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.

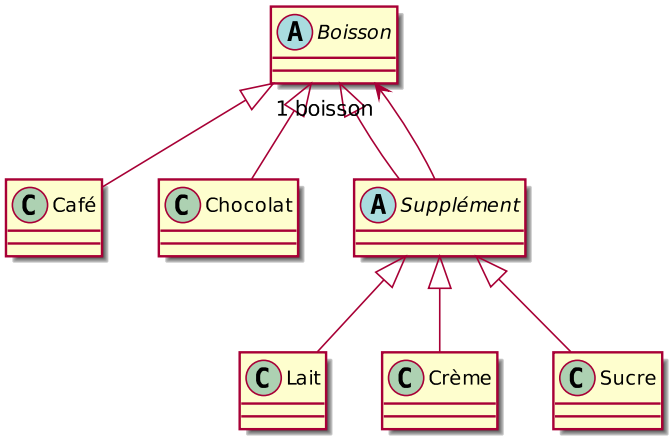
Barème indicatif :

exercice	1	2	3.1	3.2	3.3
points	4	4	4	4	4

Exercice 1 : Patron de conception Décorateur

On considère deux types de boisson, café et chocolat, pour lesquelles ont peut ajouter des suppléments : sucre, lait, crème, etc. On s'intéresse au prix d'une telle boisson qui est fonction de la boisson choisie mais aussi des suppléments sélectionnés.

1.1 Ce système peut se modéliser en utilisant le patron de conception Décorateur. Proposer le digramme de classe correspondant en indiquant les principaux éléments du patron Décorateur.
Solution : Les principaux éléments sont les types de base, ici Café et Chocolat, puis les Décorateurs (les suppléments) qui viennent compléter un composant (ici une Boisson) qui généralise les types de base et les suppléments.



1.2 Expliquer comment créer un café avec du sucre et de la crème.
Solution : new Crème(new Sucre(new Café()))

1.3 Indiquer le principal intérêt du patron de conception Décorateur.

Solution : Ajouter de nouvelles propriétés (fonctionnelles) à un objet existant.

Exercice 2 Écrire un programme qui liste toutes les méthodes publiques surchargées pour une classe dont le nom est donné en argument de la ligne de commande. Le programme n'aura pas à être robuste. Surcharge signifie qu'il existe plusieurs méthodes de même nom. On ne s'intéresse ici qu'à la surcharge entre méthodes publiques.

Le résultat de la commande `java MethodesSurchargees java.util.AbstractCollection` est donné au listing 1.

Listing 1 – Résultat de java MethodesSurchargees java.util.AbstractCollection

```
toArray :
  - public java.lang.Object[] java.util.AbstractCollection.toArray(java.lang.Object[])
  - public java.lang.Object[] java.util.AbstractCollection.toArray()
wait :
  - public final void java.lang.Object.wait(long,int) throws java.lang.InterruptedException
  - public final native void java.lang.Object.wait(long) throws java.lang.InterruptedException
  - public final void java.lang.Object.wait() throws java.lang.InterruptedException
```

Solution : La solution est donnée au listing 2

Listing 2 – Solution de l'exercice 2.

```
import java.util.*;
import java.lang.reflect.*;

class MethodesSurchargees {

        public static void main(String[] args) throws Exception {
                assert args.length == 1;
                Map<String, Set<Method>> methodes = new TreeMap<>();
                Class<?> laClasse = Class.forName(args[0]);
                for (Method m : laClasse.getMethods()) {
                        String nom = m.getName();
                        Set<Method> ens = methodes.get(nom);
                        if (ens == null) {
                                ens = new HashSet<Method>();
                                methodes.put(nom, ens);
                        }
                        ens.add(m);
                }

                for (Map.Entry<String, Set<Method>> entry : methodes.entrySet()) {
                        if (entry.getValue().size() > 1) {
                                System.out.println(entry.getKey() + " :");
                                for (Method m: entry.getValue()) {
                                        System.out.println("  - " + m);
                                }
                        }
                }
        }

}
```

Exercice 3 : Mesures de températures

On considère la DTD donnée au listing 3 pour décrire des mesures. Nous nous en servirons pour représenter des températures.

Listing 3 – DTD pour des mesures

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2
3  <!ELEMENT mesures (mesure*)>
4  <!ATTLIST mesures nature CDATA #REQUIRED>
5
6  <!ELEMENT mesure (#PCDATA)>
7  <!ATTLIST mesure date CDATA #IMPLIED>
```

3.1 On considère le document XML du listing 4.

Listing 4 – Exemple de fichier XML

```
1  <?xml version="1.0" encoding="ISO-8859-1"  standalone="no"?>
2  <!DOCTYPE mesures SYSTEM "mesures.dtd">
3  <mesures>
4      <nature valeur=températures />
5      <!-- from http://www.meteociel.fr/ -->
6      <mesure>
7          <date>2015/11/01</date>21.7
8      <mesure>
9          <date>2015/11/02</date>18.7
10     <mesure>
11         <date>2015/11/03</date>19.4
12 </mesures>
```

3.1.1 Expliquer pourquoi ce document XML est mal formé.

Solution :

1. la valeur "température" de l'attribut "valeur" de l'élément "nature" n'a pas de délimiteur (guillemet ou apostrophe).
2. Les éléments "mesure" ne sont pas fermés.

3.1.2 En supposant les erreurs précédentes corrigées, expliquer pourquoi ce document XML n'est pas valide.

Solution :

1. "nature" n'est pas un élément mais un attribut de l'élément "mesures".
2. la date n'est pas un élément mais un attribut

3.1.3 Corriger le document XML pour qu'il soit valide.

Solution : Voir le listing 5.

3.2 On considère l'interface Generateur (listing 6). Elle définit une méthode qui prend en paramètre :

Listing 5 – Le fichier de 4 bien formé et valide

```
<?xml version="1.0" encoding="ISO-8859-1"  standalone="no"?>
<!DOCTYPE mesures SYSTEM "mesures.dtd">
<mesures nature="températures">
    <!-- from http://www.meteociel.fr/ -->
    <mesure date="2015/11/01">21.7</mesure>
    <mesure date="2015/11/02">18.7</mesure>
    <mesure date="2015/11/03">19.4</mesure>
</mesures>
```

Listing 6 – L'interface Generateur

```
import java.util.Map;
import java.io.OutputStream;

public interface Generateur {
        void ecrire(Map<String, Double> mesures, String nature, OutputStream out);
}
```

1. un tableau associatif (Map) dont la clé est une chaîne de caractères correspondant à une date et la valeur un réel correspondant à une mesure,
2. la nature des mesures faites (par exemple, température, pression, etc.),
3. le flot de sortie sur lequel écrire les résultats.

Écrire une réalisation GenerateurXML de l'interface Generateur qui enregistre les mesures au format XML en respectant la DTD du listing 3. On suppose que la méthode suivante existe. Elle sait écrire un document JDom sur un flot de sortie.

```
public static void ecrire(Document document, OutputStream out);
```

Solution :

```
import java.util.Map;
import java.io.OutputStream;
import org.jdom2.*;

public class GenerateurXML implements Generateur {

    public void ecrire(Map<String, Double> mesures, String nature, OutputStream out) {
        Element racine = new Element("mesures");
        racine.setAttribute("nature", nature);
        for (Map.Entry<String, Double> entry : mesures.entrySet()) {
            Element mesure = new Element("mesure");
            mesure.setAttribute("date", entry.getKey());
            mesure.setText("" + entry.getValue());
            racine.addContent(mesure);
        }
        Document document = new Document(racine, new DocType("mesures",
                    "mesures.dtd"));
        try {
            JDOMTools.ecrire(document, out);
        } catch (Exception e) {
            System.out.println(e);
```

```
22          }
23      }
24
25  }
```

3.3 On veut définir une application graphique avec Swing pour saisir des mesures. Par défaut, on considèrera qu'il s'agit de températures.

3.3.1 Compléter le listing 7 pour obtenir l'application présentée à la figure 1.

Solution :

```
1       public MesuresSwing(final String nature, final Generateur generateur) {
2           this.fenetre = new JFrame(nature);
3           fenetre.setLayout(new BorderLayout());
4
5           JPanel commandes = new JPanel(new GridLayout(2, 2));
6           fenetre.add(commandes, BorderLayout.NORTH);
7           commandes.add(new JLabel("date"));
8           commandes.add(new JLabel(nature));
9           commandes.add(date);
10          commandes.add(mesure);
11
12          JPanel boutons = new JPanel(new FlowLayout());
13          fenetre.add(boutons, BorderLayout.SOUTH);
14          boutons.add(bValider);
15          boutons.add(bQuitter);
16
17          fenetre.pack();
18          fenetre.setVisible(true);
19      }
```

3.3.2 Quand l'utilisateur clique sur le bouton Valider, la date et la mesure saisies doivent être ajoutées dans le tableau associatif appelé `mesures`.

Solution :

```
1           bValider.addActionListener(new ActionListener() {
2               public void actionPerformed(ActionEvent ev) {
3                   try {
4                       mesures.put(date.getText(),
5                               Double.parseDouble(mesure.getText()));
6                   } catch (NumberFormatException e) {
7                       System.out.println(e);
8                   }
9               }
10          });
```

3.3.3 Quand l'utilisateur clique sur le bouton Quitter, toutes les données enregistrées dans le tableau associatif sont écrites dans un fichier en utilisant le générateur fourni en paramètre du constructeur de la classe MesuresSwing. Le nom du fichier pourra être températures.xml si la nature des mesures est température.

Solution :

```
1           bQuitter.addActionListener(new ActionListener() {
2               public void actionPerformed(ActionEvent ev) {
3                   try {
4                       generateur.ecrire(mesures, nature,
```

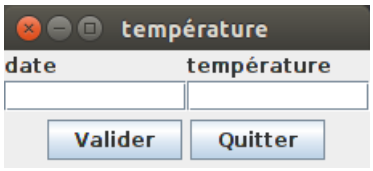


FIGURE 1 – IHM attendu pour MesuresSwing

```
                            new FileOutputStream(nature + "s.xml"));
                    // fenetre.dispose();
                    System.exit(0);
                } catch (Exception e) {
                    System.out.println(e);
                }
            }
        });
```

3.3.4 Dans le `main` de `MesuresSwing`, on utilise directement `GenerateurXML`. Expliquer comment il faudrait modifier le programme principal pour permettre à l'utilisateur de fournir le générateur à utiliser comme deuxième argument de la ligne de commande.

Solution :

```
String nature = args.length == 0 ? "température" : args[0];
String generateurName = args.length > 1 ? args[1] : "GenerateurXML";
Class<?> genClass = Class.forName(generateurName);
Generateur generateur = (Generateur) genClass.newInstance();
```

Listing 7 – Le squelette de MesuresSwing

```
1  import java.util.*;
2  import java.io.*;
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6
7  public class MesuresSwing {
8
9          final private JFrame fenetre;
10         final private JTextField date = new JTextField(10);
11         final private JTextField mesure = new JTextField(10);
12         final private JButton bQuitter = new JButton("Quitter");
13         final private JButton bValider = new JButton("Valider");
14         final private Map<String, Double> mesures = new HashMap<>();
15
16         public MesuresSwing(final String nature, final Generateur generateur) {
17
18                 fenetre.pack();
19                 fenetre.setVisible(true);
20         }
21
22         public static void main(String[] args) {
23                 String nature = args.length == 0 ? "température" : args[0];
24                 Generateur generateur = new GenerateurXML();
25
26                 EventQueue.invokeLater(new Runnable() {
27                                 public void run() {
28                                         new MesuresSwing(nature, generateur);
29                                 }
30                         }
31                 );
32         }
33
34 }
```