Examen (2 heures 30)

Préambule : Lire attentivement les points suivants.

1. La durée de l'épreuve est de 2h30 (3h20 pour les 1/3 temps).
2. Les documents ne sont pas autorisés. Seule une feuille A4 est autorisée qui devra être rendue avec la copie à la fin de l'épreuve. La feuille A4 peut être une photocopie.
3. Répondre de manière concise et précise aux questions. Ne pas mettre de commentaires de documentation sauf s'ils sont nécessaires à la compréhension.
4. Il est conseillé de répondre directement sur le sujet quand c'est possible. Sinon, il est conseillé de mettre une marque sur le sujet (par exemple le numéro de l'exercice suivi d'une lettre majuscule : 1A, 1B, 2A, etc) et d'indiquer sur la copie la marque avec le texte (ou le code) associé.
5. Rendre le sujet avec la copie (mettre son nom sur la première feuille du sujet).

Barème indicatif :

exercice	1	2	3	4	5	6
points	4	4	3	4	2	3

Exercice 1 : Comprehension du cours

Répondre de manière concise aux questions suivantes.

1.1. Indiquer ce que nous apprend la trace d'exécution suivante.

```
1 [2]
2 Exception in thread "main" java.lang.ClassCastException: Attempt to insert class
      java.lang.Double element into collection with element type class java.lang.
      Integer
3         at java.util.Collections$CheckedCollection.typeCheck(Collections.java
              :2276)
4         at java.util.Collections$CheckedCollection.add(Collections.java:2319)
5         at Main.main(Main.java:10)
```

1.2. On distingue généralement deux types d'égalité.

1.2.1. Expliquer la différence entre égalité logique et égalité physique.

1.2.2. Indiquer comment s'exprime l'égalité logique en Java

1.2.3. Indiquer comment s'exprime l'égalité physique en Java

1.3. Considérons la déclaration du listing suivant. Nous supposons qu'elle apparaît dans la classe `Point` vue en cours, TD et TP.

```
1 public class Point {
2     public static final Point origine = new Point(0, 0);
3     ...
4 }
```

1.3.1. Expliquer ce que signifient les mots-clés **`public`**, **`static`** et **`final`**.
1.3.2. Indiquer, en justifiant la réponse, si on peut être sûr que l'attribut `origine` de la classe `Point` aura toujours pour coordonnées $(0, 0)$.
1.4. On considère une interface I qui spécifie la méthode m() et une poignée p non nulle déclarée de type I, expliquer pourquoi on est sûr que p.m() a un sens, c'est-à-dire qu'il y a un code défini pour m(). Indiquer quel est le code qui sera exécuté.

Exercice 2 : Interfaces graphiques avec Swing
2.1. L'API Swing de Java concernant les interfaces graphiques définit les éléments suivants : ActionListener, ActionEvent, Event, addActionListener et actionPerformed.
2.1.1. Indiquer à quoi correspond chacun de ces éléments.
2.1.2. Dessiner un diagramme de classes UML faisant apparaître ces éléments et leurs relations.
2.2. 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 et donner les principaux éléments du code Java correspondant.

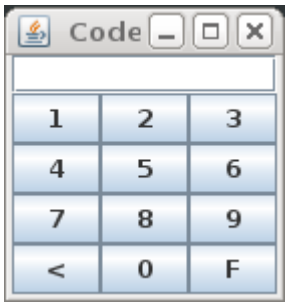

FIGURE 1 – Apparence de l'application

2.3. Rendre actifs les boutons. Les chiffres se rajoutent dans la zone de saisie, à la fin. « `<` » permet de supprimer le dernier caractère saisi. « `F` » quitte la fenêtre.
Indication : On utilisera un `JTextField` pour représenter la zone de saisie. Cette classe propose les méthodes `setText(String)` et `String getText()` pour manipuler le texte affiché.

La classe `String` fournit une `méthodesubstring(int début,int fin)` qui permet de retourner la sous-chaîne qui commence à l'indice début inclus et se termine à l'indice `fin` exclu.

Exercice 3 : Introspection
Écrire une classe principale Main qui affiche toutes les méthodes publiques à deux paramètres de la classe dont le nom est donné sur la ligne de commande.

Par exemple, le résultat de `java Main java.lang.Math` est donné au listing 1.

Exercice 4 : Annuaire
L'objectif de cet exercice est de définir un annuaire simplifié qui permet de conserver des numéros de téléphone. L'interface annuaire (listing 2) spécifie les opérations de l'annuaire de manière

Listing 1 – Résultat de `java Main java.lang.Math`

```
public static double java.lang.Math.atan2(double,double)
public static double java.lang.Math.pow(double,double)
public static int java.lang.Math.min(int,int)
public static long java.lang.Math.min(long,long)
public static float java.lang.Math.min(float,float)
public static double java.lang.Math.min(double,double)
public static int java.lang.Math.max(int,int)
public static long java.lang.Math.max(long,long)
public static float java.lang.Math.max(float,float)
public static double java.lang.Math.max(double,double)
public static double java.lang.Math.scalb(double,int)
public static float java.lang.Math.scalb(float,int)
public static double java.lang.Math.IEEEremainder(double,double)
public static double java.lang.Math.copySign(double,double)
public static float java.lang.Math.copySign(float,float)
public static double java.lang.Math.hypot(double,double)
public static double java.lang.Math.nextAfter(double,double)
public static float java.lang.Math.nextAfter(float,double)
public final void java.lang.Object.wait(long,int) throws java.lang.InterruptedException
```

Listing 2 – L'interface Annuaire

```
1 public interface Annuaire {
2
3     /** Enregistrer dans l'annuaire un nom avec son tél... */
4     void ajouter(String nom, String tel);
5
6     /** Obtenir le téléphone d'un contact à partir de son nom... */
7     String getTel(String nom);
8 }
```

Listing 3 – La classe de test AnnuaireTest

```
1  import org.junit.*;
2  import static org.junit.Assert.*;
3
4  abstract public class AnnuaireTest {
5      protected Annuaire annuaire;
6
7      @Before public void setup() {
8          this.annuaire = newAnnuaire();
9          this.annuaire.ajouter("XC", "2186");
10         this.annuaire.ajouter("MP", "2185");
11     }
12
13     /** Retourner un annuaire vide. */
14     abstract protected Annuaire newAnnuaire();
15
16     @Test public void testerGetTel() {
17         assertEquals("2185", this.annuaire.getTel("MP"));
18         assertEquals("2186", this.annuaire.getTel("XC"));
19     }
20
21     @Test(expected=ContactInconnuException.class)
22     public void testerGetTelAbsent() {
23         this.annuaire.getTel("MO");
24     }
25
26     @Test public void testerAjouter() {
27         this.annuaire.ajouter("XC", "06...");
28         assertEquals("06...", annuaire.getTel("XC"));
29 }   }
```

minimaliste. Les commentaires de documentation sont incomplets mais un programme de test JUnit est donné (listing 3) pour en préciser le fonctionnement.
4.1. Expliquer ce qu'il faudrait ajouter à l'interface Annuaire pour qu'elle constitue réellement une spécification.
4.2. Indiquer ce qui justifie que la classe de test AnnuaireTest soit abstraite.
4.3. Expliquer ce que fait la méthode `getTel()` si le nom fourni ne permet pas de trouver de numéro de téléphone.
4.4. Expliquer l'information qu'il y a dans l'annuaire si on ajoute deux numéros de téléphone différents pour un même nom.
4.5. Écrire la classe `ContactInconnuException`.
4.6. Écrire une réalisation de l'interface `Annuaire` appelée `AnnuaireConcret` en veillant à ce que :

1. l'opération `getTel(String nom)` soit efficace. Elle retourne le téléphone d'une personne à partir du nom de la personne.
2. la méthode toString() affichera l'annuaire dans l'ordre alphabétique des noms. Exemple :

```
MP  2185
XC  2186
```

On choisira avec soin les structures de données à utiliser !
4.7. `AnnuaireConcret` doit réussir les tests de `Annuaire`. Écrire la classe de test qui le fait.

Exercice 5 : XML
On considère la DTD du listing 4.
5.1. Expliquer la signification de `+`, `ID`, `EMPTY` et `|`.
5.2. Expliquer ce qu'est un document XML bien formé et valide.
5.3. Écrire un document XML bien formé et valide pour la DTD du listing 4.

Exercice 6 : XML
Considérons le code Java du listing 5.
6.1. Donner le document XML qui sera affiché sur la sortie standard par ce programme.
6.2. Donner la signature de la méthode `setAttribute` utilisée.
6.3. Proposer une DTD pour le document XML sachant qu'un point a nécessairement une abscisse et une ordonnée mais peut également avoir une couleur (information optionnelle).

Listing 4 – Une DTD

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2
3 <!ELEMENT choses ((truc|chose)+)>
4 <!ELEMENT truc (#PCDATA)>
5 <!ELEMENT chose EMPTY>
6 <!ATTLIST chose
7         a ID #REQUIRED
8         b CDATA #REQUIRED
9         c CDATA #IMPLIED>
```

Listing 5 – La classe Main

```
1  import org.jdom2.*;
2
3  class Main {
4          static Element getPoint(String name, int x, int y) {
5                  Element point = new Element("point");
6                  point.setAttribute("x", "" + x).setAttribute("y", "" + y);
7                  point.addContent(name);
8                  return point;
9          }
10         public static void main(String[] args) {
11             Element racine = new Element("polygone");
12                 racine.addContent(getPoint("O", 1, 0));
13                 racine.addContent(getPoint("N", 0, 1));
14                 racine.addContent(getPoint("E", -1, 0));
15                 racine.addContent(getPoint("S", 0, -1));
16                 Document doc = new Document(racine);
17                 JDOMTools.ecrire(doc, System.out);
18         }
19 }
```