QCM **TEST**

Introduction à la programmation orientée objet
24/10/2019

Nom et prénom :

.....

Groupe : .....

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 Soit la déclaration

```
private final int[] ra = {13, 27, 42};
```

Cherchez l'intrus : une des boucles ci-dessous ne donne pas le même résultat que les autres. Laquelle ?

☐
```
for (int i = 0; i < ra.length; i++) {
    System.out.println(i + " " + ra[i]);
}
```

☐
```
int i = 0;
while (i < ra.length) {
    System.out.println(i + " " + ra[i++]);
}
```

☐
```
// the following expression converts
// int[] into List<Integer>
List<Integer> l = IntStream.of(ra)
        .boxed().collect(Collectors.toList());
for (Iterator<Integer> it = l.iterator();
        it.hasNext();) {
    System.out.println(i++ + " " + it.next();
}
```

☐
```
for (int i = 0; i < ra.length;) {
    System.out.println(i + " " + ra[i++]);
}
```

■
```
int i = 0;
for (int j : ra) {
    System.out.println(i + " " + j);
}
```

Question 2 ⊕ Lesquelles des boucles peuvent donner lieu à une boucle infinie ?

- ■ `for`
- ☐ `for`-each
- ■ `while`
- ☐ `for` avec bon usage d'iterator

Question 3 Les déclarations ci-dessous peuvent se trouver dans la même classe :

```
void theMethod(Toto toto) {...}
void theMethod(Toto[] toto) {...}
```

- ☐ Non
- ■ Oui

Question 4 ⊕ Lesquelles des expressions ci-dessous sont `true` :

- ■
```
String fred = "Fred";
String frod = "Fred";
fred == frod;
```
- ■
```
new String("Fred").equals(new
String("Fred"));
```
- ■ `"Fred".equals(new String("Fred"));`
- ☐
```
String fred = new String("Fred")
fred == new String("Fred");
```
- ☐ `new String("Fred") == new String("Fred");`
- ☐ `"Fred" == new String("Fred");`

Question 5 ⊕ Soit la création et l'affectation d'un objet

```
String str = new String("Some ");
```

Le code

```
str = str + "random stuff"
```

- ■ crée un nouvel objet et l'affecte à `str`
- □ modifie la valeur de l'objet référencé par `str`

Question 6 Pour cette question et les deux qui suivent, donnez une réponse complète ; il ne suffit pas de simplement traduire le code en français.
Soit le code :

```
1  public class Person {
2      private final String name;
3      final private int age;
4
5      public Person(String name, int age) {
6          this.name = name;
7          this.age = age;
8      }
9
10     @Override
11     public boolean equals(Object obj) {
12         if (this == obj) {
13             return true;
14         }
15         if (!(obj instanceof Person)) {
16             return false;
17         }
18         Person other = (Person) obj;
19         return name.equals(other.name)
20                 && age == other.age;
21     }
22 }
```

(L'opérateur boolean `obj instanceof class` détermine si `obj` est une instance de la classe `class`.)

A quoi servent les lignes 12–14 ? □0 □1 □2 □3 ■4

Question 7 Pour la question précédente, à quoi servent les lignes 15–17 ?

☐0 ☐1 ☐2 ☐3 ■4

Question 8 Pour la question précédente, à quoi servent les lignes 18–20 ?

☐0 ☐1 ☐2 ☐3 ■4

Question 9 Soit le code ci-dessous pour gérer des personnes :

```
/** @author Stagiaire 2018-19. */
class Person {
    private final String name;
    final private int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    String getName() {
        return name;
    }

    @Override
    public String toString() {
        return name + " " + age + " years old";
    }
}
```

```
/** @author Stagiaire 2018-19. */
class People {
    private final List<Person> peeps;

    People(List<Person> peeps) {
        this.peeps = peeps;
    }

    @Override
    public String toString() {
        String str = "";
        for (Person p : peeps) {
            str += p.toString() + "\n";
        }
        return str;
    }
}
```

```
/** @author Client Pas Tres Malin. */
class PeopleDemo {
    public static void main(String[] args) {
        // it works, but don't ask
        final ArrayList<Person> adults
                    = new ArrayList<Person>() {{
            add(new Person("Larry", 42));
            add(new Person("Moe", 32));
            add(new Person("Curly Joe", 22));
        }};
        final People peeps = new People(adults);
        System.out.println(peeps);
        new Nasty(adults).minorize("Curly Joe", 12);
        System.out.println(peeps);
    }
}
```

```
/** @author Client Mal Intentionne */
class Nasty {
    private final List<Person> peeps;

    Nasty(List<Person> peeps) {
        this.peeps = peeps;
    }

    void minorize(String name, int age) {
        // code to develop
    }
}
```

N'ayant pas suivi le cours POO de Polytech'Groland, Stagiaire 2018-19 avait laissé une faille dans son code. Client Pas Très Malin s'est laissé convaincre par Client Mal Intentionné d'inclure du code de celui-ci dans son application, avec comme résultat à l'exécution :

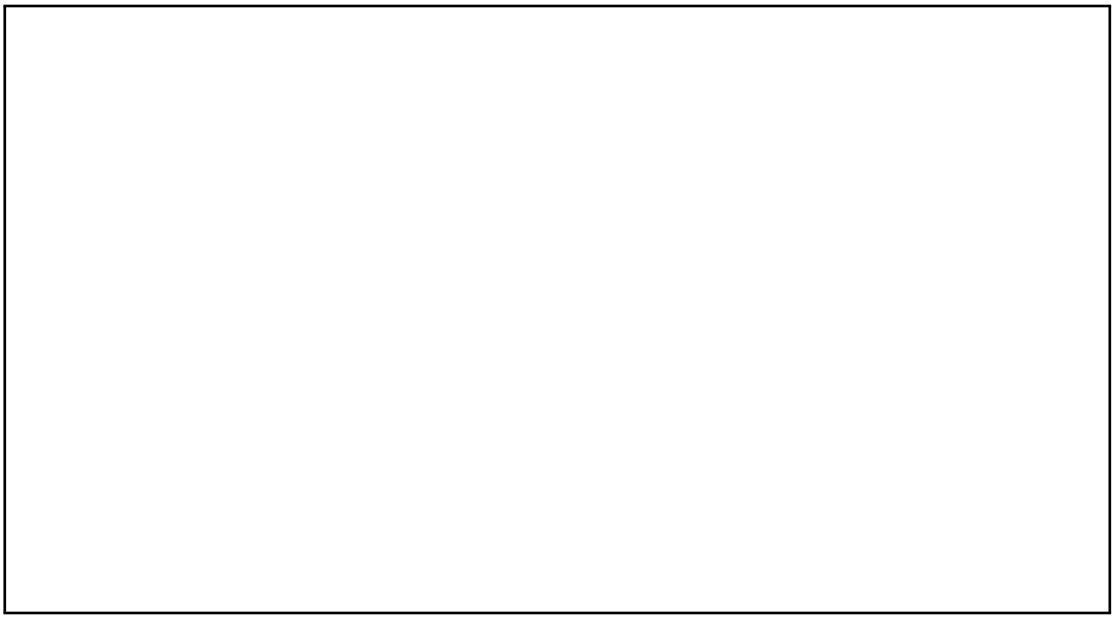
```
Larry 42 years old
Moe 32 years old
Curly Joe 22 years old

Larry 42 years old
Moe 32 years old
Curly Joe 12 years old
```

Développez le code de la méthode `Nasty#minorize` qui aurait permis la modification de l'âge du malheureux Curly Joe.

☐0 ☐1 ☐2 ☐3 ■4

Question 10 Ayant identifié la faille dans le code de la question précédente, modifiez le travail de Stagiaire 2018-19 pour l'enlever. Vous ne toucherez pas aux classes `PeopleDemo`, `Nasty`.

☐0 ☐1 ☐2 ☐3 ■4



Question 11 ⊕ Pour la question précédente, les auteurs modifient les package de leurs classes : `PeopleDemo` et `Nasty` se trouvent dans le package `main` ; `Person` et `People` dans `people`. Quels accès *doivent* être modifiés pour que tout compile :

- ■ `class Person`
- ☐ `Nasty(...)`
- ■ `Person#getName()`
- ☐ `class PeopleDemo`
- ■ `People(...)`
- ☐ `Nasty#minorize(...)`
- ■ `Person(...)`
- ■ `class People`
- ☐ `class Nasty`

Question 12 Deux élèves, Aardvark et Axolotl, ont développé les classes ci-dessous :

```
class Aardvark {
    final private double[] data;

    Aardvark(double[] data) {
        this.data = data;
    }

    double[] getData() {
        return data.clone();
    }
}
```

```
class Axolotl {
    final private double[] data;

    Axolotl(double[] data) {
        this.data = data;
    }

    double[] getSqrtData() {
        final double[] sqrtData = data.clone();
        for (int i = 0; i < data.length; i++) {
            sqrtData[i] = Math.sqrt(data[i]);
        }
        return sqrtData;
    }
```

Un troisième élève utilise les deux classes pour produire deux fois exactement le même résultat :

```
class Armadillo {
    public static void main(String[] args) {
        double[] data = {3.1, 4.1, 5.9};
        Aardvark aa = new Aardvark(data);
        for (double d : aa.getData()) {
            System.out.print(Math.sqrt(d) + ", ");
        }
        Axolotl ax = new Axolotl(data);
        for (double d : ax.getSqrtData()) {
            System.out.print(d + ", ");
        }
    }
}
```

Pourtant, une des classes est plus orientée objet que l'autre. Laquelle ?

- [x] `Axolotl`
- [ ] `Aardvark`

Question 13 ⊕ Quelles affirmations s'appliquent aux tableaux (array) :

- [ ] le nombre de leurs éléments est donné par `.size()`
- [ ] ils sont dans le package `java.util`
- [ ] ils sont p.ex. créés par `p = new Potato(14)`
- [x] ils sont indexés exclusivement par des entiers non-négatifs
- [x] ils sont de tailles fixes
- [x] ils peuvent stocker des doublons (deux fois le même élément)
- [x] ils sont p.ex. déclarés par `Potato[] p`
- [x] ils peuvent stocker des primitifs, p.ex. `double`
- [x] l'ordre de stockage de leurs éléments est bien défini

Question 14 ⊕ Quelles affirmations s'appliquent aux listes (la classe `ArrayList`) :

- [x] elles peuvent stocker des doublons (deux fois le même élément)
- [x] l'ordre de stockage de leurs éléments est bien défini
- [ ] elles sont de tailles fixes
- [ ] elles sont p.ex. déclarées par `Potato[] p`
- [x] elles sont dans le package `java.util`
- [x] le nombre de leurs éléments est donné par `.size()`
- [ ] elles sont p.ex. créées par `new Potato(14)`
- [x] elles sont indexés exclusivement par des entiers non-négatifs
- [ ] elles peuvent stocker des primitifs, p.ex. `double`

Question 15 ⊕ Quelles affirmations s'appliquent aux maps (la classe `HashMap`) :

- ☐ elles sont p.ex. initialisées par `new Potato(14)`
- ☐ l'ordre de stockage de leurs éléments est bien défini
- ☐ elles sont indexés exclusivement par des entiers non-négatifs
- ☐ elles sont p.ex. déclarées par `Potato[] p`
- ☐ elles peuvent stocker des primitifs, p.ex. `double`
- ■ le nombre de leurs éléments est donné par `.size()`
- ■ elles sont dans le package `java.util`
- ■ elles peuvent stocker des doublons (deux fois le même élément)
- ☐ elles sont de tailles fixes

Question 16 ⊕ Quelles affirmations s'appliquent aux sets (la classe `HashSet`) :

- ☐ l'ordre de stockage de leurs éléments est bien défini
- ☐ ils peuvent stocker des primitifs, p.ex. `double`
- ☐ ils sont initialsés par, p.ex. `new Potato(14)`
- ☐ ils sont de tailles fixes
- ■ le nombre de leurs éléments est donné par `.size()`
- ☐ ils peuvent stocker des doublons (deux fois le même élément)
- ☐ ils sont indexés exclusivement par des entiers non-négatifs
- ■ ils sont dans le package `java.util`
- ☐ ils sont p.ex. déclarés par `Potato[] p`

Question 17 ⊕ Soit la classe :

```
package access;

class Foo {
    public void methood() {
        System.out.println("In Foo#methood");
    }
}
```

La methode `methood` est accessible depuis :

- ☐ toute classe dans n'importe quel package
- ■ toute classe dans le package `access`

Question 18 ⊕ Soit la classe :

```
class Baz {
    public void methud() {
        Toto toto = new Toto();
        // more code
    }
}
```

La variable `toto` est accessible depuis :

- ☐ partout dans la classe `Baz`
- ■ la méthode `methud`

Question 19 ⊕ Soit la classe :

```
public class Frobnitz {
    public void methid(Toto[] toto) {
        for (int i = 0; i < toto.length; i++) {
            System.out.println("Toto" + i);
        }
        // more code
    }
}
```

La variable `i` est accessible depuis :

- ☐ partout dans la méthode `methid`
- ■ la boucle `for`

Question 20 ⊕ Soit la déclaration d'un constructor :

```
NA Toto()
```

Le niveau d'accès, `NA`, peut être déclaré :

- ■ `public`
- ■ `private`
- ■ package-private

Question 21 ⊕ Dans le code ci-dessous, l'implémentation de la méthode `calculate` manque. Quelle(s) implémentation(s) compile(nt) et à l'exécution donne(nt) le résultat `one two three` ?

```
class Array {
    private final String[] values;

    Array() {
        values = new String[]{"one", "two", null};
    }

    String[] calculate()

    @Override
    public String toString() {
        String str = "";
        for (String s : values) {
            str += s + " ";
        }
        return str;
    }

    public static void main(String[] args) {
        Array ra = new Array();
        ra.calculate();
        System.out.println(ra);
    }
}
```

■
```
String[] calculate() {
    values[values.length - 1] = "three";
    return values;
}
```

☐
```
String[] calculate() {
    return new String[]
            {"one", "two", "three"};
}
```

☐
```
String[] calculate() {
    values = new String[]
            {"one", "two", "three"};
    return values;
}
```

☐
```
String[] calculate() {
    String[] values = new String[]
            {"one", "two", "three"};
    return values;
}
```

☐
```
String[] calculate(String[] values) {
    values[values.size() - 1] = "three";
    return values;
}
```

☐ Aucune de ces implémentations.

Question 22 ⊕ Indiquez les lignes de la classe `Fibonacci` qui provoquent une erreur de *compilation* :

☐ `class Fibonacci {`
☐ `    private List<String> arrayPerson;`
☐ `    private List<Integer> arrayInt = new ArrayList<>();`
■ `    System.out.println("Fibonacci at your service")`
☐ `    public static void main(String[] args) {`
☐ `        Fibonacci fibo = new Fibonacci();`
■ `        arrayPerson = new ArrayList<>();`
☐ `        for (int i = 3; i < 10; i++) {`
☐ `            fibo.arrayInt.add(`
☐ `                    fibo.arrayInt.get(i-2) + fibo.arrayInt.get(i-1));`
☐ `        }`
☐ `        fibo.arrayPerson.add("Fred");`
☐ `        fibo.arrayPerson.add(String.valueOf('F') + ".  I. Bonacci");`
■ `        fibo.arrayPerson.add("Person" + arrayInt.get(0));`
☐ `    }`
☐ `}`