Examen (2 heures, avec document)
Corrigé

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.

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`.
Solution :

Listing 1 – L'interface ObjectInspector

```java
import java.util.Collection;
import java.lang.reflect.Field;

public interface ObjectInspector {

    /** Collect all attributes of the class c (declared in class c or
     * inherited) in the attrs collection.
     * @param c the class which attributes are collected
     * @param attrs the collection where collecte attributes are put in
     */
    void collectAttributes(Class<?> c, Collection<Field> attrs);

}
```

Listing 2 – Exemples de classes

```java
interface I {
    int i = 1;
}

class A implements I {
    int a;
    String b;
}

class B extends A implements I {
    char a = 'x';
}

public class Exemple1 extends B {
    static int ex1;
    I unI = new A();
}
```

```java
import java.util.Collection;
import java.lang.reflect.Field;

public class ConcreteInspector implements ObjectInspector{

    public void collectAttributes(Class<?> c, Collection<Field> attrs) {
        if (c == null) {
            return;     // no attribute to collect !
        }

        // collect attributes declared in the c class
        for (Field f : c.getDeclaredFields()) {
            attrs.add(f);
        }

        // collect attributes inherited by the c class
```

```
17          this.collectAttributes(c.getSuperclass(), attrs);
18
19          // collect attributes from super interfaces of c
20          for (Class<?> si : c.getInterfaces()) {
21              this.collectAttributes(si, attrs);
22          }
23      }
24
25  }
```

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`.

Solution :

```
1   import java.util.*;
2   import java.lang.reflect.Field;
3
4   public class ObjectInspectorMain {
5
6       public static void main(String[] args) throws Exception {
7           Class<?> c = (args.length == 0) ? java.util.ArrayList.class
8                   : Class.forName(args[0]);
9           Set<Field> attributs = new HashSet<Field>();
10          ObjectInspector oi = new ConcreteInspector();
11          oi.collectAttributes(c, attributs);
12
13          // Afficher les champs :
14          System.out.println("Les␣attributs␣de␣" + c.getName() + "␣sont␣:");
15          for (Field f: attributs) {
16              System.out.println("␣␣-␣" + f);
17          }
18      }
19
20  }
```

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.

Solution : On pourrait utiliser un SortedSet en utilisant un comparateur.

On pourrait utiliser la méthode Collections.sort.

Exercice 2 : Une petite visualisation avec Swing

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

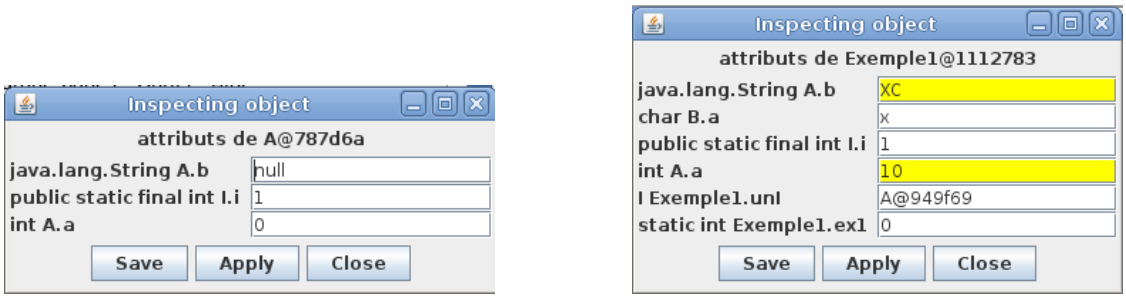

FIGURE 1 – Exemples d'interfaces utilisateur créées avec la classe du listing 3

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 }
```

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.

Solution : La classe suivante inclut également la réponse aux questions suivantes.

```
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          JPanel boutons = new JPanel(new FlowLayout());
26          boutons.add(saveButton);
27          boutons.add(applyButton);
28          boutons.add(closeButton);
29
30          Container c = frame.getContentPane();
31          c.setLayout(new BorderLayout());
32          c.add(boutons, BorderLayout.SOUTH);
33
34          JPanel titrePanel = new JPanel(new FlowLayout());
35          JLabel titre = new JLabel("attributs␣de␣" + this.object);
36          titrePanel.add(titre);
37          c.add(titrePanel, BorderLayout.NORTH);
38
39          ObjectInspector oi = new ConcreteInspector();
40          Set<Field> attrs = new HashSet<Field>();
41          oi.collectAttributes(this.object.getClass(), attrs);
42
43          JPanel main = new JPanel(new GridLayout(attrs.size(), 2));
44          for (Field f : attrs) {
45              JLabel label = new JLabel(f.toString());
46              JTextField saisie = new JTextField(15);
47
48              // initialiser avec la valeur de l'attribut
49              Object value;
50              try {
51                  value = f.get(this.object);
52                  if (value != null) {
53                      saisie.setText(value.toString());
54                  } else {
55                      saisie.setText("null");
56                  }
57              } catch (IllegalAccessException e) {
58                  saisie.setText("?");
59                  saisie.setBackground(Color.PINK);
60                  saisie.setEditable(false);
61              }
62
63              saisie.addActionListener(new ActionChange(f));
64
65              main.add(label);
66              main.add(saisie);
```

```
67          }
68          c.add(main, BorderLayout.CENTER);
69
70          closeButton.addActionListener(new ActionListener() {
71              public void actionPerformed(ActionEvent ev) {
72                  frame.dispose();
73              }
74          });
75
76          saveButton.addActionListener(new ActionSave());
77          applyButton.addActionListener(new ActionApply());
78
79          frame.pack();
80          frame.setVisible(true);
81      }
82
83      private class ActionChange implements ActionListener {
84          private Field field;
85
86          public ActionChange(Field f) {
87              this.field = f;
88          }
89
90          public void actionPerformed(ActionEvent e) {
91              JTextField jtf = (JTextField) e.getSource();
92              jtf.setBackground(Color.YELLOW);
93              mappings.put(this.field, jtf.getText());
94          }
95      }
96
97      private class ActionSave implements ActionListener {
98          public void actionPerformed(ActionEvent e) {
99              Element racine = new Element("attributs");
100             for (Map.Entry<Field, String> entry : mappings.entrySet()) {
101                 Field f = entry.getKey();
102                 // System.out.println(f + " --> " + entry.getValue());
103                 Element attrElt = new Element("attribut");
104                 attrElt.setAttribute("name", f.getName());
105                 attrElt.setAttribute("class", f.getDeclaringClass().getName());
106                 attrElt.setText(entry.getValue());
107                 racine.addContent(attrElt);
108             }
109             Document document = new Document(racine,
110                     new DocType("attributs", "attributs.dtd"));
111
112             try {
113                 ecrire(document, SwingUI.this.output);
114             } catch (Exception ex) {
115                 throw new RuntimeException(ex);
116             }
117
118         }
119     }
120
```

```
121     private class ActionApply implements ActionListener {
122
123         public void actionPerformed(ActionEvent e) {
124             for (Map.Entry<Field, String> entry : mappings.entrySet()) {
125                 Field f = entry.getKey();
126                 String value = entry.getValue();
127                 Class<?> type = f.getType();
128                 try {
129                     if (type == int.class) {
130                         f.setInt(SwingUI.this.object, Integer.parseInt(value));
131                         // XXX NumberFormatException is handled !
132                     } else if (type == char.class) {
133                         if (value.length() > 0) {
134                             f.setChar(SwingUI.this.object, value.charAt(0));
135                         }
136                         // XXX : il faut faire pareil pour tous les types de
137                         // base
138                     } else if (type == String.class) {
139                         f.set(SwingUI.this.object, value);
140                     } else {
141                         System.out.println(type + "␣is␣not␣supported␣yet!");
142                         // XXX Il faudrait que la classe puisse s'initaliser à
143                         // partir d'un String, par exemple avec valueOf ou
144                         // équivalent.
145                     }
146                 } catch (Exception ex) {
147                     System.out.println(ex);
148                 }
149             }
150         }
151
152     }
153
154     private static void ecrire(Document document, OutputStream out) {
155         try {
156             XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
157             sortie.output(document, out);
158         } catch (java.io.IOException e) {
159             throw new RuntimeException("Erreur␣sur␣écriture␣:␣", e);
160         }
161     }
162
163 }
```

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.

Solution : Voir la classe anonyme du `closeBouton.addActionListener(...)`.

Exercice 4 : Le bouton Save

L'objectif de cet exercice est de rendre le bouton Save actif. Il s'agit de sauvegarder dans un

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 }
```

fichier XML les attributs qui ont été changés ainsi que leur nouvelle valeur.

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.

Solution : Voir la classe ActionChange.

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.

Solution :

```
1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2
3 <!ELEMENT attributs (attribut)*>
4 <!ELEMENT attribut (#PCDATA)>
5 <!ATTLIST attribut
6         name CDATA #REQUIRED
7         class CDATA #REQUIRED
8 >
```

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

Solution : Voir la classe ActionSave.

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`).

Solution : Voir la classe ActionApply.