra s'assurer de compléter ses parties de projet en temps utile pour ne pas retarder son équipe, et dans la même optique il devra signaler toute difficulté ou retard en temps utile. C'est pourquoi il ne devra pas laisser traîner aucune incompréhension ni aucun élément technique dysfonctionnel

=> L'étudiant devra expliquer et même ajuster son code à la demande pour valider chaque évaluation. Si l'étudiant est incapable d'expliquer le fonctionnement d'un code inscrit dans son projet ou si il est incapable de le modifier au besoin, ce code ne sera pas considéré comme le sien et tombera sous la règle anti-plagiat.

Pour participer à ce cours, l'étudiant doit se procurer du matériel et des logiciels :

=> L'étudiant devra se procurer le matériel nécessaire :

- Ordinateur portable, écouteurs, micro & clé USB
- Partition linux & logiciels gratuits indiqués pour chaque projet

=> L'étudiant devra maintenir la configuration d'ordinateur indiquée en ce qui a trait aux logiciels requis ou interdits pour chaque chapitre du cours et utiliser les outils recommandés afin d'optimiser son apprentissage.

En résumé les exposés théoriques et les démonstrations seront suivis d'exercices pratiques effectués en atelier. De plus, tout au long de la session, afin de permettre à l'étudiant de se situer dans l'atteinte des objectifs, des rétroactions formatives s'effectueront à l'aide d'exercices et de projets dirigés supervisés.

Sommaire des activités d'évaluation

Activités d'évaluation	Moments	Pondérations
Laboratoires Mirador	**Semaine 1 à 4**	**25%**
Projet Éditeur	**Semaine 5 à 8**	**25%**
Laboratoires Patterns	**Semaine 9 à 12**	**20%**
BootCamp	**Semaine 13 à 15**	**30%**

Tous les travaux se réaliseront en utilisant l'approche de l'intégration continue, c'est-à-dire en versionnant le code et les autres fichiers du projet au fur et à mesure de l'avancement. La note des projets tiendra compte de la contribution individuelle et de son étalement tout au long du projet.

- Le code devra suivre les Normes de programmation fournies dans le cours, il devra notamment être sémantique et conforme au lexique du **dossier fonctionnel***. On doit respecter ces règles tout au long du processus (dans chaque commit) et ne pas tout renommer le code d'un seul coup à la fin.
- Des remises intermédiaires datées sous forme de TAG sont prévues dans chaque projet et devront être respectées par toute l'équipe afin d'assurer le déroulement correct du projet.
- Toutes les discussions relatives aux projets ont lieu seulement dans les espaces prévus à cette fin pour concentrer l'information et respecter la confidentialité et la traçabilité d'un projet informatique.

Les laboratoires se réalisent d'abord en classe et se complètent en devoir à la maison. Cependant la présence à chaque étape de chaque laboratoire est obligatoire pour recevoir les points associés. Toute absence non-justifiée à un laboratoire entraîne la note de 0.

*** Dossier fonctionnel :** Le dossier fonctionnel est soit l'énoncé sommaire fourni par l'enseignant (dans le cas d'un projet imposé), soit le dossier fonctionnel formel fourni par l'étudiant (dans le cas des projets au choix) et celui-ci est rédigé en langue française.

*** L'usage de l'IA générative pour rédiger le code des activités et des évaluations doit s'inscrire dans les limites de l'intégrité intellectuelle définies en classe.*

Description de l'épreuve finale de cours

Le BootCamp (30%) est l'épreuve finale de cours.

Le participant démontre sa compétence à organiser le code selon une architecture-cible, à exploiter un design pattern pertinent dans la réalisation de fonctionnalités pour l'entreprise. Tout au long du projet il documente son processus en utilisant les commentaires de commit, en renseignant le fichier README du projet et en journalisant les processus critiques.

Médiagraphie et autres ressources pertinentes

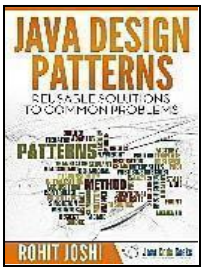
DESIGN PATTERNS principes et descriptions

https://stg-tud.github.io/eise/ (livre gratuit)
https://books.google.ca/books?id=6oHuKQe3TjQC&printsec=frontcover (aperçu gratuit)

Banques en ligne de design patterns

https://refactoring.guru/design-patterns
https://sourcemaking.com/design_patterns
https://www.dofactory.com/net/design-patterns
Liste Wikipédia en français et en anglais
https://wiki.c2.com/?DesignPatterns
https://www.tutorialspoint.com/design_pattern/index.htm
https://github.com/search?q=design+patterns&type=repositories

DESIGN PATTERNS dans les LANGAGES VARIÉS



Java Design Patterns
https://www.javacodegeeks.com/minibook/java-design-patterns

Design Patterns Christopher G. Lasater
https://books.google.ca/books?id=f7s8MsAHoucC&printsec=frontcover

Learning Python Design Patterns By Chetan Giridhar
https://books.google.ca/books?id=161KDAAAQBAJ&printsec=frontcover

APPLICATIONS des DESIGN PATTERN

Game programming patterns
http://gameprogrammingpatterns.com/

Design pattern for e-science
https://books.google.ca/books?id=pft6uAvSd34C&printsec=frontcover

Architecture logicielle

Blogs

https://www.joelonsoftware.com/
https://blog.codinghorror.com/
https://michaelfeathers.silvrback.com/
https://insights.sei.cmu.edu/blog/topics/software-architecture/

Livres

Software Architecture in Practice (Bass, Clements, Kazman) – un classique accessible avec beaucoup d'exemples
Aperçu : https://www.google.ca/books/edition/Software_Architecture_in_Practice/-II73rBDXCYC?gbpv=1
Building Evolutionary Architectures (Neal Ford, Rebecca Parsons, Patrick Kua)
https://www.google.ca/books/edition/Building_Evolutionary_Architectures/qYI2DwAAQBAJ?hl=en&gbpv=1

Principes

*Code Simplicity : **https://www.codesimplicity.com/** (livre gratuit & blog)*
*Clean Code: **https://books.google.ca/books?id=_i6bDeoCQzsC&printsec=frontcover** (aperçu Google Book)*
*37 signals : **https://37signals.com/***
12 factors : https://12factor.net/

Intégration continue

Continuous integration
http://en.wikipedia.org/wiki/Continuous_integration

Introduction to continuous integration
http://www.martinfowler.com/articles/continuousIntegration.html

Epicuriens du logiciel
https://ericminio.wordpress.com/category/scrum/

Commits
https://github.com/

Communication continue

https://online.visual-paradigm.com/
https://www.conventionalcommits.org/en/v1.0.0/
https://keepachangelog.com/en/1.1.0/

Techniques et langage

https://ocw.mit.edu/courses/6-005-software-construction-spring-2016/pages/readings/
https://introcs.cs.princeton.edu/java/11cheatsheet/
https://docs.oracle.com/en/java/javase/15/docs/api/index.html
https://quickref.me/java.html

Simulateurs en ligne :

https://www.jdoodle.com/
https://replit.com/languages/java10
https://www.onlinegdb.com/online_java_compiler
https://onecompiler.com/java
https://paiza.io/en/languages/online-java-compiler

IDE et librairies du cours

https://www.eclipse.org/
https://openjdk.org/
https://openjfx.io/