Programmation Orientée Objet
IG2I, LE3 – 2016-2017

Session 2 – 11 Avril 2017

Diego Cattaruzza, Philippe Kubiak, Maxime Ogier

Conditions de l'examen

- **durée** : 2h00 ;
- ordinateurs personnels autorisés ;
- documents et pages internet autorisées ;
- accès à Moodle autorisé ;
- les moyens de communication (mails, messageries instantanées, Facebook, Tweeter, téléphones portables, Google Drive, etc.) sont INTERDITS ; le fait qu'un moyen de communication ne soit pas listé ci-dessus ne vous donne pas le droit de l'utiliser ;
- si vous utilisez du code que vous n'avez pas développé par vous même lors du CTP, vous devez citer vos sources (dans les commentaires) ; ne pas citer ses sources est une fraude.

Rendu

Vous déposez sur Moodle, dans la zone de dépôt **Dépôt du RTP 2016/2017** une archive compressée de votre répertoire de projet. Cette archive doit être nommée **CTP_Nom_Prenom.zip** avec votre nom et votre prénom.

Un nommage différent donnera lieu à deux points de pénalisation. Cette règle est également valable pour toute proposition de nommage de ce CTP : projet, paquetages, classes, méthodes, attributs, ...

Votre application utilise une base de données qui doit impérativement être nommée **Mariage**. Le mot de passe et le nom d'utilisateur pour se connecter doivent impérativement être **rtp2017** (le mot de passe et le nom d'utilisateur sont identiques).

Un nommage différent de la base de données ou l'utilisation d'identifiants différents donnera lieu à quatre points de pénalisation.

Vérifiez bien que les fichiers sources que vous avez réalisés lors du CTP sont présents dans le dossier que vous déposez sur Moodle (oui, vérifiez bien cela : toutes les années il y a des répertoires vides ou contenant les mauvais fichiers !) Il ne sera pas possible de renvoyer une nouvelle version de votre projet après la date limite de dépôt.

Notation

Cette épreuve de CTP est évaluée sur 24 points (20 points plus 4 point de bonus pour le développement lié à la Section 6).

Pour chaque question, la moitié des points est attribuée si le code fonctionne correctement. **L'autre moitié des points est attribuée en fonction de la qualité du code** (respect des principes de la programmation orientée objet, efficacité des algorithmes) et de la présentation du code (indentation correcte, respects des conventions Java dans les noms de classe, attributs, méthodes et variables, commentaires au format Javadoc quand c'est pertinent).

Consignes

Le sujet est composé de 6 parties liées entre elles. Vous devez les traiter dans l'ordre.

Vous êtes bien sûr autorisés à rajouter les attributs ou méthodes que vous jugez nécessaires même si ce n'est pas dit explicitement dans le sujet.

Attestation sur l'honneur

Je soussigné(e) ..... certifie que le projet déposé sur moodle est l'issue d'un travail purement personnel. Je certifie aussi avoir cité toute source de code externe dans les commentaire de mon code.

Fait à .....

le .....

signature .....

Sujet

Description de l'application

Monsieur Folamour s'occupe de l'organisation de mariages. Parmi ses services, il propose de gérer la disposition des invités dans les différentes tables.

Monsieur Folamour demande aux mariés de diviser les invités en 4 catégories différentes : les *témoins*, la *famille de la mariée*, la *famille du marié* et les *amis*. Une catégorie supplémentaire, appelée *mariés* est dédiée aux jeunes mariés.

Monsieur Folamour demande aux jeunes mariés d'indiquer les personnes qui doivent forcément s'asseoir à la même table (une famille avec les enfants, par exemple) ou qui souhaitent fortement être côte à côte (des amis qui se connaissent depuis longtemps).

Il demande également d'indiquer les personnes qu'il ne faut surtout pas assoir à la même table. Eh oui, Monsieur Folamour sait très bien que chaque mariage a ses deux tantes qui ne peuvent vraiment pas se voir ! Les mettre à la même table pourrait gâcher la journée de rêve de nos mariés !

Une fois toutes ces informations collectées, Monsieur Folamour prépare le plan de la salle avec chaque invité affecté à une table. Il suit les règles suivantes :

- les mariés doivent être à table seulement avec les témoins ;
- pour les autres tables, elles ne doivent contenir que des invités de la même catégorie ;
- il est préférable de mettre ensemble les invités qui le souhaitent ;
- il est préférable de séparer les invités qui ne s'aiment pas (oui c'est vrai, on pourrait éviter d'inviter ces personnes, mais ce n'est pas toujours facile ...).

Remarquez que les deux premières règles doivent absolument être respectées (ce sont des contraintes du problème), alors que les deux autres règles peuvent être violées. Par contre, une disposition des invités qui respecte toutes les règles est de qualité meilleure qu'une autre qui ne le fait pas.

Aujourd'hui Monsieur Folamour construit les dispositions des invités à la main. C'était très sympathique au début, mais maintenant, après des centaines de mariages organisés, il le fait avec de moins en moins d'amour. Il s'adresse donc à vous pour développer une application qui puisse l'aider dans cette tâche.

Modèle conceptuel et modèle logique

Le modèle conceptuel de données ainsi que le modèle logique de données de notre application sont présentés dans les Figures 1 et 2 respectivement.

Le modèle logique de données contient cinq tables : **Mariage**, **Invite**, **Categorie**, **Adore** et **Deteste**.

Un *invite* représente toute personne invitée à un mariage. Même les mariés sont des invités à leur propre mariage ! Un *invite* est défini par un identifiant, son nom, son prénom, son age, sa catégorie et le mariage auquel il est invité.

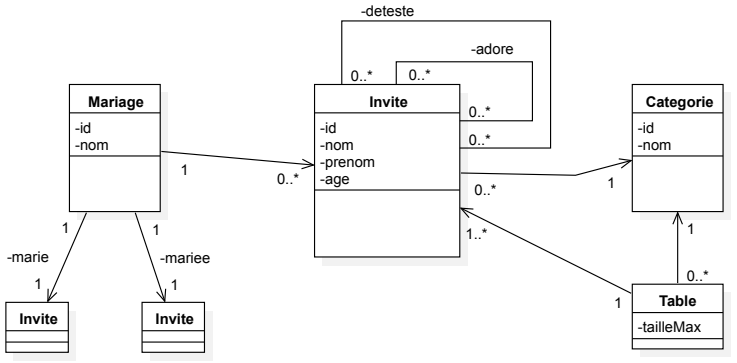

Figure 1 – Modèle conceptuel de données pour l'application de Monsieur Folamour.

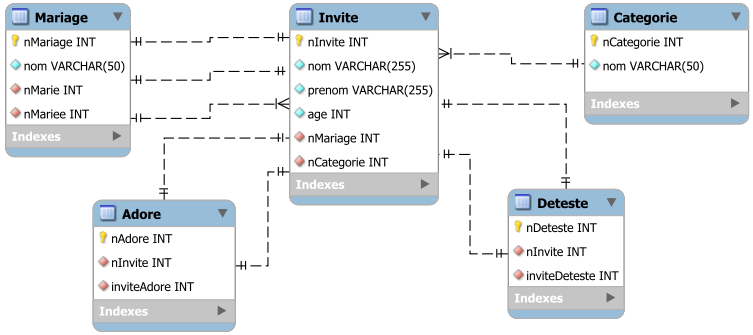

Figure 2 – Modèle logique de données pour l'application de Monsieur Folamour.

Un *mariage* représente très simplement un événement à organiser par Monsieur Folamour, défini par un identifiant, son nom, et par les références sur les jeunes mariés (qui sont des invités).

Une *catégorie* est définie par un identifiant et son nom. Enfin, les tables *Adore* et *Deteste* représentent les associations de cardinalité $n-n$ entre les invités. On suppose que si un invité A adore un autre invité B, alors B adore A, mais une seule association est mémorisée dans la base de données.

De la même manière, on suppose que si un invité A déteste un autre invité B, alors B déteste A, mais une seule association est mémorisée dans la base de données.

Le modèle conceptuel de données contient une classe supplémentaire, nom-

mée **Table**. Une *table* est définie par un nombre maximal de places, un ensemble d'invités et une catégorie. Nous ne nous intéressons pas, dans ce sujet, à la sauvegarde en base de données de la disposition des invités proposée par l'application. En conséquence, le modèle logique de données ne contient pas de table correspondant à la classe **Table**.

Par ailleurs, notez que dans le modèle conceptuel de données, *adore* et *deteste* sont des relations réflexives sur la classe **Invite**, et il n'y a pas de classes associées. Notez également que dans le modèle conceptuel, les relations *adore* et *deteste* peuvent être bidirectionnelles (si A adore B, alors B adore A).

1 Base de données et connexion avec l'application (1 point)

Question 1. En utilisant le script SQL *mariage.sql* disponible sur Moodle, mettez en place la base de données pour l'application de Monsieur Folamour. Le script est prévu pour un SGBD Derby, donc le plus simple est de créer votre base de données sur Netbeans, comme vu dans le TP2. La base de données doit impérativement être nommée **Mariage**. Le mot de passe et le nom d'utilisateur doivent impérativement être **rtp2017**. (Rappel : un nommage non conforme aux règles $\Rightarrow$ 4 points de pénalisation).

Question 2. Créez un nouveau projet nommé **CTP_Nom_Prenom** avec votre nom et votre prénom. (Rappel : un nommage non conforme aux règles $\Rightarrow$ 2 points de pénalisation). Ajoutez un paquetage **metier** et une classe **RequeteMariage**. Dans cette classe, ajoutez un attribut de type **Connection** et faites en sorte que la connexion à la base de donnée soit établie lors de l'appel au constructeur par défaut. Testez le bon établissement de la connexion à la base de données.

Remarque : n'oubliez pas d'importer le pilote JDBC dans les librairies du projet.

2 Mise en place du modèle objet (4 points)

À présent, nous souhaitons ajouter à notre projet, les classes **Categorie**, **Invite** et **Mariage** pour le modèle objet (la création de la classe **Table** sera considérée plus tard dans la suite du sujet). Vous pouvez vous référer à la Figure 1 qui présente le modèle conceptuel de données.

Question 3. Ajoutez à votre projet un paquetage **modele**. Ajoutez dans ce paquetage une classe **Categorie**, avec ses attributs, un constructeur par données, les accesseurs et mutateurs nécessaires, ainsi qu'une redéfinition de la méthode **toString**. Vérifiez le bon fonctionnement de cette classe.

Question 4. Ajoutez dans le paquetage **modele** une classe **Invite**, avec ses attributs, un constructeur par données, les accesseurs et mutateurs nécessaires, ainsi qu'une redéfinition de la méthode **toString**. Remarquez bien qu'à chaque invité, on peut associer un ensemble d'invités *adorés* et un ensemble d'invités *détestés*. Vérifiez le bon fonctionnement de cette classe.

Question 5. Ajoutez dans le paquetage **modele** une classe **Mariage**, avec ses attributs (en particulier on gardera une référence sur le marié et sur la mariée liés au mariage). On utilisera impérativement une **Map** pour mémoriser les invités du mariage. Ajoutez ainsi un constructeur par données, les accesseurs et mutateurs nécessaires, ainsi qu'une redéfinition de la méthode **toString**. Vérifiez le bon fonctionnement de cette classe. N'oubliez pas de redéfinir les méthodes nécessaires à l'utilisation de votre implémentation **Map**.

Question 6. Ajoutez à la classe **Mariage** une méthode **public boolean addInvite(Invite invite)**, qui ajoute un invité à un mariage. Vérifiez que votre code permet bien d'associer un marié et une mariée à un mariage, et que le marié et la mariée font également partie des invités. Ajoutez une méthode principale dans la classe **Mariage** pour tester son bon fonctionnement. Créez des invités et ajoutez-les au mariage. Arrivés à ce point, vous êtes encouragés à appeler les enseignants pour avoir un retour sur votre code.

3 Requêtes sur la base de données (4 points)

Question 7. Dans la classe **RequeteMariage** ajoutez une méthode **public Invite trouverInvite(int idInvite) throws SQLException** qui renvoie l'invité associé à l'identifiant **idInvite**. Testez le bon fonctionnement de cette méthode.

Question 8. Dans la classe **RequeteMariage**, ajoutez une méthode **public List<Mariage> ensMariage() throws SQLException** qui renvoie la liste des mariages contenus dans la base de données. Vous pouvez utiliser la méthode **trouveInvite(int idInvite)** précédemment développée pour récupérer les mariés associés à chaque mariage. Testez le bon fonctionnement de cette méthode.

Question 9. Dans la classe **RequeteMariage** ajoutez une méthode **public void ensInvites(Mariage m) throws SQLException**. Cette méthode récupère tous les invités associés au mariage passé en paramètre et les ajoute à ce dernier à l'aide de la méthode **addInvite(Invite invite)** précédemment développée. Idéalement (pour avoir tous les points), il faudra que la méthode développée effectue une seule requête sur la base de données.

Testez le bon fonctionnement de cette méthode.

4 Affichage de la page d'accueil (3 point)

On souhaite à présent ajouter à notre application une première fenêtre qui sera l'accueil de notre application. Sur cette fenêtre, on placera :

- une **JList** qui contient les noms des mariages ;
- une **JTextArea** pour afficher des informations liées aux mariages ;
- un **JButton** qui permettra par la suite d'accéder à la disposition des tables.

Un exemple de cette fenêtre est donné dans la Figure 3.

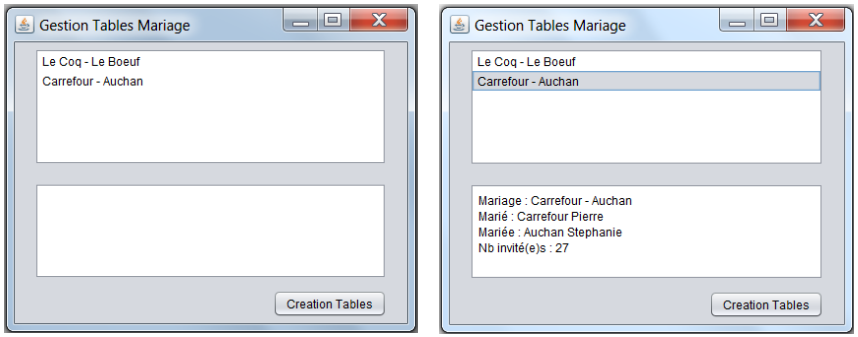

Figure 3 – Exemple d'affichage de la page d'accueil.

Question 10. Ajoutez à votre projet un paquetage **vuecontrole**. Ajoutez dans ce paquetage une fenêtre **Accueil** qui sera similaire à celle présentée dans la Figure 3. À l'ouverture de cette fenêtre, les noms des mariages à organiser seront affichés dans la liste. Ajoutez la logique de contrôle de telle sorte que lorsque l'on clique sur le nom d'un mariage, les informations suivantes soient affichées dans la zone de texte :

- le nom du mariage précédé par "Mariage :" ;
- le nom et le prénom du marié précédé par "Marié :" ;
- le nom et le prénom de la mariée précédé par "Mariée :" ;
- le nombre d'invités précédé par "Nb invité(e)s :".

Vérifiez le bon fonctionnement de cette fenêtre.

5 Gestion des tables (8 points)

On souhaite à présent mettre en place une fenêtre qui permettra d'afficher les tables à préparer pour chaque mariage. Les invités doivent bien être affectés d'une façon unique aux tables. Les règles présentées dans l'introduction doivent être respectées au maximum.

Cette fenêtre sera ouverte lors de l'appui sur le bouton "*Creation Tables*" dans la fenêtre d'accueil pour le mariage sélectionné dans la liste. Sur cette fenêtre, on placera :

- un **JTree** : la racine de l'arbre contient le nom du mariage. Puis, on considère deux niveaux de nœud. Le premier niveau contient un nœud pour chaque table. Au niveau graphique, on affichera la catégorie des invités assis à cette table. En cliquant sur le nœud correspondant à une table, nous allons pouvoir voir les invités assis à cette table (le second niveau de nœud).

Un exemple de cette fenêtre est donné dans la Figure 4.

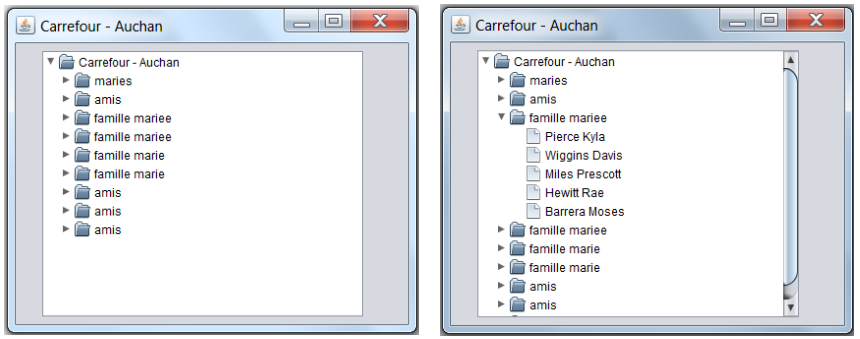

Figure 4 – Exemple d'affichage de la fenêtre de gestions de tables.

Question 11. Dans le paquetage **vuecontrole**, ajoutez une fenêtre nommée **Disposition**, et ajoutez-y les composants graphiques de telle sorte que cette fenêtre ressemble à celle de la Figure 4.

Question 12. Dans la classe **RequeteMariage**, ajoutez la méthode **public void remplirAdore(Mariage mariage) throws SQLException**. Une manière d'implémenter cette méthode est d'utiliser une requête pour sélectionner tous les enregistrements de la table *Adore* (les couples d'identifiants *nInvite*, *nInivteAdore*) tels que l'invité soit bien associé au mariage passé en paramètre. Ensuite, vous pouvez ajouter une méthode qui prend en paramètre deux identifiants d'invités dans la classe **Mariage**. Cette méthode met à jour les invités adorés pour les deux invités dont l'identifiant est passé en paramètre (utilisez astucieusement la **Map** des invités). Cette méthode est appelée lors de la création de la fenêtre **Disposition**.

Question 13. Dans la classe **RequeteMariage**, ajoutez la méthode **public void remplirDeteste(Mariage mariage) throws SQLException**. Elle est similaire à la méthode développée à la question précédente ; mais on récupère

cette fois-ci les invités détestés. Cette méthode est appelée lors de la création de la fenêtre.

Question 14. Ajoutez à votre projet, dans le paquetage **modele**, une classe **Table** avec ses attributs, un constructeur par données, les accesseurs et mutateurs nécessaires. On suppose que chaque table contient 8 places.

Question 15. Ajoutez à votre classe **Mariage** une méthode **public void createDisposition()**. Cette méthode compose les tables pour le mariage. Vous pouvez d'abord réserver une table aux mariés et aux témoins. Puis, pour chaque invité dont l'ensemble des invités adorés est non vide, vous pouvez lui réserver une table pour lui et ses adorés. Vous pouvez ensuite compléter la disposition des invités avec les autres invités en respectant les invités détestés.

Remarquez que vous pouvez ajouter les attributs et les méthodes que vous jugez nécessaires aux autres classes concernées par cette méthode (par exemple une méthode **public boolean addInvite(Invite invite)** dans la classe **Table** et une méthode **private boolean check(Invite invite)** pour vérifier si l'ajout de l'invité à la table est possible).

Question 16. Nous nous intéressons à présent à la visualisation des tables précédemment crées. Ajoutez dans la classe **Disposition** la méthode **private void visualiserTables()**. Cette méthode demande à la classe **Mariage** d'effectuer la disposition des invités, puis elle remplit l'objet **JTree** avec trois niveaux de nœuds :

- le nœud racine affiche le nom du mariage ;
- le premier niveau contient un nœud pour chaque table ; ces nœuds affichent le nom de la catégorie des invités assis (cette catégorie doit être la même pour tous les invités de la table, sauf les mariés et les témoins qui sont à la même table) ;
- le second niveau contient un nœud par invité ; ces nœuds contiennent le nom et le prénom des invités assis à la table correspondant au nœud père.

6 Bonus : Évaluation de la disposition proposée (4 points)

Pour terminer le développement de l'application demandée par Monsieur Folamour, nous allons proposer une analyse de la disposition calculée. Les informations suivantes doivent apparaître :

- le nombre de tables utilisées ;
- le nombre de tables pour chaque catégorie ;
- le nombre d'invités détestés assis à la même table ;
- le nombre d'invités adorés assis dans des tables différentes.

Vous pouvez afficher ce message par le moyen que vous jugez le plus intéressant (une **JTextArea** en dessus du **JTree**, une boite de dialogue, l'ajout d'une fenêtre avec ces informations, ouverte lors d'un clic sur un bouton, ...).

Question 17. Mettez en place une solution convenable pour que Monsieur Folamour puisse visualiser les information qu'il souhaite avoir.