Examen (1h45, avec documents)

— En cas de doute sur le sujet, notez vos choix sur la copie. Aucune réponse ne sera donnée par les surveillants.
— Il est conseillé de lire complètement le sujet avant de commencer à y répondre !
— Ne pas mettre les commentaires de documentation (sauf si nécessaires à la compréhension).
— Barème indicatif (sur 30 points) :

Exercice	1	2	3	4	5
Points	4	5	5	6	10

Exercice 1 Répondre de manière concise et précise aux questions suivantes.

1.1 Indiquer à quoi correspondent les deux mots-clés **`throw`** et **`throws`**.

1.2 Comment savoir si un objet, par exemple accessible par la poignée `o`, est une exception ?

1.3 On considère la signature suivante d'une méthode générique :

```
<T> void m(Set<? super T> l1, Set<T> l2, Set<? extends T> l3);
```

Expliquer ce que signifient les mots-clés **`extends`** et **`super`** et donner un exemple d'appel de cette méthode qui illustre l'intérêt de les avoir utilisés ici.

Exercice 2 On considère les classes du listing 1. Les classes A et B compilent sans erreurs.

Listing 1 – Les classes A, B et Main

```
1  class A {
2     void m()    { System.out.println("A.m()"); }
3     void m(int n) { System.out.println("A.m(int)"); }
4  }
5
6  class B extends A {
7     void m(String s) { System.out.println("B.m(String)"); }
8     void m(int n)  { System.out.println("B.m(int)"); }
9  }
10
11 class Main {
12    public static void main(String[] args) {
13       A p = new B();
14       p.m();
15       p.m(10);
16       p.m("dix");
17 }  }
```

2.1 Pour chacune des quatre instructions de la méthode principale, indiquer si l'instruction est acceptée par le compilateur et dans l'affirmative ce qu'elle affichera à l'exécution.

2.2 Comment évoluent les réponses de la question précédente si l'on remplace successivement la déclaration et l'initialisation de p par les trois suivantes :

```
1     B p = new B();
2     A p = new A();
3     B p = new A();
```

Exercice 3 : IntegerField

On veut définir un composant graphique `IntegerField` pour saisir un texte qui doit correspondre à un entier. Quand la souris sort de ce composant, la couleur de fond du champ sera mise au rouge (`Color.RED`) si le texte qu'il contient ne correspond pas à un entier. Quand la souris rentre sur le composant, on met la couleur de fond à blanc (`Color.WHITE`) de manière à ne pas gêner l'utilisateur dans sa saisie.

3.1 Pour écrire la classe `IntegerField`, on s'appuiera sur le composant `JTextField` de l'API Swing qui définit la méthode `getText()` pour récupérer le texte saisi, la méthode `setBackground(Color)` pour changer la couleur de fond du `JTextField`. On note qu'un `MouseListener` peut être associé à un `JTextField`. L'interface `MouseListener` définit 5 méthodes :

```
1 public void mouseClicked(MouseEvent e)
2 public void mouseEntered(MouseEvent e)
3 public void mouseExited(MouseEvent e)
4 public void mousePressed(MouseEvent e)
5 public void mouseReleased(MouseEvent e)
```

La méthode `mouseEntered` est appelée quand la souris entre dans le composant et `mouseExited` quand la souris en sort. L'API Swing définit également la classe `MouseAdapter`, réalisation de l'interface `MouseListener`, qui définit chacune de ces 5 méthodes avec un code vide.

Pour savoir si un texte correspond à un entier, on utilisera la méthode de classe `parseInt(String)` de la classe `Integer` qui retourne l'entier correspondant à la chaîne fournie en paramètre ou lève l'exception `NumberFormatException` si cette chaîne ne correspond pas à un entier.

Écrire la classe `IntegerField`.

3.2 On veut améliorer la classe `IntegerField` de manière à pouvoir vérifier des propriétés plus précises sur le texte saisie. Voici quelques exemples de propriétés possibles : 1) l'entier est positif, 2) l'entier est compris entre 1 et 12 ou plus généralement entre deux bornes, 3) c'est un nombre pair, 4) c'est un entier compris dans un ensemble d'entiers autorisés, etc.

Expliquer comment modifier la classe `IntegerField` pour intégrer cette amélioration.

1 Piloter un robot

L'objectif de ces exercices est de permettre d'écrire des petits programmes (sous forme de tâches) qui commandent un robot.

Exercice 4 : Les robots

On considère qu'un robot évolue dans un espace plat que l'on peut modéliser par un quadrillage. Un robot est ainsi repéré par son abscisse (axe des X), son ordonnée (axe des Y) et sa direction (nord, est, sud ou ouest). Abscisse et ordonnée sont des entiers. La direction est du type `Direction`, type énuméré qui définit quatre valeurs (quatre objets) : `NORD`, `EST`, `SUD` et `OUEST`.

```
public enum Direction { NORD, EST, SUD, OUEST }
```

Un robot peut avancer d'une case suivant sa direction et pivoter de 90^o vers la droite. On peut le repositionner en précisant son abscisse, son ordonnée et sa direction.

4.1 Donner le diagramme d'analyse (requêtes/commandes) de la classe `Robot`. On ne donnera ni la documentation, ni les contrats.

4.2 Quand on s'intéresse à la méthode avancer, on constate qu'il suffit d'ajouter 0, 1 ou -1 à l'abscisse et l'ordonnée en fonction de la direction du robot.

Pour écrire plus simplement la méthode avancer, on se propose de s'appuyer sur deux variables, `dx` et `dy`, abréviations de déplacements suivant l'axe des X et déplacements suivant l'axe des Y. Elles indiquent l'entier à ajouter à x et y en fonction de la direction du robot. Par exemple, si la direction est `NORD`, le déplacement est de 0 pour l'axe des X (`dx`) et de 1 pour l'axe des Y (`dy`). Si la direction est `OUEST`, `dx` donne -1 et `dy` donne 0.

4.2.1 Indiquer le type de `dx` et `dy`.

4.2.2 Écrire la méthode avancer en s'appuyant sur `dx` et `dy`.

4.3 Donner la déclaration de tous les attributs de la classe `Robot` et leur initialisation sachant que l'on souhaite deux constructeurs, le premier qui prend en paramètre l'abscisse, l'ordonnée et la direction du robot et le second qui ne prend pas de paramètre et positionne le robot à l'origine avec une direction NORD.

Exercice 5 : Les tâches

Le listing 2 constitue un exemple de l'utilisation des tâches. Le résultat de son exécution est donné listing 3). Notons que la tâche `Pour` permet de faire plusieurs fois une tâche donnée.

5.1 Donner un diagramme de classe (attributs et opérations) correspondant à l'architecture de ce système ainsi qu'un diagramme de séquence qui décrit ce qu'il se passe lors de l'exécution de la première partie du programme du listing 2 (« Faire un côté », lignes 3 à 19).

5.2 Écrire le code de `Tache`, `Pour`, `TacheComplexe` et `RobotAvancer`. On ne donnera pas les commentaires sauf s'ils sont nécessaires à la compréhension.

5.3 Indiquer les patrons de conception utilisés dans cette architecture.

Listing 2 – La classe `ExempleTacheRobot`

```
public class ExempleTacheRobot {
   public static void main(String[] args) {
      // Faire un côté
      Robot r = new Robot();
      RobotAvancer ta = new RobotAvancer(r); // tâche avancer
      RobotPivoter tp = new RobotPivoter(r); // tâche pivoter
      RobotAfficher td = new RobotAfficher(r); // tâche debug (afficher)

      final int largeur = 3;
      TacheComplexe cote = new TacheComplexe();
      cote.ajouter(new Pour(ta, largeur));
      cote.ajouter(td);
      cote.ajouter(tp);
      cote.ajouter(td);

      System.out.println("Exécution de côté :");
      cote.faire();
      System.out.println("\nDescription de la tâche 'côté' : "
            + cote.description());

      // Faire un carré
      r.positionner(0, 0, Direction.NORD);
      Tache carre = new Pour(cote, 4);

      System.out.println("\nExécution de carré :");
      carre.faire();
      System.out.println("\nDescription de la tâche 'carré' :"
            + carre.description());
}  }
```

Listing 3 – Résultat de l'exécution de la classe `ExempleTacheRobot` (listing 2)

```
Exécution de côté :
(0, 3):NORD
(0, 3):EST

Description de la tâche 'côté' :
Faire 3 fois :
   Avancer robot
Afficher robot
Pivoter robot
Afficher robot

Exécution de carré :
(0, 3):NORD
(0, 3):EST
(3, 3):EST
(3, 3):SUD
(3, 0):SUD
(3, 0):OUEST
(0, 0):OUEST
(0, 0):NORD

Description de la tâche 'carré' :
Faire 4 fois :
   Faire 3 fois :
      Avancer robot
   Afficher robot
   Pivoter robot
   Afficher robot
```