Modélisation et programmation orientées objet

GUIDE DE L'ÉTUDIANT
S2 – APP 1

Hiver 2023 – Semaines 1, 2 et 3

Département de génie électrique et de génie informatique
Faculté de génie
Université de Sherbrooke

Auteur: Charles-Antoine Brunet, Domingo Palao Munoz, Eugène Morin et Daniel Dalle
Version: 242 (2 janvier 2023 à 14:36:06)

Ce document est réalisé avec l'aide de Microsoft Word.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

1 ACTIVITÉS PÉDAGOGIQUES ET COMPÉTENCES

GEN241 – Modélisation et programmation orientées objet

1. Faire l'analyse et la modélisation objet d'un logiciel en utilisant une notation de modélisation objet standardisée et choisir les solutions appropriées pour un problème spécifique.
2. Faire l'implémentation d'un logiciel basé sur les objets en exerçant une approche disciplinée dans la conception, la codification et les tests.

Description officielle : https://www.usherbrooke.ca/admission/fiches-cours/gen241

2 SYNTHÈSE DE L'ÉVALUATION

La note attribuée aux activités pédagogiques de l'unité est une note individuelle. L'évaluation porte sur les compétences figurant dans la description des activités pédagogiques à la section 1. La pondération de chacune d'entre elles dans l'évaluation de cette unité est donnée au tableau 2.1 et d'autres détails sont donnés dans la section 11.

Tableau 2.1 Synthèse de l'évaluation de l'unité

Évaluation	**GEN241-1**	**GEN241-2**
Rapport d'APP, livrables associés et validation	55	50
Évaluation sommative théorique	145	25
Évaluation sommative pratique	-	95
Évaluation finale théorique	100	50
Évaluation finale pratique	-	80
Total	300	300

**À moins de circonstances exceptionnelles, une cote ou un niveau d'atteinte d'une qualité est calculé à partir du tableau 2.2. La grille d'indicateurs utilisée pour les évaluations est donnée au tableau 3.1.*

Tableau 2.2 Calcul d'une cote et d'un niveau d'atteinte d'une qualité

Note (%)	**<50**	**50**	**53**	**57**	**60**	**64**	**68**	**71**	**75**	**78**	**81**	**85**
Cote	E	D	D+	C-	C	C+	B-	B	B+	A-	A	A+
Niveau	N0	N1	N1	N1	N2	N2	N2	N3	N3	N3	N4	N4
Libellé	Insuffisant	Passable (seuil)			Bien			Très bien (cible)			Excellent	

3 QUALITÉS DE L'INGÉNIEUR

Les qualités de l'ingénieur visées par cette unité d'APP sont les suivantes. D'autres qualités peuvent être présentes sans être visées ou évaluées dans cette unité d'APP. Pour une description détaillée des qualités et leur provenance, consultez le lien suivant :
https://www.usherbrooke.ca/genie/etudiants-actuels/au-baccalaureat/bcapg/https://www.usherbrooke.ca/genie/fileadmin/sites/genie/documents/Etudiants_actuels/Au-baccalaureat/BCAPG-Presentation_qualites_vdf_2016-07-26.pdf

Tableau 3.1 Liste des qualités touchées et évaluées

Qualité	Libellé	Touchée	Évaluée
Q01	Connaissances en génie	✓	✓
Q02	Analyse de problèmes		
Q03	Investigation		
Q04	Conception	✓	✓
Q05	Utilisation d'outils d'ingénierie	✓	✓
Q06	Travail individuel et en équipe		
Q07	Communication		
Q08	Professionnalisme		
Q09	Impact du génie sur la société et l'environnement		
Q10	Déontologie et équité		
Q11	Économie et gestion de projets		
Q12	Apprentissage continu		

4 ÉNONCÉ DE LA PROBLÉMATIQUE

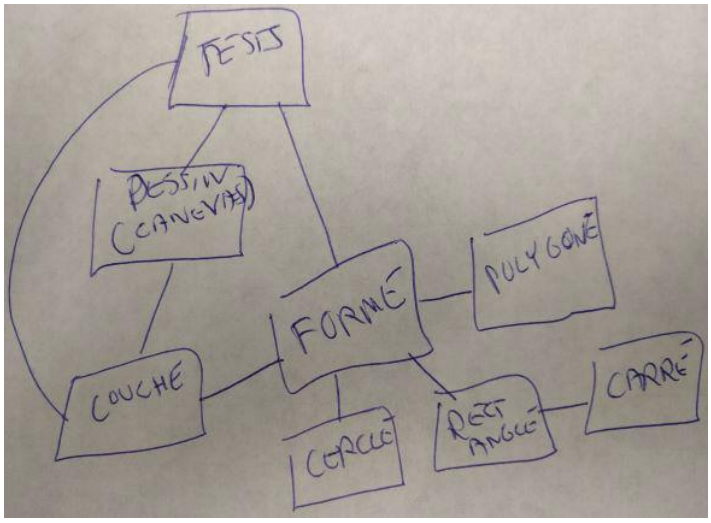
GRAPHICUS

Un collègue vous demande assistance pour développer une nouvelle version du prototype d'un logiciel de graphisme nommé **Graphicus** (graphique, en latin). Selon la spécification, la fonctionnalité de **Graphicus** est de produire et traiter des dessins composés de formes géométriques. La compagnie maintient ce type de logiciel interne depuis des décennies et veut passer de la programmation procédurale à la programmation-objet. En effet, la complexité du logiciel actuellement utilisé augmente d'année en année et la maintenance devient ingérable et il est temps d'en faire une réécriture complète.

Bien que l'entreprise ait déjà développé plusieurs produits selon la méthodologie structurée classique, elle utilise désormais une méthodologie objet avec la notation UML (*Unified Modeling Language*) pour l'analyse, la conception et la documentation de tous ses logiciels. Cette approche est instaurée pour améliorer la qualité des produits logiciels et en faciliter la maintenance et les mises à jour.

Votre collègue a déjà utilisé une méthodologie objets pour implémenter la première itération de **Graphicus**, un prototype nommé **Graphicus-01**. Il utilise une hiérarchie de classes de formes géométriques qui constitue une fondation initiale pour les classes du modèle objet. Le but du prototype est d'évaluer rapidement des stratégies de modélisation objet exploitant le concept d'héritage et de polymorphisme pour ce logiciel de graphisme. Il vous présente l'ébauche d'une hiérarchie de classes en dessinant un diagramme de classes, ou presque.

Le diagramme est montré à la figure suivante :



Les explications montrent qu'il y a une classe abstraite nommée Forme et quelques classes dérivées pour modéliser une hiérarchie de formes géométriques. Cette hiérarchie préliminaire devra être complétée et la pertinence pratique de la classe Polygone est d'ailleurs remis en cause. Les spécifications de **Graphicus** indiquent qu'il devra traiter des dessins plus élaborés qui peuvent être composés de plusieurs couches qui

chacune comportent des formes. Pour l'instant, **Graphicus-01** produit un dessin et le concept de couche est partiellement réalisé. En fait, dans **Graphicus**, un dessin est appelé un canevas.

Pour l'implémentation, faite en C++, l'ensemble des formes dans une couche devra être géré par une structure de données de type vecteur. Cela n'est pas requis pour les couches d'un canevas. Les formes géométriques devront être allouées par allocation dynamique de mémoire. C'est une exigence de l'entreprise. Un canevas devra donc regrouper un ensemble de couches qui elles regroupent chacune un ensemble de formes géométriques.

Des fonctionnalités comme activer une couche, ajouter une forme à une couche dans un canevas et retrancher une forme, afficher les informations d'un canevas, et d'autres sont à concevoir et à coder de manière efficiente dans le prochain prototype. La compagnie demande aussi que le vecteur soit réutilisable dans d'autres projets relativement facilement et vous demande d'en tenir compte dans son implémentation.

Le prototype doit être réalisé avec des unités de compilation séparée et être implémenté et testé sous Linux avec le compilateur g++ et l'utilitaire *make*. Les tests du système devront être systématisés avec l'aide d'une classe de tests. Ils doivent être planifiés à partir de scénario de tests ou de cas d'utilisation et leurs résultats documentés.

Dans un développement futur, il est prévu de réutiliser et d'intégrer les nouvelles classes dans un logiciel avec affichage graphique. Cependant, à l'étape présente, les prototypes sont réalisés seulement avec des interfaces classiques, soit une console à ligne de commande. Le but étant de valider la modélisation objet d'un canevas avant d'approuver les investissements sur ce projet interne.

Évidemment, comme vous vous en doutiez depuis le début, votre collègue vous annonce qu'il vous confie le mandat de réaliser et de valider la prochaine itération, nommée **Graphicus-02**, dont il a défini une spécification plus détaillée disponible à l'annexe A.

De son propre aveu, ce document est incomplet et mélange l'analyse et la conception.

5 CONNAISSANCES NOUVELLES

La surcharge des opérateurs (*operator overloading*) et les modèles de classes (*templates*) ne sont pas des sujets à l'étude. Si vous ne connaissez pas ces outils du C++, il vous est suggéré de ne pas les utiliser, car ils ajoutent une complexité supplémentaire au code à réaliser. Si vous les connaissez, vous êtes alors libre de les utiliser ou non.

Connaissances déclaratives (quoi)

- Notation UML : diagramme de cas d'utilisation (formes graphique et textuelle), diagramme de classes, diagramme de séquence, diagramme d'états-transitions et diagramme d'activités
- Méthodologie objet : classe, objet, modélisation statique et dynamique
- Langage C++ :
  - Classes et abstraction de données
  - Héritage
  - Polymorphisme et méthodes virtuelles
  - Classe abstraite
  - Pointeurs
  - Structures de données : allocation dynamique et vecteurs
  - *shell* de commande
  - Outils pour projet de compilation séparée

Connaissances procédurales (comment)

- Procédure de développement de logiciels
- Production d'un modèle objet dans la notation UML pour un problème spécifique
- Codage de programmes dans un bon style de programmation
- Implémentation du modèle objet en C++
- Utilisation d'un *shell* de commande
- Gestion d'un projet de compilation séparée et utilitaires
- Réalisation de tests de validation

Connaissances conditionnelles (quand)

- Choisir et formuler le diagramme adéquat pour le besoin d'un cas d'analyse.
- Choisir et implémenter les énoncés de programmation C++ pour la conception d'un modèle objet.
- Choisir et implémenter les tests de validation qui correspondent à une spécification.

6 GUIDE DE LECTURE

Il est important de mettre en pratique une méthode de lecture efficace. Les schémas de concept peuvent aider beaucoup à organiser les nouvelles connaissances. Comme le manuel sur la modélisation UML peut sembler plus abstrait que celui de programmation, il est important d'aborder la lecture des deux manuels dès le début de l'unité d'APP.

6.1 Les lectures du livre d'UML

Pratique procédurale 1

- Introduction
- Chapitre 1
- Chapitre 2 : sections 1 à 4 et 8
- Chapitre 4 : sections 1 à 4

Pratique procédurale 2

- Chapitre 3 : sections 1 et 2
- Chapitre 5 : sections 1 à 3

6.2 Les lectures du livre de C++

Pratique procédurale 1

- Leçon 2 : pages 17 à 28
  - Exercices supplémentaires pages 28 et 29
- Leçon 14 : pages 395 et 396
- Leçon 8 : pages 187 à 190 (opérateur *new* et *delete* pour allocation dynamique de mémoire)
  - Questions supplémentaires Q&A et Quiz pages 210 à 212
  - Exercices supplémentaires 1, 4, 5 et 6, pages 212 et 213
- Leçon 9 : pages 215 à 244
  - Exercices supplémentaires pages 268 à 270
- Leçon 10 : pages 271 à 290
  - Questions supplémentaires Q&A, Quiz 1 et 2, pages 302 et 303
  - Exercices supplémentaires 1 à 5, pages 303

Pratique procédurale 2

- Leçon 4 : pages 63 à 81
  - Questions supplémentaires Q&A et Quiz, pages 81 à 83
  - Exercices supplémentaires, pages 83

- Leçon 11 : pages 305 à 320
  - Questions supplémentaires Q&A et Quiz, pages 332 et 333
  - Exercices supplémentaires, pages 333
    **Dans la section Q&A, le terme ABC = Abstract Base Class*

- Leçon 17 : pages 456 à 469
  - Questions supplémentaires Q&A (1 à 4) et Quiz (1 à 5), pages 473 et 474
  - Exercices supplémentaires 1 à 3, pages 474

Les leçons 4 et 17 du livre de C++ sont lues uniquement afin de comprendre ce qu'est un vecteur et son utilisation.

La leçon 3 du livre de C++ (pages 55 à 58) est à lire pour comprendre les *Enumerations* et la définition des constantes.

Les documents d'accompagnement de la problématique sont disponibles sur la page WEB de l'unité, comme les fichiers de démarrage et le guide de pseudocode.

7 LOGICIELS ET MATÉRIEL

- Le travail se fait dans l'environnement Linux sur les stations des laboratoires du département ou sur la machine virtuelle Ubuntu fournie sur le site de l'APP (voir la documentation sur le site).
- L'environnement Linux est accessible sur les postes informatiques des laboratoires du département amorcés avec le noyau Linux. C'est un choix au démarrage de la station.
- Le compilateur utilisé est g++, le compilateur C++ de GNU.
- L'utilitaire *make* est utilisé pour gérer le projet de compilation et générer le fichier exécutable.
- Plusieurs éditeurs de texte sont disponibles sur les systèmes du département. L'éditeur de texte *gedit* est recommandé pour développer le code.
- Les diagrammes UML demandés sont typiquement faits avec un logiciel spécialisé pour les diagrammes UML, des suggestions de logiciels sont faites sur le site web de la session. Les diagrammes peuvent être également faits à la main de manière soignée et il faut alors les numériser pour le rapport.

8 SANTÉ ET SÉCURITÉ : DISPOSITIONS GÉNÉRALES

Dans le cadre de la présente activité, vous êtes réputés avoir pris connaissance des politiques et directives concernant la santé et la sécurité. Ces documents sont disponibles sur les sites web de l'Université de Sherbrooke, de la Faculté de génie et du département. Les principaux sont mentionnés ici et sont disponibles dans la section *Santé et sécurité* du site web du département:

https://www.gegi.usherbrooke.ca/santesecurite/.

- Politique 2500-004: Politique de santé et sécurité en milieu de travail et d'études
- Directive 2600-042: Directive relative à la santé et à la sécurité en milieu de travail et d'études
- Sécurité en laboratoire et atelier au département de génie électrique et de génie informatique

9 SOMMAIRE DES ACTIVITÉS

Horaire sur **3 semaines**

Semaine 1

- Première rencontre de tutorat
- Étude personnelle et exercices
- Séminaire-atelier sur la programmation dans l'environnement Linux

Semaine 2

- Formation à la pratique procédurale 1
- Formation à la pratique en laboratoire
- Formation à la pratique procédurale 2
- Consultation facultative
- Support en laboratoire
- Étude personnelle et exercices
- Validation au laboratoire

Semaine 3

- Remise des livrables d'APP
- Deuxième rencontre de tutorat
- Consultation facultative
- Évaluation sommative théorique et pratique sous Linux

10 PRODUCTIONS À REMETTRE

- Les livrables sont réalisés en équipe de 2. La même note sera attribuée à chaque membre de l'équipe.
- L'identification des membres de l'équipe doit être faite avant 16h30, le lendemain de votre 1er tutorat. Passé cette date limite, les personnes seules ou dans des équipes ne respectant pas les contraintes de formation d'équipe peuvent être affectées à une équipe par le tuteur. Le cas échéant, cette assignation est sans appel.
- La **date limite du dépôt** des livrables est **08h30** (pas 20h30!), le jour du 2e tutorat. Tout retard entraine une pénalité immédiate de 20% et de 20% par jour supplémentaire.
- La remise des livrables se fait **uniquement avec l'outil de dépôt** du département. Conséquemment, toute remise de livrables, complète ou partielle, faite autrement que par ce moyen sera ignorée.
- L'outil de dépôt ne permet qu'un seul dépôt. Si vous voulez faire un autre dépôt, il faut contacter un tuteur avant la date limite.
- Le non-respect des directives et des contraintes, comme le nom d'un répertoire ou d'un fichier, peut entrainer des pénalités.
- Seules les collaborations intra équipe sont permises. Cependant, vous devez résoudre la problématique de façon individuelle pour être en mesure de réussir les évaluations.
- Les productions soumises à l'évaluation doivent être originales pour chaque équipe, sinon l'évaluation sera pénalisée.

10.1 Schéma de concept

Le schéma de concept est une production individuelle optionnelle en vue de la deuxième rencontre de tutorat. Le schéma de concepts à faire cible la question suivante :

> *Comment la modélisation orientée objet, avec la notation UML, permet-elle de représenter un système logiciel, ses comportements et sa structure?*

10.2 Rapport et livrables associés

Afin de faciliter la gestion de la correction, une seule archive zip est déposée avec l'outil de dépôt du département et son nom est le CIP de tous les membres de l'équipe séparés par des tirets. Le contenu de l'archive est détaillé dans ce qui suit.

Rapport

Le rapport remis doit être en format **PDF** et la page couverture doit contenir les noms et les CIP des membres de l'équipe. Le nom du fichier **PDF** suit le même format que l'archive déposée : le CIP de tous les membres de l'équipe séparés par des tirets.

Le rapport contient uniquement les éléments suivants :

- Les diagrammes UML pertinents pour documenter l'itération **Graphicus-02**. Lesquels sont pertinents? Ceux qui permettent de documenter les comportements de **Graphicus-02** et des modifications de **Graphicus-01** à **Graphicus-02**. Ceci requiert donc au moins un diagramme détaillé de chaque type (cas d'utilisation, classes, séquences, états-transitions).
- Le pseudocode de la fonction d'insertion de votre classe de vecteur.
- Un plan de tests de votre cru qui décrit des scénarios afin de tester les fonctionnalités de **Graphicus-02** et un résumé des résultats de ces tests. Le plan de tests est limité à **2 pages**. Ce n'est donc pas un plan de tests exhaustif, mais un plan de tests sommaire.

Fichiers de code et autres fichiers

Deux répertoires accompagnent le fichier PDF du rapport. Ce sont les deux répertoires suivants :

- **Code** :
  - Dans ce répertoire, remettre tous les fichiers sources (.cpp, .h et makefile) de votre **Graphicus-02**. Au minimum, inscrire les noms et les CIP des membres de l'équipe dans le fichier du programme principal (le *main*).
- **Tests** :
  - Dans ce répertoire, remettre les résultats des tests développés, au minimum ceux du plan de tests donné dans le rapport. Par exemple, les résultats peuvent être les sorties à l'écran des tests qui sont redirigées dans des fichiers.

11 ÉVALUATIONS

La synthèse des évaluations est donnée à la section 2 et plus de détails sont donnés ici.
Toutes les évaluations de l'unité visent les éléments de compétences mentionnées dans la section 1.

11.1 Livrables liés à la problématique

L'évaluation des livrables est directement liée ce qui est demandé à la section 10.2 et le tableau 11.1 y réfère à l'aide d'une courte description.

Tableau 11.1 Sommaire de l'évaluation des livrables liés à la problématique

Évaluation	**GEN241-1**	**GEN241-2**
Diagrammes UML du modèle objet (pertinence et qualité)	50	
Pseudocode de l'opération d'ajout du vecteur	5	
Style de codage		5
Fonctionnalités de l'application		15
Fonctionnalités du vecteur		10
Plan de tests et résultats		5
Respect de la compilation séparée et *makefile*		5
Validation		10
Total	55	50

Quant à la qualité de la communication technique, elle ne sera pas évaluée de façon sommative, mais si votre rapport est fautif sur le plan de la qualité de la communication et de la présentation, il vous sera retourné et vous devrez le reprendre pour être noté.

11.2 Évaluation sommative et finale de l'unité

L'évaluation sommative et l'évaluation finale ont une partie théorique et une partie pratique sur ordinateur. Seulement la feuille de référence du langage C++ et le petit guide Linux seront disponibles sous forme électronique pendant les examens pratiques. Lors des évaluations théoriques, aucune documentation n'est permise.

11.3 Qualités de l'ingénieur

La grille d'indicateurs utilisée aux fins de l'évaluation est donnée au tableau 11.2. Il est à noter qu'un niveau d'atteinte d'un indicateur dans cette grille n'a pas la même signification qu'un niveau d'atteinte d'une qualité dans le tableau 2.2. Cela est normal, un indicateur et une qualité, ce sont deux choses différentes.

Tableau 11.2 Grille d'indicateurs utilisée pour les évaluations

Indicateur	Qualité	Aucun (N0)	Insuffisant (N1)	Seuil (N2)	Cible (N3)	Excellent (N4)
Notions de UML	Q01.1	... ne démontre pas une compréhension des notions d'UML.	... démontre une compréhension insuffisante des notions d'UML.	... démontre une compréhension minimale des notions d'UML.	... démontre une bonne compréhension des notions d'UML.	... démontre une excellente compréhension des notions d'UML.
Utilisation de UML	Q04.4	... ne démontre pas la capacité de définir une conception détaillée avec UML.	... utilise UML pour définir une conception détaillée insuffisante.	... utilise UML pour définir adéquatement une conception détaillée minimale.	… utilise UML pour définir adéquatement une conception détaillée presque complète.	... utilise UML pour définir adéquatement une conception détaillée complète.
Notions du C++	Q01.1	... ne démontre pas une compréhension des notions du C++.	... démontre une compréhension insuffisante des notions du C++.	... démontre une compréhension minimale des notions du C++.	... démontre une bonne compréhension des notions du C++.	... démontre une excellente compréhension des notions du C++.
Utilisation du C++	Q01.2	... ne démontre pas la capacité d'utiliser les notions du C++.	... utilise de façon insuffisante les notions du C++.	... est en mesure d'utiliser minimalement les notions du C++.	... est en mesure d'utiliser adéquatement les notions du C++.	... est en mesure d'utiliser adéquatement et efficacement les notions du C++.
Notions de structures de données	Q01.1	... ne démontre pas une compréhension des structures de données.	... démontre une compréhension insuffisante des structures de données.	... démontre une compréhension minimale des structures de données.	... démontre une bonne compréhension des structures de données.	... démontre une excellente compréhension des structures de données.
Utilisation des structures de données	Q01.2	... ne démontre pas la capacité d'utiliser les structures de données.	... utilise de façon insuffisante les structures de données.	... est en mesure d'utiliser minimalement les structures de données.	... est en mesure d'utiliser adéquatement les structures de données.	... est en mesure d'utiliser adéquatement et efficacement les structures de données.
Valider la solution	Q04.5	... ne démontre pas la capacité de valider une solution.	... valide inadéquatement ou de manière non pertinente que la solution répond aux exigences.	... valide minimalement que la solution répond aux exigences.	... valide que la solution répond aux exigences sauf dans certains cas non critiques.	... valide complètement que la solution répond aux exigences
Utiliser les techniques et outils sélectionnés selon les protocoles établis.	Q05.2	... ne démontre pas la capacité d'utiliser les techniques et outils spécifiés.	... ne démontre pas suffisamment la capacité d'utiliser les techniques et outils spécifiés.	... connait et est en mesure d'utiliser avec assistance mineure les techniques et outils spécifiés.	... connait et est en mesure d'utiliser les techniques et outils spécifiés.	... connait et est en mesure d'utiliser efficacement les techniques et outils spécifiés
Bonnes pratiques de programmation	Q01.2	... ne démontre pas la capacité à appliquer les bonnes pratiques de programmation	... démontre de la difficulté à appliquer les bonnes pratiques de programmation.	... sait appliquer la majorité des bonnes pratiques de programmation.	... sait appliquer efficacement la majorité des bonnes pratiques de programmation.	... sait appliquer efficacement les bonnes pratiques de programmation.

12 POLITIQUES ET RÈGLEMENTS

Dans le cadre de la présente activité, vous êtes réputés avoir pris connaissance des politiques, règlements et normes d'agrément suivants.

Règlements de l'Université de Sherbrooke

- Règlement des études: https://www.usherbrooke.ca/registraire/

Règlements facultaires

- Règlement facultaire d'évaluation des apprentissages / Programmes de baccalauréat
- Règlement facultaire sur la reconnaissance des acquis

Normes d'agrément

- Informations pour les étudiants au premier cycle: https://www.usherbrooke.ca/genie/fileadmin/sites/genie/documents/Etudiants_actuels/Au-baccalaureat/BCAPG-Presentation_qualites_vdf_2016-07-26.pdf
- Informations sur l'agrément: https://engineerscanada.ca/fr/agrement/a-propos-de-l-agrement

Si vous êtes en situation de handicap, assurez-vous d'avoir communiqué avec le *Programme d'intégration des étudiantes et étudiants en situation de handicap* à l'adresse de courriel prog.integration@usherbrooke.ca.

13 INTÉGRITÉ, PLAGIAT ET AUTRES DÉLITS

Dans le cadre de la présente activité, vous êtes réputés avoir pris connaissance de la déclaration d'intégrité relative au plagiat : https://www.usherbrooke.ca/ssf/enseignement/evaluation-des-apprentissages/passeurs-integrite/ressources/antiplagiat

14 PRATIQUE PROCÉDURALE 1

Note : Il est important de faire les lectures demandées avant de se présenter au procédurale [Section 6].

Buts de l'activité

- Comprendre la notion de classe, d'objet et de vecteur.
- Analyser l'implémentation en C++ de modèles objet.
- Comprendre la construction et la destruction d'objets en C++.
- Construire des diagrammes de classes, de cas d'utilisation et d'états-transitions en UML.

14.1 Exercices

E.1 Classes et objets

1. Qu'est-ce qu'une classe dans la vie de tous les jours et en C++? Donnez des exemples.
2. Qu'est-ce qu'un objet dans la vie de tous les jours et en C++ ? Donnez des exemples.
3. Pourquoi fait-on des objets ou de la Programmation Orientée Objet ?
4. Qu'est-ce qu'un vecteur? À quoi peuvent-ils servir? Donnez des exemples.

E.2 Diagramme de classes

Analysez le code de l'annexe B et faites-en le diagramme de classes.

E.3 Analyse de code

En analysant le code de l'annexe B :

1. Identifiez les classes et les objets instanciés.
2. Quels objets sont construits statiquement et dynamiquement?
3. Décrivez quelle sortie à l'écran produira ce code sans le compiler ni l'exécuter.

E.4 Diagramme de cas d'utilisation et d'états-transitions

Le logiciel *MenuFact* gère le menu, les factures et la facturation d'un restaurant. *MenuFact* permet de maintenir à jour le menu des plats offerts et de l'afficher sur des tablettes pour les clients. Il permet aussi d'assister la gérante, ou le gérant, et les serveuses, ou les serveurs, avec la gestion des factures. Il permet donc de gérer ce qui est commandé par les clients, incluant de payer. Le dernier volet est la facturation qui permet de faire la gestion administrative des ventes de la journée.

Pour la gestion du menu, il y a des fonctionnalités comme ajouter et retirer des plats au menu et sauvegarder et afficher et publier un menu. D'autres fonctionnalités sont surement aussi à prévoir. Pour certains plats au menu, les plats qui sont identifiés comme étant santé, le menu doit inclure et afficher des

renseignements de nature diététique. Le client passe sa commande à la serveuse ou au serveur et non pas directement sur la tablette.

Pour la gestion des factures, des fonctionnalités comme ouvrir et fermer une facture, ajouter un plat, retirer un plat, modifier des quantités, afficher, payer et imprimer sont à prévoir et surement d'autres aussi. Évidemment, toute opération ne peut pas être effectuée en tout temps. Par exemple, il ne devrait pas être permis d'ajouter ou de retirer un plat sur une facture qui a été fermée ou payée. Il faut gérer les opérations valides selon l'évolution du repas et de sa conclusion éventuelle.

- Faites un diagramme (ou des diagrammes?) de cas d'utilisation du logiciel *MenuFact*.
- Faites aussi un diagramme d'états-transitions de la facture.

15 PRATIQUE LABORATOIRE 1

Buts de l'activité

Le but de cette activité est d'expérimenter avec le développement de logiciels par objets en C++ sous Linux.

- Compiler un programme simple.
- Compiler un programme possédant plusieurs unités de compilation.
- Analyser un programme en y apportant une modification simple.
- Analyser le fonctionnement de l'application de la problématique.

Procédure de développement et de programmation

- Démarrer votre station de travail en choisissant l'amorçage de *Linux* (*Ubuntu*).
- Se référer à la documentation de l'atelier.
- Les exemples de l'atelier, du laboratoire et les codes sources requis sont disponibles sur le site de l'APP.
- Utiliser un éditeur de texte, comme *gedit*, dans l'environnement *Linux*.

E.1 Première compilation

Le but de cet exercice est de faire une première compilation afin de se familiariser avec les commandes de base de l'environnement *Linux* et de faire la compilation d'un programme simple. Ce premier exercice devrait être relativement rapide puisqu'il rappelle simplement les apprentissages faits à l'atelier. Les étapes principales sont les suivantes.

1. Créer dans votre compte un répertoire pour le laboratoire.
2. Copier les fichiers du projet *TableauEntier* (étant fournis sur le site de l'APP).
3. Compiler le programme sur la ligne de commande avec l'utilitaire *make*. Le *makefile* est fourni.
4. Exécuter le programme sur la ligne de commande.
5. Modifier le code original en ajoutant quelques lignes, comme indiqué à l'annexe C.
6. Exécuter le programme sur la console.
7. Expérimentez avec l'ensemble du code afin de bien le comprendre.

E.2 Évaluation

Cet exercice consiste à monter un projet C++ sous Linux et de travailler avec une hiérarchie de classes. Il faut monter un projet nommé *Evaluations* à partir du code fourni pour cet exercice. Les étapes principales sont les suivantes :

1. Dans le répertoire laboratoire, copier les fichiers du projet *Evaluations* (étant fournis sur le site de l'APP).

2. Le *makefile* a un problème et vous pouvez le constater en lançant la compilation avec *make*. Vous devez régler ce problème. Si nécessaire, vous pouvez consulter les notes du séminaire-atelier de l'unité.
3. Compiler avec un *makefile* corrigé et exécuter le programme généré.
4. Modifier et tester le programme pour que la fonction d'affichage devienne une méthode de la classe *Evaluation*. Voici un exemple avec le programme principal modifié.

```
for (int count = 0; count < NUM_TESTS; count++)
{
    cout << "Test #" << (count + 1) << ":\n";
    tests[count]->afficher();
    cout << endl;
}
```

E.3 Code du programme Items

Cet exercice consiste à réaliser un projet C++ sous *Linux* avec l'utilisation de fichiers .cpp (code source) et .h (*header*).

À partir de l'exemple de code donné à l'Annexe B, créer un projet en C++ dont le code doit être séparé dans différents fichiers. Les fichiers du projet sont :

- Item.h
- Item.cpp
- Item_Simple.h
- Item_Simple.cpp
- Item_Complexe.h
- Item_Complexe.cpp
- Item_Main.cpp (contenant la méthode « main »)

Ensuite :

- Compléter le fichier « *makefile* » afin de compiler l'application, et pouvoir l'exécuter.
- Valider la sortie en console selon l'analyse fait au procédural 1 (E.3)

E.4 Code du programme Items (pointeur)

En réutilisant le code de l'exercice précédent (E.3), effectuer la modification suivante :

- Dans le fichier « *Item_Complexe.h* », modifier l'objet « *Item_Simple c;* » en tant que pointeur, soit :
    - *Item_Simple *pc;*

- Effectuer les modifications nécessaires au code du fichier « *Item_Complexe.cpp* », afin que le code compile à nouveau et que la sortie en console soit la même qu'à l'exercice précédent.

E.5 Construire Graphicus-01

Il s'agit de monter le projet **Graphicus-01** (le code source se trouve sur le site de l'APP).

- Une bonne pratique est de vous créer un répertoire pour le code source fourni **Graphicus-01**.
- Puis de créer un second répertoire nommé **Graphicus-02**, contenant le même code source (sera utilisé pour votre développement).
  *Ainsi, vous conserverez une copie du code source original **Graphicus-01** (en cas de besoin).

Il faut commencer par examiner le code source et faire une liste des unités de compilation. Ensuite, il faut établir les dépendances de compilation entre ces unités et proposer un graphe qui décrit leurs dépendances. Cela fait, il faut concevoir et réaliser un fichier *makefile* pour pouvoir ensuite compiler ce programme en utilisant *make*.

E.6 Exécution de Graphicus-01

Le but est d'expérimenter avec l'exécution du programme **Graphicus-01**. Les tests de l'application effectués sont décrits dans la classe Tests. Lorsque le programme est exécuté, il produit toutes les sorties sur la console. Il est intéressant de faire en sorte que toutes les sorties sur la console soient redirigées dans un fichier, comme pour archiver des résultats de tests. La commande donnée sur la ligne est alors la suivante :

./graphicus-01 > test.log

Les sorties normalement faites sur la console sont alors redirigées dans le fichier **test.log**.

*Évidemment, un autre nom de fichier peut être donné. Ce mode de fonctionnement est pratique pour réaliser les tests, car il permet de produire des rapports dans des fichiers.

16 PRATIQUE PROCÉDURAL 2

Note : Il est important de faire les lectures demandées avant de se présenter au procédurale [Section 6].

Buts de l'activité

- Analyser l'implémentation en C++ de modèles objet.
- Comprendre le polymorphisme et les méthodes virtuelles en C++.
- Construire des diagrammes de classes, de séquence et d'activités en UML.
- Concevoir des algorithmes pour une classe de vecteur.

16.1 Exercices

E.1 Diagramme de classes

Analysez le code de l'annexe D et faites-en le diagramme de classes.

E.2 Analyse de code

En analysant le code de l'annexe D :

1. Identifiez les classes et les objets instanciés.
2. Quels objets sont construits statiquement et dynamiquement?
3. Décrivez quelle sortie à l'écran produira ce code sans le compiler ni l'exécuter.

E.3 Diagramme de séquence et d'activités

Avec le contexte de *MenuFact* du problème E.4 du procédural 1, faites un diagramme de séquence d'un scénario typique d'un couple, nommément madame et monsieur, qui dinent au restaurant à partir du moment où ils donnent chacun leur choix au menu jusqu'au moment de payer la facture.

Aussi, donnez un diagramme d'activités pour votre scénario.

E.2 Ensemble de nombre

Cet exercice repose sur la classe *EnsembleDeNombres* (annexe E) dont le fichier d'entête et un exemple d'utilisation sont donnés à l'annexe E. Il s'agit de trouver des algorithmes pour les méthodes de la classe *EnsembleDeNombres*. Ici, il s'agit de faire un travail conceptuel. La conception peut être faite graphiquement, avec du texte, avec pseudocode ou tout autre formalisme équivalent. Le rôle des méthodes peut être défini intuitivement comme suit :

Afficher : Écriture à l'écran des toutes les valeurs stockées dans le vecteur.
Ajout à la fin : Ajout de la valeur donnée à la suite de la dernière valeur ajoutée.
Supprimer au début : Suppression de la première valeur stockée, la plus ancienne.

17 VALIDATION AU LABORATOIRE

Le but de cette activité est de valider l'implémentation de la solution qui a été développée pour le logiciel **Graphicus-02**. Un scénario de test sera fourni sur la page web de l'unité.

C'est ce scénario qui est utilisé pour faire la validation des fonctions essentielles du nouveau programme. Ce scénario de tests n'est que pour la validation et il ne constitue pas votre plan de tests à remettre dans votre rapport.

Lors de la validation, chaque équipe doit présenter ses travaux de réalisation de la problématique afin de valider les apprentissages faits. L'évaluation des diagrammes UML est formative. Une période de temps sera allouée à chaque équipe lors de l'activité de validation. Le moment de cette période et une directive sur la logistique de la validation seront publiés sur la page web de l'unité...

A SPÉCIFICATIONS INITIALES DE GRAPHICUS

Graphicus est un logiciel de dessin qui est à l'état de prototype et dont les fonctionnalités sont très limitées. La fonction du prototype est centrée sur le canevas, les couches et les formes géométriques. Le but étant de valider la conception du canevas et des classes sous-jacentes.

Il s'agit de créer, de manipuler et d'afficher des informations sur le canevas et toutes ses parties. Le prototype affiche tous les résultats à l'écran sur la console. Les spécifications qui suivent documentent le passage de **Graphicus-01** à **Graphicus-02**. Certaines parties relatives à l'itération **Graphicus-02** sont donc incomplètes.

Les spécifications données sont un minimum qui doit être implémenté, mais rien n'empêche d'ajouter à ces spécifications, tant que ces ajouts ne viennent pas contredire, invalider ou modifier ce qui est décrit ici.

Les tests sont faits avec l'aide d'une classe de tests spécifique qui simule les interactions d'un utilisateur avec le canevas. L'usager n'interagit donc pas directement avec le prototype, car tous les tests sont automatisés avec la classe de tests.

Les différentes classes impliquées dans les itérations **Graphicus-01** et **Graphicus-02** et qui doivent être conçues et implémentées sont maintenant décrites. La figure de votre collègue est rappelée ici à la figure A.1. Il peut être utile de suivre les explications suivantes avec le code existant de **Graphicus-01**. Aussi, la sortie attendue lors de l'affichage d'un canevas avec plusieurs couches et formes est donnée un peu plus loin à la figure A.2.

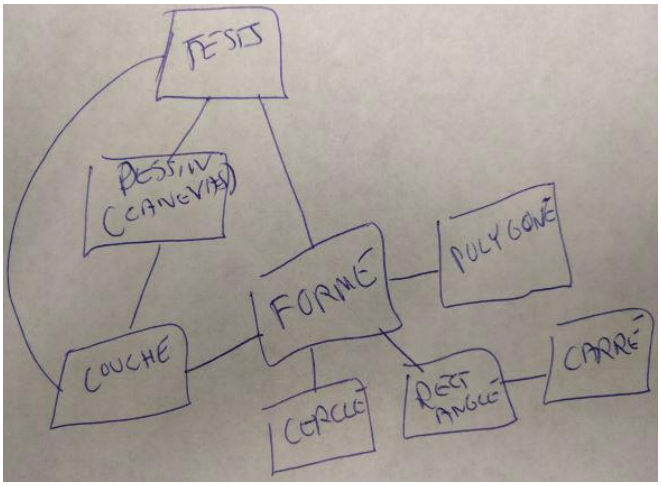


Figure A.1 Ébauche d'un diagramme de classes, ou presque.

A.1 Classes

Il est important de noter que dans les classes qui suivent, il n'y a aucun affichage qui est fait. Seules les méthodes qui permettent de faire l'affichage en font. De la même manière, il n'y a aucune sollicitation qui est faite à l'usager, comme au clavier.

A.1.1 Classes de formes géométriques

Avant de donner le détail des classes de formes géométriques, il est important de noter qu'elles utilisent une structure utilitaire, la structure *Coordonnee*. Cette structure permet de stocker le point d'ancrage et les coordonnées x et y dans le plan où se trouve la forme.

La coordonnée est représentée par des entiers, car les valeurs sont en pixels. La structure *Coordonnee* est fournie et ne doit pas être modifiée.

Classe Forme
C'est une classe abstraite de laquelle sont dérivées toutes les autres formes géométriques. Les classes dérivées de Forme doivent être concrètes. Cette classe est fournie, mais ne doit pas être modifiée.

Classe Rectangle
Un rectangle est spécifié par sa largeur, sa hauteur et son point d'ancrage. Le point d'ancrage est le coin inférieur gauche. Par défaut, lors de sa création, son point d'ancrage est à l'origine et ses dimensions sont unitaires. Évidemment, lors de sa création, il doit être possible de spécifier autrement. Cette classe n'est pas fournie et doit être réalisée. La classe doit fournir les méthodes nécessaires pour accéder aux différents attributs afin d'en connaitre la valeur ou d'en modifier la valeur.

Classe Carre
Un carré est spécifié par la longueur de son côté et son point d'ancrage. Le point d'ancrage est le coin inférieur gauche. Par défaut, lors de sa création, son point d'ancrage est à l'origine et ses dimensions sont unitaires. Évidemment, lors de sa création, il doit être possible de spécifier autrement. Cette classe n'est pas fournie et doit être réalisée. La classe doit fournir les méthodes nécessaires pour accéder aux différents attributs afin d'en connaitre la valeur ou d'en modifier la valeur.

Classe Cercle
Un cercle est spécifié par la longueur de son rayon et son point d'ancrage. Le point d'ancrage est le centre du cercle. Par défaut, lors de sa création, son point d'ancrage est à l'origine et son rayon est unitaire. Évidemment, lors de sa création, il doit être possible de spécifier autrement. Cette classe n'est pas fournie et doit être réalisée. La classe doit fournir les méthodes nécessaires pour accéder aux différents attributs afin d'en connaitre la valeur ou d'en modifier la valeur.

A.1.2 Classe Couche

Cette classe permet de gérer un ensemble de formes géométriques avec l'aide d'un vecteur, comme décrit à la section A.1.3 un peu plus loin. La classe *Couche* n'est pas fournie; elle donc à faire. La classe *Couche* est responsable de plusieurs tâches et des contraintes sont imposées :

1. Elle stocke les formes qui font partie de la couche. Comme toutes les formes sont allouées dynamiquement, elle stocke des pointeurs à des formes.
2. Elle utilise un vecteur pour stocker les formes. En l'occurrence, elle utilise un vecteur qui stocke des pointeurs de formes.

3. Elle permet l'ajout d'une forme à la suite des autres formes déjà présente. La forme à ajouter est spécifiée par son pointeur. Une valeur de retour booléenne indique si l'opération est une réussite ou un échec.
4. Elle permet le retrait d'une forme de la couche. La forme à retirer est spécifiée par son index dans le vecteur. Si l'opération est une réussite, alors la valeur de retour est le pointeur sur la forme en question. Si l'opération est un échec, comme avec un index invalide, un pointeur nul est retourné. La forme retirée de la couche n'est pas détruite.
5. Elle permet d'obtenir une forme stockée dans la couche. La forme à obtenir est spécifiée par son index dans le vecteur. Si l'opération est une réussite, alors la valeur de retour est le pointeur sur la forme en question. Si l'opération est un échec, comme avec un index invalide, un pointeur nul est retourné. La forme n'est pas retirée du vecteur et elle n'est pas détruite.
6. Elle offre la possibilité d'obtenir l'aire totale de la couche. L'aire totale de la couche est définie comme la somme des aires des formes qui la compose. La valeur de retour, l'aire totale, est un nombre réel.
7. Elle offre la possibilité de translater la couche. La translation d'une couche est réalisée en faisant la translation de toutes les formes qui la compose. Les valeurs de translation horizontale et verticale, des entiers, sont passées en paramètres. Une valeur de retour booléenne indique si l'opération est une réussite ou un échec.
8. Elle offre la possibilité de réinitialiser la couche. L'état de la couche est donc ramené à son état initial, comme lors de sa création. Cette opération vide donc le vecteur et toutes les formes sont détruites. Une valeur de retour booléenne indique si l'opération est une réussite ou un échec.
9. Elle offre la possibilité de changer son état. Les états possibles d'une couche sont : *initialisée*, *active* et *inactive*. L'état désiré est spécifié en paramètre. Une valeur de retour booléenne indique si l'opération est une réussite ou un échec. Les détails sur les états et la manière dont ils affectent les opérations de la couche sont donnés un peu plus loin.
10. Elle permet l'affichage du contenu d'une couche, donc de toutes ses formes. L'endroit à afficher (le flot) est spécifié en paramètre. La figure A.2 montre comment une couche doit être affichée. On peut d'ailleurs y observer que l'état de la couche affecte l'affichage.

Une couche peut être dans un de trois états : *initialisée*, *active* et *inactive*.

Initialisée : C'est l'état initial de la couche lorsqu'elle est créée. C'est aussi son état suite à sa réinitialisation, comme décrite au point 8 ci-haut. Cet état ne peut pas être atteint avec l'appel de changement d'état, comme décrit au point 9 ci-haut. L'aire d'une couche dans cet état est zéro.

Active : Cet état est atteint avec l'appel de changement d'état, comme décrit au point 9 ci-haut. Une couche doit être active afin de pouvoir y apporter des modifications sur les formes : comme une demande d'ajout, de retrait ou de translation.

Inactive : Cet état est atteint avec l'appel de changement d'état, comme décrit au point 9 ci-haut. Lorsqu'une couche est inactive, aucune modification ne peut y être apportée sur les formes : comme une demande d'ajout, de retrait et de translation. Une demande de modification se solde alors en un échec.

A.1.3 Classe Vecteur

Cette classe permet de stocker les formes géométriques, comme mentionné dans la section sur la classe Couche. Cette classe n'est pas fournie; elle donc à faire. Cette classe concerne directement un sujet d'étude et doit donc être réalisée sans l'aide de librairies réalisant le même type de fonctionnalité, comme la classe *vector* de la librairie standard qui vient avec le compilateur.

Comme demandé par la compagnie, la classe doit (autant que possible) être facilement réutilisable dans un autre projet qui n'a possiblement aucun lien avec les formes géométriques. Il faut donc minimiser les endroits où doivent être faites les modifications (noms des méthodes, des variables, etc.), si le vecteur était réutilisé pour stocker un autre type de données.

La classe *Vecteur* est responsable de plusieurs tâches et des contraintes sont imposées :

1. Les items stockés sont des pointeurs de formes.
2. Les items sont toujours contigus en mémoire.
3. Elle permet de connaitre la capacité du vecteur. La valeur de retour est donc un entier. La capacité est le nombre maximal d'items qui peuvent être stockés dans le vecteur.
4. Elle permet de connaitre la taille du vecteur. La valeur de retour est donc un entier. La taille est le nombre actuel d'items stockés dans le vecteur. La valeur de la taille est donc dans l'intervalle [0, capacité].
5. Lorsqu'il manque d'espace dans le vecteur pour ajouter une forme, la capacité du vecteur est doublée.
6. Elle permet de vider le vecteur en un seul appel.
7. Lorsque le vecteur est détruit ou qu'il est vidé, les formes pointées sont alors détruites. Pour toutes les autres opérations, les formes ne sont pas détruites.
8. Elle permet de savoir si le vecteur est vide. La valeur de retour est donc un booléen : vrai si le vecteur est de taille zéro, faux autrement.
9. Elle permet l'ajout d'une forme à la suite des autres formes déjà présente. La forme à ajouter est spécifiée en passant en paramètre son pointeur. Une valeur de retour booléenne indique si l'opération est une réussite ou un échec.
10. Elle permet le retrait d'une forme. La forme à retirer est spécifiée par son index dans le vecteur. Si l'opération est une réussite, alors la valeur de retour est le pointeur sur la forme en question. La forme n'est pas détruite. Si l'opération est un échec, comme avec un index invalide, un pointeur nul est retourné.
11. Elle permet d'obtenir une forme stockée dans le vecteur. La forme à obtenir est spécifiée par son index dans le vecteur. Si l'opération est une réussite, alors la valeur de retour est le pointeur sur la forme en question. Si l'opération est un échec, comme avec un index invalide, un pointeur nul est retourné.
12. Elle permet l'affichage du contenu du vecteur, donc de tous les items. L'endroit à afficher (le flot) est spécifié en paramètre.

A.1.4 Classe Canevas

Un canevas (dessin) permet de gérer un ensemble de couches. Un squelette de cette classe est fourni et elle doit être complétée. La classe *Canevas* est responsable de plusieurs tâches et des contraintes sont imposées :

1. Elle contient un tableau de couches.
2. Lorsqu'un canevas est créé toutes les couches sont dans l'état initialisée, sauf la couche zéro qui elle est active.
3. Elle offre la possibilité de réinitialiser le canevas. Le canevas est donc ramené à son état initial, comme lors de sa création. Une valeur de retour booléenne indique si l'opération est une réussite ou un échec.
4. Elle offre la possibilité d'activer une couche. La couche qui doit changer d'état est spécifiée par son index dans le tableau de couches. Une valeur de retour booléenne indique si l'opération est une réussite ou un échec, comme avec un index invalide.
   **Il y a toujours une seule couche active à la fois. Si une couche était active, elle est désactivée (inactive). L'état des autres couches n'est pas affecté. La couche active est celle qui sera affectée par les opérations d'ajout et de retrait de formes.*
5. Elle offre la possibilité d'ajouter une forme. La forme est ajoutée dans la couche active. Une valeur de retour booléenne indique si l'opération est une réussite ou un échec.
6. Elle offre la possibilité de retirer une forme. La forme est retirée de la couche active. La forme à retirer est spécifiée par son index dans la couche. Une valeur de retour booléenne indique si l'opération est une réussite ou un échec.
7. Elle offre la possibilité d'obtenir l'aire totale du canevas. L'aire totale du canevas est la somme des aires des couches qui la compose. La valeur de retour, l'aire totale, est un nombre réel.
8. Elle offre la possibilité de translater une couche. La couche translatée est la couche active, s'il y en a une. Les valeurs de translation horizontale et verticale, des entiers, sont passées en paramètres. Une valeur de retour booléenne indique si l'opération est une réussite ou un échec.
9. Elle permet l'affichage du contenu du canevas, donc de toutes ses couches et de leurs contenus. L'endroit à afficher (le flot) est spécifié en paramètre. La figure A.2 montre comment un canevas doit être affiché.

A.1.5 Classe Tests

Cette classe permet de gérer les tests de vos développements pour **Graphicus**. Elle sert autant pour faire les tests unitaires, comme tester une classe de forme géométrique, que pour les tests applicatifs de **Graphicus**. Une version initiale est fournie et vous devez l'adapter pour vos besoins en accord avec votre plan de tests et vos tests unitaires.

C'est une bonne idée d'ajouter des méthodes à la classe Tests au fur et à mesure de votre avancement, comme pour vos tests unitaires de la classe cercle et de la classe couche. Ainsi, l'ensemble de vos tests seront conservés tout au long de vos développements. Éventuellement, vous pouvez retourner aux tests antérieurs, si le besoin s'en fait sentir.

Évidemment, la classe de tests peut faire de l'affichage à l'écran afin d'informer l'usager de ce qui se déroule lors de l'exécution et des résultats de tests. Elle permet d'avoir une fonction main très synthétique qui gère les tests que vous voulez effectuer, selon où vous en êtes rendus. Des appels à différentes méthodes de la classe de tests permettent de faire différents tests et il n'est pas nécessaire de toujours exécuter tous les tests.

A.2 Sortie à la console

La figure A.2 montre l'affichage d'un canevas avec plusieurs couches et formes. Cette figure montre la sortie attendue lorsqu'un canevas est affiché et donc aussi ce que devraient afficher les méthodes d'affichage des différentes classes.

```
----- Couche 0
Rectangle(x=2, y=3, l=1, h=1, aire=1)
Carre (x=0, y=0, c=3, aire=9)
Cercle (x=7, y=8, r=6, aire=113.097)
----- Couche 1
Couche initialisee
----- Couche 2
Rectangle(x=5, y=6, l=3, h=4, aire=12)
Carre (x=3, y=4, c=1, aire=1)
Cercle (x=0, y=0, r=3, aire=28.2743)
----- Couche 3
Couche initialisee
----- Couche 4
Couche initialisee
```

Figure A.2 Affichage d'un canevas

**À noter que l'indication « Couche initialisée », indique une couche n'ayant pas de contenu.*

B CODE DU PROGRAMME ITEMS

```
1   class Item
2   {
3   public:
4       Item();
5       Item(int n);
6       ~Item();
7   };
8
9   class Item_simple: public Item
10  {
11      int x;
12  public:
13      Item_simple();
14      Item_simple(int n);
15      ~Item_simple();
16  };
17
18  class Item_complexe: public Item
19  {
20  public:
21      Item_complexe();
22      Item_complexe(int n);
23      ~Item_complexe();
24  private:
25      Item_simple c;
26  };
27
28  Item::Item()
29  { cout << "It::It()" << endl; }
30
31  Item::Item(int n)
32  { cout << "It::It(" << n << ")"
33    << endl; }
34
35  Item::~Item()
36  { cout << "It::~It()" << endl; }
37
38  Item_complexe::Item_complexe()
39  { cout << "Ic::Ic()" << endl; }
40
41  Item_complexe::Item_complexe(int n)
42     : Item(5*n)
43  { cout << "Ic::Ic(" << n << ")"
44     << endl; }
45
46  Item_complexe::~Item_complexe()
47  { cout << "Ic::~Ic()" << endl; }
48
49  Item_simple::Item_simple()
50  { cout << "Is::Is()" << endl; }
51
52  Item_simple::Item_simple(int n)
53  { cout << "Is::Is(" << n << ")"
54     << endl; }
55
56  Item_simple::~Item_simple()
57  { cout << "Is::~Is()" << endl; }
58
59  int main()
60  {
61      Item it;
62      Item_complexe* pitc;
63      Item_complexe itcplx(10);
64      cout << "0------" << endl;
65      it = Item(30);
66      cout << "1------" << endl;
67      pitc = new Item_complexe;
68      cout << "2------" << endl;
69      delete pitc;
70      cout << "3------" << endl;
71      return 0;
72  }
```

C CODE DU PROGRAMME TABLEAU D'ENTIER

```
int main()
{
    const int TAILLE = 20;
    TableauEntier nombres(TAILLE);
    int table[TAILLE];    //+ Ligne ajoutee
    int val, x;
    val = 0;    //+ Ligne ajoutee

    // Sauvegarde des 9 dans le tableau
    // et affiche un asterisque si succes
    for (x = 0; x < TAILLE; x++)
    {
        nombres.setElement(x, 9);
        table[x] = val++;    //+ Ligne ajoutee
        cout << "* ";
    }
    cout << endl;

    // Affichage des valeurs.
    for (x = 0; x < TAILLE; x++)
    {
        val = nombres.getElement(x);
        cout << val << " ";
        cout << table[x] << endl;    //+ Ligne ajoutee
    }
    cout << endl;

    // Tentative hors du tableau.
    nombres.setElement(50, 9);

    return 0;
}
```

D CODE DU PROGRAMME VÉHICULES

```
1   class Vehicule
2   {
3   public:
4       virtual void modele() const;
5       void nserie() const;
6   };
7
8   class Automobile : public Vehicule
9   {
10  public:
11      virtual void modele() const;
12      void nserie() const;
13  private:
14      int x;
15  };
16
17  class Cabriolet : public Automobile
18  {
19  public:
20      void nserie() const;
21  private:
22      int y;
23  };
24
25  void Vehicule::modele() const
26  { cout << "vehicule" << endl; }
27
28  void Vehicule::nserie() const
29  { cout << "v000" << endl; }
30
31  void Automobile::modele() const
32  { cout << "automobile" << endl; }
33
34  void Automobile::nserie() const
35  { cout << "a000" << endl; }
36
37  void Cabriolet::nserie() const
38  { cout << "c000" << endl; }
```

```
39
40  int main()
41  {
42      Vehicule v;
43      Automobile a;
44      Cabriolet c;
45      Vehicule *pv = new Vehicule;
46      Vehicule* pvc = new Cabriolet;
47      Automobile *pa = new Automobile;
48      Cabriolet *pc = new Cabriolet;
49      cout << "v----" << endl;
50      v.modele();
51      v.nserie();
52      cout << "a----" << endl;
53      a.modele();
54      a.nserie();
55      cout << "c----" << endl;
56      c.modele();
57      c.nserie();
58      cout << "pv----" << endl;
59      pv->modele();
60      pv ->nserie();
61      cout << "pvc----" << endl;
62      pvc->modele();
63      pvc ->nserie();
64      cout << "pa----" << endl;
65      pa->modele();
66      pa ->nserie();
67      cout << "pc----" << endl;
68      pc->modele();
69      pc ->nserie();
70
71      return 0;
72  }
```

E CODE D'ENSEMBLE DE NOMBRES

```
class EnsembleDeNombres
{
public:
    EnsembleDeNombres(int laCapacite = 10);
    ~EnsembleDeNombres();
    void afficher() const;
    void ajoutFin(double);
    void supprimerDebut();
private:
    double *tableau;
    int taille;
    int capacite;
};

int main()
{
    EnsembleDeNombres ensemble;
    ensemble.ajoutFin(0.1);
    ensemble.ajoutFin(1.2);
    ensemble.ajoutFin(2.3);
    ensemble.afficher();
    return 0;
}
```

LISTE DES RÉFÉRENCES

[1] B. Charroux, A. Osmani, et Y. Thierry-Mieg, Langage UML 2 : Pratique de la modélisation, 3e édition, séries Collection Synthex. Pearson Education, 2010.

[2] S. Rao, Sams Teach Yourself C++ in One Hour a Day, 8e édition. Sams publishing, Pearson Education, 2017.