UNIVERSITÉ DE
SHERBROOKE

MODULE - Programmation

Guide de l'étudiant

APP3 - S3

GIF 350 - Modèles de conception - *Design patterns*

Semaines x et y

Département de génie électrique et de génie informatique
Faculté de Génie

2020

Note : En vue d'alléger le texte, le masculin est utilisé pour désigner les femmes et les hommes.

Document guideGIF350.pdf

L'APP existe depuis l'été 2018, il y a eu des modifications mineures: correction de fautes d'orthographe et amélioration du texte.

Version 2020.1en 2020, par: Ruben Gonzalez-Rubio et Eugène Morin

Réalisé à l'aide de LaTeX, TeXShop, et TeXnicCenter

Contents

"*The object-oriented version of spaghetti code is, of course, 'lasagna code'. Too many layers.*" Roberto Waltman

"*Good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun.*" Linus Torvalds

"*A good programmer is someone who looks both ways before crossing a one-way street.*" Doug Linder

"*Simplicity and elegance are unpopular because they require hard work and discipline to achieve and education to be appreciated.* Edsger Dijkstra

1 Éléments de compétences visés par l'unité

GIF 350 - Modèles de conception - *Design patterns*

- Compétences
  1. Maîtriser les concepts de la programmation orientée objet et appliquer ces concepts pour produire un logiciel de qualité;
  2. Identifier de situations où un *modèle de conception* peut être utilisé;
  3. Valider et implémenter un logiciel en utilisant les *modèles de conception*.
- Contenu
  - Les concepts nécessaires pour maîtriser la programmation orientée objet.
  - Les concepts nécessaires pour maîtriser les *modèles de conception*
  - Bonnes pratiques et méthodologie de développement.

2 Énoncé de la problématique

La compagnie *s3I* doit développer l'application : **EMR Application**, qui apparaît comme un bon moyen pour gagner de l'argent.

Comme vous venez d'être embauché, votre chef: Tim, vous demande de faire des prototypes en utilisant des *modèles de conception*. Il veut connaître vos compétences dans ce domaine.

Votre équipe va devoir ajouter de nouvelles fonctionnalités au logiciel et devra s'occuper de sa maintenance tout au long de sa durée de vie. Avant de continuer le développement, *s3I* veut faire un premier audit afin de s'assurer de garder son image de marque dans les développements de haute qualité et aussi, planifier la suite.

L'application devra être écrite en Java, en utilisant le framework JavaFX et les *modèles de conception*.

Pour démarrer, il faut que l'application puisse avoir une interface utilisateur « classique », voir Figure 3. Le logiciel va dans un premier temps avoir une surface de travail (*canvas*) où un concepteur de modèle **EMR** devra être en mesure de dessiner un schéma semblable à celui de la figure 1.

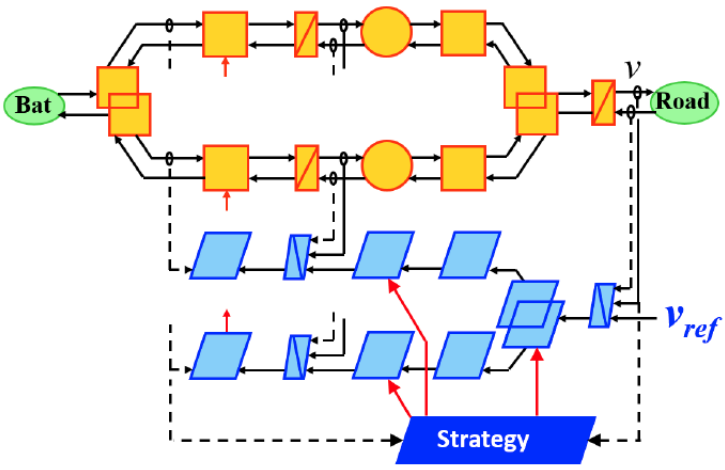

Figure 1: Exemple de schéma

Afin de pouvoir réaliser le dessin, il faudra offrir, sur le côté de la surface de dessins, une palette présentant les « composants » de dessin disponibles. Ces éléments sont présentés dans la figure 2.

Il sera nécessaire de consulter le document en annexe **??** et sur la page Web qui présente les dimensions des composants à intégrer. Les composants doivent être dessinés à l'aide de JavaFX, afin de respecter les spécifications.

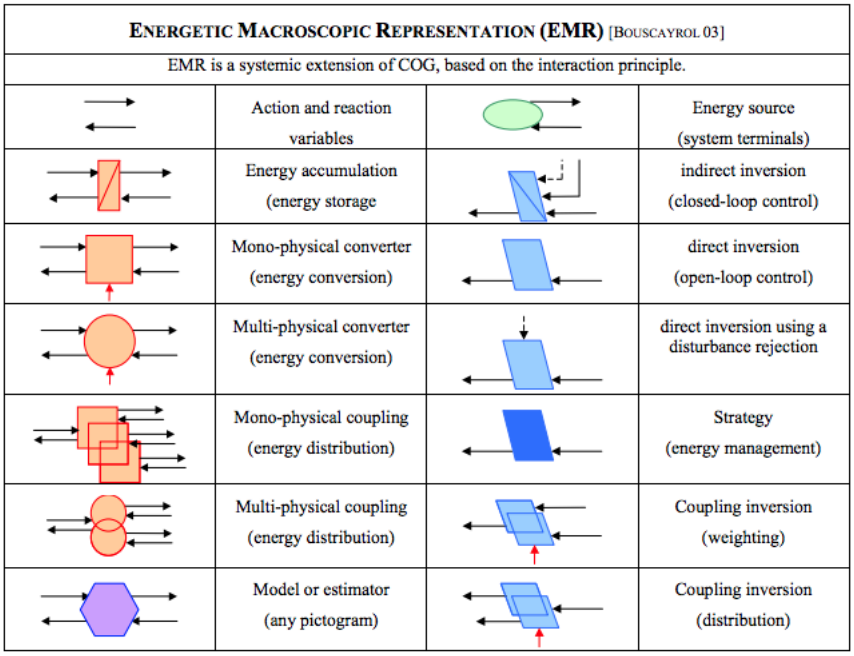

Figure 2: Les éléments pour constituer un schéma

Avant de commencer les développements, il faut s'assurer de bien maîtriser les concepts suivants:

- Le langage de programmation Java, Eclipse, la création d'un projet, sa compilation et le développement en équipe. (`git` ou `svn` sont optionnels).
- Le concept de classe et le concept d'objet, ainsi que leurs relations.
- Les classes abstraites, les classes concrètes, les classes imbriquées (*nested*), les classes imbriquées anonymes, les interfaces et le lambda.
- Les principes de base de la programmation orientée-objet (OOP): abstraction, encapsulation, héritage et polymorphisme.
- La compilation et l'exécution d'un programme en Java.
- La notation UML.
- Les dépendances entre classes.
- JavaFX.

Les *modèles de conception* "officiels[1]" viennent du livre [FR04] on peut aussi s'inspirer de ceux présentés sur les sites suivants:

- Iterator Pattern :
  `https://www.oodesign.com/iterator-pattern.html`
- Composite Pattern :
  `https://www.oodesign.com/composite-pattern.html`
- Command Pattern :
  `https://www.oodesign.com/command-pattern.html`
- Factory Pattern :
  `https://www.oodesign.com/factory-pattern.html`
- MVC Pattern :
  `https://www.upgrad.com/blog/mvc-architecture-in-java/`
- Observer Pattern :
  `https://www.oodesign.com/observer-pattern.html`
- Chain of responsability Pattern :
  `https://www.oodesign.com/chain-of-responsibility-pattern.html`
- Strategy Pattern :
  `https://www.oodesign.com/strategy-pattern.html`
- State Pattern :
  `https://www.tutorialspoint.com/design_pattern/state_pattern.htm`

Bien entendu, en deux semaines il est impossible de réaliser le logiciel au complet. C'est pourquoi il vous est uniquement demandé de développer plusieurs prototypes (4) montrant quelques fonctionnalités dans chaque prototype. Dans certains cas un prototype peut réutiliser du code d'un autre prototype. En plus de 4 prototypes, il a un rapport à faire, l'annexe C indique avec précision le contenu du rapport. Aussi, il y a la validation à faire, voir D.

Ceci permettra plus tard de définir l'architecture de l'application **EMR**. Cette façon de développer vient des concepts « agile-scrum ». Les prototypes permettent de se familiariser avec les problèmes, à trouver des solutions et à simplifier les intégrations.

De plus, le projet devra être réalisé dans le respect des concepts, vous devrez d'abord valider la faisabilité de l'application en prenant en compte, quelques problèmes critiques possibles.

1. Officiels veut dire ceux qui seront pris pour corriger les travaux de l'APP.

Voici le fonctionnalités de chaque prototype :

1. Dessiner de composants dans l'espace de dessin.
   - Réaliser la conception de l'interface utilisateur du **EMR Application**, voir Figure 3, qui devra avoir :
     - un menu classique,
     - une barre d'outils,
     - un espace de travail pour le modèle/dessin,
     - une barre indiquant l'état où est le logiciel, qui est le status bar,
     - et une palette pour choisir la pictogramme à dessiner sur le schéma.

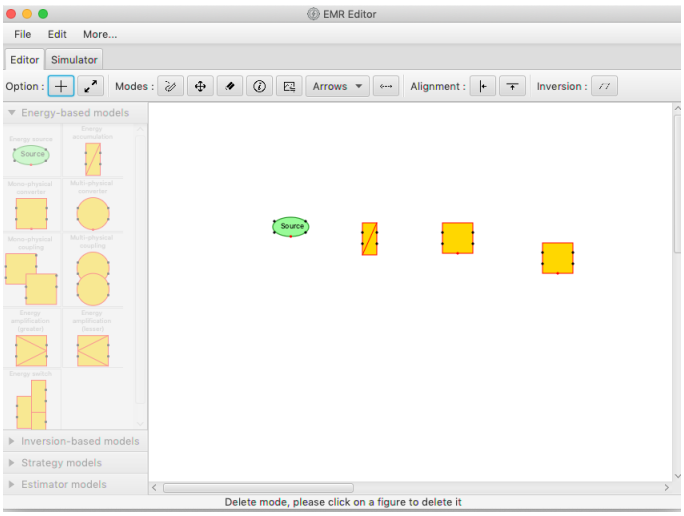

Figure 3: L'interface de l'application EMR

2. Dessiner de composants et de flèches, qui vont connecter les composants
3. Arranger les composants et flèches pour avoir de diagrammes avec un espacement équivalent entre les composants, les flèches doivent être horizontales. Ou faire do et undo.
4. Faire la sauvegarde de diagrammes dans de fichiers, aussi ouvrir un fichier permettant de retrouver un diagramme dans l'espace de travail.

La validation, voir annexe D:

- Vous devrez, en équipe, présenter l'ensemble des développements réalisés et démontrer que les fonctionnalités demandées sont opérationnelles.

3 Connaissances à acquérir par la résolution de cette problématique

Connaissances déclaratives: QUOI

- Utiliser les principes de base de la programmation orientée objet.
- Utiliser la notation UML. Savoir lire et savoir écrire des diagrammes UML.
- Connaître plusieurs *modèles de conception*.
- Expliquer comment implémenter un *modèle de conception*.

Connaissances procédurales: COMMENT

- Savoir comment organiser un projet Java avec JavaFX.
- Savoir implémenter un *modèle de conception* dans un projet.
- Savoir comment indiquer les liens entre les classes formant un *modèle de conception*.
- Savoir comment tester du code qui utilise le framework JavaFX.
- Travailler en équipe sur un projet logiciel.
- Effectuer l'implémentation d'un *modèle de conception* avec un minimum de classes auxiliaires pour le démontrer.

Connaissances conditionnelles: QUAND

- Savoir reconnaître les situations qui nécessitent l'utilisation d'un *modèle de conception* ou des situations où cela n'est pas nécessaire.
- Savoir ajouter un test pour éviter la régression dans le code.
- Savoir quand ajouter un test pour éviter la régression dans le code.
- Savoir intégrer la nouvelle fonctionnalité dans un code au bon endroit et en respectant les principes de la POO.

4 Références

Clean Code [Mar09]

Clean Architecture [Mar18]

Test-Driven Development [Bec02]

Head First : Design Patterns [FR04]

Refactoring [Fow99] et `http://www.refactoring.com/catalog/index.html`.

`http://rcardin.github.io/programming/oop/dependency-dot.html`

`https://fr.wikipedia.org/wiki/Diagramme_de_classes`

`https://en.wikipedia.org/wiki/Class_diagram`

Les sites présentés dans la section 2.

References

[Bec02] K. Beck. *Test-Driven Development*. Addison-Wesley, 2002.

[Fow99] M. Fowler. *Refactoring. Improving the Design of Existing Code*. Addison-Wesley, 1999.

[FR04] E. Freeman and E. Robson. *Head First Design Patterns*. O'Reilly, 2004.

[Mar09] R. C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice-Hall, 2009.

[Mar18] R. C. Martin. *Clean Architecture, A Craftsman's Guide to Software Structure and Design*. Prentice-Hall, 2018.

5 Sommaire des activités liées à la problématique

Activités de la semaine 1:

Si à l'horaire il n'y a pas d'activité indiquée, les activités peuvent être : étude personnelle ou travail en équipe ou travail pour le projet.

L'horaire sur le Web est plus à jour que ce document, à consulter en tout temps pendant l'APP.

- *Lundi* - Première rencontre de tutorat. Préparation des activités.
- *Mercredi* - Formation à la pratique procédurale: Programmation Java et programmation orientée-objet, les *modèles de conception*.
- *Mercredi* - Formation à la pratique en laboratoire: Implémentation de deux *modèles de conception*.
- *Vendredi* - Formation à la pratique procédurale: Étude de plusieurs *modèles de conception*.

Activités de la semaine 2:

- *Lundi* - Consultation facultative. Aide à faire marcher les projets de l'APP.
- *Mardi* - Validation. Validation du bon fonctionnement des projets demandés.
- *Mercredi* - Deuxième rencontre de tutorat.
- *Vendredi* - Consultation facultative pour l'examen.
- *Vendredi* - Dépôt de travaux.
- *Vendredi* - Évaluation sommative.

6 Semaine 1 - Procédural 1

Prérequis:

Lecture de documents.

- Lectures pour comprendre les concepts de base de la programmation orientée-objet.
  - Le langage de programmation Java, Eclipse, la création d'un projet, sa compilation et le développement en équipe. (`git` ou `svn` sont optionnels).
  - Le concept de classe et le concept d'objet, ainsi que leurs relations.
  - Les classes abstraites, les classes concrètes, les classes imbriquées (*nested*), les classes imbriquées anonymes, les interfaces et le lambda.
  - Les principes de base de la programmation orientée-objet (OOP): abstraction, encapsulation, héritage et polymorphisme.

  - La compilation et l'exécution d'un programme en Java.
  - La notation UML.
  - Les dépendances entre classes.
  - JavaFX.

- Clean Code;
- `http://rcardin.github.io/programming/oop/dependency-dot.html`.
- `https://docs.oracle.com/en/java/javase/11/`
- Pour Java FX :
  - `http://code.makery.ch/library/javafx-8-tutorial/part1/`
  - `https://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm`
- Pour les modèles de conception :
  - Le livre Head First : Design Patterns [FR04].
  - Les liens présentés dans l'énoncé.
  - `https://users.encs.concordia.ca/home/PDF/se_design_patterns.pdf`
  - Les sites de Design Patterns, dans la section 2.

But de l'activité

Comme introduction, nous étudions le langage Java. Depuis le concept de classe et d'objet jusqu'à la programmation en Java FX. Les principes de base de la programmation orientée-objet: abstraction, encapsulation, héritage et polymorphisme. Le tout en utilisant la notation UML.

6.1 Préalable aux développements, notion de qualité

Donnez une définition de la qualité.

Donnez une définition de la qualité interne et externe d'un logiciel.

Pourquoi faire un code de qualité?

Est-ce qu'il est possible de développer un code de qualité, sans maîtriser les concepts de la programmation?

Est-ce que faire un code de qualité va coûter plus cher? Avantages et désavantages.

Quelle est l'utilité des tests? Comment faire des tests avec `JUnit`.

Quoi tester.

6.2 Concepts de base

Décrivez les différences entre un langage procédural et un langage orienté objet.

Décrivez comment un code Java est un langage orienté objet.

Décrivez comment un code Java est compilé et exécuté, Notions de: compilation, éditeur de liens, machine virtuelle, machine réelle et langage assembleur.

Comment on reconnaît que Java est-il un langage fortement typé? Donnez des avantages et des désavantages de cet aspect.

Comment faire cohabiter `int` et `Integer`.

Donnez la différence entre classe et objet.

Donnez une définition pour : les classes abstraites, les classes concrètes, les classes imbriquées *nested*, les classes anonymes, les interfaces et les lambda.

Expliquez la phrase: Java is a general-purpose programming language that is class-based, object-oriented, and designed to have as few implementation dependencies as possible. Le mot clé ici est dépendencies.

6.3 Principes

Les principes de base de la programmation orientée objet sont: abstraction, encapsulation, polymorphisme et héritage.

Donnez des exemples d'utilisation de chaque principe.

Donnez des exemples d'utilisation de chaque principe en utilisant la notation UML.

Comment peut-on identifier une classe avec des problèmes d'encapsulation?

Quels sont les symptômes d'une mauvaise conception?

Définissez la liaison dynamique (*dynamic binding*). Quand devrait-elle être utilisée?

Décrivez comment un code Java est compilé et exécuté, dans le cas précédent.

6.4 Le *modèle de conception* Strategy

Exemple provenant du premier chapitre de [FR04].

6.5 Les *modèles de conception*

6.5.1 Le *modèle de conception*: *Iterator*

Le *modèle de conception*: *Iterator*.

6.6 Java FX

Quelles sont les notions de base du framework FX servant à développer un logiciel?

Définissez les notions d'`Application`, `Stage`, `Scene` et `Nodes`?

Montrer comment JavaFX implémente le *modèle de conception* MVC.

Définissez les notions d'événement `Event` et d'interface `EventHandler`?

Définissez les mécanismes de `setOnAction()`

Comment peut-on interfacer l'application et la souris?

7 Semaine 1 - Laboratoire 1

But de l'activité: Réaliser un projet Java, JavaFX, coder d'un *modèle de conception* à partir d'un schéma UML

Il faut vérifier que tous les étudiants aient un ordinateur avec eclipse et que les programmes suivants Puissent être exécutés.

Projet Java

Classique : HelloWorld, notion de projet, notion de package, compilation et exécution.

Projet JavaFX

Première partie d'un projet JavaFX site:
`http://code.makery.ch/library/javafx-8-tutorial/part1/`. Scene Builder, Vue, controller, main, exécution.

Factory

Note: Un schéma UML sera fourni sur le site web de l'app (voir le fichier : GIF350-Labo1_FactoryUML.pdf)

Le code à réaliser est basé sur le *modèle de conception* Factory.

Préciser les fonctionnalités à implémenter et définir une stratégie pour réussir leur implémentation.

Les étapes.

1. Définir le problème mettant en oeuvre le *modèle de conception* Factory,
2. Bien comprendre le *modèle de conception* Factory, identification des classes et des interfaces, ainsi que leurs interactions.
3. Dans l'environnement de Eclipse, créer un projet de type « JavaFX », puis assurez vous que l'Application peut être exécutée (devrait afficher une fenêtre vide).
4. Débutez par l'ajout de « packages » au répertoire « src » du projet. Respecter les noms donnés dans le fichier UML fourni selon le *modèle de conception* MVC (voir les noms entre parenthèses « ... »).
5. Commencer par coder les classes concrètes et la classe abstraite du package « models ». Tester la classe factory à partir de la méthode « main » ou via des tests unitaires avec « JUnit ».
6. Ajouter le fichier « fxml » au package « views », puis à l'aide de SceneBuilder créer l'interface utilisateur (voir le fichier UML fourni pour un aperçu un l'interface à réaliser).
7. Modifier l'`Application` afin de s'assurer que l'interface utilisateur, nouvellement créé, s'affiche correctement.
8. Ajouter la classe requise au package « controllers », puis lier les événements de bouton et les champs textes de l'interface utilisateur (fxml) avec cette classe.
9. Vérifier que les événements de bouton sont capturés par les méthodes dédiées dans la classe du package « controllers ».
10. Ajouter les appels entre la classe « FactoryController » et la classe « ShapeFactory », puis afficher le retour d'information requis dans le champ texte dédié de l'interface utilisateur.

Analyse du projet :

1. L'organisation du projet.
2. Le framework JavaFX.
    - Les actions.
    - La gestion d'erreurs, exceptions.
    - Liens entre fichiers `.fxml` et `controlleurs`
    - Autres

8 Semaine 2 - Procédurale 2

Prérequis:

Lecture de documents

- Lectures pour comprendre les concepts de base de la programmation orientée-objet.
    - Le langage de programmation Java, Eclipse, la création d'un projet, sa compilation et le développement en équipe. (`git` ou `svn` sont optionnels).
    - Le concept de classe et le concept d'objet, ainsi que leurs relations.
    - Les classes abstraites, les classes concrètes, les classes imbriquées (*nested*), les classes imbriquées anonymes, les interfaces et le lambda.
    - Les principes de base de la programmation orientée-objet (OOP): abstraction, encapsulation, héritage et polymorphisme.
    - La compilation et l'exécution d'un programme en Java.
    - La notation UML.
    - Les dépendances entre classes.
    - JavaFX.
- Le livre Head First : Design Patterns [FR04].
- Clean Code;
- `http://rcardin.github.io/programming/oop/dependency-dot.html`.
- `https://docs.oracle.com/en/java/javase/11/`
- Les sites de Design Patterns énumérés dans l'énoncé et dans la section 2.

But de l'activité

Ce procédural étudie les *modèle de conception*: `MVC`,`Command`, `Observer`, `Chain of Responsibility`, `State` et `Strategy`.

8.1 Retour sur les *modèles de conception*

8.2 Le *modèle de conception* `Iterator`

À quoi sert-il?

À quel type de problème s'applique-t-il?

8.3 Le *modèle de conception* `Composite`

À quoi sert-il?

À quel type de problème s'applique-t-il?

8.4 Le *modèle de conception* `Factory`

À quoi sert-il?

À quel type de problème s'applique-t-il?

8.5 Lecture et compréhension d'un *modèle de conception*

Quelles sont les sections à lire dans une description d'un *modèle de conception*.

8.6 Le *modèle de conception* `MVC`

À quoi sert-il?

À quel type de problème s'applique-t-il?

Avantages et désavantages?

Variations.

8.7 Le *modèle de conception* `Command`

À quoi sert-il?

À quel type de problème s'applique-t-il?

Avantages et désavantages?

Variations.

8.8 Le *modèle de conception* `Observer`

À quoi sert-il?

À quel type de problème s'applique-t-il?

Avantages et désavantages?

Variations.

8.9 Le *modèle de conception* `Chain of Responsibility`

À quoi sert-il?

À quel type de problème s'applique-t-il?

Avantages et désavantages?

Variations.

8.10 Le *modèle de conception* `State`

À quoi sert-il?

À quel type de problème s'applique-t-il?

Avantages et désavantages?

Variations.

8.11 Le *modèle de conception* `Strategy`

À quoi sert-il?

À quel type de problème s'applique-t-il?

Avantages et désavantages?

Variations.

9 Évaluation de l'unité

La note attribuée aux activités pédagogiques de l'unité est une note individuelle (sommatif et final) et par équipe (code, rapport et validation). L'évaluation portera sur les compétences figurant dans la description des activités pédagogiques. Ces compétences, ainsi que la pondération de chacune d'entre elles dans l'évaluation de cette unité, sont :

Comme indiqué à la section 1:

- l'activité GIF 350 comprend 3 éléments de compétences qui seront désignés par 1, 2, 3.

Il a trois livrables : le code, le rapport et la validation

Voir dates et heures dans la section 13.

Les qualités et critères du BCAPG concernant cet APP:

- Qualité 2 (Q2) Analyse de problèmes.
  - Qualité 02 (Q2_1) Définir un problème applicant les concepts a programmation orientée objet.
  - Qualité 02 (Q2_5) Mettre en oeuvre la solution retenue, bonne utilisation d'un *modèle de conception*.
- Qualité 4 (Q4) Conception.
  - Qualité 04 (Q4_5) Faire la conception détaillée.
  - Qualité 04 (Q4_6) Valider et implémenter la solution.

9.1 Code

L'annexe B précise comment le code doit être déposé et les critères demandés.

Le code sera évalué comme précisé ci-dessous en se basant sur les qualités du BCAPG:

Q2: critère 1. Application de concepts de la programmation orientée objet	60 pts
- considère tous les paramètres nécessaires à la résolution du problème.	100 %
- considère la majorité des paramètres nécessaires à la résolution du problème et réalise qu'il en manque pour le solutionner, sans toutefois pouvoir les identifier.	85 %
- considère quelques paramètres nécessaires à la résolution du problème, sans toutefois réaliser que ce dernier ne pourra être solutionné.	60 %
- considère divers paramètres sans que ceux-ci ne puissent conduire à la résolution du problème.	25 %
- no fait.	0 %

Q4: critère 5. Faire de la conception détaillée	100 pts
- définit l'architecture de la solution. Raffine le concept pour obtenir une conception détaillée et une implémentation, en s'appuyant sur les règles de l'art. Justifie ses choix en fonction des exigences identifiées.	100 %
- définit l'architecture de la solution. Raffine le concept pour obtenir une conception détaillée et une implémentation, en s'appuyant sur quelques règles de l'art. Justifie ses choix.	85 %
- raffine le concept pour obtenir une conception détaillée et une implémentation, sans respecter des règles	60 %
-passe rapidement du concept à une conception détaillée et à l'implémentation, avec des erreurs.	25 %
- no fait.	0 %

9.2 Rapport

L'annexe **??** précise comment le rapport doit être déposé et les critères demandés. À rendre voir section 13. Le rapport sera évalué comme précisé ci-dessous en se basant sur les qualités du BCAPG:

Q2: critère 1. Application de concepts de la programmation orientée objet	20 pts
- considère tous les paramètres nécessaires à la résolution du problème.	100 %
- considère la majorité des paramètres nécessaires à la résolution du problème et réalise qu'il en manque pour le solutionner, sans toutefois pouvoir les identifier.	85 %
- considère quelques paramètres nécessaires à la résolution du problème, sans toutefois réaliser que ce dernier ne pourra être solutionné.	60 %
- considère divers paramètres sans que ceux-ci ne puissent conduire à la résolution du problème.	25 %

Q4: critère 6. Solution retenue ayant une bonne utilisation d'un *modèle de conception*	20 pts
- met en oeuvre la solution retenue en se référant aux connaissances et aux principes de la programmation orientée objet, nécessaires pour ce faire et en tenant compte de plusieurs autres principes pertinents.	100 %
- met en oeuvre la solution retenue en se référant à la plupart des connaissances et des principes aux principes de la programmation orientée objet,	85 %
- met en oeuvre la solution retenue en se référant aux principes de la programmation orientée objet.	60 %
- n'est pas en mesure de mettre en oeuvre la solution retenue, mais est capable d'expliquer pourquoi.	25 %

9.3 Validation

L'annexe D précise les critères à valider. À faire pendant la période indiquée dans la section 13. Un document précisant ce qui est attendu à la validation sera disponible sur la page Web.

La validation sera évaluée comme précisé ci-dessous en se basant sur une qualités du BCAPG:

Q4: critère 6. Valider et implémenter la solution	20 pts
- élabore un prototype, de grande qualité. Valide adéquatement que la solution répond aux exigences. Propose des améliorations. - élabore un prototype. Valide adéquatement que la solution répond presque à toutes exigences. - élabore un prototype. Valide partiellement que la solution répond aux exigences. - élabore un prototype. Ne valide pas que la solution répond aux exigences.	100 % 85 % 60 % 25 %

10 Évaluation sommative

Elle portera sur les compétences de l'APP et sur l'atteinte des objectifs d'apprentissage. Il y aura des questions théoriques et du développement de codes en Java. Les références et le matériel de procéduraux et de laboratoires et du rapport fera partie de l'examen.

11 Évaluation finale

Elle portera sur les compétences de l'APP et sur l'atteinte des objectifs d'apprentissage. Il y aura des questions théoriques et du développement de codes en Java. Les références et le matériel de procéduraux et de laboratoires et du rapport fera partie de l'examen.

12 Tableau récapitulatif de points pour les évaluations.

GIF 350 - Modèles de conception - *Design patterns*

Activités et éléments de compétence	*Code*	*Rapport*	*Validation*	*Sommatif*	*Final*
1. Maîtriser les concepts de la programmation orientée objet	60	20		50	70
2. Identifier de situations où un *modèle de conception* peut être utilisé		20		50	70
3. Valider et implémenter un logiciel avec de *modèles de conception*	100		20	60	80
Total: GIF 350 (**600**)	160	40	20	160	220

13 Dates et heures pour les livrables

Livables GIF350	*Mode de livraison*	*Date*	*Heure*
Validation	toute l'équipe à distance	mar 2 juin 2020	13h30
Code	selon procedure de dépôt	ven 5 juin 2020	13h00
Rapport	selon procedure de dépôt	ven 5 juin 2020	13h00

14 Grille d'assignation de notes

Note	Points
A+:	90
A:	86
A-:	82
B+:	78
B:	75
B-:	71
C+:	67
C:	62
C-:	58
D+:	54
D:	50
E:	0

A Les règles graphiques of EMR

Le document Graphical rules of EMR. continent tous les renseignements pour coder les composants de base de la EMR.

Ici il y a un sous-ensemble pour appliquer à l'APP.

A.1 Couleurs RGB et HEX des composants

Figure 4: - Liste des 6 composants en couleur

1. Energy source control (system terminals) :
   - light green background: RGB = (152,251,152) = #98FB98
   - dark green border (size b pt): RGB = (0,128,0) = #008000

2. Energy-based control :
   - orange background: RGB=(255,215,0) = #FFD700
   - red border (size b pt): RGB = (255,0,0) = #FF0000

3. Model or estimator control :
   - purple background: RGB=(238,130,238) = #EE82EE
   - dark blue border (size b/2 pt): RGB = (0,0,255) = #0000FF

4. Inversion-based control :
   - light blue background: RGB (135,206,235) = #87CEEB
   - dark blue border (size b/2 pt): RGB = (0,0,255) = #0000FF

5. Strategy control (energy management):
   - dark blue background: RGB = (0,0,255) = #0000FF
   - dark blue border (size b/2 pt): RGB = (0,0,255) = #0000FF

6. Flèches (noir et rouge):
    - black arrow: RGB = (0,0,0) = #000000
    - red arrow: RGB = (255,0,0) = #FF0000

A.2 Dimension des composants

Les composants à utiliser sont les pictogrammes pour *power elements* et *control elements* Sauf les *switching elements*.

Les flèches doubles servent à connecter les *power elements* et le flèches rouges à connecter des *control elements* ou un *control element* à un *power element*. Le flèches rouges sont une seule ligne, on clique sur le composant du départ et ensuite sur le composant d'arrivée. La pointe de la flèche va pointer sur le composant d'arrivée.

B Le Code

Le livrable: code doit respecter les consignes suivantes:

Vous devez avoir 4 projets, ces projets doivent s'appeler eXXp1, eXXp2, eXXp3 et eXXp4. Le lettres XX doit être le numéro d'équipe. Les projets doivent contenir uniquement les développements demandés.

Comme vous allez déposez votre code. Il faut que: le fichier contenant les projets, doit se nommer gif350aeXX, sans accent et XX le numéro de l'équipe. Le code doit être déposé dans la boîte de dépôt.

Le non-respect de ces consignes donnera une perte de 6 points.

Il est interdit d'utiliser Maven.

Les projets ne doivent pas contenir de fichiers de git ou de svn.

B.1 Organisation de projets

Chaque projet doit être construit en utilisant gradle 6.4. Normalement chaque projet doit ouvrir avec eclipse (vous pouvez le tester).

À la correction, si le projet n'ouvre pas sur eclipse. Un nouveau projet gradle sera fait. Ainsi, les fichiers `build.gradel` et `settings.gradle` seront copiés.

Les répertoires `src/main/java`, `src/main/resources`, `src/test/java` et `src/test/resources` seront aussi copiés.

Aucune autre manipulation ne sera faite pour pouvoir corriger votre code

L'application de la programmation orientée-objet et la bonne implémentation de *modèle de conception* sera prise en compte pour l'évaluation du code. Ainsi que la bonne implémentation de fonctionnalités.

B.2 Le code et Clean code

Le code doit être lisible, et respecter autant que possible les consignes données dans le livre "Clean code" [Mar09].

B.3 Le code et le *modèles de conception*

Dans votre code, lorsqu'un ensemble de classes forme un modèle de conception. Vous devez mettre en commentaire avant la déclaration de la classe.

// This class makes part of the design pattern X together with class Y, class Z ...

X doit prendre le nom du design patter correspondant. Y et Z sont des noms de classes.

Des fois, une classe peut faire parie de deux ou plus que deux design patterns. Un commentaire par appartenance est nécessaire.

B.4 Les projets

Les requis de projets sont détaillés dans l'annexe validation D. Les projets doivent faire ce qui est demandé, il est impossible de donner tous les détails de fonctionnement. Dans la correction du code, il y a une nouvelle validation (quelques points sont accordés), tout doit marcher, une autre partie de la correction vérifie que les mdcs sont bien implémentés.

B.5 Les points

Prototypes	Comp 1	Comp 2
fichiers bien organisés	6 points	6 points
prototype 1	14 points	24 points
prototype 2	10 points	20 points
prototype 3	10 points	20 points
prototype 4	20 points	30 points
Total	60 points	100 points

C Rapport

Le livrable: rapport doit contenir entre autres:

1. Une section contenant les diagrammes de classes (UML) de modèles de conception utilisés dans votre code. Les digrammes et le code doivent avoir exactement la même information.

   Les diagrammes doivent indiquer dans quel projet se trouve le modèle de conception. Si un modèle de conception est présent dans plus qu'un projet, il doit être présenté une seule fois indiquant les projets où il est utilisé.

   Les diagrammes doivent respecter les règles d'utilisation d'UML.

2. Une section sur les dépendances. Expliquer les dépendances. Expliquer comment vous avez évité certaines dépendances dans votre code. Donnez deux exemples de modèles de conception, permettant d'éviter de dépendances.

3. Une section sur les format de données.

   Donnez le format de données pour les fichiers .xml et texte.

Le rapport doit être dépose sur la boîte de dépôt.

D Validation

1. À la validation, toute l'équipe doit être présente.

2. Vous devez avoir les quatre applications prêtes à être évaluées.

3. Vous pouvez utiliser un ou plusieurs ordinateurs pour faire la validation.

Les projets à montrer doivent respecter les consignes indiquées pour le code, en particulier noms et organisation de projets.

Les projets

1. Dessiner de composants dans l'espace de dessin.
2. Dessiner de composants et de flèches, qui vont connecter les composants
3. Arranger les composants et flèches pour avoir de diagrammes avec un espacement équivalent entre les composants, les flèches doivent être horizontales.
4. Faire la sauvegarde de diagrammes dans de fichiers, aussi ouvrir un fichier permettant de retrouver un diagramme dans l'espace de travail.

D.1 Dessiner de composants

1. Les requis de développement sont les suivants:
   - Réaliser la conception de l'interface utilisateur du *EMR*, qui devra avoir :
     - un menu classique,
     - une barre d'outils,
     - un espace de travail pour le modèle/dessin,
     - une barre indiquant l'état où est le logiciel, qui est le status bar,
     - et une palette pour choisir le pictogramme à dessiner sur le schéma.
   - Utiliser *scene builder* afin de définir la vue. Coder le contrôleur et le modèle.
   - Démontrer que le menu fonctionne. Soit lorsqu'une option du menu est sélectionnée, afficher une indication décrivant de ce que l'option pourrait faire comme tâche sur la status bar.
   - Démontrer que la barre d'outils fonctionne: lorsqu'un outil est sélectionné, afficher une indication de ce que l'outil pourrait faire comme tâche sur le status bar,
   - Offrir la possibilité de prendre un dessin de la palette et le déposer sur la surface de travail.
   - Pouvoir dessiner ou tracer des flèches sans connexion avec les composants.

D.2 Dessiner de flèches

1. Faire un menu flèches où il peut y avoir de flèches doubles (Noires) ou de flèches simples(rouges).
2. Utilisez la barre d'outils ou utilisez la barre de menu, il doit y doit avoir le *modèle de conception* State Pattern. Selon l'état, il est possible de dessiner des composants, dessiner de flèches rouges ou noires. Il faut réaliser un démonstrateur qui permettra l'exécution de certaines actions, en fonction de l'état de l'application. Par exemple, lorsque l'application est dans un état dessin, il sera alors possible de prendre un dessin de la palette et l'amener sur la surface de travail. Lorsque l'application passera à l'état connexion, il sera alors possible de choisir deux composants et de les relier avec une connexion avec deux flèches. Les flèches doivent toucher la bordure des composants, pas aller à l'intérieur du composant, pas à l'extérieur du composant.
    - Faire la conception d'un automate d'états finis analysant les états nécessaires pour l'application.
    - Coder le *modèle de conception* State Pattern, avec uniquement une barre d'outils comme interface de l'application *EMR*, ensuite, afficher les actions effectuées et indiquer lorsqu'il y a un passage d'un état à un autre.

D.3 Arranger les composants et flèches ou Undo-Redo

1. Faire un menu où il peut y avoir des arrangements du dessin. Par exemple, lorsque l'espace de travail contient plusieurs composants et flèches, et que le menu arrange est activé. Les composants seront distribués dans un espace raisonnable, et les flèches seront droites (à l'horizontale). On doit pouvoir sélectionner des composants et flèches pour constituer de groupes qui peuvent s'aligner.
2. Code un *modèle de conception* Composite. L'utiliser pour faire de regroupements.

Ou

1. Faire un menu où il peut y avoir und et redo. Si la commande undo est exécutée la dernière action faite par l'utilisateur est défaite, si un redo est exécuté la action "efffacée" par l'undo est faite. Il doit être possible de revenir à une surface de travail vide en faisant autant de undo que nécessaire. Bien gérer le menus redo et undo.
2. Code un *modèle de conception* Command servant pour le undo et redo.

D.4 Sauvegarde et lecture de fichiers

1. Sauvegarde et chargement en format texte et en XML utilisant le *modèle de conception* Strategy Pattern. On suppose que dans une classe Model, nous avons une liste contenant les coordonnées et l'identité de chaque élément. Il faut pouvoir choisir le format de sauvegarde (texte ou XML), ainsi que permettre le chargement des données des fichiers de sauvegarde dans la classe Model. Il faudra donc :

   - Définir le format des données dans la classe Model.
   - Définir le format des données pour le fichier texte.
   - Définir le format des données pour le fichier XML.
   - Et coder le *modèle de conception* Strategy Pattern pour pouvoir:
     - Écrire les données de la classe Model dans le fichier texte ou XML.
     - Lire les données du fichier texte ou XML qui seront importées dans la classe Model.

   Nous suggérons que le format, à utiliser pour la sauvegarde, indique sur une ligne avec les deux (2) informations suivantes:

     - le type de composant
     - et la position du composant. * La position étant représentée par un point ou les coordonnées x et y d'un point de référence du composant. Utiliser la position du coin supérieur gauche du composant.
     - Les connexions peuvent être sauvegardées en donnant les indexes de composants qui sont reliés.

Résumé de points:

fichiers bien organisés	3 points
prototype 1	4 points
prototype 2	4 points
prototype 3	4 points
prototype 4	5 points
Total	20 points