Programmation orienté objet avancée

- Contraintes pour l'examen -

Informatique de gestion

UE 209 : Projet informatique intégré

1 Directives générales

1.1 Examen

La cotation attribuée pour le cours de Programmation orientée objet avancé (théorie et labo) est basée d'une part sur un travail de groupe et d'autre part sur un examen oral.

Le ***travail*** est un programme écrit en java qui doit être réalisé par groupe de deux étudiants. La constitution des groupes doit être validée par le professeur de laboratoire.

L'examen oral consiste en une défense du travail de groupe et des questions portant sur la théorie.

Une première note sera attribuée par le professeur après évaluation du travail. Cette note servira de base à l'examen oral ; elle sera augmentée ou diminuée sensiblement en fonction des réponses données par l'étudiant aux questions posées par le(s) professeur(s).

Lors de l'examen oral, l'étudiant devra être capable d'expliquer ***n'importe quelle ligne de code du programme java réalisé en groupe***. Une attention particulière sera apportée à

- La gestion des exceptions
- La gestion des évènements
- L'architecture en couches (+ avantages)
- L'accès aux bases de données relationnelles
- Le clean code
- Les tests unitaires
- Les validations et la sécurité

De plus, l'étudiant devra être capable de répondre à des questions portant sur le langage java en général, comme

- L'héritage
- Le polymorphisme
- Les liens entre classes
- Les protections et les gettors/settors
- Les variables et méthodes de classe (static)
- Les classes abstraites et les interfaces,
- La **sérialisation d'objets**
- La **synchronisation des threads**

1.2 Domaine d'application

Le domaine d'application est soit celui choisi par votre groupe dans le cadre du cours d'analyse du processus métier, soit un domaine d'application au choix qui devra alors être validé par votre professeur de labo.

Les fonctionnalités sont laissées à votre appréciation, à condition qu'elles respectent les contraintes et fonctionnalités minimales énoncées ci-après.

L'**énoncé du programme**, constitué d'une description du domaine d'application, du schéma de la base de données et des fonctionnalités de l'application, doit être proposé au professeur de laboratoire pour validation. Il est évident que l'énoncé du programme doit être validé par votre professeur avant que vous ne commenciez la programmation.

Les énoncés doivent être différents d'un groupe à l'autre.

1.3 Agenda

1.3.1 Dossier de l'énoncé

Vous devez rendre à votre professeur de laboratoire un dossier reprenant l'énoncé afin que celui-ci puisse valider votre énoncé.

Ce dossier est à rendre au professeur de laboratoire au plus tard lors de la dernière séance de laboratoire avant les vacances de Pâques.

Le dossier de l'énoncé doit comprendre :

- Une présentation du **domaine d'application** (une demi-page) ;
- Le **schéma conceptuel** (entités-associations) de la base de données, le **schéma logique des tables**, ainsi que la **documentation** de ces tables (définition de chaque table et chaque colonne, en précisant les identifiants et clés étrangères) ;
- Les **fonctionnalités** minimales de l'application (cfr point 2)
    - Pour les fonctionnalités 2.1 à 2.4, il suffit de préciser dans le dossier de l'énoncé la table sur laquelle ces fonctionnalités s'appliqueront.

- Pour les fonctionnalités de type recherche (cfr point 2.5), vous devez spécifier dans le dossier de l'énoncé l'objectif de la recherche et précisez les entrées et sorties :
  - Entrées : énumérez les critères de recherche en précisant sous quelle forme ils seront affichés à l'utilisateur (ex : JComboBox, JTextField, JRadioButton…) ;
  - Sortie : énumérez les colonnes de la JTable qui affichera les résultats de la recherche.

Pour décrire chacune des recherches, utilisez le formulaire ci-dessous.

Recherche numéro …

Titre de la recherche : …

Objectif de la recherche : …

…

Jointure entre les tables suivantes (au moins 3) :

Table 1 : … Table 2 : … Table 3 : …

(Table 4 : … Table 5 : …)

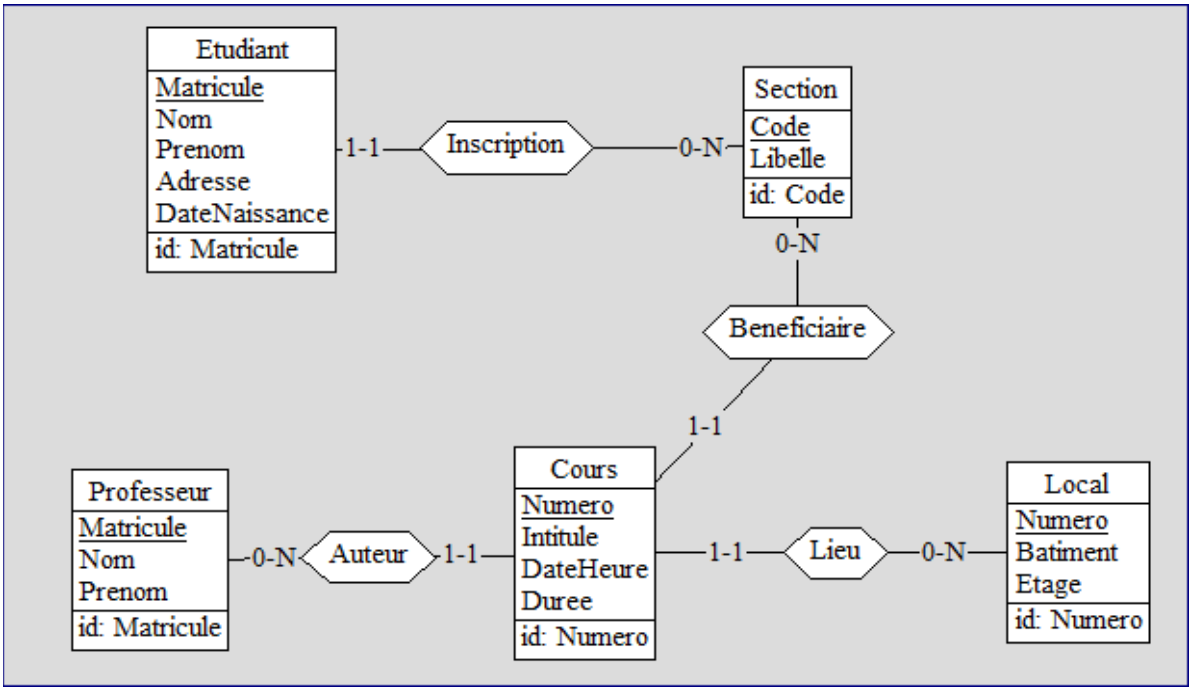
Entrées :

Critère de recherche	**Format** (type de composant swing)
…	…
…	…
…	…

…

Exemple de recherche

Soit la base de données



Recherche numéro 2

Titre de la recherche : Cours dispensés à un étudiant entre 2 dates

Objectif de la recherche : Informations sur les cours suivis par un étudiant choisi par l'utilisateur entre deux dates sélectionnées par l'utilisateur

Jointure entre les tables suivantes :
Table 1 : Etudiant Table 2 : Section Table 3 : Cours Table 4 : Professeur Table 5 : Local

Entrées :

Critère de recherche	**Format**
Liste des étudiants	JComboBox proposant le prénom + le nom
Date début	JSpinner
Date de fin	JSpinner

Sorties (dans une JTable) :

Nom de la colonne provenant de la table	**Nom de la table**
Intitule	Cours
DateHeure	Cours
Batiment	Local
Nom	Professeur
Prenom	Professeur

La rédaction du dossier doit suivre les consignes reprises dans le document intitulé ***"Guide pour la rédaction d'un document ou d'un rapport***".

1.3.2 Dossier final

Vous devez rendre le dossier contenant le résultat de votre travail au plus tard lors de la dernière séance de laboratoire, à savoir **le 15 mai 2019** !

Des points de pénalités seront attribués en cas de retard.

Votre travail sera installé et testé en votre présence lors d'une des dernières séances de laboratoire, **au plus tard le 15 mai** !

Le dossier final doit comprendre

- Une copie du **dossier de l'énoncé** (tel que validé par le professeur de labo) reprenant le domaine d'application, les fonctionnalités de votre programme et les schémas et **documentation** de la base de données (cfr 1.3.1) ;
- Le **code de votre programme** (attention au clean code) ;
- Le **script SQL de création** de votre **base de données MySQL**.

Le dossier doit être complet et impérativement comprendre ces 4 parties.

La procédure de dépôt des travaux vous sera communiquée ultérieurement.

Votre travail sera testé avant l'examen. C'est donc le travail que vous remettrez alors qui sera évalué et coté, et en aucun cas, une version ultérieure du programme (que vous apporteriez éventuellement le jour de l'examen).

1.4 Environnement IntelliJ

Le programme doit être réalisé en java et doit impérativement tourner dans l'environnement **IntelliJ**.

2 Fonctionnalités minimales de l'application

Au lancement de l'application, la première fenêtre qui apparaît doit contenir un message d'accueil et proposer les différentes fonctionnalités de l'application sous forme de **menus**.

Les différentes fonctionnalités que l'application doit proposer au minimum sont listées ci-dessous.

2.1 Insertion via un formulaire d'encodage

Une option de menu doit proposer l'encodage via le formulaire le plus convivial possible d'une ***nouvelle*** *entité* et l'insertion de cette nouvelle entité (nouvelle ligne) dans la table correspondante dans la base de données.

Le formulaire d'insertion doit permettre l'ajout d'une nouvelle ligne dans une table qui doit obligatoirement :

- Contenir des ***colonnes de types différents*** (au moins une colonne de type ***texte***, au moins une colonne de type ***numérique***, au moins une colonne de type ***date*** et au moins un ***booléen***) ;
- Contenir ***plusieurs colonnes facultatives*** ;
- Contenir au moins une ***colonne clé étrangère*** vers une autre table de la base de données.

Tout champ du formulaire correspondant à une ***colonne clé étrangère*** doit être une ***JComboBox*** ou une ***JList***. Celle-ci doit proposer la liste des valeurs présentes dans la table reliée via la clé étrangère. De plus, cette ***liste des valeurs doit être la plus conviviale possible*** pour l'utilisateur : éviter quand c'est possible de lister simplement les identifiants si des libellés plus explicites sont disponibles dans la base de données.

Exemple : soit la table Article qui contient une clé étrangère (FKCategorie) vers la table Categorie, cette dernière contenant les colonnes IDCategorie (identifiant) et LibelleCategorie. Dans le formulaire d'encodage d'un nouvel article, la catégorie (champ correspondant à la colonne FKCategorie) doit être proposée via une JComboBox ou une JList contenant non pas la liste des identifiants des catégories existantes, mais bien la liste des LIBELLES des catégories existantes.

Dans le formulaire d'encodage, vous devez permettre et ***gérer les attributs facultatifs***. L'utilisateur peut donc ne pas introduire de valeur dans les champs (facultatifs) du formulaire correspondants aux colonnes facultatives.

Un ***message d'erreur*** doit être affiché à l'utilisateur ***si tous les champs obligatoires ne sont pas remplis***. Ce message d'erreur doit être suffisamment explicite pour que l'utilisateur identifie quels sont les champs obligatoires qu'il n'a pas remplis.

2.2 Modification

Une option de menu doit proposer à l'utilisateur de ***modifier*** les données **d'une** entité (= une ligne d'une table). L'entité à modifier doit être proposée à l'utilisateur sous forme d'un ***formulaire d'encodage***. Par facilité, proposez de modifier le même type d'entité (même table) que celui que vous aurez choisi pour la fonctionnalité n° 2.1. Vous pourrez ainsi récupérer le formulaire d'encodage que vous aurez créé pour l'insertion.

Même remarque à propos de la gestion des attributs facultatifs et obligatoires (cfr point 2.1).

2.3 Suppression

Une option de menu doit proposer la ***suppression*** d'une ou plusieurs lignes existant dans une table.

Le contenu de cette table doit être proposé à l'utilisateur sous forme d'une JTable. L'utilisateur doit alors pouvoir sélectionner la ou les lignes à supprimer dans cette JTable.

S'il existe, dans d'autres tables, des clés étrangères référençant la table dans laquelle vous permettez les suppressions, vous devez ***gérer les suppressions des lignes reliées dans ces autres tables***. Vous devez alors demander confirmation à l'utilisateur avant de supprimer les autres lignes reliées.

2.4 Listing de table

Une option de menu doit proposer de lister le contenu de la table *dans laquelle l'utilisateur peut insérer ou modifier de nouvelles lignes* (cfr 2.1 et 2.2) (Dans ce listing doivent apparaître les lignes correspondant aux nouvelles insertions ou

modifications éventuellement effectuées par votre application lors de la session en cours).

2.5 Recherches

Au moins **trois** recherches doivent être proposées à l'utilisateur via des options de menu.

Un clic sur une de ces options de menu doit afficher un ***formulaire de saisie de critères de recherche***.

Le formulaire doit contenir un ou plusieurs critère(s) de recherche ainsi qu'un bouton *Recherche*.

Si les différentes valeurs possibles pour un critère de recherche sont enregistrées dans la base de données, cette liste de valeurs doit être ***récupérée dans la base de données*** et affichée à l'utilisateur via une ***JComboBox*** ou ***JList***.

Par exemple, si vous proposez comme fonctionnalité le listing des étudiants habitant une certaine localité, et que la table Localite *existe dans la base de données, vous devez proposer la liste des localités disponibles dans cette table via une liste ou combobox. Cette liste ou combobox sera donc remplie via exécution d'une requête SQL qui récupérera le contenu de la table* Localite *dans la base de données. De plus, ne vous contentez pas d'afficher dans cette liste les identifiants des localités, mais bien les combinaisons des noms des localités et codes postaux.*

Au moins une de ces recherches doit faire intervenir une ***date comme critère de recherche***. Vous devez alors utiliser la classe ***JSpinner*** pour proposer des dates sous forme d'une liste déroulante. A vous de vous documenter sur l'utilisation de cette classe.

Un clic sur le bouton *Recherche* lance la recherche des données qui satisfont les critères sélectionnés par l'utilisateur. Chacune de ces recherches doit nécessiter l'exécution d'une requête SQL contenant ***impérativement au moins une jointure entre trois tables*** !

Le résultat de la recherche doit être affiché sous forme d'une ***JTable***.

2.6 Tâche métier

Implémentez une tâche plus complexe (que nous appellerons tâche métier) et implémentez-la **dans la couche métier**.

Attention, cette tâche métier doit nécessiter des calculs ou traitements à effectuer dans la couche métier ; la fonctionnalité que vous choisissez d'implémenter ne peut donc pas se contenter d'effecteur des accès à la base de données.

Cette tâche métier pourrait être testée par des JUnit.

3 Contraintes

3.1 MVC – 3 tiers

Appliquez les principes de **l'architecture 3-tiers** et **Model – View – Controller**.

Pour rappel :

- Prévoyez au minimum les ***packages*** : viewPackage, controllerPackage, businessPackage, dataAccessPackage, modelPackage et exceptionPackage. Veillez à prévoir les imports corrects entre ces packages.

- Les **entrées** effectuées par l'utilisateur doivent être **testées** au maximum dès la couche **View** (valeurs obligatoires, bon type de données (ex : de type numérique), bon format (ex : email)…)

- La couche ***Business*** est chargée de ***vérifier*** que les ***données*** passées en argument de méthodes sont ***valides***. En cas d'erreur, une exception doit être remontée à la couche View qui se charge alors d'afficher un message à l'utilisateur.

- Veillez à respecter le ***découplage des couches au niveau des exceptions***, notamment en ne remontant pas de SQLException vers les couches supérieures, ni des exceptions dont les noms feraient référence à la notion de base de données (ex : BDException).

- ***Aucun affichage*** ni via la console ni via une boîte de dialogue ne peut être fait à destination de l'utilisateur ***dans une couche autre que la***

couche View, même pour afficher un message d'erreur. Si une erreur survient dans une autre couche que la couche View, une exception doit être remontée vers la couche View qui se chargera, elle, d'afficher un message d'erreur à destination de l'utilisateur.

3.2 Base de données

La persistance des données se fait via une ***base de données MySQL.***

La base de données doit être sécurisée par un ***mot de passe***.

La base de données doit contenir ***au moins 5 tables reliées par des clés étrangères***.

Ajoutez le ***maximum de contraintes*** possibles au niveau de la définition des tables :

- Type correct des colonnes (ex : colonne de type Date pour gérer les dates et non une colonne de type Texte) ;
- Colonnes obligatoires ou facultatives ;
- **Contraintes** via des checks sur les valeurs permises (ex : valeur numérique > 0) ; Ces **checks doivent apparaître dans le script de création des tables**, même si la BD est de type MySQL (qui ne vérifie pas les checks !) ;
- Clés primaires ;
- Clés étrangères.

Les tables devront être ***remplies avec suffisamment de valeurs*** pour pouvoir tester efficacement les différentes fonctionnalités du programme.

L'objet de type **Connection** doit être stocké en appliquant le ***Design Pattern singleton***.

Evitez les injections SQL en utilisant la classe ***PreparedStatement***.

3.3 Interface DataAccess

Vous devez implémenter le **Design Pattern DAO**. Vous devez donc prévoir et utiliser des interfaces qui reprennent **toutes les méthodes de la couche DataAccess** qui peuvent être appelées par la couche Business. Ceci afin de permettre de facilement changer l'implémentation de la couche DataAccess en modifiant le moins possible de code dans les couches appelantes (découplage des couches). Il suffira que les classes de la couche DataAccess s'engagent à implémenter ces interfaces.

Attention, dans la couche Business, veillez à **déclarer des variables de type interface** et à les initialiser avec des objets provenant de classes de la couche DataAccess qui implémentent ces interfaces.

3.4 Thread

Vous devez inclure **au moins un thread** supplémentaire (autre que le thread du programme principal).

Ce thread supplémentaire doit impérativement être ***différent du thread d'affichage de l'heure***.

3.5 Test unitaires

Vous devez prévoir plusieurs tests unitaires (JUnit).

4 Pénalités

Des points de pénalités seront retirés à la cotation du travail pour chacune des contraintes ci-dessus non respectées, notamment :

- 8 points sur 20 si vous n'avez pas appliqué l'architecture 3 tiers ou MVC ;
- 3 points sur 20 si la liste des critères n'est pas proposée à l'utilisateur lorsque plusieurs choix sont possibles et que la liste de ces choix est disponible dans la base de données ;
- 4 points sur 20 si le formulaire d'encodage d'une nouvelle ligne dans une table ne permet pas d'encoder des valeurs nulles (dans les colonnes

facultatives) ou si vous avez choisi de permettre l'encodage dans une table qui ne contient aucune colonne facultative ;

- 2 points sur 20 si vous n'avez pas de thread supplémentaire ;
- 4 points sur 20 s'il y a retard dans la remise du dossier de l'énoncé ou le dossier final.

Bon travail !