Examen (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
points	8	4	6	2

Sorte de calculette

L'objectif de ces exercices est de réaliser une sorte de calculette qui permet d'utiliser les méthodes d'une classe Java qui prennent en paramètre un réel [1] et retourne un réel. La figure 1 donne l'IHM de cette calculette pour la classe `java.lang.Math`.

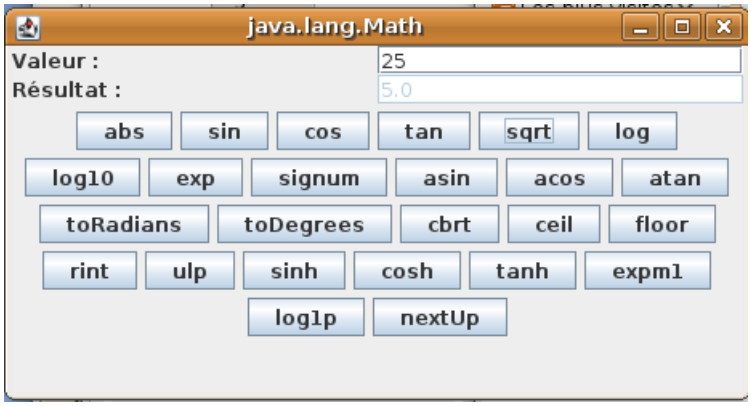

FIGURE 1 – Calculette pour la classe `java.lang.Math`

La classe principale s'appelle `Calculette`. Elle prend en paramètre le nom d'une classe Java. Par exemple :

1. Dans ce sujet, réel signifie réel double précision.

```
java Calculette java.lang.Math
```

La partie supérieure de la fenêtre contient deux champs (`JTextField`). Le premier permet à l'utilisateur de saisir une valeur réelle. Le second ne peut pas être modifié et est destiné à recevoir le résultat calculé pour la valeur saisie dans le champ précédent. La partie principale de la fenêtre contient des boutons. Chaque bouton correspond à une méthode de la classe donnée en argument de la calculette. Plus précisément, seules les méthodes de classe qui sont publiques, n'ont qu'un seul paramètre de type réel et retournent un réel sont prises en compte. Le libellé du bouton est le nom de la méthode correspondante. C'est quand l'utilisateur clique sur un bouton que le résultat est calculé pour la valeur saisie. Par exemple, sur la figure 1, l'utilisateur a saisi la valeur 25. Il a cliqué sur le bouton « sqrt ». Le résultat 5.0 apparaît dans le champ « résultat ».

Les exercices suivants ont pour objectif de compléter le squelette de cette application (donné au listing 1) et d'ajouter quelques fonctionnalités supplémentaires.

Listing 1 – Squelette de Calculette.java

```
1
2  // Ce code correspond à l'exercice 1, partie Swing.
3
4
5  // Ce code correspond à l'exercice 1, partie Introspection (mais
6  // qui bien sûr utiliser des composants Swing.
7
8
9  // Ce code correspond à l'exercice 2.
10
11
12 // Ce code correspond à l'exercice 3.
13
14 import java.awt.*;
15 import javax.swing.*;
16
17 import java.awt.event.*;
18
19
20 import java.lang.reflect.*;
21
22
23 import java.util.*;
24
25
26
27 import org.jdom.*;
28 import org.jdom.output.*;
29 import java.io.*;
30
31
32 public class Calculette {
33
34     private Class laClasse;
35
36     private JFrame fenetre;
37     private JTextField valeur = new JTextField(10);
```

```
38      private JTextField resultat = new JTextField(10);
39
40      private Map<String, Collection<Couple<Double, Double>>> table;
41
42
43
44      final private ActionListener fonctionListener = new ExecuterFonction();
45
46
47      public Calculette(Class laClasse) {
48          this.laClasse = laClasse;
49
50          this.table = new TreeMap<String, Collection<Couple<Double, Double>>>();
51
52
53          // Construire l'IHM
54          this.fenetre = new JFrame(laClasse.getName());
55          this.resultat.setEditable(false);
56
57
58          this.fenetre.addWindowListener(new SaveItems());
59
60
61          Container contenu = this.fenetre.getContentPane();
62          contenu.setLayout(new BorderLayout());
63          // les valeurs et résultats
64          JPanel dialogue = new JPanel(new GridLayout(2, 2));
65          dialogue.add(new JLabel("Valeur : "));
66          dialogue.add(valeur);
67          dialogue.add(new JLabel("Résultat : "));
68          dialogue.add(resultat);
69          contenu.add(dialogue, BorderLayout.NORTH);
70
71          // les boutons
72          JPanel boutons = new JPanel(new FlowLayout());
73
74          for (Method m : laClasse.getMethods()) {
75              Class<?>[] parametres = m.getParameterTypes();
76              if (parametres.length == 1 && parametres[0] == double.class
77                      && Modifier.isPublic(m.getModifiers())
78                      && Modifier.isStatic(m.getModifiers())
79                      && m.getReturnType() == double.class) {
80                  JButton bouton = new JButton(m.getName());
81                  bouton.addActionListener(fonctionListener);
82                  boutons.add(bouton);
83              }
84          }
85
86          contenu.add(boutons, BorderLayout.CENTER);
87
88
89          this.fenetre.setSize(466, 246);
90          this.fenetre.setVisible(true);
91
```

```
92      }
93
94
95      private class ExecuterFonction implements ActionListener {
96
97          public void actionPerformed(ActionEvent ev) {
98              try {
99                  JButton bouton = (JButton) ev.getSource();
100                 String name = bouton.getText();
101                 System.out.println("ActionListener : " + name);
102                 Method fn = laClasse.getMethod(name, double.class);
103                 double arg = Double.parseDouble(valeur.getText());
104                 Double r = (Double) fn.invoke(null, new Double(arg));
105                 resultat.setText("" + r);
106
107                 Collection<Couple<Double, Double>> collection = table.get(name);
108                 if (collection == null) {
109                     collection = new LinkedList<Couple<Double, Double>>();
110                     table.put(name, collection);
111                 }
112                 collection.add(new Couple<Double, Double>(arg, r));
113                 System.out.println("Tous les résultats : " + table);
114
115             } catch (NumberFormatException e) {
116                 resultat.setText("la valeur doit être un réel !");
117             } catch (NoSuchMethodException e) {
118                 assert false : "Mauvaise construction de la liste des fonctions";
119             } catch (IllegalAccessException e) {
120                 assert false : "Mauvaise construction de la liste des fonctions";
121             } catch (InvocationTargetException e) {
122                 assert false : "Mauvaise construction de la liste des fonctions";
123             }
124         }
125
126     }
127
128
129
130     private class SaveItems extends WindowAdapter {
131
132         public void windowClosed(WindowEvent e) {
133             System.out.println("Closed !");
134         }
135
136         public void windowClosing(WindowEvent e) {
137             System.out.println("Closing !");
138             // Créer le Document JDOM
139             Element racine = new Element("results");
140             for (String fn: table.keySet()) {
141                 Element eltFn = new Element("function")
142                         .setAttribute("name", fn);
143                 racine.addContent(eltFn);
144                 for (Couple<Double, Double> c : table.get(fn)) {
145                     eltFn.addContent(new Element("run")
```

```
146                             .setAttribute("arg", "" + c.a)
147                             .setAttribute("res", "" + c.b));
148                 }
149             }
150             Document document = new Document(racine, new DocType("results",
151                     "results.dtd"));
152             try {
153                 ecrire(document, new FileOutputStream(laClasse.getName() + ".xml"));
154             } catch (Exception ex) {
155                 throw new RuntimeException(ex);
156             }
157             System.exit(0);
158         }
159
160     }
161
162     public static void ecrire(Document document, OutputStream out) {
163         try {
164             XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
165             sortie.output(document, out);
166         } catch (java.io.IOException e) {
167             throw new RuntimeException("Erreur sur écriture : ", e);
168         }
169     }
170
171
172
173     public static void main(String[] args) throws Exception {
174         if (args.length != 1) {
175             System.out.println("Il faut un argument : le nom de la classe");
176         } else {
177
178             Class c = Class.forName(args[0]);
179
180             new Calculette(c);
181         }
182     }
183
184 }
```

Exercice 1 : Version initiale de la calculette

Compléter le listing 1 pour rendre opérationnelle l'application. Elle doit ressembler à la figure 1 avec le comportement décrit ci-avant.

Solution : Il s'agit du code avec fond bleu (Swing) et rose (Introspection) sur le listing 1.

Exercice 2 : Conserver tous les calculs faits

On souhaite conserver tous les calculs faits dans un tableau associatif (Map). La clé sera le nom de la méthode appliquée (sqrt, sin, cos, etc.), l'information associée sera une collection de couples composés d'une valeur et du résultat correspondant.

2.1 Définir une classe Couple qui permet de conserver deux valeurs que l'on notera `a` et `b`. Cette classe doit être générale et fonctionner quelque soit le type de `a` et le type de `b`. Dans le cas de la calculette, le type de `a` et `b` sont identiques : des réels double précision.

Solution :

```
1  public class Couple<A, B> {
2      public A a;
3      public B b;
4
5      public Couple(A a, B b) {
6          this.a = a;
7          this.b = b;
8      }
9
10     public String toString() {
11         return "" + a + " -> " + b;
12     }
13 }
```

2.2 Compléter l'application pour que chaque nouveau calcul soit conserver dans le tableau associatif.
Solution : Il s'agit du code avec fond vert sur le listing 1.

Exercice 3 : Persistance en XML
On souhaite conserver dans un fichier XML tous les calculs réalisés depuis le lancement de l'application (ceux conservés dans le tableau associatif). Le listing 2 donne un exemple d'un tel fichier. Cette sauvegarde doit être faite lorsque l'application est quittée. Le listing 3 donne un extrait de la documentation de `WindowListener`.

3.1 Écrire une DTD à laquelle le fichier du listing 2 serait conforme.
Solution :

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2
3  <!ELEMENT results       (fn*)>
4  <!ELEMENT function      (run+)>
5  <!ATTLIST function
6                  name    ID      #REQUIRED
7  >
8  <!ELEMENT run   (EMPTY)>
9  <!ATTLIST run
10                 arg     ID      #REQUIRED
11                 res     ID      #REQUIRED
12 >
```

3.2 Compléter l'application pour implanter cette persistance. On pourra utiliser la méthode suivante :

```
1      public static void ecrire(Document document, OutputStream out) {
2          try {
3              XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
4              sortie.output(document, out);
5          } catch (java.io.IOException e) {
6              throw new RuntimeException("Erreur sur écriture : ", e);
7          }
8      }
```

Solution : Il s'agit du code avec fond jaune sur le listing 1.

Listing 2 – Exemple de fichier XML produit

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE results SYSTEM "results.dtd">

<results>
  <function name="ceil">
    <run arg="-3.5" res="-3.0" />
  </function>
  <function name="floor">
    <run arg="-3.5" res="-4.0" />
  </function>
  <function name="log10">
    <run arg="10.0" res="1.0" />
  </function>
  <function name="sqrt">
    <run arg="9.0" res="3.0" />
    <run arg="25.0" res="5.0" />
  </function>
</results>
```

Listing 3 – Extrait de l'interface WindowListener

```
public interface WindowListener extends EventListener {
 void windowActivated(WindowEvent e)
        // Invoked when the Window is set to be the active Window.
 void windowClosed(WindowEvent e)
        // Invoked when a window has been closed as the result of
        // calling dispose on the window.
 void windowClosing(WindowEvent e)
        // Invoked when the user attempts to close the window from the
        // window's system menu.
 void windowDeactivated(WindowEvent e)
        // Invoked when a Window is no longer the active Window.
 void windowDeiconified(WindowEvent e)
        // Invoked when a window is changed from a minimized to a
        // normal state.
 void windowIconified(WindowEvent e)
        // Invoked when a window is changed from a normal to a
        // minimized state.
 void windowOpened(WindowEvent e)
        // Invoked the first time a window is made visible.
}
```

Exercice 4 : Amélioration de l'architecture de l'application

Les différents ajouts réalisés ont conduit à intervenir directement dans le code de l'application. Ceci traduit une application peu évolutive. Expliquer comment il aurait été possible de structurer initialement l'application pour permettre les évolutions sans avoir à modifier le code initial.

Solution : Utiliser le patron Observateur. À chaque fois qu'un calcul est fait, avertir.