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	4	5
points	4	4	4	4	4

Exercice 1 : Lister les fabriques statiques d'une classe

Écrire une réalisation de l'interface `Listeur` qui retourne la liste de toutes les fabriques statiques d'une classe dont le nom a été donné.

```
1  import java.util.*;
2  import java.lang.reflect.*;
3
4  interface Listeur {
5      java.util.List<Method> getMethodes(String nomClasse) throws Exception;
6  }
```

Une fabrique statique d'une classe est une méthode :

1. qui est de classe (mot-clé **`static`**),
2. qui a pour type de retour cette classe,
3. dont aucun paramètre n'a pour type cette classe,
4. qui a un droit d'accès quelconque.

On ne traitera aucune exception.

Voici le résultat obtenu, si on affiche le contenu de la liste retournée par getMethodes("C") avec la classe C ci-après.

```
1  Fabriques statiques de C :
2   - private static C C.p(int)
3   - public static C C.q(double)
```

```
1  class C {
2      public C m(double d) {return null;}
3      public static C n(C autre) {return null;}
4      private static C p(int n) {return null;}
5      public static C q(double d) {return null;}
6      public static Object s(double d) {return null;}
7  }
```

Solution :

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

class ListeurFabriquesStatiques implements Listeur {

    public List<Method> getMethodes(String nomClasse) throws Exception {
        List<Method> fabriques = new ArrayList<>();
        Class<?> classe  = Class.forName(nomClasse);
        for (Method m : classe.getDeclaredMethods()) {
            if (Modifier.isStatic(m.getModifiers())
                    && m.getReturnType().equals(classe))
            {
                // Déterminer si classe est le type d'un des paramètres
                boolean ok = true;
                for (Class<?> type : m.getParameterTypes()) {
                    if (type.equals(classe)) {
                        ok = false;
                        break;
                    }
                }

                if (ok) {
                    fabriques.add(m);
                }

            }
        }
        return fabriques;
    }

}
```

Questions à réponses courtes (QRC)

Exercice 2 : Questionnaire et questions à réponses courtes

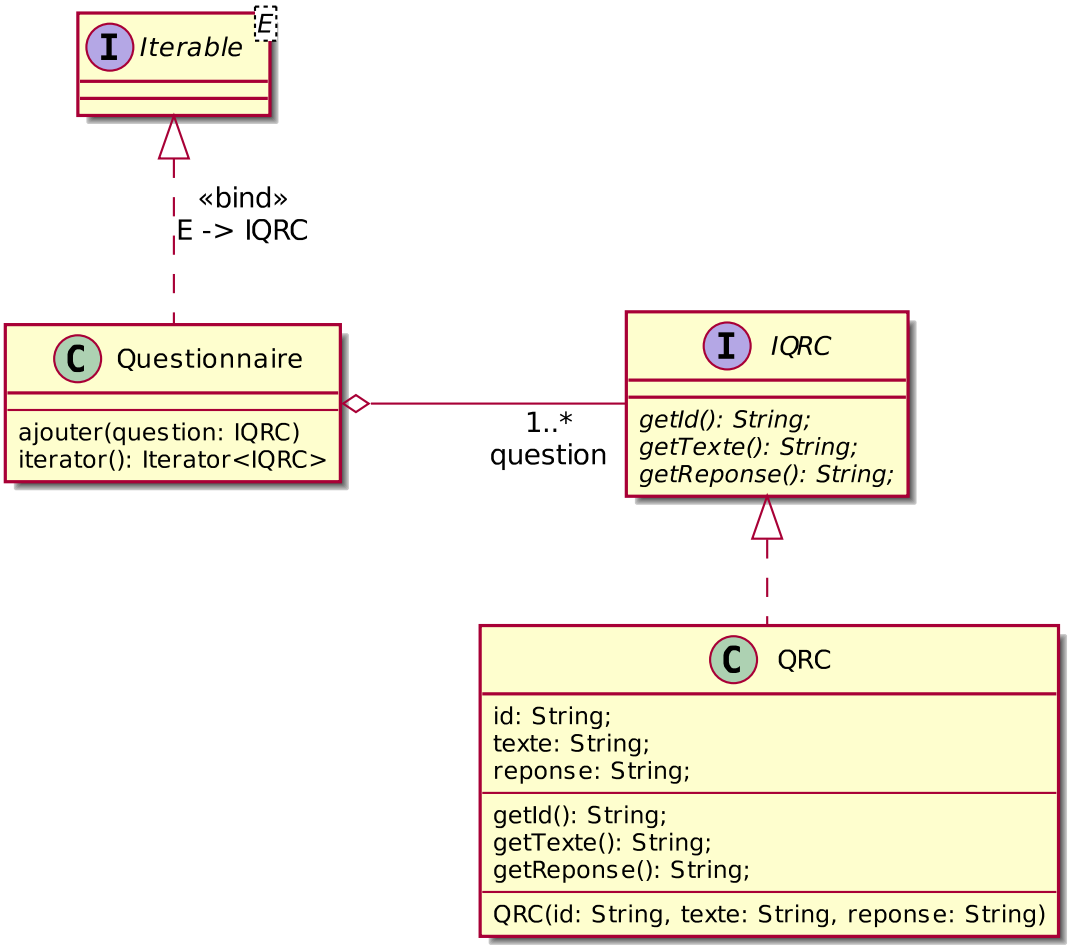
On souhaite réaliser un questionnaire composé de questions à réponses courtes (QRC). Chaque question est caractérisée par un identifiant unique (id), le texte de la question (texte) et la réponse attendue (réponse).

On considère l'interface `IQRC` (listing 1) qui spécifie de telles questions, la classe `Questionnaire` qui correspond à une liste de questions et la classe `QRC` qui réalise l'interface `IQRC` et définit un constructeur naturel qui prend en paramètre l'identifiant, le texte et la réponse à une question.

Un questionnaire sera défini comme une liste de questions (`IQRC`). On veut pouvoir itérer sur toutes ses questions avec le `foreach` de Java comme dans l'exemple du listing 2.

2.1. Donner le diagramme de classe qui fait apparaître `IQRC`, `QRC` et `Questionnaire`.

Solution :



2.2. Écrire la classe `Questionnaire` (en Java). Ne définir que ce qui est nécessaire à l'exécution du listing 2.

Solution :

```java
import java.util.*;

public class Questionnaire implements Iterable<IQRC> {

    private List<IQRC> questions;

    public Questionnaire() {
        this.questions = new ArrayList<>();
    }

    public void ajouter(IQRC question) {
        this.questions.add(question);
    }

    public Iterator<IQRC> iterator() {
        return this.questions.iterator();
    }

}
```

Listing 1 – L'interface `IQRC`

```java
public interface IQRC {
    String getId();
    String getTexte();
    String getReponse();
}
```

Listing 2 – La classe ExempleQuestionnaire

```java
/**
 * Exemple d'utilisation d'un questionnaire.
 */
public class ExempleQuestionnaire {
    public static void main(String[] args) {
        // Création d'un questionnaire
        Questionnaire qObjet = new Questionnaire();
        qObjet.ajouter(new QRC("Q1", "Accessible à tous ?", "public"));
        qObjet.ajouter(new QRC("Q2", "Non modifiable ?", "final"));
        qObjet.ajouter(new QRC("Q3", "2 * 3 + 4 ?", "10"));

        // Afficher le texte des questions
        for (IQRC question : qObjet) {
            System.out.println(question.getTexte());
        }
    }
}
```

Exercice 3 : IHM de saisie d'un questionnaire

On souhaite proposer une interface graphique pour aider l'utilisateur à saisir son questionnaire.

L'interface homme/machine (IHM) proposée à l'utilisateur doit correspondre à celle de la figure 1. Elle permet de saisir l'identifiant, le texte et la réponse attendue d'une question. Trois boutons permettent d'ajouter la question au questionnaire, d'annuler les informations saisies dans les champs (tous les champs sont vidés) et quitter l'application. La technologie choisie est Swing. L'écriture de la classe implantant cette IHM a été commencée (listing 3). Le constructeur de cette classe `SaisirQuestionnaire` prend en paramètre le questionnaire à compléter.

3.1. Compléter le listing 3 pour respecter l'apparence choisie pour la saisie utilisateur.

Solution : On ajoute en ligne 20 le code suivant.

```
1          // Construire la vue
2          // Construire la zone de saisies
3          Container contenu = fenetre.getContentPane();
4          contenu.setLayout(new BorderLayout());
5          JPanel zoneSaisie = new JPanel(new GridLayout(3, 2));
6          contenu.add(zoneSaisie, BorderLayout.CENTER);
7          zoneSaisie.add(new JLabel("Identifiant de la question : "));
8          zoneSaisie.add(id);
9          zoneSaisie.add(new JLabel("Texte de la question"));
10         zoneSaisie.add(texte);
11         zoneSaisie.add(new JLabel("Réponse à la question : "));
12         zoneSaisie.add(reponse);
13
14         // Définir les boutons
15         JPanel boutons = new JPanel(new FlowLayout());
16         contenu.add(boutons, BorderLayout.SOUTH);
17         boutons.add(enregistrer);
18         boutons.add(annuler);
19         boutons.add(quitter);
```

3.2. Rendre actifs les boutons de cette IHM.

Solution : On ajoute à la suite du code de la question précédente le code suivant.

```
1          // Définir les réactions
2          fenetre.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
3
4          quitter.addActionListener(new ActionListener() {
5              public void actionPerformed(ActionEvent ev) {
6                  fenetre.dispose();
7              }
8          });
9
10         annuler.addActionListener(new ActionListener() {
11             public void actionPerformed(ActionEvent ev) {
12                 id.setText("");
13                 texte.setText("");
14                 reponse.setText("");
15             }
16         });
17
18         enregistrer.addActionListener(new ActionListener() {
19             public void actionPerformed(ActionEvent ev) {
20                 String qId = id.getText();
21                 String qTexte = texte.getText();
22                 String qReponse = reponse.getText();
23                 QRC nouvelle = new QRC(qId, qTexte, qReponse);
24                 questionnaire.ajouter(nouvelle);
```

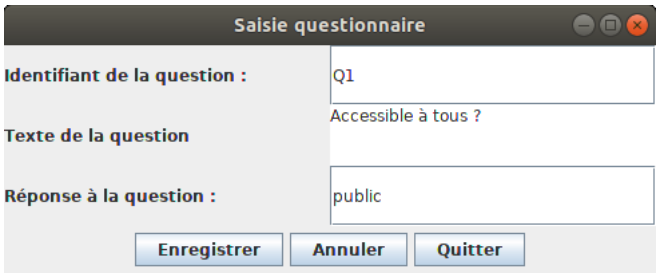


FIGURE 1 – IHM souhaitée pour saisir les notes

```
25              }
26          });
```

Listing 3 – Squelette de la classe SaisirQuestionnaire

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  public class SaisirQuestionnaire {
6
7      private JFrame fenetre = new JFrame("Saisie questionnaire");
8      private Questionnaire questionnaire;
9      private JTextField id = new JTextField(20);
10     private JTextArea texte = new JTextArea(3, 20);
11     private JTextField reponse = new JTextField(20);
12
13     private JButton enregistrer = new JButton("Enregistrer");
14     private JButton annuler = new JButton("Annuler");
15     private JButton quitter = new JButton("Quitter");
16
17     public SaisirQuestionnaire(Questionnaire questionnaire) {
18         this.questionnaire = questionnaire;
19
20
21
22         fenetre.pack();
23         fenetre.setVisible(true);
24     }
25
26
27 }
```

Exercice 4 : Sérialisation en XML

On souhaite sérialiser en XML un questionnaire. On utilise l'interface JDOM le faire. Une DTD a été définie (listing 4). Le listing 5, document XML valide pour cette DTD, correspond au questionnaire construit dans l'exemple du listing 2.

4.1. Compléter la classe `QuestionnaireUtil` du listing 6 pour réaliser cette sérialisation.

Solution :

```java
import org.jdom.*;
import org.jdom.output.*;

public class QuestionnaireUtil {

    public static void sauverXML(Questionnaire questionnaire, java.io.Writer out)
            throws java.io.IOException
    {
        Element questionnaireElt = new Element("questionnaire");
        for (IQRC question : questionnaire) {
            Element questionElt = new Element("question");
            questionnaireElt.addContent(questionElt);
            questionElt.setAttribute("type", "qrc");
            questionElt.setAttribute("id", question.getId());

            // Créer le texte
            Element texteElt = new Element("texte");
            questionElt.addContent(texteElt);
            texteElt.setText(question.getTexte());

            // Créer la réponse
            Element reponseElt = new Element("reponse");
            questionElt.addContent(reponseElt);
            reponseElt.setText(question.getReponse());
        }

        DocType docType = new DocType("questionnaire", "questionnaire.dtd");
        Document document = new Document(questionnaireElt, docType);

        // Production du fichier XML
        XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
        sortie.output(document, out);
}   }
```

Listing 4 – La DTD questionnaire.dtd

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT questionnaire (question+)>
<!ELEMENT question (texte, reponse)>
<!ATTLIST question
    type CDATA #REQUIRED
    id ID #REQUIRED
>
<!ELEMENT texte (#PCDATA)>
<!ELEMENT reponse (#PCDATA)>
```

Exercice 5 : Amélioration des QRC

Nous souhaitons apporter deux améliorations à nos QRC.

1. Une QRC peut avoir plusieurs réponses justes. La réponse attendue n'est donc pas unique.
2. Pour chaque QRC, il y a des erreurs qui reviennent souvent. Dans ce cas, on souhaite pouvoir expliquer à l'utilisateur l'erreur qu'il a commise quand il a mal répondu. Par exemple,

Listing 5 – Le document XML correspondant au questionnaire du listing 2

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

<questionnaire>
  <question type="qrc" id="Q1">
    <texte>Accessible à tous ?</texte>
    <reponse>public</reponse>
  </question>
  <question type="qrc" id="Q2">
    <texte>Non modifiable ?</texte>
    <reponse>final</reponse>
  </question>
  <question type="qrc" id="Q3">
    <texte>2 * 3 + 4 ?</texte>
    <reponse>10</reponse>
  </question>
</questionnaire>
```

Listing 6 – Squelette de la classe QuestionnaireUtil

```java
import org.jdom.*;
import org.jdom.output.*;

public class QuestionnaireUtil {

    public static void sauverXML(Questionnaire questionnaire, java.io.Writer out)
            throws java.io.IOException
    {

        DocType docType = new DocType("questionnaire", "questionnaire.dtd");
        Document document = new Document(........., docType);

        // Production du fichier XML
        XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
        sortie.output(document, out);
}   }
```

pour la question Q3, on peut lui dire « attention, la multiplication est prioritaire sur l'addition » s'il répond « 14 ». Bien sûr, il y autant de messages explicatifs que de mauvaises réponses classiques.

5.1. Indiquer comment compléter l'interface `IQRC` et la classe `QRC`. On donnera simplement les nouvelles méthodes et attributs avec leurs signatures et types.

Solution : Si plusieurs réponses sont possibles, il faut remplacer l'attribut `String reponse` par `Set<String> reponse`.

On utilise `Set<String>` car une même réponse ne peut apparaître qu'une seule fois et qu'il n'y a pas d'ordre sur les réponses.

La méthodes getReponse() peut devenir getReponses() mais il serait plus logique de définir une méthode estReponseValide(String) qui retourne un booléen. On garde ainsi les réponses cachées. Dans la version originale, il aurait été préférable de faire la même chose !

Pour donner les indications, on peut ajouter un attribut de type Map<String, String>. Le premier type String est la clé, la réponse erronée, et le second, l'explication associée.

On peut ajouter deux méthodes :

```
1 void  ajouterAide(String reponse, String explication);
2 String getIndication(String reponse);
```

5.2. Proposer une extension de la DTD qui prenne en compte ces améliorations et donner le document XML pour un questionnaire qui ne contient que la question Q3 avec l'indication ci-dessus pour la réponse 14.

Solution : Voici la nouvelle DTD :

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <!ELEMENT questionnaire (question+)>
4  <!ELEMENT question (texte, reponse+, explication*)>
5  <!ATTLIST question
6      type CDATA #REQUIRED
7      id ID #REQUIRED
8  >
9  <!ELEMENT texte (#PCDATA)>
10 <!ELEMENT reponse (#PCDATA)>
11 <!ELEMENT explication (reponse, message)>
12 <!ELEMENT message (#PCDATA)>
```

Voici le document XML pour la question Q3 :

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE questionnaire SYSTEM "questionnaire.dtd">
3
4  <questionnaire>
5    <question type="qrc" id="Q3">
6      <texte>2 * 3 + 4 ?</texte>
7      <reponse>10</reponse>
8      <explication>
9      <reponse>14</reponse>
10     <message>attention, la multiplication est prioritaire sur l´addition</message>
11     </explication>
12   </question>
13 </questionnaire>
```