Polytech Nice Sophia – SI4 P. Collet
Décembre 2017

Contrôle de conception logicielle

Durée : 3h.
Seuls documents autorisés : support de cours et notes personnelles

Toute ambiguïté que vous pourriez rencontrer dans ce sujet devra être résolue en décrivant brièvement le choix que vous avez fait.

Il y a deux études de cas dans le sujet, une centrée sur UML, l'autre centrée uniquement sur les classes et les patrons de conception. Prenez bien du temps pour aborder les deux études, il y a des points à prendre assez facilement des deux côtés.

1. Petites questions (4 pts)

a) Lorsqu'un cas d'utilisation est relié à deux acteurs, cela signifie-t-il que la présence d'un des deux acteurs ou des deux acteurs est nécessaire pour réaliser le cas ? (1 pt)

b) Dans un diagramme de séquence, un objet peut-il envoyer un message à lui-même ? Si oui, comme cela est exprimé ? (1 pt)

c) Expliquez rapidement pourquoi et comment les patrons *AbtractFactory* et *Singleton* sont souvent utilisés ensemble. (1 pt)

d) Expliquez (donc ne donnez pas juste le nom du résultat, expliquez les fonctionnalités obtenues) ce qu'on peut obtenir lorsqu'on couple les patrons *Command* et *Composite*. (1 pt)

2. Conception UML – étude de cas (9 pts)

Un organisme gérant les logements pour les étudiants souhaite mettre en place une application Web pour gérer les visites. Cette application doit gérer des fiches descriptives des logements accessibles en libre service par les étudiants. Les logements devront être classés par type (chambre, studio, T1, T1bis, T2, etc.) et des informations descriptives des logements devront être conservées (quartier, superficie, équipements, etc.). Le prix du loyer, celui des charges lorsqu'elles ne sont pas comprises dans le loyer, et le nombre de mois de caution doivent aussi être précisés. Il faudra proposer plusieurs modes de recherche de logements libres aux étudiants : par quartier, par prix, etc.

L'étudiant intéressé par un logement doit remplir une demande de visite en ligne, qui est ensuite validée par l'intermédiaire d'une secrétaire de l'organisme via l'application à développer. Pour avoir accès à ce service, l'étudiant doit s'acquitter d'une somme forfaitaire de 10 euros. On conservera alors le numéro de l'étudiant, son nom, son prénom, son adresse et son numéro de téléphone. Cette somme n'est à verser qu'une seule fois et permet à l'étudiant de faire autant de demandes de visite qu'il le souhaite pendant toute l'année scolaire.

Une demande de visite porte sur un seul logement. Il peut y avoir plusieurs demandes en attente en même temps sur le même logement. Un étudiant ne peut demander à visiter plus de deux logements à la fois. Sur la demande de visite visible par l'étudiant apparaissent les informations sur le logement et sur le propriétaire afin que l'étudiant puisse le contacter et organiser une visite (les propriétaires n'ont

pas de compte, pas d'accès à l'application, ce sont des opérateurs de l'entreprise de location qui les contacte). Une fois la visite faite, l'étudiant peut compléter la demande en indiquant s'il souhaite ou non louer le logement. S'il répond par la positive, le logement est bloqué pour lui, un opérateur doit contacter le propriétaire pour obtenir son aval et ainsi réaliser un contrat de location (la première version de l'application ne permet pas l'édition du contrat de location, elle marque juste que le logement est loué jusqu'à une certain date). Si le propriétaire refuse, l'opérateur saisit la décision, l'étudiant est notifié et l'application passe au suivant (soit au suivant qui a visité et annoncé qu'il voulait ce logement, soit au suivant qui a demandé une visite).
Si aucune décision d'étudiant ne parvient dans un délai de deux semaines après la date de demande de visite initiale, la décision est considérée comme négative, et la demande n'est plus considérée comme en attente.

a) Donnez le(s) diagramme(s) de cas d'utilisation de l'application à développer. *La description détaillée de type Cockburn de chaque cas d'utilisation n'est **pas** demandée*. Attention donc à bien choisir un nom **explicite** pour chaque cas d'utilisation et/ou à donner quelques lignes d'explication afin de montrer que l'ensemble du sujet est couvert. (2,5 pts)

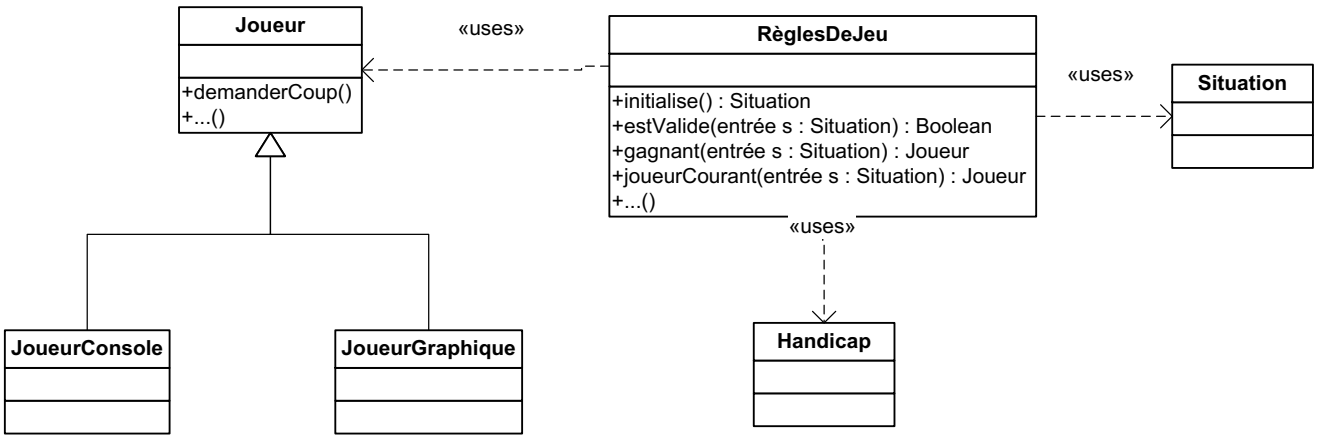
b) Donnez le diagramme de classe de cette application. Pour chaque classe, donnez une description détaillée, avec les attributs et leurs types, les opérations *mais sans forcément leur signature typée complète*. (3 pts)

c) Donnez le diagramme de séquence correspondant à la demande d'une visite d'un logement par un étudiant, dans le cas où celui-ci n'a aucune demande de visite en cours. (1,5 pts)

d) Ecrivez un scénario *Gherkin* (format d'entrée de *Cucumber*) couvrant le diagramme de séquences précédent (définissez un *Background* si nécessaire). Expliquez ensuite, en français, quelles actions vous effectueriez sur quels objets pour chaque phrase du background et du scénario Gherkin afin de l'implémenter (2 pts)

3. Conception objet et patrons de conception – étude de cas (7 pts)

On veut modéliser de manière assez générique des applications de jeu qui fonctionne par tours (chaque joueur joue à tour de rôle dans un ordre prédéfini). On souhaite permettre à différents types de joueurs d'effectuer une partie. Les types de joueurs dépendent du médium qu'ils utilisent : console texte, écran graphique, etc.
Bien sûr, on veut que les classes permettant de gérer le jeu et celles permettant de gérer les joueurs dépendent le moins possible les unes des autres. Le diagramme de classes ci-dessous donne une vue partielle de la modélisation de départ (les liens uses indiquent juste qu'une classe en utilisent une autre).



La classe RèglesDeJeu fournit des primitives pour initialiser une partie (initialise), pour vérifier si une situation est valide selon les règles, s'il y a un gagnant pour une situation donnée, etc. La classe Situation modélise l'état du jeu à un moment donné, comme par exemple l'état d'un plateau avec le contenu des cases.
Le déroulement d'un jeu consiste donc au minimum à initialiser une situation à partir des règles, à déterminer quel est le joueur courant, à lui demander un coup à jouer (par la méthode demanderCoup()), à vérifier que la situation résultante est valide et à passer au suivant jusqu'à ce que le jeu soit terminé.
Le handicap qu'on peut donner à un joueur en cours de partie est modélisé par une interface quelconque. Ce handicap est variable selon les règles de jeu et les joueurs : saut d'un tour, réduction de points, etc.
On ne se focalise ici que sur les joueurs et les règles du jeu. Vous n'avez, par exemple, pas besoin de connaître comme sont modélisés les coups joués, ni comment on évalue qu'un joueur a gagné, ni si les joueurs marquent des points ou pas.

Pour répondre aux questions, vous pouvez effectuer un seul diagramme de classes complet, puis expliquer dans chaque réponse vos choix de conception.

a) Indiquez brièvement quels sont les éléments **à l'intérieur des classes** de la conception de départ qui ne vous semblent pas vraiment orientés objets et proposez une correction. (0,5 pt)

b) Définissez l'association, avec ses cardinalités, entre Joueur et RèglesDeJeu. (0,5 pt)

c) On souhaite intégrer un nouveau type de joueur, JoueurAutomatique, pour simuler différents joueurs par l'ordinateur. Intégrez la classe JoueurAutomatique dans le diagramme de classes. (1 pt)

d) Quelle serait la solution (en appliquant un ou des patrons de conception) pour créer des joueurs automatiques dont on verrait les actions se dérouler sur une console de texte ? Etendez le diagramme et justifiez aussi vos choix de conception. (1,5 pt)

e) Quel patron utiliser pour attribuer un handicap à un joueur en cours de partie ? Donnez les modifications sur le diagramme de classes ou les classes Java. (1,5 pt)

f) Quels patrons utiliser pour représenter différentes variantes d'implémentation de l'intelligence d'un joueur automatique, sachant aussi que plusieurs implémentations sont déportées sur de gros serveurs de calcul distants. Donnez les modifications sur le diagramme de classes. (2 pts)