Examen (avec documents)

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	4	4	4	4	4

Vérificateur de propriétés sur une classe

Ces exercices ont pour but de définir un vérificateur de propriétés sur des classes Java. L'idée est de pouvoir facilement ajouter de nouvelles propriétés pour enrichir le vérificateur.

Nous commençons par définir quelques exemples de propriétés (exercice 1). Nous écrivons ensuite un évaluateur de telles propriétés (exercice 2). Nous proposons ensuite une description en XML pour représenter les propriétés qui pourra être exploitée pour construire une collection de propriétés à vérifier (exercice 3), description XML qui pourra être engendrée à partir d'une classe considérée comme modèle (exercice 4). Enfin, nous définissons une petite interface graphique pour cette application (exercice 5).

Exercice 1 : Quelques propriétés concrètes

L'objectif étant de pouvoir ajouter facilement de nouvelles propriétés, nous définissons une interface `Propriete` qui spécifie deux méthodes : `getTexte()` qui explique la vérification faite par la propriété et `verifier(Class<?> c)` qui évalue la propriété sur une classe. Les commentaires de documentation expliquent ces deux méthodes. Un exemple d'utilisation de propriétés concrètes est donné dans l'exercice suivant.

```
1  public interface Propriete {
2      /** Vérifier si cette propriété est satisfaite sur la classe c.
3       * @param c la classe à analyser
4       * @exception RuntimeException si la vérification échoue (le
5       * message explique l'échec).
6       */
7      void verifier(Class<?> c);
8
9      /** Obtenir l'objectif de cette propriété.
10      * @return l'objectif de cette propriété
```

```
11       */
12      String getTexte();
13  }
```

1.1 Écrire la classe NomEnMajuscule qui est une propriété vérifiant que l'initial du nom de la classe est une majuscule.

1.2 Écrire la classe MethodeExiste, propriété qui vérifie qu'une méthode est bien déclarée dans la classe. Le constructeur de cette propriété prendra en paramètre le nom de la méthode (du type String) et les types de ses paramètres (un tableau de `Class<?>`).

Exercice 2 : Vérificateur de propriétés

On considère maintenant l'interface `IVerificateur` ci-dessous. Sa méthode `evaluation` vérifie si une classe respecte chaque propriété d'une collection. La valeur retournée est une chaîne de caractères qui contient sur chaque ligne le texte de la propriété évaluée et le résultat de son évaluation sur la classe.

```
1  import java.util.Collection;
2
3  public interface IVerificateur {
4      String evaluation(Collection<? extends Propriete> props, Class<?> c);
5  }
```

Le listing 1 donne un exemple d'utilisation du vérificateur. La classe Exemple1 est donnée au listing 2. Le résultat de l'exécution est donné au listing 3.

Listing 1 – "Exemple d'utilisation de la classe Verificateur"

```
1  import java.util.*;
2  public class ExempleVerificateur {
3    public static void main(String[] args) {
4      Collection<Propriete> props = new ArrayList<Propriete>();
5      props.add(new NomEnMajuscule());
6      props.add(new MethodeExiste("translater", Double.class, Double.class));
7      props.add(new MethodeExiste("distance", Exemple1.class));
8      props.add(new MethodeExiste("toString"));
9      props.add(new MethodeExiste("m", String.class, Integer.class));
10
11     IVerificateur v = new Verificateur();
12     String r = v.evaluation(props, Exemple1.class);
13
14     System.out.println(r);
15   }
16 }
```

Listing 2 – "La classe Exemple1"

```
1  public class Exemple1 {
2      public void translater(Double dx, Double dy) {}
3      private void distance(Exemple1 autre) {}
4  }
```

Listing 3 – "Résultats de l'exécution de la classe ExempleVerificateur"

```
1  Est-ce que le nom commence par une majuscule ?  OUI
```

```
Est-ce que la méthode translater(java.lang.Double, java.lang.Double)
    existe ?  OUI
Est-ce que la méthode distance(Exemple1) existe ?  OUI
Est-ce que la méthode toString() existe ?  java.lang.NoSuchMethodException
    : Exemple1.toString()
Est-ce que la méthode m(java.lang.String, java.lang.Integer) existe ?
    java.lang.NoSuchMethodException: Exemple1.m(java.lang.String, java.lang
    .Integer)
Fin.
```

2.1 Expliquer succinctement pourquoi il faut utiliser `Collection<? extends Propriete>` et pas seulement `Collection<Propriete>` pour le type du paramètre de `evaluation`.

2.2 Écrire la réalisation `Verificateur` de l'interface `IVerificateur`.

Exercice 3 : Propriétés en XML

On se propose de définir un fichier XML pour représenter les propriétés. L'objectif sera par exemple de pouvoir paramétrer une application de vérification grâce à un tel fichier XML. La DTD de ces fichiers XML est donnée ci-dessous.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!ELEMENT proprietes ((nom-majuscule|meth-def)*)>
<!ELEMENT nom-majuscule EMPTY>
<!ELEMENT meth-def (param*)>
<!ATTLIST meth-def
    nom CDATA #REQUIRED >

<!ELEMENT param EMPTY>
<!ATTLIST param
    type CDATA #REQUIRED >
```

Le fichier XML du listing 4 est conforme à la DTD précédente. Il correspond à l'exemple du listing 1.

Listing 4 – Fichier de propriétés en XML

```
<?xml version="1.0" encoding="ISO-8859-1"  standalone="no"?>

<!DOCTYPE proprietes SYSTEM "proprietes.dtd">

<proprietes>
    <nom-majuscule />
    <meth-def nom="translater">
        <param type="java.lang.Double" />
        <param type="java.lang.Double" />
    </meth-def>
    <meth-def nom="distance">
        <param type="Exemple1" />
    </meth-def>
    <meth-def nom="toString" />
    <meth-def nom="m">
        <param type="java.lang.String" />
        <param type="java.lang.Integer" />
    </meth-def>
</proprietes>
```

3.1 On souhaite utiliser SaX pour lire le fichier XML et ajouter toutes les propriétés qu'il contient dans une collection de propriétés. Compléter le code donné au listing 5. Il s'agit en fait de compléter la classe `ChargeurXMLHandler`.

Listing 5 – Squelette de la classe ChargeurXML

```
1  import java.io.*;
2  import javax.xml.parsers.*;
3  import org.xml.sax.*;
4  import org.xml.sax.helpers.*;
5  import java.util.*;
6
7  public class ChargeurXML {
8
9      public void charger(InputStream in, Collection<? super Propriete> list)
10         throws SAXException, ParserConfigurationException, java.io.IOException
11     {
12         SAXParserFactory spf = SAXParserFactory.newInstance();
13         XMLReader xmlReader = spf.newSAXParser().getXMLReader();
14         xmlReader.setFeature("http://xml.org/sax/features/validation", true);
15         ChargeurXMLHandler handler = new ChargeurXMLHandler(list);
16         xmlReader.setContentHandler(handler);
17         xmlReader.parse(new InputSource(in));
18     }
19 }
20
21 class ChargeurXMLHandler extends DefaultHandler {
22
23     private Collection<? super Propriete> proprietes;
24
25         // à compléter
26
27
28     public ChargeurXMLHandler(Collection<? super Propriete> proprietes) {
29         this.proprietes = proprietes;
30     }
31
32         // à compléter
33
34 }
```

3.2 Expliquer pourquoi le type `Collection<? super Propriete>` est utilisé.

Exercice 4 : Utiliser une classe existante comme modèle

On souhaite pouvoir utiliser une classe comme modèle pour construire le fichier de propriétés. Plus précisément, étant donnée une classe Java, on souhaite engendrer un fichier de propriétés XML qui vérifie que chaque méthode de la classe modèle est bien présente dans la classe à tester.

Par exemple, les méthodes déclarées dans la classe Modele1 (listing 6) correspondent aux propriétés décrites dans le fichier exemple1.xml (listing 4).

Listing 6 – La classe Modele1

```
1  public abstract class Modele1 {
2      abstract void translater(Double dx, Double dy);
```

```
3       abstract void distance(Exemple1 autre);
4       abstract public String toString();
5       abstract void m(String s, Integer i);
6   }
```

Compléter le code de la classe ProprietesBuilder (listing 7) qui utilise JDom pour construire le fichier de propriété.

Listing 7 – Squelette de la classe ProprietesBuilder

```
1   import java.io.*;
2   import java.util.*;
3   import org.jdom.*;
4   import org.jdom.output.*;
5
6   public class ProprietesBuilder {
7
8           // à compléter
9
10
11      public void genererXML(OutputStream out, Class<?> modele) {
12
13          // à compléter
14
15      }
16
17      public void ecrire(Document document, OutputStream out) {
18          try {
19              XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
20              sortie.output(document, out);
21          } catch (java.io.IOException e) {
22              throw new RuntimeException("Erreur␣sur␣écriture␣:", e);
23          }
24      }
25
26  }
```

Exercice 5 : Interface graphique simplifiée

On veut proposer une interface graphique en Swing qui permet à l'utilisateur de donner le nom de la classe à vérifier et le nom de la classe servant de modèle. Un bouton *vérifier* lance la vérification et affiche les résultats dans l'onglet *résultats*. Un bouton *quitter* permet de quitter l'application. Le squelette de la classe correspondante est donnée au listing 8.

5.1 Dessiner l'apparence de l'interface graphique lorsque l'application du listing 8 est lancée.

5.2 Compléter l'application de manière à ce que les deux boutons deviennent actifs.

Listing 8 – Squelette de la classe VerificateurSwing

```
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 VerificateurSwing extends JFrame {
8
9      private JTextField nomClasse = new JTextField("Exemple1", 20);
10     private JTextField modeleClasse = new JTextField("Modele1", 20);
11     private JButton bVerifier = new JButton("Verifier");
12     private JButton bQuitter = new JButton("Quitter");
13     private JTextArea resultats  = new JTextArea(10, 40);
14
15     public VerificateurSwing() {
16         super("Vérificateur");
17         JTabbedPane onglets = new JTabbedPane();
18         this.getContentPane().add(onglets);
19
20         JPanel parametres = new JPanel(new BorderLayout());
21         onglets.addTab("paramètres", parametres);
22
23         // Construire la partie supérieure
24         JPanel saisies = new JPanel(new GridLayout(2, 2));
25         saisies.add(new JLabel("Classe à vérifier : "));
26         saisies.add(nomClasse);
27         saisies.add(new JLabel("Modèle de classe  : "));
28         saisies.add(modeleClasse);
29         parametres.add(saisies, BorderLayout.NORTH);
30
31         // Construire les boutons de commandes inférieurs
32         JPanel boutons = new JPanel();
33         boutons.setLayout(new FlowLayout());
34         boutons.add(bVerifier);
35         boutons.add(bQuitter);
36         parametres.add(boutons, BorderLayout.SOUTH);
37
38         // Construire l'onglet résultat
39         onglets.addTab("résultats", new JScrollPane(resultats));
40
41         // à compléter
42
43         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
44         this.pack();
45         this.setVisible(true);
46     }
47
48
49         // à compléter
50
51
52     public static void main(String[] args) {
53         EventQueue.invokeLater(new Runnable() {
54             public void run() {
55                 new VerificateurSwing();
56             }
57         });
58     }
59 }
```