Examen (2 heures, avec document)

Préambule : Répondre de manière concise et précise aux questions. Ne pas mettre de commentaires de documentation sauf s'ils sont nécessaires à la compréhension.

Il est conseillé de répondre directement dans 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.

Les exercices sont relativement indépendants.

Barème indicatif :

exercice	1	2	3	4	5
points	5	5	2	7	1

Inspecteur d'objet

L'objectif de ces exercices est d'écrire une petite application qui permet de visualiser la valeur des attributs d'un objet grâce à une interface graphique en Swing. Il est aussi possible de modifier certaines valeurs et de les sauvegarder dans un fichier XML.

Exercice 1 : Retrouver les attributs

Commençons par récupérer tous les attributs présents dans une classe. L'interface `ObjectInspector` (listing 1) présente la spécification d'une méthode `collectAttributes` qui a pour objectif d'ajouter tous les attributs de la classe en paramètre dans la collection aussi en paramètre.

Listing 1 – L'interface ObjectInspector

```
1  import java.util.Collection;
2  import java.lang.reflect.Field;
3
4  public interface ObjectInspector {
5
6      /** Collect all attributes of the class c (declared in class c or
7       * inherited) in the attrs collection.
8       * @param c the class which attributes are collected
9       * @param attrs the collection where collecte attributes are put in
10      */
11     void collectAttributes(Class<?> c, Collection<Field> attrs);
12
13 }
```

Listing 2 – Exemples de classes

```
1  interface I {
2      int i = 1;
3  }
4
5  class A implements I {
6      int a;
7      String b;
8  }
9
10 class B extends A implements I {
11     char a = 'x';
12 }
13
14 public class Exemple1 extends B {
15     static int ex1;
16     I unI = new A();
17 }
```

Étant données les classes du listing 2, on obtiendra par exemple les résultats suivants (résultats du programme demandé à la question 1.2) :

```
Les attributs de A sont :
  - java.lang.String A.b
  - public static final int I.i
  - int A.a

Les attributs de B sont :
  - java.lang.String A.b
  - char B.a
  - public static final int I.i
  - int A.a
```

```
Les attributs de Exemple1 sont :
  - java.lang.String A.b
  - char B.a
  - public static final int I.i
  - int A.a
  - I Exemple1.unI
  - static int Exemple1.ex1
```

1.1 Écrire une réalisation de cette interface appelée `ConcreteInspector`.

1.2 Écrire une classe `ObjectInspectorMain` qui affiche les attributs de la classe dont le nom est donné en argument de la ligne de commande.

Les exemples ci-dessus sont obtenus en faisant :

```
java ObjectInspectorMain A
java ObjectInspectorMain B
java ObjectInspectorMain Exemple1
```

Remarque : Dans les exemples ci-dessus, on a simplement afficher l'objet Java de type `Field`.

1.3 Expliquer comment il serait possible d'avoir les attributs triés dans l'ordre alphabétique de leur nom. On donnera simplement le principe en quelques phrases sans écrire le code.

Exercice 2 : Une petite visualisation avec Swing

Maintenant que nous sommes capables de récupérer les attributs d'une classe, nous allons construire

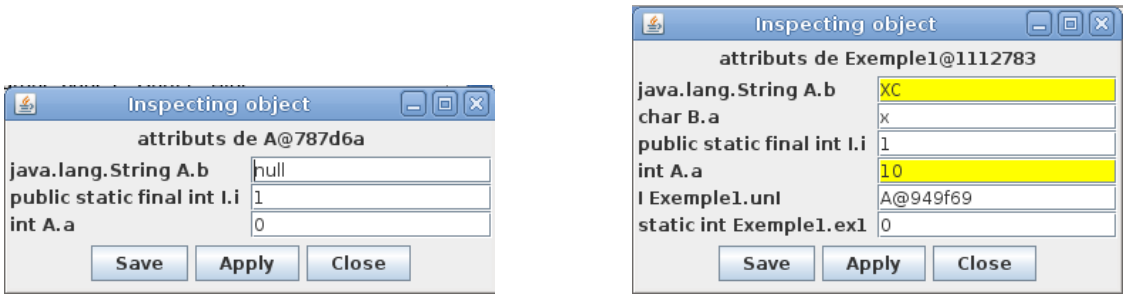

FIGURE 1 – Exemples d'interfaces utilisateur créées avec la classe du listing 3

Listing 3 – La classe SwingUIMain

```
1  import java.io.FileOutputStream;
2
3  public class SwingUIMain {
4
5      public static void main(String[] args) throws Exception {
6          Exemple1 e = new Exemple1();
7          new SwingUI(e, new FileOutputStream("/tmp/e.xml"));
8          A a = new A();
9          new SwingUI(a, new FileOutputStream("/tmp/a.xml"));
10     }
11
12 }
```

une petite application Swing qui permet de les visualiser. Dans l'exercice suivant, nous la compléterons pour pouvoir aussi agir sur les valeurs des attributs. Dans cet exercice, seuls les aspects visualisation des attributs d'un objet et ergonomie de l'application sont traités.

La figure 1 donne un exemple de l'apparence des fenêtres créées à partir d'un objet pour en inspecter la valeur des attributs. La première ligne joue le rôle de titre. Elle précise ce qui est affiché en dessous (les attributs de l'objet). L'objet lui-même est affiché en utilisant `toString()`. En dessous, la liste des attributs et de leur valeur est affiché. En fin, en bas de la fenêtre, trois boutons permettent à l'utilisateur d'interagir avec l'application. C'est la classe principale du listing 3 qui a permis de construire les deux fenêtres de la figure 1. Le fond jaune indique une valeur d'attribut qui a été modifiée. Cet aspect ne sera traité qu'à la question suivante.

2.1 Compléter le code du listing 4.

Exercice 3 : Le bouton Close

L'objectif de cet exercice est de rendre le bouton Close actif.

3.1 Donner le code à ajouter au listing 4 pour qu'un clic sur le bouton Close provoque la fermeture de la fenêtre.

Exercice 4 : Le bouton Save

L'objectif de cet exercice est de rendre le bouton Save actif. Il s'agit de sauvegarder dans un fichier XML les attributs qui ont été changés ainsi que leur nouvelle valeur.

Listing 4 – Squelette de la classe SwingUI

```
1  import java.io.OutputStream;
2  import org.jdom.*;
3  import org.jdom.output.*;
4  import java.util.*;
5  import javax.swing.*;
6  import java.awt.*;
7  import java.awt.event.*;
8  import java.lang.reflect.*;
9
10 public class SwingUI {
11
12     private Object object;
13     private OutputStream output;
14     private JButton saveButton = new JButton("Save");
15     private JButton applyButton = new JButton("Apply");
16     private JButton closeButton = new JButton("Close");
17     private JFrame frame = new JFrame("Inspecting␣object");
18
19     private Map<Field, String> mappings = new HashMap<Field, String>();
20
21     public SwingUI(Object o, OutputStream output) {
22         this.object = o;
23         this.output = output;
24
25
26
27         frame.pack();
28         frame.setVisible(true);
29     }
30
31
32     private static void ecrire(Document document, OutputStream out) {
33         try {
34             XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
35             sortie.output(document, out);
36         } catch (java.io.IOException e) {
37             throw new RuntimeException("Erreur␣sur␣écriture␣:␣", e);
38         }
39     }
40
41 }
```

4.1 *Conserver les attributs modifiés.* Le principe est de conserver dans un tableau associatif (Map), l'attribut qui a été changé (Field) et la nouvelle valeur (String). Un attribut est modifié quand une nouvelle valeur est saisie dans la zone de saisie correspondante. On utilisera l'ActionListener du JTextField pour détecter de tels changements.

Quand la valeur d'un attribut est changée (même si la nouvelle valeur est la même que l'ancienne), le fond du JTextField sera mis à jaune (Color.YELLOW) en utilisant la méthode setBackground.

Compléter le code du listing 4.

4.2 *Sauver en XML.* Quand on clique sur le bouton « Save », les données enregistrées dans le tableau associatif sont écrites dans le OutputStream output au format XML. Le résultat pour l'objet Exemple1 de la figure 1 est le suivant :

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE attributs SYSTEM "attributs.dtd">
3
4 <attributs>
5   <attribut name="b" class="A">XC</attribut>
6   <attribut name="a" class="A">10</attribut>
7 </attributs>
```

4.2.1 Donner une DTD pour le fichier XML précédent.

4.2.2 Compléter le listing 4 pour que le bouton « Save » écrive le document XML sur l'attribut « output ».

Exercice 5 : Le bouton Apply

Expliquer en quelques phrases, sans écrire le code correspondant, comment faire pour que le bouton « Apply » applique les modifications enregistrées sur l'objet (`this.object`).