Programmation orienté objet avancée

- Contraintes pour l'examen -

UE Projet informatique intégré

1 Directives générales

1.1 Examen

L'évaluation de l'UE est intégrée. L'évaluation est basée sur les connaissances de l'étudiant en SQL, un travail de groupe et sur la défense orale de ce travail.

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 appréciation sera attribuée par le professeur après évaluation du travail. Cette appréciation 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 protections et les gettors/settors
- La **sérialisation d'objets**
- La **synchronisation des threads**

L'acquis d'apprentissage lié aux **Design Patterns** (pour rappel : *Identifier les "Design Patterns" (bonnes pratiques de programmation prédéfinies) à appliquer et les implémenter*) sera évalué **par écrit**. Une interro dispensatoire sera

organisée hors session. Si l'interro n'est pas réussie, l'étudiant sera de nouveau interrogé par écrit à la fin de son examen oral pendant la session.

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 remplir et soumettre à votre professeur de laboratoire le dossier reprenant votre énoncé afin que celui-ci puisse valider votre travail pour l'examen (cf. fichier intitulé « Dossier énoncé examen Java – 2024 » disponible sur Moodle).

Ce dossier est à rendre au professeur de laboratoire au plus tard fin mars.

1.3.2 Dossier final

Vous devez rendre le dossier contenant le résultat de votre travail au plus tard **le deuxième jour de la session de juin.**

Des points de pénalités seront attribués en cas de retard.

Le dossier final doit comprendre

- Une copie du **dossier de l'énoncé** (tel que validé par le professeur de labo) (cf. 1.3.1) ;
- Le lien vers le **code de votre programme** (sur un repository) ;
- Le **script SQL de création** de votre **base de données MySQL**.

Le dossier doit être complet et impérativement comprendre ces 3 parties.

La procédure de dépôt des travaux est la suivante :

Dépôt sur Moodle d'un seul fichier qui doit respecter les contraintes suivantes :

- Type de fichier : **un seul fichier compressé** (peut contenir un répertoire contenant plusieurs fichiers, comme un fichier .docx, un fichier .sql…)
- Nom du fichier : Concaténation des prénoms-noms des deux étudiants du projet
- Contenu :
  - Le fichier contenant le dossier de l'énoncé
  - Un fichier .txt contenant les informations suivantes :
    - Prénoms et noms des étudiants auteurs du projet ainsi que leur adresse mail
    - Tous les logins et mots de passe éventuels pour accéder aux différentes fonctionnalités du programme
    - **Le lien vers un repository GitHub ou autre** contenant le **code** de votre application Java auquel vous aurez **donné l'accès à votre professeur de labo**
  - Le script SQL de **création** des tables de la base de données
  - Le script SQL de **remplissage** des tables de la base de données

Votre travail sera testé avant l'examen. C'est donc le travail que vous remettrez alors qui sera évalué, 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.

Les opérations CRUD (Create – Read – Update – Delete) doivent être implémentées sur une même table (cf. 2.1 à 2.4).

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*** *occurrence d'entité* et l'insertion de cette nouvelle occurrence d'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 ***combobox*** ou une ***liste***. 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 **item** (stockant les articles) qui contient une clé étrangère (**category_fk**) vers la table **category**, cette dernière contenant les colonnes **category_id** (identifiant) et **label** (libellé de la catégorie, unique).*

*Dans le formulaire d'encodage d'un nouvel article, la catégorie (champ correspondant à la colonne **category_fk**) doit être proposée via une comboBox ou une liste contenant non pas la liste des identifiants des catégories existantes (category_id), mais bien la liste des **LIBELLES** des catégories existantes (colonne label).*

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** occurrence d'entité (= une ligne d'une table). L'occurrence d'entité à modifier doit être proposée à l'utilisateur sous forme d'un ***formulaire d'encodage***. Par facilité, proposez de modifier la même entité (même table) que celle que vous aurez choisie 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 (cf. point 2.1).

2.3 Suppression

Une option de menu doit proposer la ***suppression*** d'une ou plusieurs lignes existant dans la même table que celle choisie pour les points 2.1 et 2.2.

Le contenu de cette table doit être proposé à l'utilisateur sous forme d'un composant graphique de type table (ex : JTable). L'utilisateur doit alors pouvoir sélectionner la ou les lignes à supprimer dans cette table.

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, modifier ou supprimer des lignes* (cf. 2.1, 2.2 et 2.3) (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 ***comboBox*** ou ***une liste***.

Par exemple, si vous proposez comme fonctionnalité le listing des étudiants habitant une certaine localité, et que la table city *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* city *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****, ce qui est de loin plus convivial pour l'utilisateur.*

Au moins une de ces recherches doit faire intervenir une ***date comme critère de recherche***. Vous devez alors proposer des dates sous forme d'une liste déroulante (ex : JSpinner). 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*** ! Attention, une seule instruction SQL à prévoir par recherche (instruction de jointure).

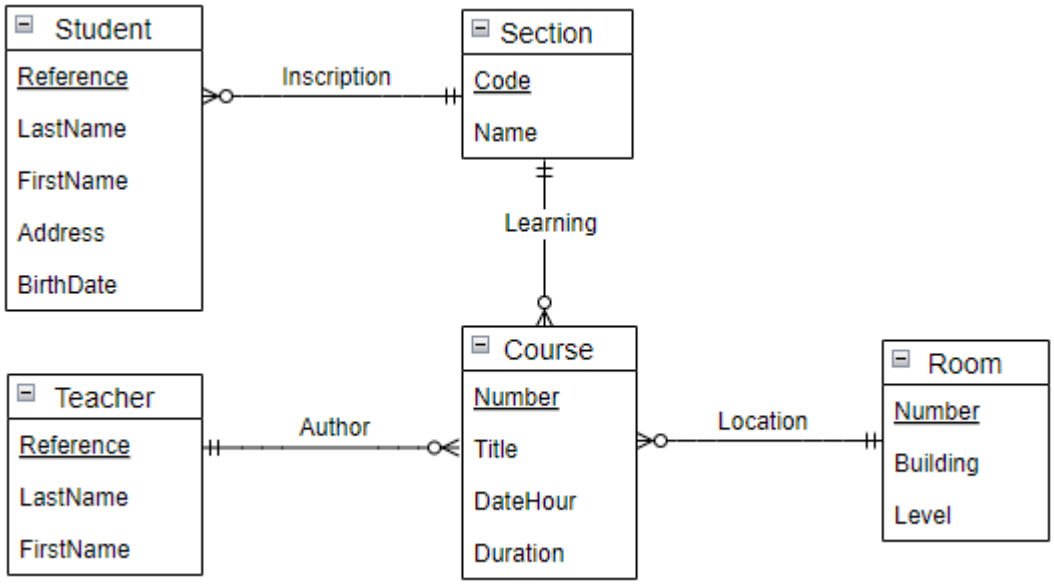
Le résultat de la recherche doit être affiché sous forme d'un composant graphique représentant une table (ex : JTable) : les sorties doivent être composées de plusieurs lignes, chacune contenant des colonnes provenant d'au moins 3 tables différentes.

Pour chaque recherche, 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 du composant graphique de type table (ex : JTable) qui affichera les résultats de la recherche.

Exemple de recherche

Soit la base de données



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

Objectif de la recherche : Lister les 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 : Student Table 2 : Section Table 3 : Course Table 4 : Teacher
Table 5 : Room

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**
Title	Course
DateHour	Course
Building	Room
FirstName	Teacher
LastName	Teacher

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 Code

Les noms des classes, variables et méthodes doivent être en **anglais**.

Les régles du **clean code** seront d'application.

3.2 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) …)

- En cas d'erreur détectée dans la couche **business**, 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.

3.3 Base de données

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

Les noms de table et de colonne doivent être en **anglais**.

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***.

Dans le script de création des tables (à fournir dans le dossier), 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**.

- 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***.

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

3.4 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.

3.5 Thread

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

Ne proposez pas un thread qui nécessite l'accès en base de données !

Créez par exemple une animation graphique en rapport avec votre domaine d'application.

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

3.6 Test unitaires

Vous devez impérativement tester des méthodes via plusieurs tests unitaires (en utilisant au minimum JUnit).

4 Pénalités

Des pénalités seront attribuées lors de l'évaluation du travail pour chacune des contraintes ci-dessus non respectées, notamment :

- Si vous n'avez pas appliqué l'architecture 3 tiers ou MVC ou si les couches sont couplées ;
- Si vous n'avez pas implémenté les Designs Patterns DAO et Singleton ;
- 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 ;
- 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 ;
- Si vous n'avez pas de thread supplémentaire ;
- S'il y a retard dans la remise du dossier de l'énoncé ou le dossier final.

Bon travail !