

QCM

TEST

**Introduction à la programmation  
orientée objet  
9/11/2017**

Nom et prénom :

STROBBE Nathan

Groupe : 2

Cochez les cases en mettant une X.

Le symbole  $\oplus$  indique que la question peut avoir zéro, une ou plusieurs bonnes réponses. Pour ces questions, cocher une bonne réponse apporte des points positifs ; cocher une mauvaise réponse peut apporter des points négatifs.

Dans tout le code, les **package** et les **import** sont censés être correctement déclarés. Toute classe est supposée être dans le bon package, dans le bon fichier, avec les bons import.

**Question 1** Quel est l'accès le plus restrictif par lequel on pourrait remplacer \_\_\_\_ ?

```
package toto;

public class Toto {
    ____ String doSomething() {
        // does something
    }
}
```

☐ package-private (default)☒ private☐ public

**Question 2** Quel est l'accès le plus restrictif par lequel on pourrait remplacer \_\_\_\_ ?

```
package toto;

public class Toto {
    ____ String doSomething() {
        // does something
    }
}
```

```
package toto;

class Foobar {
    private Toto toto = new Toto();

    private String doSomething() {
        toto.doSomething();
    }
}
```

☐ private☐ public☒ package-private (default)

**Question 3** Quel est l'accès le plus restrictif par lequel on pourrait remplacer \_\_\_\_ ?

```
package toto;

public class Toto {
    ____ String doSomething() {
        // does something
    }
}
```

```
package foobar;

class Foobar {
    private Toto toto = new Toto();

    private String doSomething() {
        toto.doSomething();
    }
}
```

☐ package-private (default)☐ private☒ public



**Question 4** ⊕ Quelles affirmations s'appliquent aux maps (la classe `HashMap`) :

- ☐ sont initialisés par, eg, `new Person(14)`
- ☒ peuvent stocker des doublons (deux fois le même élément)
- ☒ sont dans le package `java.util`
- ☐ l'ordre de stockage des éléments est bien défini
- ☐ sont déclarés par, eg, `Person[] p`
- ☐ peuvent stocker des primitifs, eg, `double`
- ☒ le nombre d'éléments est donné par `.size()`
- ☐ sont de taille fixe
- ☐ sont indexés exclusivement par des entiers non-négatifs

**Question 5** ⊕ Quelles affirmations s'appliquent aux listes (la classe `ArrayList`) :

- ☒ l'ordre de stockage des éléments est bien défini
- ☒ sont dans le package `java.util`
- ☐ sont de taille fixe
- ☐ peuvent stocker des primitifs, eg, `double`
- ☒ le nombre d'éléments est donné par `.size()`
- ☐ sont déclarés par, eg, `Person[] p`
- ☒ peuvent stocker des doublons (deux fois le même élément)
- ☐ sont créés par, eg, `new Person(14)`
- ☒ sont indexés exclusivement par des entiers non-négatifs

**Question 6** Le code

```
class Protagonist {  
    private String name = "Fred";  
  
    private void print() {  
        System.out.println("My name is " + name);  
    }  
  
    public static void main(String... args) {  
        print();  
    }  
}
```

ne compile pas. Le compilateur dit

```
Protagonist.java:11: error: non-static method print() cannot be referenced from a static context  
    print();  
    ^
```

1 error

Deux possibilités se présentent, laquelle est le meilleur choix :

- ☐ Modifier `main` : `new Protagonist().print()`
- ☒ Déclarer : `private static void print()`



**Question 7** Justifiez votre réponse à la question précédente.

☐ 0 ☐ 1 ☐ 3

**Question 8** ⊕ Lesquelles des expressions déclarent, construisent et initialisent un tableau ?

☐ `int myList = {4,9,7,0};`

☐ `int[] myList = {"1", "2", "3"};`

☐ `int[] myList = (5, 8, 2);`

☒ `int myList[] = {4, 3, 7};`

**Question 9** ⊕ On souhaite écrire une application dans un package `foobar`. Il est obligatoire de...

☐ mettre tous les fichiers sources dans un même `.jar`

☐ compiler les fichiers avec la commande `javac -package foobar *.java`

☐ écrire `import foobar.*;` en début de tous les fichiers source

☒ écrire `package foobar;` en début de tous les fichiers source

**Question 10** ⊕ Nous souhaitons compiler le code source :

```
package main;

class Main {
    private Toto toto;

    public static void main(String... args) {
        // some code
    }
}
```

```
package main;

class Toto {}
```

Le code est dans le dossier `myproject/src/main`, et on aimerait que le bytecode soit généré dans le dossier `myproject/bin`. Tout en restant dans `myproject`, lesquelles des commandes feraient l'affaire :

☒ `javac -d bin src/main/*.java`

☒ `javac -d bin src/main/Main.java src/main/Toto.java`

☐ `javac -d bin src/main/Main src/main/Toto`

☐ `javac src/main/*.java`

☐ `javac -d bin *.java`



**Question 11** ⊕ Pour exécuter le code de la question précédente, ça serait :

☐ `java -cp bin Main`

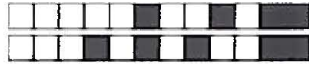
☒ `java -cp bin main.Main`

☐ `java main.Main`

☐ `java -cp bin Main.java`

☐ `java -cp bin Main.class`

☐ `java bin/Main.class`



**Question 12** Soit le code à Polytech'Groland pour stocker des notes, afficher les notes, et une classe **Main** de mise en exécution :

```
package admin;

class Marks {
    // this is voodoo, but it correctly initializes marks
    private final Map<String, int[]> marks
        = new HashMap<String, int[]>(){
        put("Barney", new int[]{12, 8});
        put("Fred", new int[]{7, 9});
        put("Wilma", new int[]{15, 13});
    };

    int[] getMarks(String student) {
        return marks.get(student);
    }

    Set<String> getStudents() {
        return marks.keySet();
    }
}
```

```
package admin.sploit;

public class Sploit {
    // code to be supplied for the following method
    public void haxMyMarks
}
```

```
package admin;

class Consuler {
    private final Marks marks;

    Consuler(Marks marks) {
        this.marks = marks;
    }

    void displayMarks(String student) {
        System.out.print(student + ": ");
        for (int m : marks.getMarks(student)) {
            System.out.print(m + " ");
        }
        System.out.println();
    }
}
```

```
package admin;

public class Main {
    public static void main(String... args) {
        Marks marks = new Marks();
        Consuler consuler = new Consuler(marks);
        // administration consults student marks
        marks.getStudents().forEach(s
            -> consuler.displayMarks(s));
        // Wilma introduces exploit
        new Sploit().haxMyMarks(marks.getMarks("Wilma"));
        // administration consults student marks again
        marks.getStudents().forEach(s
            -> consuler.displayMarks(s));
    }
}
```

En fonctionnement normal, tout cela donne le résultat à gauche :

Barney: 12 8  
Wilma: 15 13  
Fred: 7 9

Barney: 12 8  
Wilma: 20 20  
Fred: 7 9

Hélas, une élève rusée a trouvé le moyen d'introduire du code dans la classe **Sploit** pour exploiter une faille dans le système, afin d'améliorer ses notes. Cela donne le résultat à droite.

Démontrez comment elle aurait pu arriver à ce résultat en complétant la classe **Sploit**, *sans toucher aux autres classes*.

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4



**Question 13** Soit le code de la question précédente. Quelle parade dans la classe **Marks**, et seulement dans la classe **Marks**, aurait pu éviter ce désagrément pour Polytech'Groland ?

☐ 0 ☐ 1 ☐ 2

**Question 14**  $\oplus$  Soit la déclaration :

```
private final String[] names = {"Fred"};
```

Quelles expressions sont permises dans une méthode de la même classe ?

- |   |  |
|---|--|
| <input type="checkbox"/> names = new String[1]; | <input type="checkbox"/> names[0] == "Barney";         |
| <input type="checkbox"/> names[0] = "Barney";   | <input type="checkbox"/> names = new String("Barney"); |
| <input type="checkbox"/> names = "Barney";      |  |

**Question 15**  $\oplus$  Soit la déclaration :

```
private String[] names = {"Fred"};
```

Quelles expressions sont permises dans une méthode de la même classe ?

- |  |  |
|--|--|
| <input type="checkbox"/> names = "Barney";                 | <input type="checkbox"/> names = new String("Barney"); |
| <input checked="" type="checkbox"/> names[0] = "Barney";   | <input type="checkbox"/> names[0] == "Barney";         |
| <input checked="" type="checkbox"/> names = new String[1]; |  |



**Question 16** Corrigez toutes les erreurs dans le code :

```
/**
 * Print all the values in the marks array that are
 * greater than mean.
 * @param marks in array of mark values.
 * @param mean The mean (average) mark.
 */
void printGreater(double marks, double mean) {
    for (index = 0; index <= marks.length(); index++) {
        if (marks[index] > mean) {
            System.out.println(marks[index]);
        }
    }
}
```

☐ 0 ☐ 1 ☐ 2 ☒ 3

**Question 17** ⊕ Quelles affirmations s'appliquent aux tableaux :

- |  |   |
|--|---|
| <input type="checkbox"/> le nombre d'éléments est donné par <code>.size()</code>             | <input checked="" type="checkbox"/> sont de taille fixe                                     |
| <input checked="" type="checkbox"/> peuvent stocker des doublons (deux fois le même élément) | <input checked="" type="checkbox"/> sont déclarés par, eg, <code>Person[] p</code>          |
| <input type="checkbox"/> sont créés par, eg, <code>p = new Person(14)</code>                 | <input checked="" type="checkbox"/> l'ordre de stockage des éléments est bien défini        |
| <input type="checkbox"/> sont dans le package <code>java.util</code>                         | <input checked="" type="checkbox"/> sont indexés exclusivement par des entiers non-négatifs |
| <input checked="" type="checkbox"/> peuvent stocker des primitifs, eg, <code>double</code>   |   |