Examen

Nom : **Prénom :**

- Il est conseillé de lire complètement le sujet avant de commencer à y répondre !
- Penser à mettre son nom sur le sujet et à le rendre avec la copie puisque certaines réponses peuvent être données directement sur le sujet.
- Barème indicatif :

Exercice	1	2	3	4	5	6	7	8
Points	4	2	2	3	4	2	2	1

1 Interfaces graphiques avec Swing

Exercice 1 Répondre de manière concise aux questions suivantes.

1.1 L'API Swing de Java concernant les interfaces graphiques définit les éléments suivants : ActionListener, ActionEvent, Event, addActionListener et actionPerformed.

1.1.1 Indiquer à quoi correspond chacun de ces éléments.

1.1.2 Dessiner un diagramme de classes UML faisant apparaître ces éléments et leurs relations.

1.2 On considère une application Swing dont l'apparence est donnée à la figure 1. Elle permet de saisir un code qui s'affiche dans la zone de saisie (partie haute de la fenêtre) au fur et à mesure que l'utilisateur clique sur l'un des boutons correspondant à un chiffre.

Expliquer comment construire la vue de cette application en Swing.

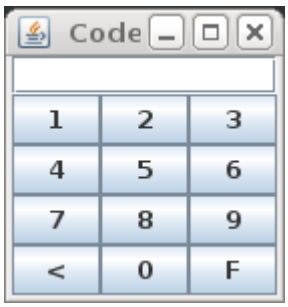

FIGURE 1 – Apparence de l'application

2 Agendas hiérarchiques

Nous souhaitons modéliser des agendas simplifiés (exercices 2, 3 et 4) et des groupes hiérarchiques (exercices 5 et 6). L'exercice 7 demande de dessiner le diagramme de classe du système. L'exercice 8 propose de généraliser les agendas.

Exercice 2 : Agenda
Pour simplifier, on considère un agenda pour une seule journée découpée en 24 créneaux d'une heure. Les créneaux sont repérés par un numéro de 0 à 23 correspondant à l'heure du rendez-vous. Par exemple, 10 correspond au créneau de 10h à 11h.

Un agenda fournit quatre opérations :

- enregistrer pour un créneau une information qui sera ici limitée à une chaîne de caractères (`String`). Ainsi on peut enregistrer le rendez-vous "Examen" pour le créneau 14. Si le créneau est déjà occupé, une exception `OccupeException` est levée,
- annuler le rendez-vous prévu pour un créneau donné. Par exemple, on peut supprimer le rendez-vous du créneau 14. Si le créneau ne contient pas de rendez-vous, l'agenda n'est pas modifié.
- obtenir le rendez-vous correspondant à un créneau. Si ce créneau ne contient pas de rendez-vous enregistré, une erreur est signalée grâce à l'exception `LibreException`,
- obtenir le nom d'un agenda.

2.1 Écrire la classe `OccupeException` sachant qu'elle est contrôlée par le compilateur. Indiquer ce qui fait la différence entre une exception contrôlée ou non contrôlée et quel est l'intérêt d'une exception contrôlée.

La classe `LibreException` est aussi contrôlée. On ne demande pas d'écrire la classe Java correspondante.

2.2 Écrire en Java une interface Agenda.

Expliquer pourquoi, il est préférable de définir une interface plutôt qu'une classe.

Exercice 3 : AgendaAbstrait
On définit maintenant une classe abstraite qui réalise l'interface `Agenda` et possède un attribut pour stocker le nom de l'agenda.

3.1 Indiquer s'il est possible de définir un constructeur sur une classe abstraite.

3.2 Écrire la classe `AgendaAbstrait`.

3.3 Expliquer l'intérêt d'avoir à la fois l'interface `Agenda` et la classe abstraite `AgendaAbstrait`.

Exercice 4 : AgendaIndividuel
On s'intéresse maintenant à une réalisation de l'agenda avec la classe `AgendaIndividuel`.

4.1 Pour stocker les rendez-vous, on peut utiliser les tableaux de Java ou la collection `List` des API Java. Indiquer les avantages et inconvénients de ces deux solutions.

4.2 Écrire la classe `AgendaIndividuel`.

Exercice 5 : GroupeAgenda
Un groupe d'agendas est utilisé pour manipuler plusieurs agendas, qu'ils soient des agendas

individuels ou des groupes d'agendas. Ceci permet en particulier de positionner une rendez-vous sur tous les agendas d'un groupe ou d'obtenir le rendez-vous commun à l'ensemble des agendas du groupe.

Quand on enregistre un rendez-vous sur un groupe, il est enregistré dans chaque agenda du groupe libre pour la date consiérée.

Quand on récupère un rendez-vous, on obtient l'exception `LibreException` si tous les agendas sont libres pour la date considérée, un rendez-vous si tous les agendas occupés le sont pour le même rendez-vous, **`null`** dans les autres cas.

5.1 Dessiner un diagramme de séquence qui correspond au scénario suivant :
- On crée un groupe d'agendas appelé g1 ;
- On crée et on ajoute dans le groupe g1 trois agendas a1, a2 et a3 ;
- On enregistre le rendez-vous "Examen" à 14h dans l'agenda a2 ;
- On enregistre le rendez-vous "Pause" à 14 h dans l'agenda g1 ;
- On vérifie qu'à 14h, le rendez-vous de a1 et a3 est "Pause", celui de a2 est "Examen" et celui de g1 est **`null`**.

5.2 Expliquer quel outil pourrait être utilisé pour écrire un programme de test correspondant au scénario précédent.

5.3 Écrire la classe `GroupeAgenda` sachant que l'on doit utiliser l'interface `java.util.List` pour stocker les agendas contenus dans le groupe.

Exercice 6 : GroupeAgendaTransactionnel

On veut définir une classe `GroupeAgendaTransactionnel`. Le principe est que soit tous les agendas d'un groupe ont le nouveau rendez-vous enregistré, soit aucun. Dans le cas où le rendez-vous n'a pas pu être enregistré, l'exception `OccupeException` est bien sûr levée.

6.1 Écrire la classe `GroupeAgendaTransactionnel`.

Exercice 7 : Diagramme de classe

Dessiner le diagramme de classe du système faisant apparaître toutes les classes et interfaces de cette partie. On ne fera pas apparaître les constructeurs, attributs et opérations.

Exercice 8 : Généralisation

Dans la solution proposée, un rendez-vous est réduit à une chaîne de caractères. Ceci est suffisant pour ce que l'on voulait faire. Dans un autre cas, on pourrait vouloir indiquer en plus un numéro de salle pour le rendez-vous. On pourrait aussi dans un autre contexte vouloir préciser du matériel à utiliser (vidéo-projecteur, etc.), ou un ordre du jour, etc.

Indiquer comment il serait possible de définir des agendas qui puissent simplement être adaptés à ces types de changement.