STROBBE Nathan
Mark: 13.9/20 (total score: 12.8/18.5)

+75/1/23+

QCM **TEST**

Introduction à la programmation orientée objet
9/11/2017

Nom et prénom : STROBBE Nathan

Groupe : 2

Cochez les cases en mettant une ×.
Le symbole ⊕ 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
    }
}
```

0.33/0.33

- ☐ 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();
    }
}
```

0.33/0.33

- ☐ `private`
- ☒ package-private (default)
- ☐ `public`

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();
    }
}
```

0.33/0.33

- ☐ 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

1/1

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

1/1

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()`

0/1

Question 7 Justifiez votre réponse à la question précédente. ☒0 ☐1 ☐3 0/1

Si l'on modifiait le main en utilisant le new, on ferait appel à un nouvel objet référencé Protagonist pour le print et donc on ne ferait pas référence à l'objet en question lors de l'exécution du main. Or en utilisant le static, on peut faire référence à l'objet en question. il n'y a aucun objet

Question 8 ⊕ Lesquelles des expressions déclarent, construisent et initialisent un tableau ? 2/2

- ☐ `int myList = {4,9,7,0};`
- ☐ `int[] myList = (5, 8, 2);`
- ☐ `int[] myList = {"1", "2", "3"};`
- ☒ `int myList[] = {4, 3, 7};`

Question 9 ⊕ On souhaite écrire une application dans un package `foobar`. Il est obligatoire de... 0.5/0.5

- ☐ 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 : 0.5/0.5

- ☒ `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`

+75/4/20+

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

0.5/0.5

- [ ] `java -cp bin Main`
- [x] `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 initalizes 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;

class Consulter {
    private final Marks marks;

    Consulter(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.sploit;

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

```
package admin;

public class Main {
    public static void main(String... args) {
        Marks marks = new Marks();
        Consulter consulter = new Consulter(marks);
        // administration consults student marks
        marks.getStudents().forEach(s
                -> consulter.displayMarks(s));
        // Wilma introduces exploit
        new Sploit().haxMyMarks(marks.getMarks("Wilma"));
        // administration consults student marks again
        marks.getStudents().forEach(s
                -> consulter.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 3/3

```
public void haxMyMarks(int[] marks) {
    for(int mark : marks) {
        mark = 20;
    }
}
```

faudrait boucle avec
for(int i=0;...;...)
marks[i] = 20;

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

0/1

Il faudrait que le tableau d'entiers dans la map soit "final". Cela éviterait de pouvoir changer ces notes à partir d'autres classes.
Donc par exemple : put("Barney", new final int[] {12, 8});
son contenu pourrait tjr être modifié

Question 14 ⊕ Soit la déclaration :

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

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

☐ `names = new String[1];`
☒ `names[0] = "Barney";`
☐ `names = "Barney";`
☒ `names[0] == "Barney";`
☐ `names = new String("Barney");`

0/2

Question 15 ⊕ Soit la déclaration :

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

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

☐ `names = "Barney";`
☒ `names[0] = "Barney";`
☒ `names = new String[1];`
☐ `names = new String("Barney");`
☒ `names[0] == "Barney";`

1.33/2

Question 16 Corrigez toutes les erreurs dans le code :

```
/**
 * Print all the values in the marks array that are
 * greater than mean.
 * @param marks An 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 1/1

```
void printGreater(double[] marks, double mean) {
    for(int index=0; index < marks.length; index++){
        if(marks[index] > mean) {
            System.out.println(marks[index]);
        }
    }
}
```

Question 17 ⊕ Quelles affirmations s'appliquent aux tableaux :

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

1/1