Sorbonne Université – L3
3I012
Année 2018-2019

Introduction à la modélisation des systèmes logiciels avec UML

3I012

L'équipe pédagogique :

Olga Melekhova (TD/TME)
Cédric Besse (TD/TME)
Olivier Sigaud (TD/TME)
Tewfik Ziadi – (Amphi et Responsable)

TD/TME 1 à 11 – Année 2018-2019

Sorbonne Université – L3
3I012
Année 2018-2019

TD1. Un curieux besoin de modélisation

À partir du code donné en annexe, répondez aux questions suivantes.

Question 1. En une phrase, quels sont les rôles de chacune des classes ?

Question 2. Peut-on dire qu'il existe des classes représentant des données et des classes représentant des interfaces graphiques ? Si oui, pourquoi et quelles sont ces classes ?

Question 3 Est-il possible que le numéro de téléphone d'une personne soit
+33 1 44 27 00 00 ?

Question 4 Est-il possible que l'adresse e-mail d'une personne soit
« je_ne_veux_pas_donner_mon_email » ?

Question 5 Quelles sont les fonctionnalités proposées par les menus graphiques de cette application ?

Question 6 Quelles sont les fonctionnalités réellement réalisées par cette application ?

Question 7 Est-il possible de sauvegarder un répertoire dans un fichier ?

Question 8 Si vous aviez à rédiger un document décrivant tout ce que vous savez sur cette application afin qu'il puisse être lu par un développeur qui veut réutiliser cette application et un chef de projet qui souhaite savoir s'il peut intégrer cette application, quelles devraient être les caractéristiques de votre document ?

Pour répondre à la question suivante, formez des groupes de travail de 4 à 6 personnes.

Question 9 Elaborez autant de schémas qu'il vous semble nécessaire pour comprendre la structure de l'application MyAssistant. Vous préciserez la légende de vos schémas.

Sorbonne Université – L3
3I012
Année 2018-2019

TME1

Introduction : Un curieux besoin de modélisation

1 - MyAssistant

Q1 : Compilez et exécutez l'application MyAssistant (java repertoire.MyAssistant).

Q2 : Lorsque vous utilisez l'application MyAssistant, quelles-sont les questions du TD1 auxquelles il est possible de répondre sans lire le code ?

Q3 : Est-il possible de comprendre l'architecture technique de l'application MyAssistant sans avoir lu le code ?

2 – Réutiliser, maintenir ou faire évoluer une application existante

Q4 : Téléchargez l'exécutable du projet Domination à l'adresse suivante :
http://sourceforge.net/projects/domination/
Installez le jeu et jouez une partie.

Q5 : Un utilisateur souhaite que le défenseur puisse visualiser les dés de l'attaquant avant de lancer ses dés. Un autre souhaite que la carte soit affichée en 1280*800. Ces demandes ont-telles été réalisées ?

Q6 : Décompressez les sources du projet Domination qui sont dans "src.zip".

Q7 : Combien de développeurs ont participé à ce projet ?

Q8 : Comment est documenté le code source du projet ?

Q9 : Quelles classes faut-il modifier pour réaliser les fonctionnalités demandées (celles identifiées en Q5) ?

Sorbonne Université – L3
3I012
Année 2018-2019

TD2. La vue statique
Diagrammes de classe UML, et d'objets

Question 1 Définissez les classes UML classes Etudiant et Enseignant, Cours représentant respectivement :

- Un étudiant caractérisé, entre autres, par un identifiant, un nom, un prénom et une date de naissance.
- Un enseignant, caractérisé, entre autres, par un identifiant, un nom, un prénom et une date de naissance.
- Un cours, caractérisé par un identifiant, un nom, le nombre d'heures de cours magistral, le nombre d'heures de travaux dirigés et un nombre d'heures de travaux pratiques que doit suivre un étudiant

Question 2 Définissez les associations qui peuvent exister entre un enseignant et un cours

Question 3 Définissez la classe UML GroupeEtudiant représentant un groupe d'étudiants. Ajoutez les associations nécéssaires entre cette classe et les classes Etudiant et Cours.

Question 4 Définissez l'association possible entre un groupe d'étudiants et un cours

Question 5 Définissez un diagramme d'objets (diagramme d'instances) modélisant des objets instances des classes Etudiant et Enseignant, Cours et GroupeEtudiant

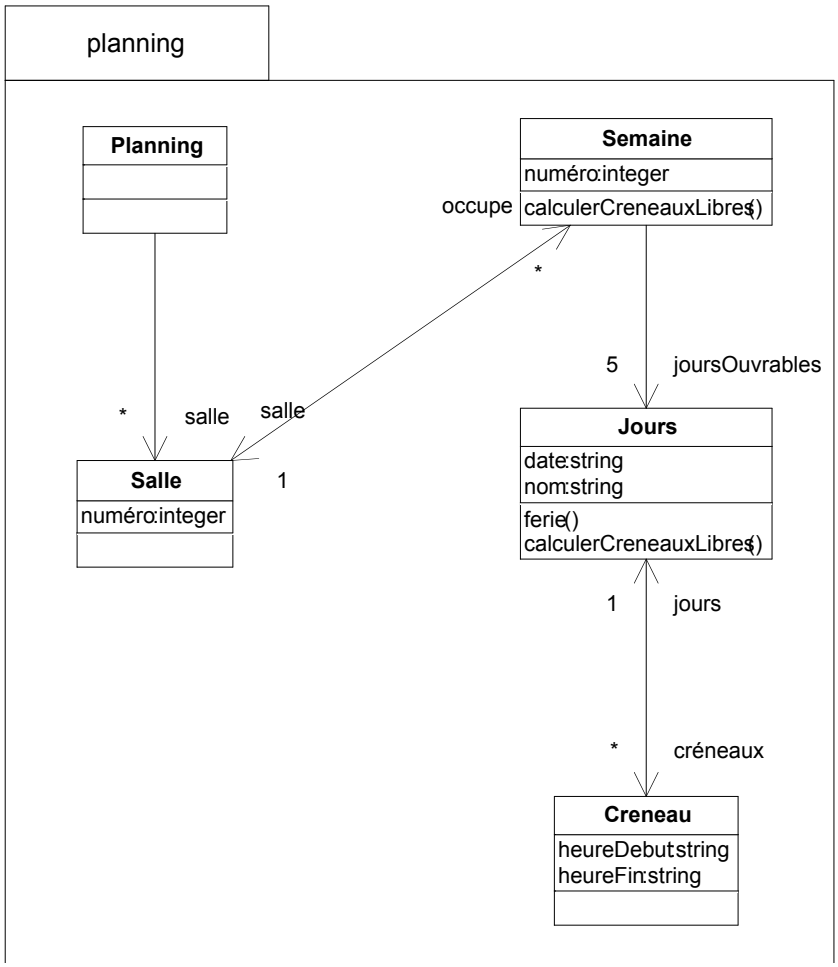
Question 5

- Pensez-vous qu'il soit possible de définir un lien d'héritage entre les classes UML représentant respectivement les étudiants et les enseignants
- Pensez-vous qu'il soit possible de définir un lien d'héritage entre les classes UML représentant respectivement les étudiants et les groupes d'étudiants.
- Proposez une utilisation correcte du lien d'heritage.

Question 6 Considérons l'opération *coursDeLEtudiant()* permettant d'obtenir l'ensemble des cours suivis par un étudiant. Pour chacune des signatures proposées ci-dessous pour l'opération, dites dans quelle(s) classe(s) positionner l'opération, ainsi que les modifications à apporter aux associations préalablement identifiées pour que votre solution soit réalisable.

1) coursDeLEtudiant():Cours[*]
2) coursDeLEtudiant(in e:Etudiant): Cours[*]
3) coursDeLEtudiant(in ens:Enseignant): Cours[*]

Question 7 Expliquez le diagramme de classe représenté à la figure suivante



Question 8 Positionnez toutes vos classes (Etudiant, Enseignant, Cours, GroupeEtudiant) dans un package nommé Personnel.

Question 9 Liez vos classes pour faire en sorte qu'à partir d'un créneau on puisse accéder à l'intitulé du cours correspondant.

Sorbonne Université – L3
3I012
Année 2018-2019

TME2

Le diagramme de classes UML

1. Projet Modelio

Q1 : Construisez un nouveau projet Modelio UML

Nommez le « GNum_Groupe»_« Nom_Binome »_TP2_Projet1

Q2 : Construisez le modèle UML correspondant au diagramme de classes fait en TD2.

Pour créer un diagramme UML, dans l'espace de travail (Workspace):

- *Clic droit sur le package devant contenir le diagramme ➔ Créer un diagramme ➔ « type du diagramme »*

On peut ajouter un élément au modèle :

- *en passant par le diagramme de classes (clic sur l'icone correspondant à l'élément), l'élément sera aussi visible dans l'espace de travail*
- *directement dans l'espace de travail:*
  Clic droit sur l'élement existant (package, classe, opération, ...) devant contenir le nouvel élément ➔ Créer un élément ➔ "type d'élément"

Q3 : Spécifiez bien les associations (nom de l'association, nom des rôles, multiplicité de rôle).

Q4 : Spécifiez bien les types des attributs.

Q5 : Spécifiez bien les signatures des opérations.

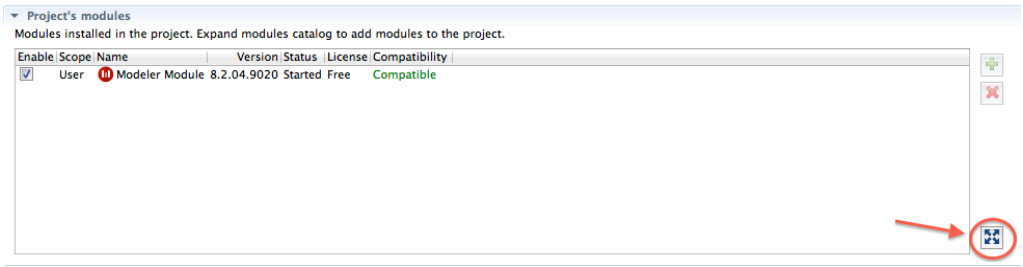
Q6 : Construisez un diagramme qui ne montre que les associations entre les classes (ce diagramme doit cacher les attributs et les opérations des classes).

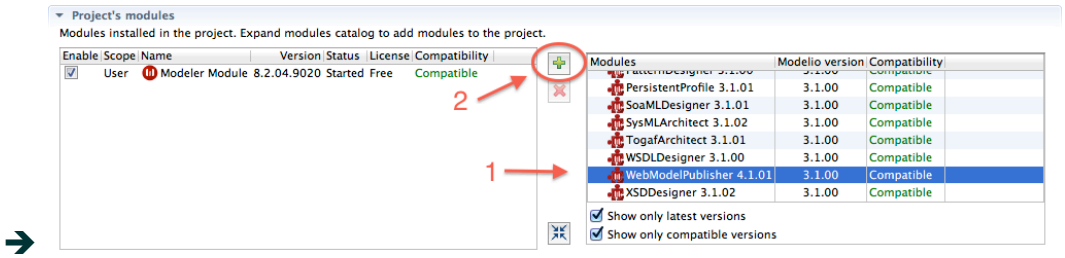
Q7. Dans cette question, vous utilisez module WebModelPublisher de Modelio pour la génération de la documentation à partir des modèles réalisés dans le TP.

- Chargez le module WebModelPublisher

Menu: Configuration ➔ Modules

➔





➔ WebModelPublisher apparaît dans la listre des modules du projet

- Générez la documentation de votre projet :
  1. Sélectionnez dans l'explorateur Modèle la racine du projet
  2. Clic droit sur la racine ➔ Menu Web Model Publisher ➔ Générer
  3. Renseignez les champs "Nom" et "Répertoire de destination" et validez
  4. la documentation sera générée dans le –sous-repertoire:
     <Répertoire destination>/<Nom>"

2 – MyAssistant

Q8 : Sans regarder le code de l'application, et en relisant les schémas que vous avez construits en TD1, construisez le modèle de l'application MyAssistant (vous construirez par exemple les classes Repertoire, Personne, etc).

Sorbonne Université – L3
3I012
Année 2018-2019

TD3. Reverse Engineering

Les opérations de Reverse Engineering présentées dans ce TD portent sur le code Java de l'application MyAssistant donné au TD1. Nous appliquons ici les règles de correspondance Java vers UML décrites vues en cours.

Question 1 Effectuez le Reverse Engineering de la classe Adresse.

Question 2 Effectuez le Reverse Engineering de la classe Personne. Discutez de l'association entre Personne et Adresse.

Question 3 Effectuez le Reverse Engineering de la classe Repertoire.

Question 4 Discutez de l'association entre la classe Repertoire et la classe Personne.

Question 5 Effectuez le Reverse Engineering de la classe UIPersonne.

Question 6 Faut-il que le code Java soit compilable (sans erreur) pour pouvoir faire un reverse engineering ?

Question 7 Discutez les différences entre "*Modèle*" et "*Diagramme*"?

Question 8 Que produit l'opération de Reverse Engineering?

Question 9 Quels sont les diagrammes de classe qu'il faudrait construire juste après avoir exécuté un reverse engineering ?

Sorbonne Université – L3
3I012
Année 2018-2019

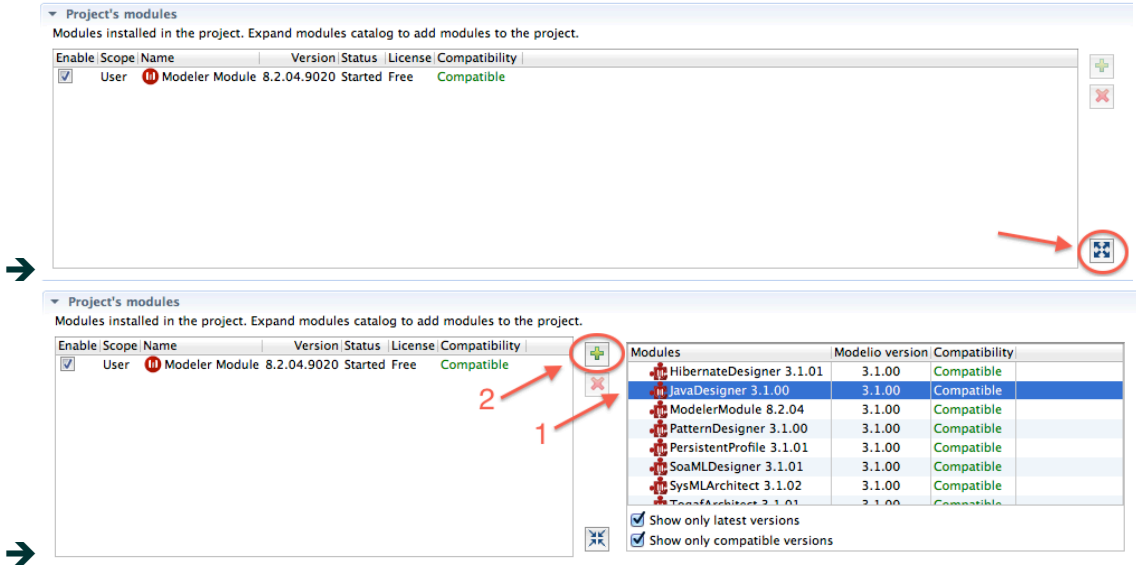
TME3

Reverse Engineering

ATTENTION ! *À compter de ce TME vous ne devez travailler que dans le seul projet Modelio créé aujourd'hui, en structurant votre projet à l'aide de packages correspondant aux séances: TME3, TME4, TME5, etc. Tout ce qui concerne MyAssistant sera évalué en dernière scéance de TME.*

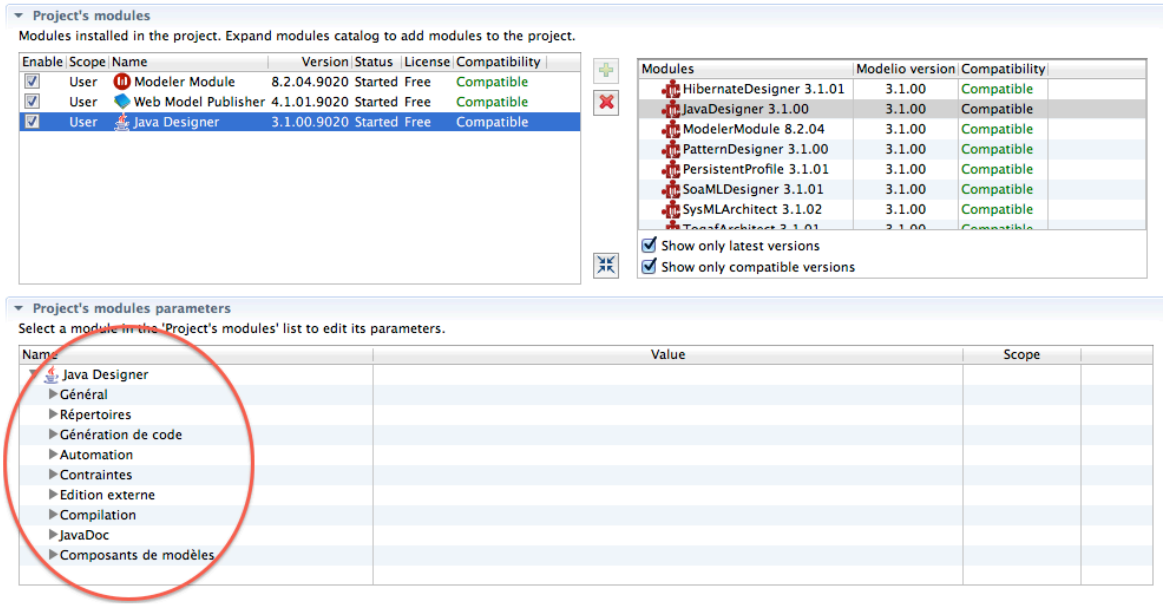
Q1 : Utilisez Modelio pour effectuer le reverse engineering de l'application « MyAssistant ». Pour ce faire :

5. Créez un projet Modelio. Pour intégrer Java dans un projet:
   Menu : Configuration ➔Modules



6. Configurez le module JavaDesigner:

Sorbonne Université – L3
3I012
Année 2018-2019



➔
➔ Répertoires ➔ JDK : il faut préciser le chemin du JDK: */usr/lib/jvm/j2sdk1.7-oracle/*

3. Et enfin démarrez le reverse **<u>à partir de la racine</u>** de votre nouveau projet:
   - Clic droit sur la racine de votre projet ➔ Java Designer ➔ Reverse ➔ Reverser des sources
   - Dans l'onglet **"Fichiers à réverser"**, renseignez le chemin jusqu'aux sources de MyAssistant
   - Sélectionnez tous les fichiers sources .java
   - Cliquez sur le bouton **Suivant**
   - ouvrez l'onglet **"Jars du reverse"** et ajoutez un jar externe (1er bouton):

le .jar en question, **"rt.jar** se trouve dans le sous répertoire **"jre/lib"** du répertoire d'installation du **jdk (**il contient les classes compilées du runtime Java) –Lancez l'opération de reverse engineering (bouton **Reverse**)

SOUS MACOSX:
JDK: /System/Library/Frameworks/JavaVM.framework/Versions/CurrentJDK/Home
rt.jar => classes.jar : /System/Library/Frameworks/JavaVM.framework/Classes/**classes.jar**

Q2 : Faites un diagramme de classes représentant la totalité de l'application.

Q3 : Faites un diagramme de classes représentant les classes l'interface graphique.

Q4 : Faites un diagramme de classes représentant les classes de données.

Q5 : Faites un diagramme de classes pour chacune des classes (on montrera uniquement les associations de la classe avec ses voisines).

Q6 : Faites un diagramme de classes de la classe <u>Repertoire</u> qui affiche les notes correspondant à ses méthodes.

Q7: Comparez votre modèle avec le modèle de MyAssistant obtenu au TME2

Sorbonne Université – L3
3I012
Année 2018-2019

TD4. Retroconception & Patrons de conception

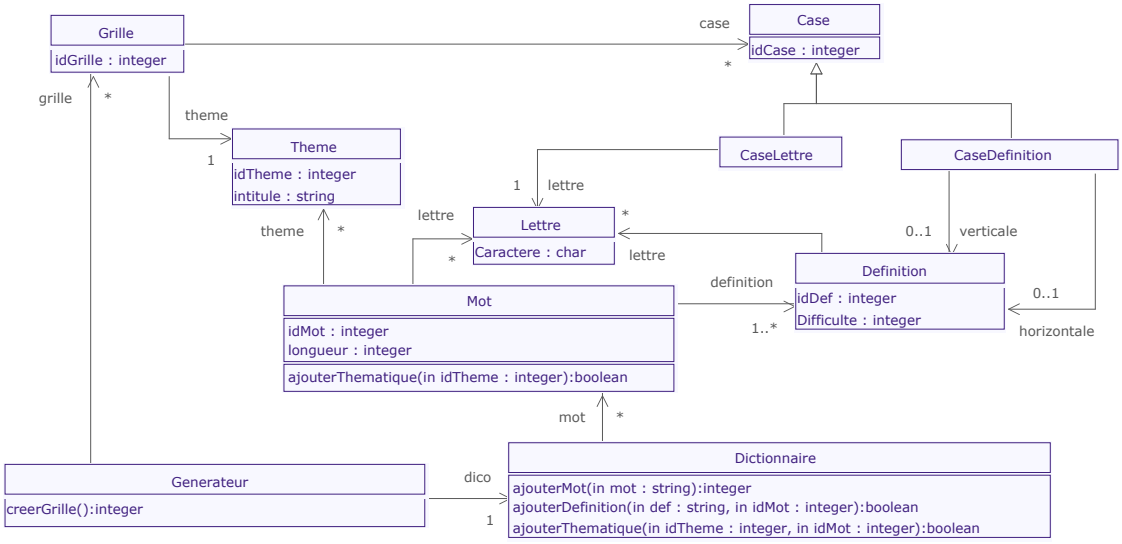
Partie 1: Cycles de dépendance entre packages

Le diagramme de classe ci-dessous est le résultat de l'application de reverse engieering à partir de code Java de l'application **MotFleches**.

Question 10 Découpez l'application en 2 packages :

- un package **GestionMots** contenant les classes **Dictionnaire**, **Mot**, **Lettre**, et **Definition**
- un package **GestionGrilles** contenant les classes restantes. Quel problème se pose alors ?

Question 11 Proposez une solution pour le résoudre (sans déplacer les classes existantes). Vous présenterez de façon explicite les dépendances entre packages. Discutez votre solution dans le cas ou nous souhaitons reutiliser le package GestionMots sans le package GestionGrilles.

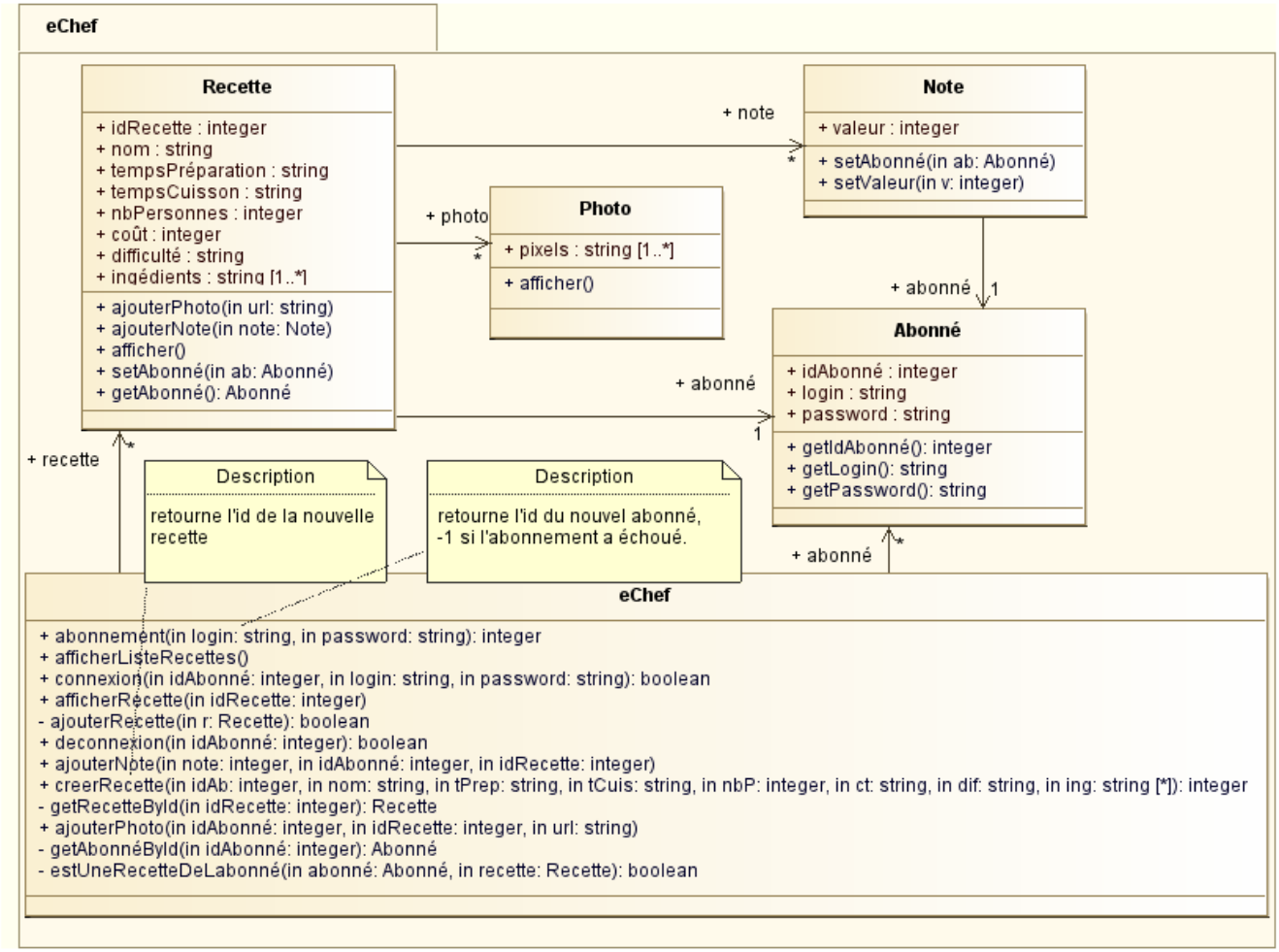


Question 2 Proposez une solution pour le résoudre (sans déplacer les classes existantes). Vous présenterez de façon explicite les dépendances entre packages.

Exercice 2

Nous considérons une application web *eChef* qui a pour objectif la réalisation d'un logiciel de gestion de recettes de cuisine, permettant à ses utilisateurs de stocker leurs recettes, les consulter, les rechercher, les noter. Tout internaute peut consulter librement les recettes proposées par *eChef*. Par contre pour ajouter une recette, la supprimer, ou la noter, il faut absolument être abonné et être connecté. Lorsqu'un internaute s'abonne à l'application *eChef*, il choisit un login et un password. Tout utilisateur peut déposer une demande de recette au cas où celle-ci ne serait pas déjà présente sur le site. Les autres utilisateurs pourront ainsi répondre à ce type de demandes. Une recette a un identifiant, un nom ainsi qu'une description. Cette description détaille : les ingrédients (nom et quantité) nécessaires à sa réalisation (paramètre *ing* dans l'opération *creerRecette()*), le temps de cuisson (paramètre *tCuis* dans l'opération *creerRecette()*), le temps de préparation (paramètre *tPrep* dans l'opération

creerRecette()), le coût de la recette (paramètre *ct* dans l'opération *creerRecette()*), le nombre de personnes (paramètre *nbP* dans l'opération *creerRecette()*), la difficulté (paramètre *diff* dans l'opération *creerRecette()*). Seul l'abonné qui crée la recette a le droit d'ajouter des photos présentant le résultat final. Bien sûr un abonné, en plus d'ajouter une recette, a le droit de la consulter, ou de la supprimer. N'importe quel abonné peut également noter les recettes des autres abonnés (de 0 à 10).



Question 1 Découpez l'application en 2 packages :

- un package **Principal** contenant les classes ***eChef, Recette*** et ***Abonné***
- un package **Secondaire** contenant les classes ***Note*** et ***Photo***. Quel problème se pose alors ?

Question 2 ***Sachant que l'on souhaite que le package Secondaire soit réutilisable par d'autres applications***, proposez une solution pour le résoudre (sans déplacer les classes existantes),. Vous présenterez de façon explicite les dépendances entre packages.

Partie 2: Patrons de conception (Design Patterns)

Exercice 1

Un étudiant qui a suivi le cours LI342 à l'UPMC est admis pour un stage d'été au sein d'une entreprise qui développe des systèmes de contrôle automatique de rames de métro (comme la ligne 14 dans la région parisienne). L'étudiant a défini la classe UML ***Meteor*** qui contrôle automatiquement le train. Cette classe est définie par une propriété *etat* de type *integer* qui enregistre l'état courant du train.

Nous distinguons quatre états possibles selon les valeurs de *etat* :

- etat =0 ; cela signifie que le train est dans l'état PortesOuvertes.
- etat =1 ; cela signifie que le train est dans l'état PorteFermées
- etat =2 ; cela signifie que le train est dans l'état EnMarche
- etat =3 ; cela signifie que le train est dans l'état EnArrêt

La classe ***Meteor*** est définie aussi par l'opération *actionner()* qui contrôle le changement des états du train pour assurer son bon fonctionnement. En effet, si l'état courant du train est PortesOuvertes, l'exécution de cette opération entraîne le changement de l'état vers PortesFermées. Une fois les portes fermées, l'invocation de l'opération *actionner()* entraîne le changement de l'état vers EnMarche. Une nouvelle invocation de cette opération provoque l'arrêt du train (EnArrêt). L'invocation suivante de *actionner()* ramène le train à l'état PortesOuvertes. On retrouve ces informations dans le diagramme de classe de la figure 2.

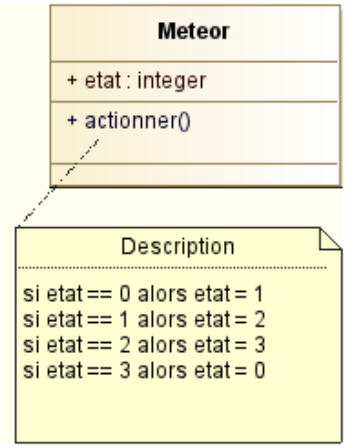


Figure 2. Diagramme de classes de Meteor

Nous souhaitons améliorer cette gestion des états de ***Meteor*** en utilisant le design pattern State (illustré en annexe).

Question 1 Faites un état des lieux avant application du pattern, i.e.:

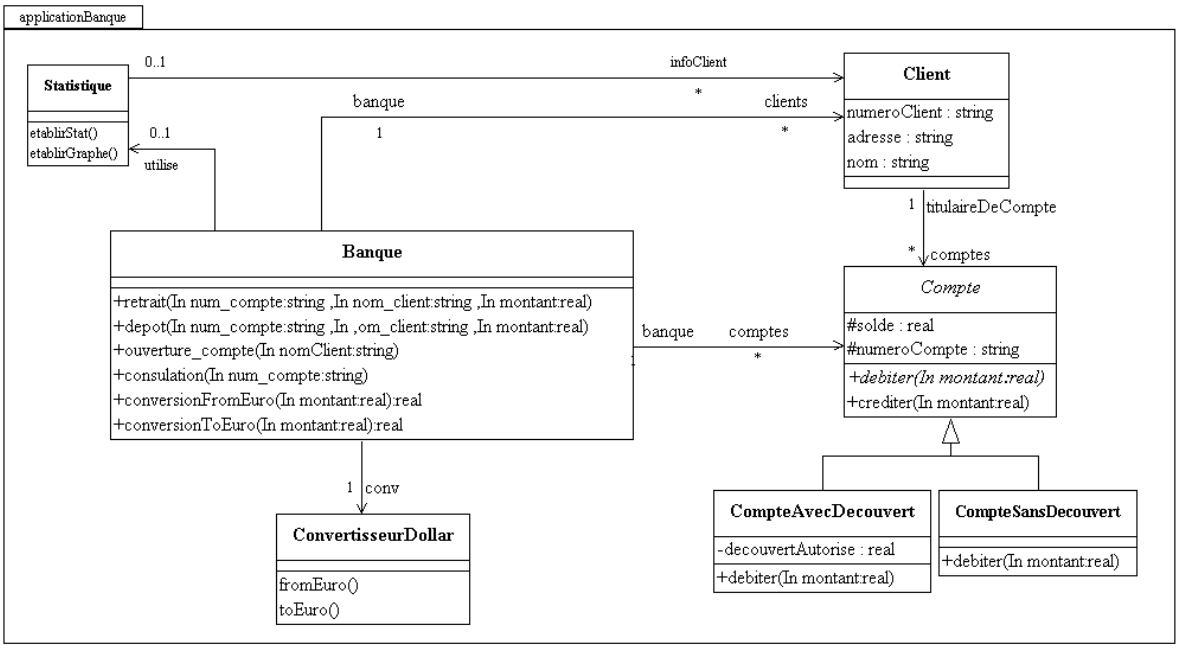
- quelles classes existantes peuvent jouer l'un des rôles définis dans le pattern? Quelles sont les opérations de ces classes qui correspondent à des opérations définies dans le pattern? quelles sont les opérations manquantes ?
- quels sont les rôles encore à attribuer ?

Question 2 Donnez le nouveau diagramme de classe obtenu après application du pattern. Vous expliciterez les ajouts dans le code des opérations utilisées par le pattern.

Exercice 2

Nous considérons dans cet exercice une application de gestion de comptes bancaires. Chaque client peut posséder plusieurs comptes en banque. Un client est défini par un numéro, un nom et une adresse. Un compte est défini par son numéro et le solde. Deux types de comptes existent, l'un autorisant un certain découvert au client, l'autre pas. L'application offre la possiblité d'établir des statistiques concernant les mouvements des comptes de clients au travers de la classe Statistique. L'application contient aussi une classe ConvertisseurDollar

qui permet de traduire les montants des transactions du Dollar vers l'euro et inversement.



La conception proposée pour l'application bancaire impose que pour exécuter les différentes fonctionnalités bancaires (ouverture d'un compte, débit, crédit.. etc), les usagers doivent toujours invoquer les opérations correspondantes de la classe *Banque*. Ces opérations sont critiques pour l'application bancaire et il ne semble pas judicieux d'autoriser les usagers à les invoquer directement. Il est donc souhaitable de définir des commandes de haut niveau qui gèrent les opérations bancaires en utilisant les opérations de base définies dans la classe *Banque*.
Les commandes de haut niveau que l'on souhaite intégrer dans l'application bancaire sont les suivantes :
Format : NOM_COMMANDE <param 1> <param 2> …. <Param n>
OUVERTURE_COMPTE *nom_client*
//Créer un compte dont le nom de client est nom_client
DEPOT *num_compte nom_client montant*
// Déposer une somme sur un compte
RETRAIT *num_compte nom_client montant*
// Retrait d'une somme
CONSULTATION *num_compte*
// Consulter le solde d'un compte dont le numéro est num_compte
CONVERSION_FROM_EURO *montant*
// Convertir de l'Euro vers le Dollar
CONVERSION_TO_EURO *montant*
// Convertir du Dollar vers l'Euro
Nous souhaitons utiliser le design pattern « Commande » présenté en annexe pour fournir une solution.
Question 1 Faites un état des lieux avant application du pattern, i.e.:

- quelles classes existantes peuvent jouer l'un des rôles définis dans le pattern? Quelles

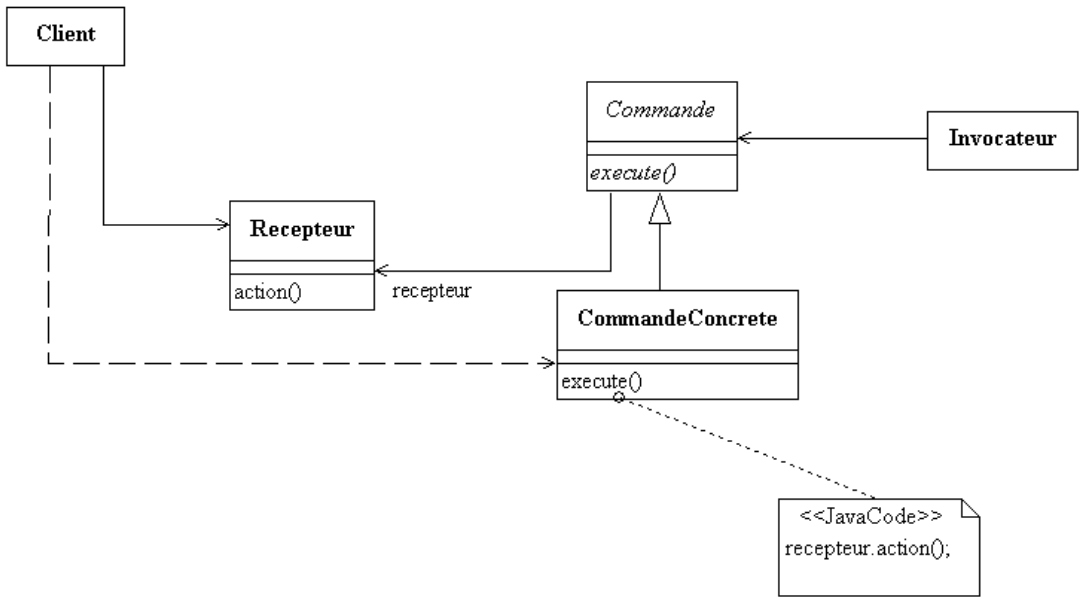
sont les opérations de ces classes qui correspondent à des opérations définies dans le pattern? quelles sont les opérations manquantes ?

- quels sont les rôles encore à attribuer ?

Question 2 Donnez le nouveau diagramme de classe obtenu après application du pattern. Vous expliciterez les ajouts dans le code des opérations utilisées par le pattern.

ANNEXE: Le patron de conception COMMANDE

Structure



Constituants

- **Commande** : déclare une interface pour exécuter une opération
- **CommandeConcrete** : Concrétise *execute()* par l'invocation des opérations adéquates du récepteur.
- **Client** : crée un objet *CommandeConcrete* et positionne son récepteur.
- **Invocateur** : demande à la commande d'entreprendre la requête.
- **Récepteur** : Sait comment effectuer les opérations associées avec le traitement d'une requete. Tout type de classe peut servir de récepteur.

Remarque

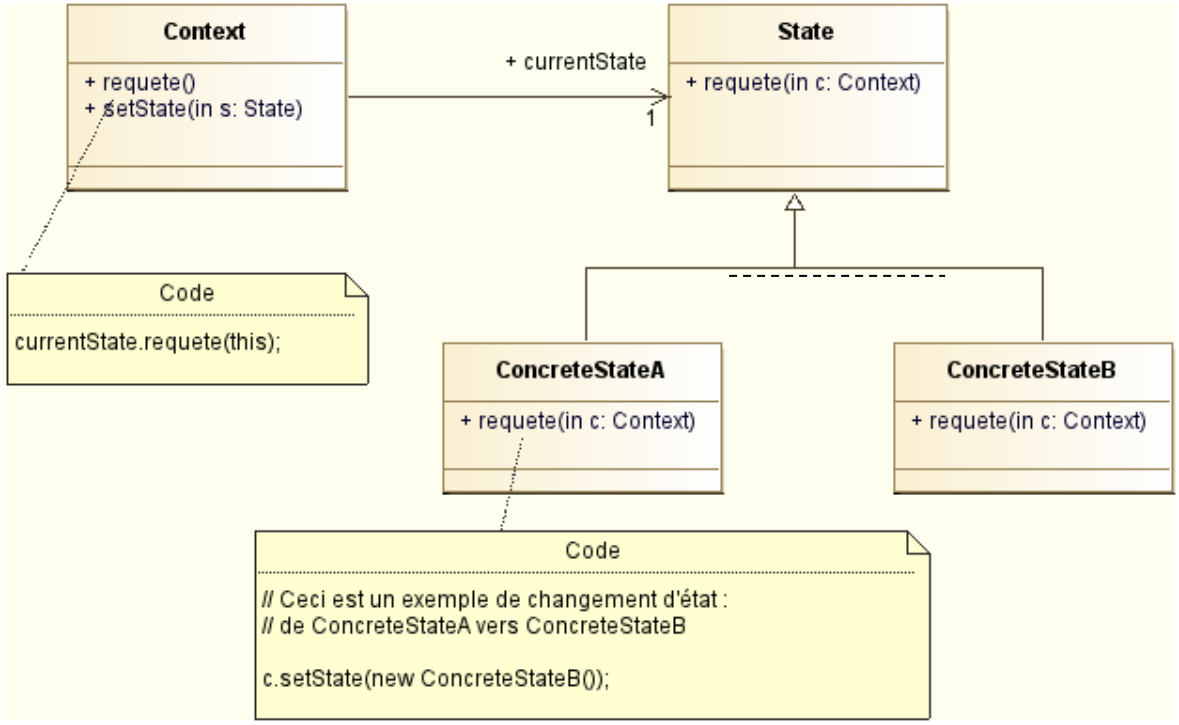
Si la commande concrète définit des paramètres, ceux-ci peuvent être ajoutés comme des attributs de la classe *CommandeConcrete.*

ANNEXE: Le patron de conception STATE

Objectif

Le patron de conception STATE permet à un objet d'altérer son comportement quand son état interne change.

Structure



Participants

- **Context** : Définit l'objet dont ont veut gérer l'état. L'attribut *currentState* définit l'état courant de l'objet. Cet attribut est lui-même un objet implémentant l'interface « State ».
- **State** : Définit une classe abstraite pour encapsuler le comportement correspondant à un état de l'objet de classe « Context ».
- **ConcreteStateA, ConcreteStateB** : Sous-classes de « State » définissant chacune un état concret ainsi que le comportement associé à cet état.

Fonctionnement

- « Context » délègue les invocations des opérations à l'objet « ConcreteState » représentant l'état courant, tout en lui fournissant sa référence.
- Le changement d'état d'un objet de classe « Context » est défini dans les opérations des sous-classes « ConcreteState ».

Sorbonne Université – L3
3I012
Année 2018-2019

TME4

Retro-Conception et patrons de conception

Partie 1 : Découpe en packages

Q1 : Découpez l'application Repertoire en deux packages : Un package représentant l'IHM,

Un package contenant le reste.

Q2 : Faites un diagramme représentant les dépendances entre packages.

Partie 2 : Application du patron « Observer »

On veut créer un nouveau composant graphique qui affiche le nombre de personnes actuellement saisies dans le répertoire.

Q1 : Construisez la (ou les) classe(s) correspondant à ce composant.

Q2 : Le composant graphique, pour afficher des informations correctes, doit être averti de chaque ajout de personne. Mettez en œuvre le patron de conception Observer pour que cela soit possible.

Q3 : Faites un diagramme ne contenant que les éléments participant à la mise en œuvre du patron de conception.

Sorbonne Université – L3
3I012
Année 2018-2019

TD5. Génération de code

Partie 1: Génération

Question 1 Écrivez le code généré à partir de la classe Document illustrée à la figure 1.

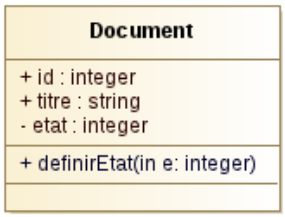


Figure 1 Classe Document

Question 2 Écrivez le code généré à partir de la classe Bibliothèque illustrée à la figure 2.

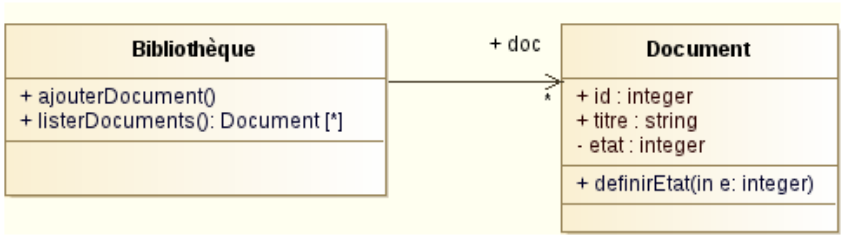


Figure 2 Classe Bibliothèque et Document

Question 3 Écrivez le code généré à partir des classes Livre, CD, Revue (voir figure 3).

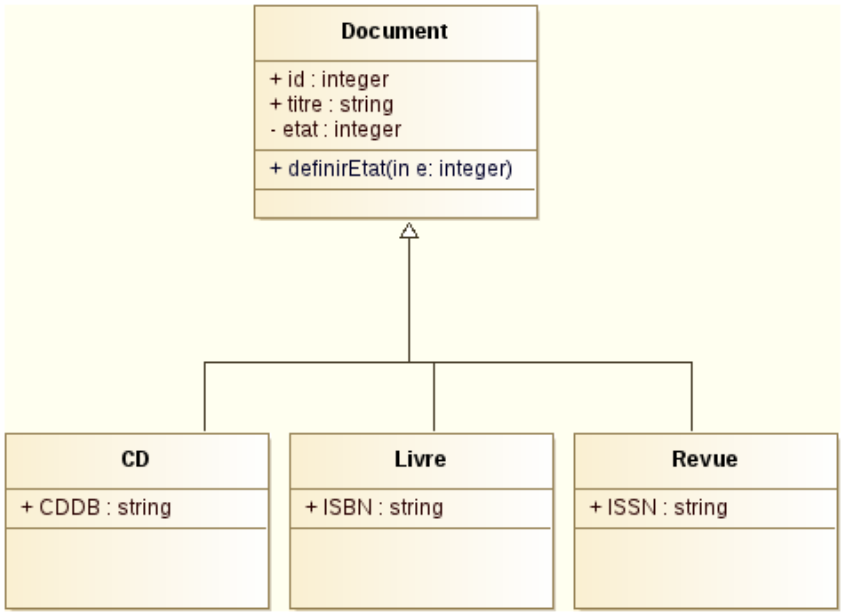


Figure 3 Classes CD, Livre et Revue

Question 4 Écrivez le code généré à partir de l'association CDdeLivre représentée à la figure4.

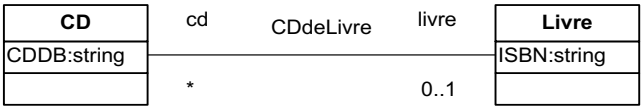


Figure 4 Association CDdeLivre

Question 5 Écrivez le code généré à partir des classes représentées à la figure 5 après avoir défini les règles de génération de code que vous comptez utiliser.

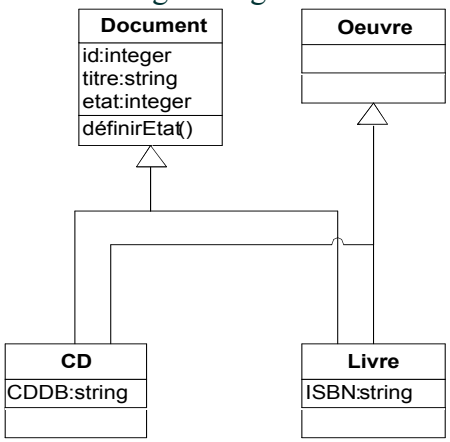


Figure 5 Héritage multiple

Partie 2 : Synchronisation Génération + Reverse

Après une première génération de code à partir d'un modèle UML, le modèle UML et le code sont synchronisés. Afin de maintenir cette synchronisation, supposons qu'un mécanisme d'*update* permette de faire remonter automatiquement dans le modèle UML les modifications manuelle apportées au code Java.
Nous considérerons pour l'instant que le mécanisme d'update correspond à une opération de Reverse Engineering dans laquelle seuls les nouveaux éléments du code sont automatiquement ajoutés dans le modèle.

Question 6 Construisez le modèle UML de la classe Bibliothèque (dont vous avez fourni le code à la question 2) obtenu par update après avoir ajouté dans le code Java les attributs « nom », « adresse » et « type » dont les types sont des String.

Question 7 Nous voulons maintenant, toujours dans le code Java, modifier l'attribut « type » en attribut « domaine ». Pensez-vous qu'il soit possible, après un update, que les deux attributs « type » et « domaine » puissent être présents dans le modèle ? Si oui, à quoi est dû ce comportement bizarre ?

Question 8 Proposez un nouveau mécanisme d'update ne souffrant pas des défauts présentés à la question 7.

Question 9 Proposez le mécanisme inverse de l'update permettant de modifier un modèle UML déjà synchronisé avec du code et de mettre à jour automatiquement le code Java.

Question 10 Dans quelle approche de programmation par modélisation (Model Driven, Code Driven et Round Trip) ces mécanismes d'update sont-ils fondamentaux ?

Sorbonne Université – L3
3I012
Année 2018-2019

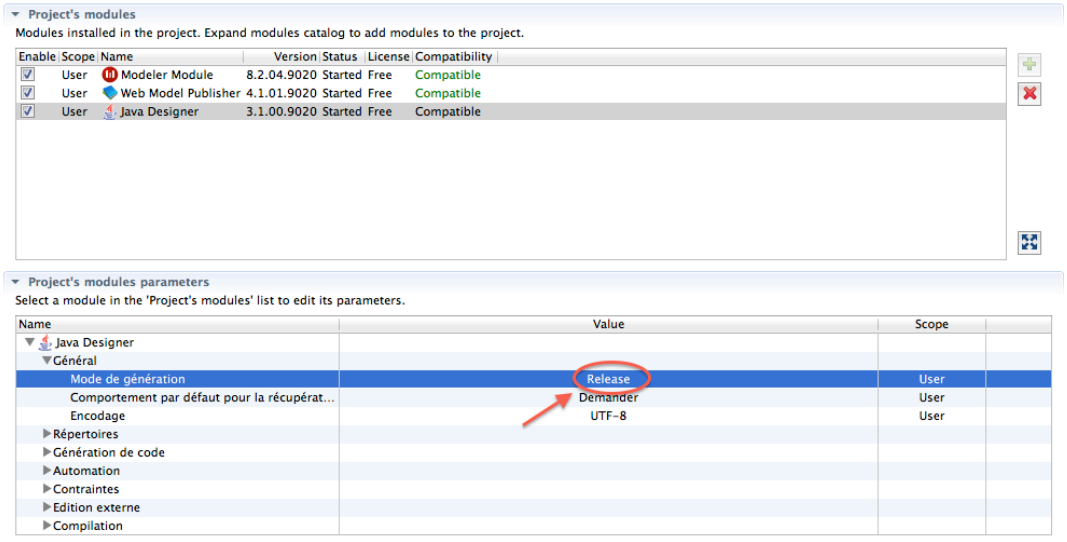
TME5

Génération de code

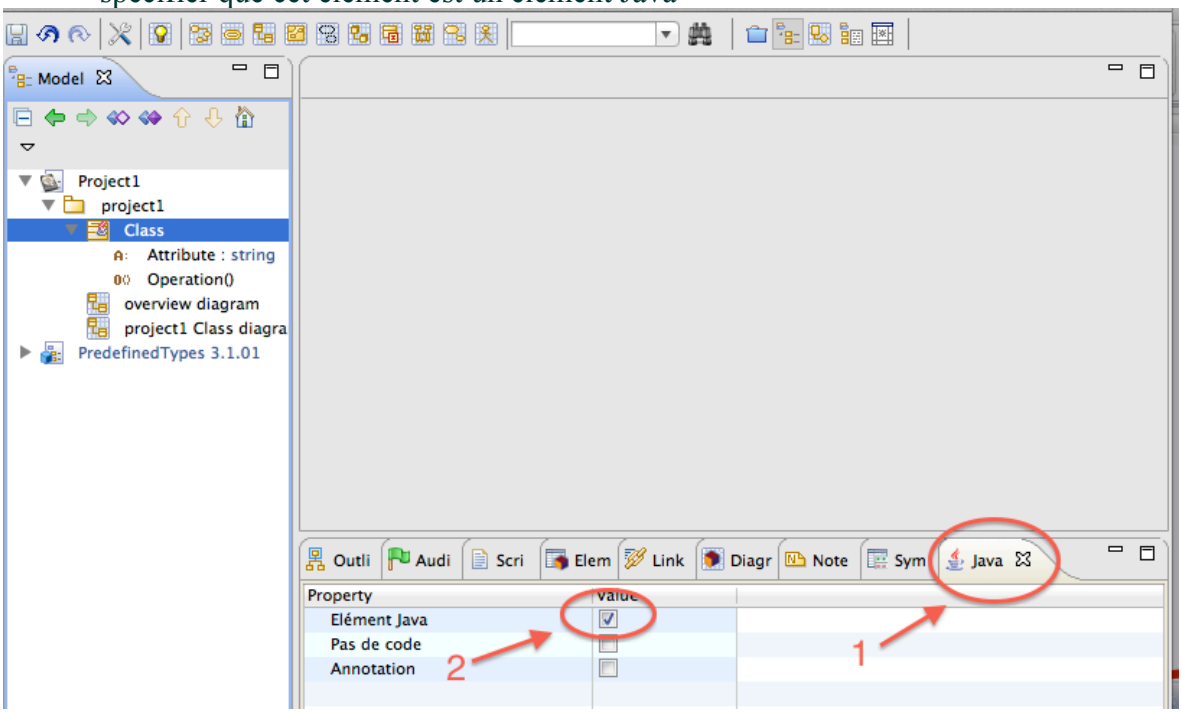
1- Comment marche la génération de code

Configurer Modelio pour permettre une génération de code :

- il faut que le module JavaDesigner sous déployé pour le projet (cf TP3)
- il faut positionner dans les préférences générales de JavaDesigner le mode de génération à « Release » :
  Menu *Configuration* ➔ *Modules*



➔ il suffit ensuite, pour chaque élément du modèle que l'on souhaite générer, de spécifier que cet élément est un élément Java



Une autre possibilité pour désigner un élément comme élément Java:

Clic droit sur l'élément ➔ Ajouter un (des) stéréotype(s) ➔ Java Designer ➔ "Java Package" ou "Java Class" etc.

- enfin, on peut générer le code :
  - Clic droit sur le package associé au TP5
  - Menu : Java Designer -> générer : le code est généré dans le répertoire défini pour la génération de code dans les préférences du module Java Designer
- il ne reste plus qu'à compiler (javac) et lancer (java) l'application

Q1 : Construisez un simple modèle (3 ou 4 classes) afin de tester la génération de code ainsi que la compilation (compilation en utilisant javac).

2 – Modification et Génération de code

Nous allons reprendre l'exercice de la semaine dernière et générer l'application avec le compteur graphique de personne.

Nous appellerons UINbPersonne le compteur graphique (L'observer du pattern Observer).

Q1 : Ajoutez le code nécessaire à l'enregistrement du composant selon le pattern Observer (mis en place la semaine dernière).

Q2 : Générez le code et vérifiez que votre composant s'affiche !

Q3 : Effectuez les modifications suivantes :

7. Faites que la classe UINbPersonne hérite de JLabel.
8. Ajoutez le code nécessaire à la classe UINbPersonne pour afficher dans le label le nombre de personnes actuellement enregistrées.
9. Ajoutez dans la méthode init de la classe UIRepertoire le code permettant l'affichage du UINbPersonne (un getContentPane().add(uiNbPers) devrait être suffisant si uiNbPers est une instance de UINbPersonne)

Q4 : Lors de l'exécution de l'application, ajoutez des personnes et vérifiez que le composant graphique se rafraîchit.

Q5 : Faites le reverse engineering du code généré.

Q6 : Inspectez le modèle obtenu.

Sorbonne Université – L3
3I012
Année 2018-2019

TD6. Diagrammes de séquence UML

Partie 1: Approche classes-puis-diagrammes de séquence

L'application ChampionnatDEchecs, qui devra permettre de gérer le déroulement d'un championnat d'échecs est actuellement en cours de développement. L'équipe de développement n'a pour l'instant réalisé qu'un diagramme de classe de cette application *(voir figure 1)*.

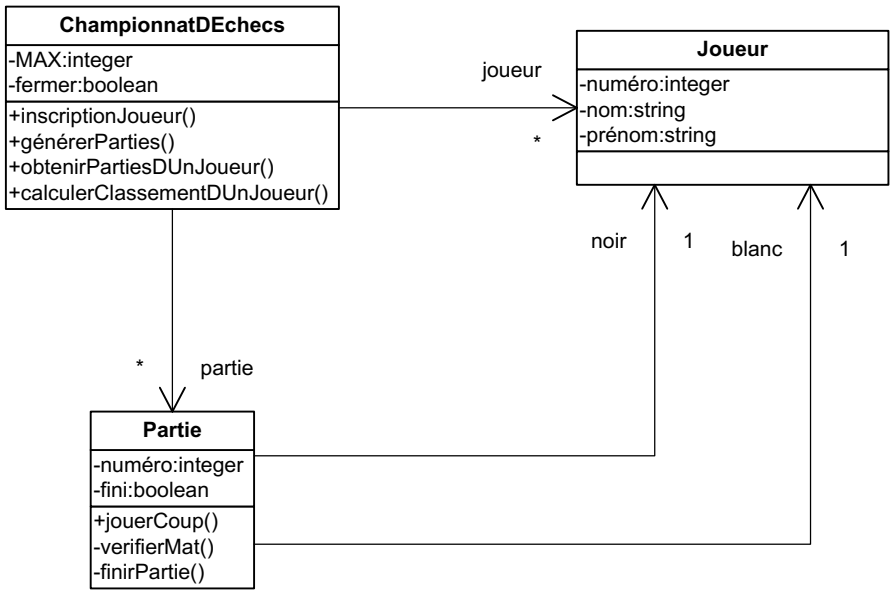


Figure 1 Classes de l'application ChampionnatDEchecs

La classe ChampionnatDEchecs représente un championnat d'échecs. Un championnat se déroule entre plusieurs joueurs (voir classe Joueur) et se joue en plusieurs parties (voir classe Partie). La propriété MAX de la classe ChampionnatDEchecs correspond au nombre maximal de joueurs que le championnat peut comporter. La propriété « fermer » permet de savoir si le championnat est fermé ou si de nouveaux joueurs peuvent s'inscrire.
ChampionnatDEchecs possède les opérations suivantes :

- inscriptionJoueur(in nom:string, in prénom:string) : integer permettant d'inscrire un nouveau joueur dans le championnat si le nombre de joueurs inscrits n'est pas déjà égal à MAX et si le championnat n'est pas déjà fermé. Si l'inscription est autorisée, cette opération crée le joueur et retourne son numéro dans le championnat.
- générerPartie() permet de fermer le championnat et de générer toutes les parties nécessaires.
- obtenirPartieDUnJoueur(in numéro :integer) : Partie[*] permet d'obtenir la liste de toutes les parties d'un joueur (dont le numéro est passé en paramètre).
- calculerClassementDUnJoueur(in numéro :integer) : integer permettant de calculer le classement d'un joueur (dont le numéro est passé en paramètre) pendant le championnat.

La classe Partie représente une des parties du championnat. La classe ChampionnatDEchecs est d'ailleurs associée avec la classe Partie, et l'association précise qu'un championnat peut

contenir plusieurs parties. Une partie se joue entre deux joueurs. Un joueur possède les pièces blanches et commence la partie alors que l'autre joueur possède les pièces noires. Les associations entre les classes Partie et Joueur précisent cela. La propriété « numéro » correspond au numéro de la partie (celui-ci doit être unique). La propriété « finie » permet de savoir si la partie a déjà été jouée ou pas.
La classe Partie possède les opérations suivantes :

- jouerCoup(in coup:string) permet de jouer un coup tant que la partie n'est pas finie. Le traitement associé à cette opération fait appel à l'opération « vérifierMat » afin de savoir si le coup joué ne met pas fin à la partie. Si tel est le cas, l'opération « finirPartie » est appelée.
- vérifierMat() : boolean permettant de vérifier si la position n'est pas mat.
- finirPartie() permet de préciser que la partie est finie. Il n'est donc plus possible de jouer de nouveaux coups.

La classe Joueur représente les joueurs du championnat. La classe Joueur est d'ailleurs associée avec la classe ChampionnatDEchecs, et l'association précise qu'un championnat peut contenir plusieurs joueurs. La propriété « numéro » correspond au numéro du joueur (celui-ci doit être unique). Les propriétés « nom » et « prénom » permettent de préciser le nom et le prénom du joueur.
Un championnat d'échecs se déroule comme suit :

1. Un administrateur de l'application crée un championnat avec une valeur MAX.
2. Les participants peuvent s'inscrire comme joueurs dans le championnat.
3. L'administrateur crée l'ensemble des parties.
4. Les participants, une fois inscrits, peuvent consulter leur liste de parties.
5. Les participants, une fois inscrits, peuvent jouer leurs parties. Nous ne nous intéressons qu'aux coups joués par chacun des deux joueurs. Nous ignorons l'initialisation de la partie (identification du joueur qui a les pions blancs et donc qui commence la partie).
6. Les participants peuvent consulter leur classement.

Question 1 Dans les questions suivantes, nous allons spécifier des exemples d'exécution de ChampionnatDEchecs avec des diagrammes de séquence. Comment modéliser les administrateurs et les participants ?

Question 2 Représentez par un diagramme de séquence le scénario d'exécution correspondant à la création d'un championnat et à l'inscription de deux joueurs. Vous assurerez la cohérence de votre diagramme avec le diagramme de classe fourni à la figure 1.

Question 3 Représentez par un diagramme de séquence le scénario d'exécution correspondant à la création de l'ensemble des parties pour le championnat créé à la question 2. Vous assurerez la cohérence de votre diagramme avec le diagramme de classe fourni à la figure 1.

Question 4 Représentez par un diagramme de séquence le scénario d'exécution correspondant au déroulement de la partie d'échecs entre deux joueurs. Vous pouvez considérer une partie qui se termine en quatre coups. Vous assurerez la cohérence de votre

diagramme avec le diagramme de classe fourni à la figure 1.

Question 5 Est-il possible de générer automatiquement le code d'une opération de cette application à partir de plusieurs diagrammes de séquence ?

Question 6 Quel est le type de diagrammes de séquence qu'on peut construire à partir du code d'une application ?

Question 7. Construisez le diagramme de séquence que nous pouvons obtenir à partir de code Java de la méthode *enregistrerCompte()* définie dans une classe *Main* ci-dessous:

```
public void enregistrerCompte(String name, int age){

        ArrayList<Account> liste = new ArrayList<Account>();
        Account accoount = new Account();

        account.setId(1);
        account.setName(name);
        account.setAge(age);
        liste.add(account);

 }

}
```

Partie 2: Approche diagrammes de séquence-puis-classes

Une équipe de développement souhaite réaliser une application Calculus qui permet à des utilisateurs d'effectuer des opérations arithmétiques simples sur des entiers : addition, soustraction, produit, division. Cette application a aussi une fonction mémoire qui permet à l'utilisateur de stocker un nombre entier qu'il pourra ensuite utiliser pour n'importe quelle opération. Les opérations peuvent directement s'effectuer sur la mémoire. L'utilisateur se connecte et ouvre ainsi une nouvelle session. Puis, dans le cadre d'une session, l'utilisateur peut demander au système d'effectuer une suite d'opérations.

Question 8 Utilisez des diagrammes de séquences pour représenter les différents scénarios d'exécution du service Calculus.

Question 9 Pour chacune des instances apparaissant dans votre diagramme de classe, créez la classe correspondante.

Sorbonne Université – L3
3I012
Année 2018-2019

TME6. Diagrammes de séquence

Partie 1 : Approche classes-puis-séquences

Dans un nouveau package de votre projet Modelio. Construire deux classes :

- **Livre** avec les attributs Type, Prix, Année, Auteur, etc.
- **Librairie** (qui contient des livres) Avec les opérations *ajouterLivre()*, *supprimerLivre()*.

Définissez un diagramme de séquences pour chaque opération. Pour chaque nouvelle ligne de vie de votre diagramme, créez une instance (explorateur UML dans Modelio), liez la classe correspondante à cette instance et liez l'instance à la ligne de vie (onglet Propriétés de l'élément concerné).

Partie 2 - Approche séquences-puis-classes

Dans un nouveau package de votre projet Modelio. Avant de construire vos classes, définissez le diagramme de séquences représentant, pour une application de commerce électronique, la création d'un panier, l'ajout d'un article dans le panier et la validation de la commande.

Pour chaque ligne de vie de votre diagramme de séquences, créez une instance et une classe correspondantes (explorateur UML dans Modelio) et liez la classe à l'instance et l'instance à la ligne de vie (onglet Propriétés de l'élément concerné).

Partie 3 : Sur l'application MyAssistant

Construire les diagrammes de séquences permettant de représenter les scénarii suivants :

1. Enregistrement d'un observer auprès du répertoire

2. Notification de l'ajout d'une personne auprès d'un observer

3. Ajout d'une personne dans le répertoire

Sorbonne Université – L3
3I012
Année 2018-2019

TD7. Diagrammes de séquence et tests

La classe Partie de l'application de gestion de championnat d'échecs présentée au TD6 représente une partie d'échecs. Elle permet aux joueurs de jouer leur partie en appelant l'opération jouerCoup(). Chaque fois qu'un coup est joué, l'opération vérifierMat() est appelée afin de vérifier que la position n'est pas mat. Si tel est le cas, la partie est finie. Aucun coup ne peut alors être joué (voir TD6 pour la modélisation de classe Partie ainsi qu'un diagramme de séquence spécifiant un cas nominal de déroulement d'une partie entre deux joueurs).

Question 1. Identifiez une faute qui pourrait intervenir lors du déroulement d'une partie.

Question 2. Définissez un cas de test abstrait visant à révéler cette faute.

Question 3. Construisez un diagramme de séquence de test modélisant le cas de test abstrait de la question précédente.

Question 4. Écrivez le pseudo-code Java du cas de test exécutable correspondant au cas de test abstrait de la question précédente.

Question 5. Si ce cas de test ne révèle pas de faute, est-ce que cela signifie que l'application ne contient pas de défaillance ?

Question 6. Combien de cas de test faudrait-il élaborer pour améliorer la qualité de l'application ?

L'application permettant la gestion de championnat d'échecs contient aussi la classe ChampionnatDEchecs, qui est associée à la classe Partie et qui permet de gérer l'inscription des joueurs et la création des parties (voir TD6).

Question 7. Identifiez une faute qui pourrait intervenir lors de la création des parties d'un championnat. Définissez un cas de test abstrait visant à révéler cette faute, et construisez un diagramme de séquence de test modélisant ce cas de test abstrait.

Question 8. Est-il possible de lier les deux cas de test abstrait que vous avez définis (un à la question 2, l'autre à la question 7) ?

Sorbonne Université – L3
3I012
Année 2018-2019

TD8. Réflexion sur les classes : vers l'indépendance des plates-formes d'exécution

Question 1. Le diagramme de l'agence de voyage représenté à la figure 1 correspond-t-il à un modèle conceptuel ou à un modèle physique ?

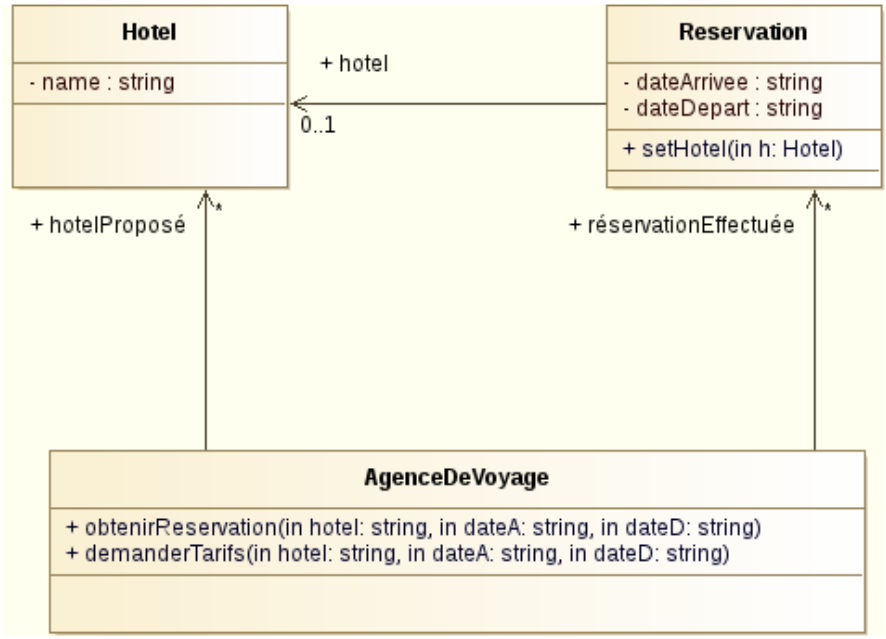


Figure 1 Classes de l'agence de voyage

Question 2. Pensez-vous qu'il soit intéressant d'appliquer des patrons de conception sur les modèles conceptuels ?

Question 3. Le diagramme de séquence représenté à la figure 2 est-il conceptuel ou physique ? Vous noterez qu'il fait intervenir une opération qui n'apparaît pas dans le diagramme de classe initial. Quelle classe doit posséder cette opération ?

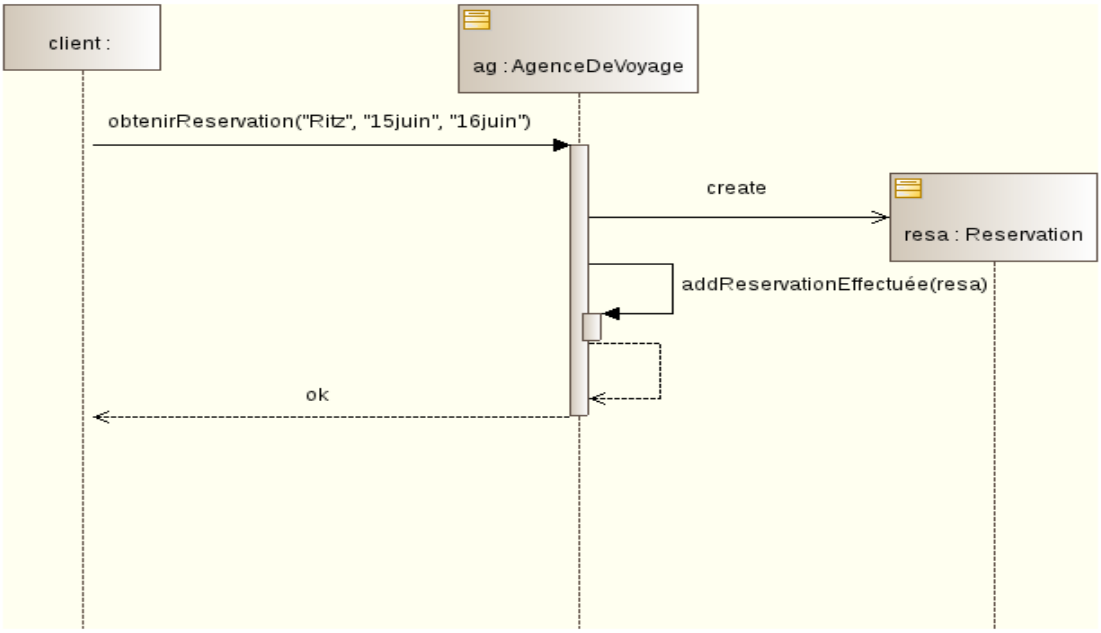


Figure 2 Interaction représentant une réservation

Question 4. Serait-il possible de spécifier en « 100 % UML » le comportement de l'agence de voyage ?

Question 5. Serait-il possible de spécifier en « 100 % UML » des tests pour l'agence de voyage ? Justifiez l'intérêt de ces tests.

Question 6. Le diagramme représenté à la figure 3 est une concrétisation du diagramme conceptuel de l'agence de voyage. Exprimez les relations d'abstraction entre les éléments des deux diagrammes.

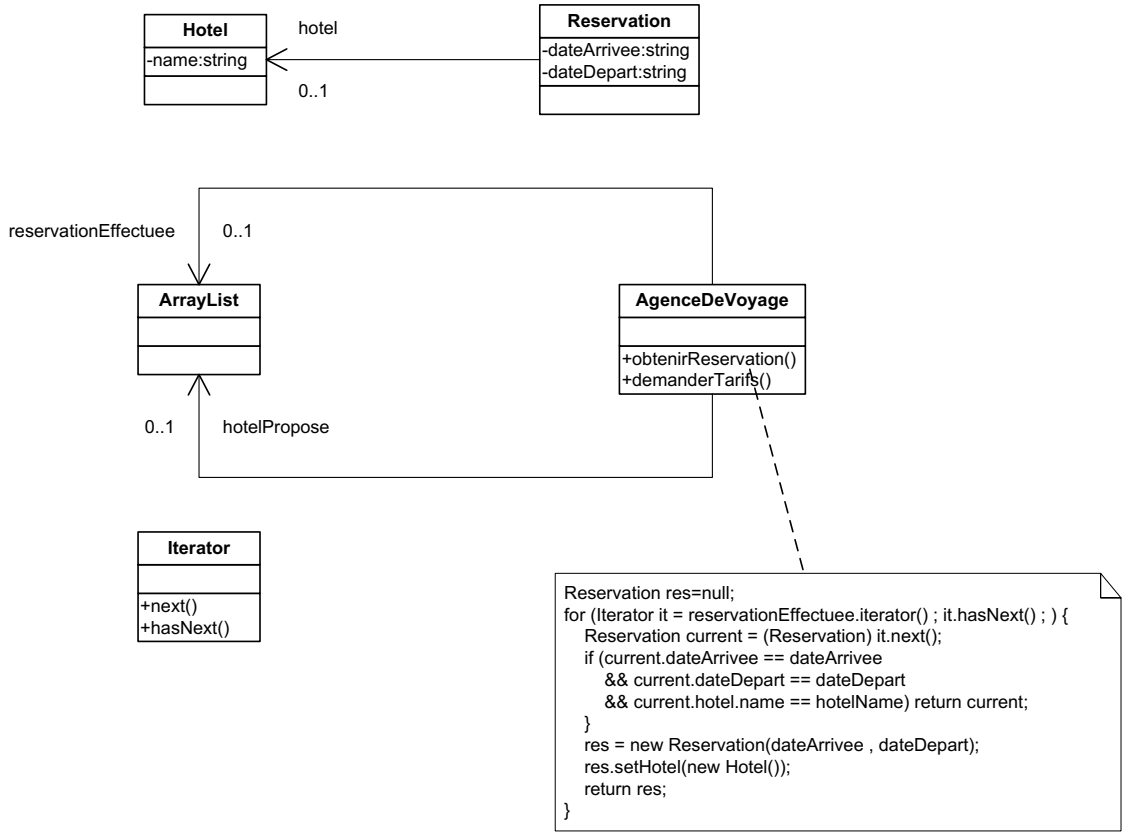


Figure 3 Classes du niveau physique de l'agence de voyage

Question 7. Quel est l'intérêt d'avoir fait apparaître les classes ArrayList et Iterator dans le modèle concret (considérez en particulier la génération de code et le Reverse Engineering) ?

Question 8. Construisez le diagramme de séquence concrétisant le diagramme de séquence présenté à la question 1.

Question 9. Exprimez les relations d'abstraction entre les diagrammes de séquence.

Sorbonne Université – L3
3I012
Année 2018-2019

TME8

Réflexion sur les classes : vers l'indépendance des plates-formes techniques.

Partie 1 : MyAssistant Conceptuel vs Physique

Q1 - A partir de l'application MyAssistant, créez un nouveau package contenant les abstractions des classes des packages que vous avez déjà construits (classes conceptuelles). Vous vous arrangerez pour que ces classes n'aient aucune dépendance avec les classes de l'API Java™. Vous proposerez notamment des moyens pour abstraire les concepts Java™ tels que JPanel, Vector, etc.

Q2 - Dans Modelio, faites en sorte que ces classes ne permettent pas de génération de code.

Q3 - Trouvez un moyen dans Modelio pour exprimer les liens d'abstraction entre les classes conceptuelles et les classes physiques.

Q4 - Reproduisez les diagrammes de séquences que vous avez déjà fait sur ces classes (diagrammes de séquences conceptuels). Pensez vous qu'il soit possible de générer des tests à partir de ces diagrammes de séquences ?

Q5 - Trouvez un moyen dans Modelio pour exprimer les liens d'abstraction entre les diagrammes de séquences conceptuels et les diagrammes de séquences physiques.

Q6 - Générez une documentation de votre nouveau projet et arrangez vous pour que l'on distingue bien le niveau conceptuel et le niveau qui permet la génération de code (appelé niveau physique).

Q7 – Pensez-vous qu'il soit possible d'abstraire ainsi tous les concepts des langages de programmation (Socket, Graphique 2D/3D, Impression, XML, etc.) ?

Q8 – Pensez-vous qu'il soit possible de transformer automatiquement le niveau physique en niveau conceptuel ? Et l'inverse ?

Sorbonne Université – L3
3I012
Année 2018-2019

TD9. Le diagramme de cas d'utilisation UML

Le diagramme de cas d'utilisation de la figure 1 représente les fonctionnalités d'une agence de voyage classique.

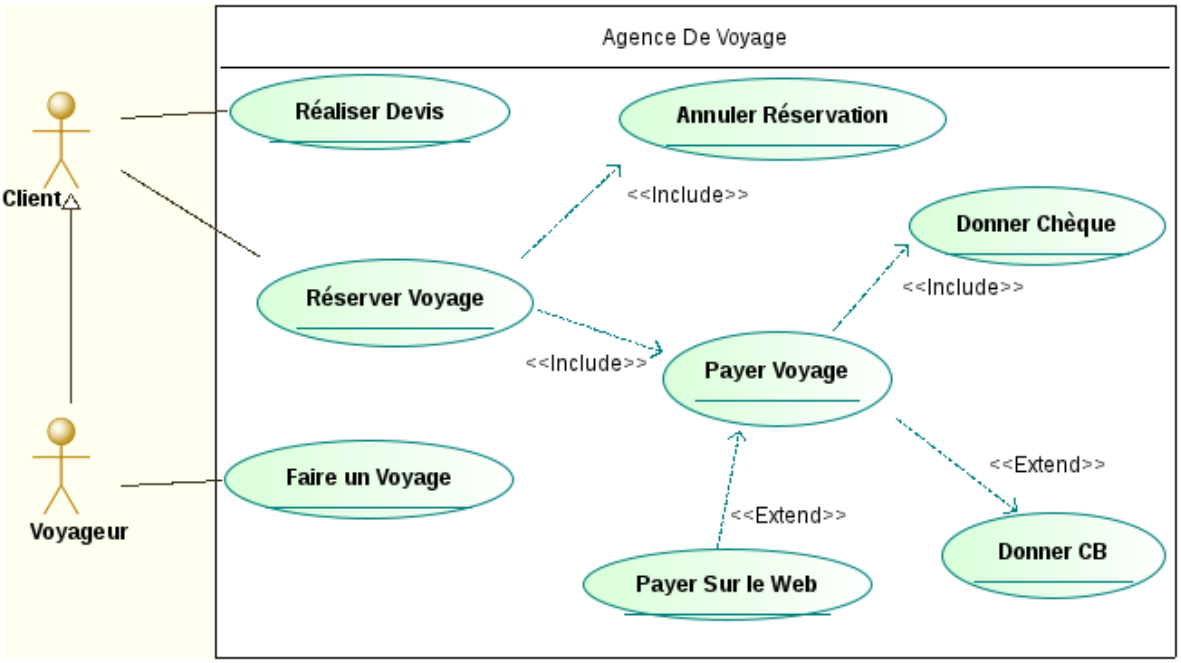


Figure 1 Diagramme de cas d'utilisation de l'agence de voyage

Question 1. Commentez les acteurs du diagramme de cas d'utilisation.

Question 2. Commentez les cas d'utilisation du diagramme de cas d'utilisation.
Nous souhaitons réaliser le diagramme de cas d'utilisation du championnat d'échecs présenté au TD6.

Question 3. Donnez la liste des acteurs du système.

Question 4. Donnez la liste des cas d'utilisation du système en les liant aux acteurs.

Question 5. Donnez le diagramme de cas d'utilisation du système.

Question 6. Reprenez les diagrammes de séquence réalisés au TD6 pour l'application de championnat d'échecs, et expliquez comment les relier au diagramme de cas d'utilisation obtenu à la question précédente.

Sorbonne Université – L3
3I012
Année 2018-2019

TME9

Le diagramme de Use Case UML

ATTENTION : les résultats de ce TP seront utilisés en TD10

Application Databirds

Une association d'ornithologie vous confie la réalisation du système logiciel de recueil et de gestion des observations réalisées par ses adhérents (le logiciel DataBirds). L'objectif est de centraliser toutes les données d'observation arrivant par différents canaux au sein d'une même base de données, qui permettra ensuite d'établir des cartes de présence des différentes espèces sur le territoire géré par l'association.
Les données à renseigner pour chaque observation sont les suivantes :

10. Nom de l'espèce concernée. Il y a environ trois cents espèces possibles sur le territoire en question. Si l'observation concerne plusieurs espèces, renseigner plusieurs observations.
11. Nombre d'individus.
12. Lieu de l'observation.
13. Date de l'observation.
14. Heure de l'observation.
15. Conditions météorologiques lors de l'observation.
16. Nom de chaque observateur.

Quelle que soit la façon dont sont collectées les données, celles-ci sont saisies dans la base dans un état dit « à valider ». Tant que les données ne sont pas validées par les salariés de l'association, des modifications peuvent être faites sur les données.
La validation des données se fait uniquement par les salariés de l'association qui ont le droit de modifier la base de DataBirds. Ils doivent vérifier que les données saisies sont cohérentes. Plus précisément, ils doivent valider les noms des observateurs (les noms doivent correspondre à des noms d'adhérents) et l'espèce (celle-ci doit correspondre à une espèce connue sur le territoire).
Après validation, une saisie se trouve soit dans l'état dit « validé », soit dans l'état dit « non validé ». Les saisies dans l'état « non validé » sont automatiquement purgées de la base une fois par semaine.
Grâce aux données saisies et validées, l'association souhaite pouvoir établir différents types de cartes de présence des différentes espèces :

- Cartes géographiques par espèce présentant un cumul historique des populations. Ce traitement peut être demandé par un adhérent.
- Cartes des observations réalisées par chaque observateur. Ce traitement peut être demandé par un salarié uniquement.

Ces cartes de présence des oiseaux sont générées par DataBirds et accessibles soit par le Web, soit par demande *via* un courrier électronique ou postal.

Q1 : Faites le diagramme de cas d'utilisation du niveau besoin de l'application.

Q2 : Pour chaque cas d'utilisation identifié, faites un diagramme de séquence représentant un fonctionnement correct et un diagramme de séquence représentant un fonctionnement provoquant une erreur. N'oubliez pas que ces diagrammes de séquence doivent faire apparaître les acteurs identifiés en question 1.

TD10. Application de notre méthode de développement

Application DataBirds

Vous avez déjà réalisé en TP les étapes 1 et 2 de la méthode de développement présentée en cours. Vous avez donc déjà à votre disposition un diagramme de cas d'utilisation du niveau fonctionnel et des diagrammes de séquence précisant les cas d'utilisation.

Question 1 : Construisez un diagramme de classe représentant les données spécifiées dans l'application (étape 3 de la méthode). Faites bien attention à retrouver dans ce diagramme les classes que vous avez utilisées pour la réalisation des diagrammes de séquences que vous avez faits en TP. Les diagrammes faits en TP9 et celui de cette question modélisent le niveau besoin de l'application.

Nous nous intéressons maintenant au niveau conceptuel de l'application.

Question 2 : Identifiez les composants du système et pour chacun d'eux produisez un diagramme de cas d'utilisation (étape 4 de la méthode). Précisez les relations de réalisation entre les cas d'utilisation du niveau besoin et ceux du niveau conceptuel.

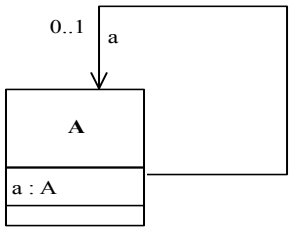
Question 3 : Pour chaque cas d'utilisation identifié en question 2, faites un diagramme de séquence représentant un fonctionnement correct et un diagramme de séquence représentant un fonctionnement provoquant une erreur (étape 5 de la méthode). N'oubliez pas que ces diagrammes de séquence doivent faire apparaître les acteurs identifiés en question 2.

Question 4 : Construisez un diagramme de classe par composant de l'application (étape 6 de la méthode). Vous préciserez les dépendances entre composants. Faites bien attention à retrouver dans ce diagramme les classes que vous avez utilisées pour la réalisation des diagrammes de séquences de la question 3. Les diagrammes élaborés en questions 2, 3 et 4 modélisent le niveau conceptuel de l'application.

Sorbonne Université – L3
3I012
Année 2018-2019

TD11
Synthèse et conclusion

Diagrammes de classe

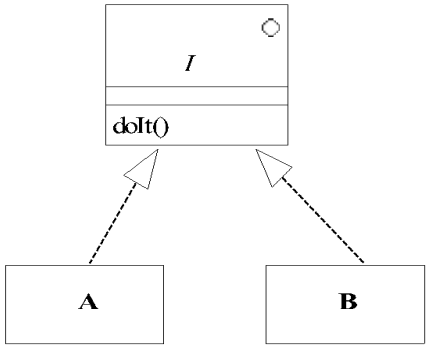


Q1 : Un attribut d'une classe peut-il être typé par sa propre classe ?

Q2 : Une association peut-elle avoir une même classe comme source et comme cible ?

Q3 : Quelles sont les différences entre l'association et l'attribut de la figure précédente ?

Q4 : A quoi servent les noms des rôles des associations ? Donnez un exemple illustrant leur intérêt.

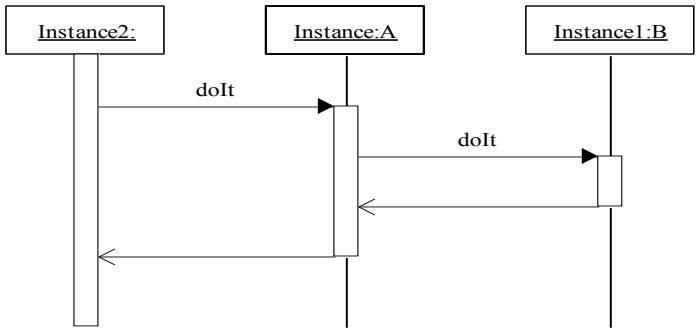


Q5 : Quelle est la signification de ce diagramme pour les objets instances de A et de B ?

Q6 : Un objet peut-il être instance de A **et** de B ? (Justifiez votre réponse)

Sorbonne Université – L3
3I012
Année 2018-2019

Diagrammes de séquences

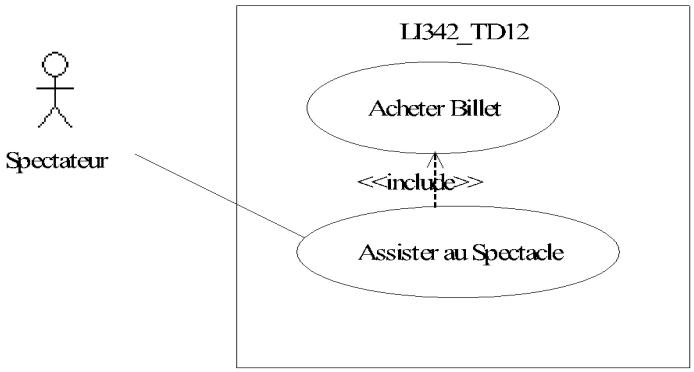


Q7 : Que manque-t-il à ce diagramme pour être suffisamment complet ?

Q8 : Peut-on dire que les instances de A appellent l'opération doIt sur des instances de B à chaque fois que l'on leur demande de réaliser l'opération doIt ?

Q9 : Comment faudrait-il compléter ce diagramme de séquences pour modéliser complètement le comportement de l'opération doIt ?

Diagrammes de cas d'utilisation



Q10 : Expliquez en quoi ce diagramme de cas d'utilisation n'est pas bien fait ?

Q11 : Proposez un ou plusieurs diagrammes de cas d'utilisation illustrant les relations d'héritage, d'inclusion et d'extension entre cas d'utilisation.

Méthode

Q12 : Est-il obligatoire de faire toutes les étapes de la méthode vue dans ce cours pour construire une application ?
Q13 : La méthode vue dans ce cours est-elle uniquement applicable pour la construction d'application Java ?
Q14 : Quels sont pour vous les trois avantages majeurs de UML ainsi que les trois inconvénients ?

Annexe code d'un carnet d'adresses

```
/-----------------------------------------------------------------------------------------------------------/
package repertoire;

public class MyAssistant {
	public static void main(String[] args) {
		UIRepertoire ihm = new UIRepertoire();
	}

}
/-----------------------------------------------------------------------------------------------------------/
package repertoire;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;

import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JScrollPane;

import javax.swing.JSplitPane;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

public class UIRepertoire extends JFrame {

	Repertoire theRepertoire;
	UIMenuActionListener menuListener;

	JMenuBar menu_barre;
	JMenu repertoire_menu, fonction_menu, aide_menu;
	JMenuItem repertoire_menu_ouvrir,
		repertoire_menu_enregistrer,
		repertoire_menu_enregistrersous,
		repertoire_menu_nouveau,
		fonction_menu_ajouterPersonne,
		fonction_menu_rechercherPersonne,
		aide_menu_item;

	JSplitPane splitPane;

	JList repertoireView;
	UIPersonne uipersonne;

	public Repertoire getTheRepertoire() {
		return theRepertoire;
	}

	public void setTheRepertoire(Repertoire theRepertoire) {
		this.theRepertoire = theRepertoire;
		refreshUIRepertoire();
	}

	public UIRepertoire() {
		super("Mon Repertoire");
		menuListener = new UIMenuActionListener(this);
		WindowListener l = new WindowAdapter() {
			public void windowClosing(WindowEvent e) {
				System.exit(0);
```

```
                }
                public void windowClosed(WindowEvent e) {
                        System.exit(0);
                }
        };
        addWindowListener(l);
        init();
}

public UIRepertoire(Repertoire rep) {
        super("Mon Repertoire");
        theRepertoire = rep;
        menuListener = new UIMenuActionListener(this);
        WindowListener l = new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                        System.exit(0);
                }
                public void windowClosed(WindowEvent e) {
                        System.exit(0);
                }
        };
        addWindowListener(l);
        init();
        refreshUIRepertoire();
}

void init() {
        //Barre de Menu
        menu_barre = new JMenuBar();
        setJMenuBar(menu_barre);

        // Menu FICHIER
        repertoire_menu = new JMenu("Fichier");
        menu_barre.add(repertoire_menu);

        repertoire_menu_nouveau = new JMenuItem("Nouveau");
        repertoire_menu.add(repertoire_menu_nouveau);
        repertoire_menu_nouveau.addActionListener(menuListener);
        repertoire_menu_ouvrir = new JMenuItem("Ouvrir");
        repertoire_menu.add(repertoire_menu_ouvrir);
        repertoire_menu_ouvrir.addActionListener(menuListener);
        repertoire_menu_enregistrer = new JMenuItem("Enregistrer");
        repertoire_menu.add(repertoire_menu_enregistrer);
        repertoire_menu_enregistrer.addActionListener(menuListener);
        //fichier_menu_enregistrer.setMnemonic(KeyEvent.VK_S);
        repertoire_menu_enregistrersous = new JMenuItem("Enregistrer Sous");
        repertoire_menu.add(repertoire_menu_enregistrersous);
        repertoire_menu_enregistrersous.addActionListener(menuListener);

        // Menu FONCTION
        fonction_menu = new JMenu("Organisation");
        menu_barre.add(fonction_menu);

        fonction_menu_ajouterPersonne =
                new JMenuItem("Ajouter Nouvelle Personne");
        fonction_menu.add(fonction_menu_ajouterPersonne);
        fonction_menu_ajouterPersonne.addActionListener(menuListener);

        fonction_menu_rechercherPersonne =
                new JMenuItem("Rechercher Personne(s)");
        fonction_menu.add(fonction_menu_rechercherPersonne);
        fonction_menu_rechercherPersonne.addActionListener(menuListener);
```

```
		// Menu AIDE
		aide_menu = new JMenu("Aide");
		menu_barre.add(aide_menu);

		aide_menu_item = new JMenuItem("A Propos");
		aide_menu_item.addActionListener(menuListener);
		aide_menu.add(aide_menu_item);

		//Mettre un SplitPane
		splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
		getContentPane().add(splitPane);

		setVisible(true);
		pack();

	}

	public void refreshUIRepertoire() {
		// Mettre la JList ‡ gauche
		repertoireView = new JList(theRepertoire.listerPersonnes());
		repertoireView.addListSelectionListener(new ListSelectionListener() {
			public void valueChanged(ListSelectionEvent e) {
				System.out.println("Ok");
				Personne p = (Personne) repertoireView.getSelectedValue();
				uipersonne.setPersonne(p);

			}
		});

		splitPane.setLeftComponent(new JScrollPane(repertoireView));

		//Test ‡ droite
		if (theRepertoire.listerPersonnes().length!=0) {
			uipersonne = new UIPersonne(theRepertoire.listerPersonnes()[0]);
			splitPane.setRightComponent(uipersonne);
		}

	}

}
/-------------------------------------------------------------------------------------------------------/
package repertoire;

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class UIPersonne extends JPanel {

	Personne personne;

	JTextField nomTF,
		prenomTF,
		telMaisonTF,
		telPortTF,
		telBurTF,
		faxTF,
		titreTF,
```

```
            socTF,
            addTF,
            mailTF;

    public UIPersonne() {
            super();
            init();
    }

    public UIPersonne(Personne p) {
            super();
            personne = p;
            init();
    }

    public Personne getPersonne() {
            return personne;
    }

    public void setPersonne(Personne personne) {
            this.personne = personne;
            prenomTF.setText(personne.getPrenom());
            nomTF.setText(personne.getNom());
            telBurTF.setText(personne.getTelephoneBureau());
            telMaisonTF.setText(personne.getTelephoneMaison());
            telPortTF.setText(personne.getTelephonePortable());
            faxTF.setText(personne.getFax());
            titreTF.setText(personne.getTitre());
            socTF.setText(personne.getSociete());
            //Adresse
            mailTF.setText(personne.getMail());
    }

    public void init() {
            this.setLayout(new GridLayout(0, 2));
            add(new JLabel("nom"));
            nomTF = new JTextField("");
            add(nomTF);

            add(new JLabel("prenom"));
            prenomTF = new JTextField("");
            add(prenomTF);
            add(new JLabel("telephone maison"));
            telMaisonTF = new JTextField("");
            add(telMaisonTF);
            add(new JLabel("telephone portable"));
            telPortTF = new JTextField("");
            add(telPortTF);
            add(new JLabel("telephone bureau"));
            telBurTF = new JTextField("");
            add(telBurTF);
            add(new JLabel("fax"));
            faxTF = new JTextField("");
            add(faxTF);
            add(new JLabel("titre"));
            titreTF = new JTextField("");
            add(titreTF);
            add(new JLabel("sociÈtÈ"));
            socTF = new JTextField("");
            add(socTF);
            add(new JLabel("adresse"));
            addTF = new JTextField("");
            add(addTF);
```

```
		add(new JLabel("mail"));
		mailTF = new JTextField("");
		add(mailTF);
		JButton save = new JButton("Save");
		save.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				personne.setPrenom(prenomTF.getText());
				personne.setNom(nomTF.getText());
				personne.setTelephoneBureau(telBurTF.getText());
				personne.setTelephoneMaison(telMaisonTF.getText());
				personne.setTelephonePortable(telPortTF.getText());
				personne.setFax(faxTF.getText());
				personne.setTitre(titreTF.getText());
				personne.setSociete(socTF.getText());
				//personne.setAdresse(addTF.getText());
				personne.setMail(mailTF.getText());
			}
		});
		add(save);
		JButton cancel = new JButton("Cancel");
		cancel.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				prenomTF.setText(personne.getPrenom());
				nomTF.setText(personne.getNom());
				telBurTF.setText(personne.getTelephoneBureau());
				telMaisonTF.setText(personne.getTelephoneMaison());
				telPortTF.setText(personne.getTelephonePortable());
				faxTF.setText(personne.getFax());
				titreTF.setText(personne.getTitre());
				socTF.setText(personne.getSociete());
				//Adresse
				mailTF.setText(personne.getMail());
			}
		});
		add(cancel);

	}

}
/-------------------------------------------------------------------------------------------------------------/
package repertoire;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JMenuItem;

public class UIMenuActionListener implements ActionListener {

	UIRepertoire uirep;

	public UIMenuActionListener(UIRepertoire uirep) {
		super();
		this.uirep = uirep;
	}

	public void actionPerformed(ActionEvent ev) {
		JMenuItem test = (JMenuItem) ev.getSource();
		if (test.getText() == "A Propos")
			System.out.println("Aide");
		else if (test.getText() == "Rechercher Personne(s)") {
			System.out.println("LOAD ");
		}
```

```
                else if (test.getText() == "Ajouter Nouvelle Personne") {
                        System.out.println("Ajouter Nouvelle Personne ");
                        Personne p = new Personne();
                        uirep.getTheRepertoire().ajouterPersonne(p);
                        uirep.refreshUIRepertoire();

                }
                else if (test.getText() == "Rechercher Personne(s)") {
                        System.out.println("LOAD ");

                }
                else if (test.getText() == "Nouveau") {
                        System.out.println("Nouveau ");
                        uirep.setTheRepertoire(new Repertoire());
                }
                else if (test.getText() == "Enregistrer Sous") {
                        System.out.println("LOAD ");

                }
                else if (test.getText() == "Enregistrer") {
                        System.out.println("LOAD ");

                }
                else if (test.getText() == "Ouvrir") {
                        System.out.println("LOAD ");

                }
        }

}
/-------------------------------------------------------------------------------------------------------/
package repertoire;

public class Adresse {
        String pays;
        String region;
        String codePostal;
        String ville;
        String rue;

        public String getCodePostal() {
                return codePostal;
        }

        public void setCodePostal(String codePostal) {
                this.codePostal = codePostal;
        }

        public String getPays() {
                return pays;
        }

        public void setPays(String pays) {
                this.pays = pays;
        }

        public String getRegion() {
                return region;
        }

        public void setRegion(String region) {
                this.region = region;
        }
```

```
	public String getRue() {
		return rue;
	}

	public void setRue(String rue) {
		this.rue = rue;
	}

	public String getVille() {
		return ville;
	}

	public void setVille(String ville) {
		this.ville = ville;
	}

}
/-----------------------------------------------------------------------------------------------------------/
package repertoire;

public class Personne {
	String nom;
	String prenom;
	String telephoneMaison;
	String telephonePortable;
	String telephoneBureau;
	String fax;
	String titre;
	String societe;
	Adresse adresse;
	String mail;

	public Adresse getAdresse() {
		return adresse;
	}

	public void setAdresse(Adresse adresse) {
		this.adresse = adresse;
	}

	public String getFax() {
		return fax;
	}

	public void setFax(String fax) {
		this.fax = fax;
	}

	public String getMail() {
		return mail;
	}

	public void setMail(String mail) {
		this.mail = mail;
	}

	public String getNom() {
		return nom;
	}

	public void setNom(String nom) {
		this.nom = nom;
```

```
	}

	public String getPrenom() {
		return prenom;
	}

	public void setPrenom(String prenom) {
		this.prenom = prenom;
	}

	public String getSociete() {
		return societe;
	}

	public void setSociete(String societe) {
		this.societe = societe;
	}

	public String getTelephoneBureau() {
		return telephoneBureau;
	}

	public void setTelephoneBureau(String telephoneBureau) {
		this.telephoneBureau = telephoneBureau;
	}

	public String getTelephoneMaison() {
		return telephoneMaison;
	}

	public void setTelephoneMaison(String telephoneMaison) {
		this.telephoneMaison = telephoneMaison;
	}

	public String getTelephonePortable() {
		return telephonePortable;
	}

	public void setTelephonePortable(String telephonePortable) {
		this.telephonePortable = telephonePortable;
	}

	public String getTitre() {
		return titre;
	}

	public void setTitre(String titre) {
		this.titre = titre;
	}

	public String toString() {
		return nom+" "+prenom;
	}

}
/-----------------------------------------------------------------------------------------------------/
package repertoire;

import java.util.Iterator;
import java.util.ArrayList;

public class Repertoire {
```

```
        ArrayList <Personne> personnes;

        public void ajouterPersonne(Personne p) {
                personnes.add(p);
        }

        public void supprimerPersonne(Personne p) {
                personnes.remove(p);
        }

        public Personne[] rechercherPersonnesParNom(String nom) {
                ArrayList <Personne> success = new ArrayList<Personne>();
                for (Iterator it = personnes.iterator() ; it.hasNext();) {
                        Personne current = (Personne) it.next();
                        if (current.getNom().compareTo(nom)==0) success.add(current);
                }
                Personne[] res = new Personne[0];
                return (Personne[]) success.toArray(res);
        }

        public Personne[] listerPersonnes() {
                Personne[] res = new Personne[0];
                return personnes.toArray(res);
        }

        public Repertoire() {
                personnes = new ArrayList<Personne>();
        }

}
```

Sorbonne Université – L3
3I012
Année 2018-2019

Examen LI342 – 16 Décembre 2011

Tous documents autorisés !
Toute réponse doit impérativement être justifiée !!!

Questions de cours (4 points)

C1. Dans quel(s) cas doit-on utiliser un patron de conception ?
C2. Un cas d'utilisation peut-il hériter d'un cas d'utilisation qu'il étend ?
C3. Quand peut-on dire qu'une classe A dépend d'une classe B ?
C4. Selon quel(s) critère(s) peut-on rendre une association navigable ?

Exercice (16 points)

Nous considérons l'application eSROA, propriété de la Société de Restauration d'Oeuvres d'Art (SROA), qui offre à ses clients de gérer la restauration de leurs oeuvres d'art (dépôt d'un dossier de demande de restauration, choix d'un devis particulier et établissement d'un contrat de restauration). Outre la nature (nom, description, photo(s)) de l'oeuvre à restaurer, un client peut exprimer dans son dossier ses souhaits concernant la restauration (e.g. nettoyage, restauration des couleurs d'origine, réparation du support physique de l'oeuvre, etc.). Il finalise ensuite sa demande en fermant le dossier. Pour traiter le dossier d'un client (on peut imaginer que les dossiers sont traités dans l'ordre dans lesquels ils ont été déposés sur le site), l'un des experts de la SROA doit le récupérer, afficher les informations qu'il contient, le prendre officiellement en charge, poser un diagnostic général concernant l'état de l'oeuvre et proposer un ou plusieurs devis, chaque devis détaillant un ensemble de restaurations (pas forcément limité aux souhaits des clients) et reflétant une certaine qualité de service offerte au client. Le client peut afficher les différents devis concernant son dossier. Il peut alors choisir l'un des devis ou les rejetter en bloc et demander à ce que son dossier soit pris en charge par un autre expert. Quand un devis satisfait le client, il est validé par celui-ci et un contrat définitif est automatiquement généré par eSROA à partir des informations contenues dans le devis. Une copie papier de ce contrat sera ensuite apportée au client pour signature par un coursier de la SROA.

E1.Définissez le diagramme de cas d'utilisation de l'application eSROA.

E2.Définissez le diagramme de classe d'analyse, i.e. de niveau Besoin, de l'application eSROA.

Considérons à présent le diagramme de classes conceptuel de l'application eSROA, présenté dans la figure 2.
Attention: il n'existe pas dans le modèle d'autres opérations que celles présentées dans le diagramme de classes en figure 2.

E3. Présentez dans un diagramme de séquence le dépôt d'un dossier complet par un client qui ajoute une photo de l'oeuvre à restaurer (oeuvre non répertoriée auparavant dans eSROA) au dossier et qui exprime deux souhaits concernant sa restauration.
E4. Présentez dans un diagramme de séquence le scénario suivant: un expert de la SROA obtient le dossier créé à la question E3, pose un diagnostic général et établit un devis de restauration. Dans votre scénario, le client concerné doit regarder ce devis puis le valider.
E5. Définissez le diagramme de classe physique de l'application eSROA. On ne considérera pas les notes de code des opérations.
E6. Définissez une faute pouvant se produire lors du dépôt d'un dossier par un client. Présentez un diagramme de séquence de test permettant de révéler cette faute.

E7. Le SRT souhaite offrir la possibilité à ses clients les plus fidèles d'assister à des cours (Master Class) de conservation préventive de leurs oeuvres les plus précieuses. Pour pouvoir gérer ce nouveau service au travers de eSROA, les concepteurs de eSROA proposent d'intégrer la classe Java MasterClass.java dont le code est donné ci-dessous. Présentez le diagramme de classes physique de eSROA mis à jour après le reverse engineering de la classe java MasterClass.java et l'ajout de deux opérations à la classe eSROA: listerToutesLesMasterClasses() et inscriptionClientMasterClass(). Vous devrez définir les signatures de ces opérations et modifier (si besoin est) le diagramme de classes en conséquence.

```
import java.util.ArrayList ;
public class MasterClass {
   int idMasterClass ;
   String intituleCours ;
   int dateCours ;
   ArrayList clients = new ArrayList() ;
   ajouterClient(Client c) {}
   getIdMasterClass() {}
   getIntituleCours() {}
   getDateCours() {}
}
```

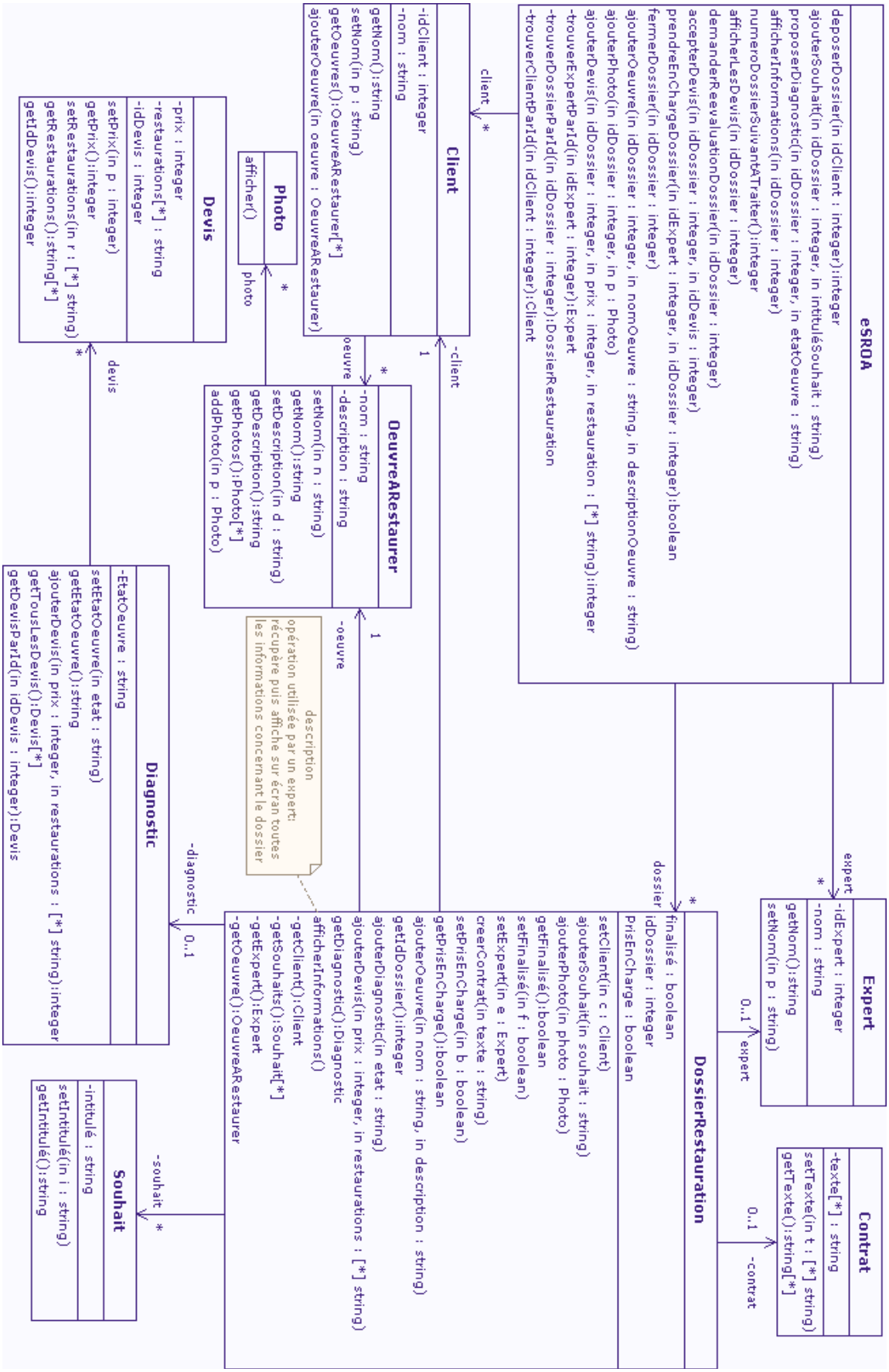


Figure 2: Diagramme de classes conceptuel de eSROA