Examen (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
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`.

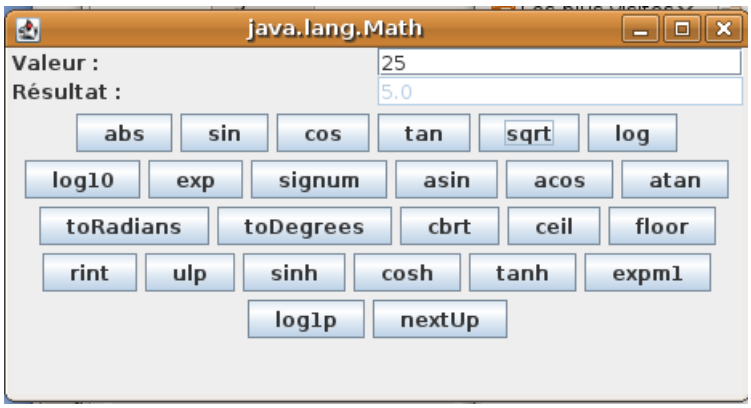

FIGURE 1 – Calculette pour la classe `java.lang.Math`

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

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

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

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  import java.awt.*;
2  import javax.swing.*;
3  import java.lang.reflect.*;
4
5  public class Calculette {
6
7      private Class laClasse;
8
9      private JFrame fenetre;
10     private JTextField valeur = new JTextField(10);
11     private JTextField resultat = new JTextField(10);
12
13
14     public Calculette(Class laClasse) {
15         this.laClasse = laClasse;
16
17         // Construire l'IHM
18         this.fenetre = new JFrame(laClasse.getName());
19         this.resultat.setEditable(false);
20
21
22         this.fenetre.setSize(466, 246);
23         this.fenetre.setVisible(true);
24
25     }
26
27
28     public static void main(String[] args) throws Exception {
29         if (args.length != 1) {
30             System.out.println("Il faut un argument : le nom de la classe");
31         } else {
32             Class c = Class.forName(args[0]);
33             new Calculette(c);
34         }
35     }
36
37 }
```

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.

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.

2.2 Compléter l'application pour que chaque nouveau calcul soit conserver dans le tableau associatif.

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.

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      }
```

Listing 2 – Exemple de fichier XML produit

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE results SYSTEM "results.dtd">
3
4  <results>
5    <function name="ceil">
6      <run arg="-3.5" res="-3.0" />
7    </function>
8    <function name="floor">
9      <run arg="-3.5" res="-4.0" />
10   </function>
11   <function name="log10">
12     <run arg="10.0" res="1.0" />
13   </function>
14   <function name="sqrt">
15     <run arg="9.0" res="3.0" />
```

```
16      <run arg="25.0" res="5.0" />
17    </function>
18  </results>
```

Listing 3 – Extrait de l'interface WindowListener

```
1  public interface WindowListener extends EventListener {
2   void windowActivated(WindowEvent e)
3          // Invoked when the Window is set to be the active Window.
4   void windowClosed(WindowEvent e)
5          // Invoked when a window has been closed as the result of
6          // calling dispose on the window.
7   void windowClosing(WindowEvent e)
8          // Invoked when the user attempts to close the window from the
9          // window's system menu.
10  void windowDeactivated(WindowEvent e)
11         // Invoked when a Window is no longer the active Window.
12  void windowDeiconified(WindowEvent e)
13         // Invoked when a window is changed from a minimized to a
14         // normal state.
15  void windowIconified(WindowEvent e)
16         // Invoked when a window is changed from a normal to a
17         // minimized state.
18  void windowOpened(WindowEvent e)
19         // Invoked the first time a window is made visible.
20 }
```

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.