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 une lettre majuscule : A, B, C, 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	3	3	4	4	6

Exercice 1 Indiquer au moins trois patrons de conception utilisés dans Swing en indiquant clairement où ils apparaissent dans Swing.

Gestion de préférences

L'objectif de ces exercices est de proposer un moyen de définir des préférences. Les préférences permettent de paramétrer une application en associant une valeur à un nom. L'exercice 2 définit la classe `Preference`. L'exercice 3 permet de sauvegarder les préférences au format XML. L'exercice 4 permet d'extraire les préférences à partir d'une classe Java. Enfin, l'exercice 5 propose une application Swing pour saisir les valeurs des préférences.

Exercice 2 : La classe Preference
On considère qu'une préférence est définie par un nom, un type, une description et une valeur. La description peut ne pas être donnée (**`null`**). La valeur initiale est indéterminée (**`null`**) et pourra donc être positionnée ultérieurement.

Le type permet de connaître le type de l'information et est utilisé pour savoir comment initialiser la valeur à partir d'une chaîne de caractères. Le principe est d'utiliser la méthode `valueOf` prenant en paramètre un `String` ou, à défaut, un `Object`. Cette méthode doit bien sûr exister sur le type considéré. Par exemple, les classes `Integer`, `Boolean` ou `Double` définissent la méthode `valueOf(String)` qui retourne un élément du type considéré, construit à partir d'une chaîne de caractères. La classe `String` définit la méthode `valueOf(Object)` qui retourne la représentation de l'objet sous la forme d'une chaîne de caractères (en utilisant `Object.toString()`).

```
1 Double d = Double.valueOf("3.14");
2 Integer i = Integer.valueOf("123");
3 Boolean b = Boolean.valuOf("true");
4 String s = String.valueOf(new Date());
```

On suppose que l'on pourra utiliser les types élémentaires (double, int, boolean, etc.) qui seront remplacés par le type enveloppe correspondant (`Double`, `Integer`, `Boolean`, etc.). De même, on pourra utiliser « string » pour désigner la classe `String`.

Le code partiel de la classe `Preference` est donné au listing 1.

2.1 Indiquer quand est exécuté le code qui commence à la ligne 18.

2.2 Compléter le code du constructeur. Lire attentivement sa documentation.

2.3 Compléter le code de `setValeur(String)`. Lire attentivement sa documentation.

Listing 1 – La classe Preference (code partiel)

```
1  import java.lang.reflect.Method;
2  import java.util.*;
3
4  /** Préférence typée avec initialisation grâce à ce type. */
5  public class Preference {
6      private String nom;
7      private String nomType;
8      private Object valeur;
9      private String description;
10
11     private Method valueOf;
12     private Class<?> type;
13         //@ invariant value != null ==>
14         //@     type.getDeclaringClass() == value.getClass();
15
16     static Map<String, Class<?>> typesPredefinis;
17
18     static {
19         typesPredefinis = new HashMap<String, Class<?>>();
20         typesPredefinis.put("string", String.class);
21         typesPredefinis.put("int", Integer.class);
22         typesPredefinis.put("double", Double.class);
23         typesPredefinis.put("float", Float.class);
24         typesPredefinis.put("boolean", Boolean.class);
25         // XXX cette liste n'est pas complète
26     }
27
28     /** Initialiser cette préférence à partir de son nom et son type.
29      * Notons que pour les types prédéfinis (int, double, etc) on
30      * utilisera les classes enveloppes correspondantes.  Le type <<
31      * string >> correspondra au type java.lang.String.
32      * @param nom le nom de la propriété
33      * @param type le type de cette propriété
34      * @throws IllegalArgumentException si la classe correspondant à
35      * type n'existe pas (ClassNotFoundException) ou si elle ne contient
36      * pas la méthode valueOf adéquate (NoSuchMethodException).
37      */
38     public Preference(String nom, String type) {
39         this.nom = nom;
40         this.nomType = type;
41
42         // Initialiser this.type (la classe qui correspond au type
43         // this.type et this.valueOf sa méthode valueOf qui prend un
```

```
44          // String en paramètre, à défaut un Object.
45          Class<?> typePredefini = typesPredefinis.get(type);
46          try {
47              if (typePredefini != null) {
48                  this.type = typePredefini;
49              } else {
50
51                  // À COMPLÉTER
52
53              }
54
55              // À COMPLÉTER....
56
57          } catch (ClassNotFoundException e) {
58              throw new IllegalArgumentException("Type inconnu : " + type, e);
59          }
60      }
61
62      public Object getValeur() {
63          return this.valeur;
64      }
65
66      /** Mettre à jour la valeur de cette préférence à partir de la
67       * chaîne de caractères valeur.  Cette mis à jour est réalisée au
68       * moyen de la méthode this.valueOf.
69       * @param valeur valeur à utiliser pour l'initialisation
70       * @throws IllegalArgumentException si l'application de la méthode
71       * valueOf signale un problème (InvocationTargetException).
72       */
73      public void setValeur(String valeur) {
74
75          // À COMPLÉTER...
76
77      }
78
79      /** Initialiser directement la valeur de l'objet.
80       * @param valeur nouvelle valeur de l'objet
81       */
82      public void setValeur(Object valeur) {
83          this.valeur = valeur;
84      }
85
86      public Class<?> getType() {
87          return this.type;
88      }
89
90      public String getNomType() {
91          return this.nomType;
92      }
93
94      public String getNom() {
95          return this.nom;
96      }
97
```

```
98      public void setDescription(String description) {
99          this.description = description;
100     }
101
102     public String getDescription() {
103         return this.description;
104     }
105
106     @Override
107     public String toString() {
108         return "<" + this.nom + ": " + this.type + " = " + this.valeur + ">";
109     }
110
111 }
```

Exercice 3 : Engendrer une représentation XML des préférences

On souhaite pouvoir écrire une collection de préférences en XML dans un fichier. L'interface suivante est définie dans cet objectif.

```
1  import java.util.Collection;
2  import java.io.OutputStream;
3
4  public interface IPreferencesXML {
5      void genererXML(OutputStream out, Collection<Preference> preferences);
6  }
```

La DTD suivante a été définie pour définir la structure du document XML.

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2
3  <!ELEMENT preferences   (preference*)>
4  <!ELEMENT preference    (description?)>
5  <!ATTLIST preference
6                  nom     ID      #REQUIRED
7                  type    CDATA   #REQUIRED
8                  valeur  CDATA   #IMPLIED
9  >
10 <!ELEMENT description   (PCDATA)>
```

Voici un exemple de fichier XML conforme à la DTD précédente.

```
1  <?xml version="1.0" encoding="ISO-8859-1"  standalone="no"?>
2
3  <!DOCTYPE preferences SYSTEM "preferences.dtd">
4
5  <preferences>
6      <preference nom="debug" type="java.lang.Boolean" >
7          <description>
8              afficher des informations de mise au point
9          </description>
10     </preference>
11     <preference nom="taille" type="int" valeur="10" >
12     </preference>
13     <preference nom="fichierLog" type="java.lang.String"
14                 valeur="/tmp/log.txt" >
```

```
15          <description>
16              Le nom du fichier dans lequel les informations de mise au point
17              doivent être affichées.
18          </description>
19      </preference>
20  </preferences>
```

3.1 Expliquer l'intérêt d'utiliser `OutputStream` comme type du paramètre alors que l'on veut écrire dans un fichier.

3.2 Écrire une réalisation `PreferencesXML` qui utilise JDom. On supposera qu'il existe une méthode `ecrire(Document, OutputStream)` qui écrit un document JDom sur un flot de sortie.

Exercice 4 : Construire des préférences à partir d'une classe

On souhaite maintenant pouvoir construire les préférences à partir d'une classe. Les préférences seront stockées dans un tableau associatif (`Map`) dont la clé sera le nom de la préférence et la valeur la préférence elle-même.

On veut définir la méthode suivante dans la classe `Preference` :

```
public static Map<String, Preference> getPreferences(String nomClasse);
```

Le principe est de considérer que tous les modifieurs de la classe (méthodes dont le nom commence par « set » et qui n'ont qu'un seul paramètre) correspondent à une préférence. Le nom de cette préférence est le nom de la méthode [1] et son type est le type de l'unique paramètre.

Voici un exemple d'un telle classe.

```
1  public class Exemple1 {
2      public void setTitre(String v) { }
3      public void setNom(String v) { }
4      public void setVolume(double v) { }
5      public void setVivant(boolean v) { }
6      public void setNombre(int v) { }
7
8      public void setCartesien(double x, double y) { }
9  }
```

La dernière méthode ne sera pas considérée comme une préférence car elle prend deux paramètres.

4.1 Expliquer l'intérêt d'utiliser un tableau associatif (`Map`) plutôt qu'une collection pour stocker les préférences.

4.2 Écrire la méthode `getPreferences(String)`.

Exercice 5 : Construire une interface graphique

On veut écrire une application Swing pour renseigner les préférences récupérées d'une classe. Une capture est donnée figure 1. L'utilisateur commence par saisir le nom de la classe (ici « Exemple1 ») qui servira à définir les préférences. Ensuite, il clique sur charger. Les préférences récupérées de la classe sont affichées dans la partie centrale de la fenêtre. L'utilisateur peut alors appuyer sur le bouton « MAJ » qui réalise la mise à jour de la collection `preferences` (collection

1. En fait, le nom de la préférence devrait être obtenu en supprimant « set » et en mettant en minuscule la première lettre. Ceci n'est pas demandé.

initialement passée en paramètre du constructeur ou positionnée ensuite par `setPreferences`) en fonction des valeurs saisies dans les zones de saisies correspondantes (utilisation de la méthode `Preference.setValeur(String)`). Le bouton XML permet d'engendrer le fichier XML « output.xml » correspondant aux préférences. Enfin, le bouton « Annuler » permet de quitter l'application.

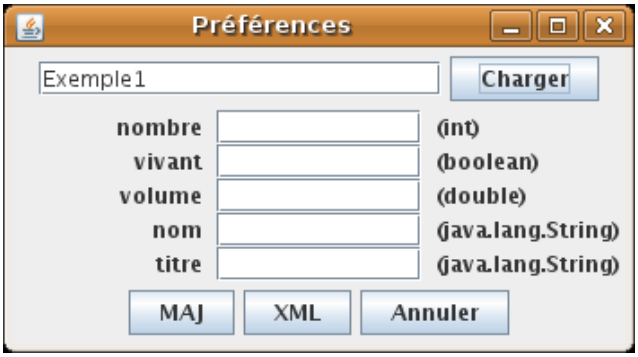

FIGURE 1 – L'application PreferencesSwing, une fois la classe Exemple1 chargée

Le code partiel de la classe `PreferencesSwing` est donné ci-dessous.

5.1 Compléter le code du constructeur. Les « ..... » correspondent à une seule instruction.

5.2 Compléter le code de la méthode `setPreferences` qui met à jour l'attribut `preferences` et recrée la partie centrale de la fenêtre (pour chaque préférence, un `JLabel` pour le nom, un `JTextField` pour la valeur et une `JLabel` pour le type). Notons que les `JTextField` sont conservées dans l'attribut « zonesDeSaisie ».

5.3 Expliquer l'intérêt de l'attribut « zonesDeSaisie » de type `Collection<JTextField>`.

5.4 Rendre actifs les différents boutons.

```
1  import java.io.*;
2  import java.util.*;
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6
7  public class PreferencesSwing extends JFrame {
8
9      private JTextField nomClasse = new JTextField("Exemple1", 20);
10     private JButton bCharger = new JButton("Charger");
11     private JButton bAnnuler = new JButton("Annuler");
12     private JButton bMAJ = new JButton("MAJ");
13     private JButton bXML = new JButton("XML");
14     private Collection<JTextField> zonesDeSaisie;
15     private Collection<Preference> preferences;
16     private JPanel pPrefs;      // le panel contenant les préférences
17
18     public PreferencesSwing(Collection<Preference> preferences) {
19         super("Préférences");
20         Container c = this.getContentPane();
```

```
21
22          // ....................................................
23          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24
25          // Construire la partie supérieure
26          JPanel chargementPreferencePanel = new JPanel();
27
28          // ....................................................
29          chargementPreferencePanel.add(nomClasse);
30          chargementPreferencePanel.add(bCharger);
31
32          // ....................................................
33
34          // Construire les boutons de commandes inférieurs
35          JPanel boutons = new JPanel();
36
37          // ....................................................
38          boutons.add(bMAJ);
39          boutons.add(bXML);
40          boutons.add(bAnnuler);
41
42          // ....................................................
43
44          // Construire la partie préférences
45          this.pPrefs = new JPanel();
46
47          // ....................................................
48          if (preferences != null) {
49              this.setPreferences(preferences);
50          }
51
52          // Positionner les réactions
53
54          // À COMPLÉTER
55
56
57          this.pack();
58          this.setVisible(true);
59      }
60
61      public void setPreferences(Collection<Preference> preferences) {
62          this.preferences = preferences;
63          pPrefs.removeAll();     // supprimer tous les composants de pPrefs
64
65          // À COMPLÉTER
66
67          this.pack();    // recalculer les dimmensions optimales de la fenêtre
68      }
69
70
71          // À COMPLÉTER
72
73      public static void main(String[] args) {
74          new PreferencesSwing(null);
```

```
75          }
76
77  }
```