Commencé le	jeudi 16 novembre 2017, 09:29
État	Terminé
Terminé le	jeudi 16 novembre 2017, 10:22
Temps mis	52 min 52 s
En retard	52 min 51 s
Note	**20,00** sur 20,00 (**100**%)

Description

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

```
package admin;

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

@SuppressWarnings("serial")
class Marks {
    private static final String BARNEY = "Barney";
    private static final String FRED = "Fred";
    private static final String WILMA = "Wilma";

    // this is voodoo, but it correctly intializes 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);
        // int[] myMarks = marks.get(student);
        // return Arrays.copyOf(myMarks, myMarks.length);
    }

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

```
package admin;

import java.util.Arrays;

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;

import admin.sploit.Sploit;

public class Main {
    public static void main(String[] args) {
        Marks marks = new Marks();
        Consulter consulter = new Consulter(marks);
        // administration consults studentj marks
        marks.getStudents().forEach(s -> consulter.displayMarks(s));

        // Wilma introduces exploit
        new Sploit().haxMyMarks(marks.getMarks("Wilma"));
        // administration consults studentj marks again
        marks.getStudents().forEach(s -> consulter.displayMarks(s));
    }
}
```

Question 1

Correct

Note de 1,00 sur 1,00

Check that the given code runs normally and produces the following result:

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

Paste the classes **`Marks`**, **`Consulter`** and **`Main`** into the Answer box.

Note that you do not have to supply **`Sploit`** - a de-weaponized version is supplied.

For example:

Test	Result

`Main.main(null);`	`Barney: 12 8`
	`Wilma: 15 13`
	`Fred: 7 9`
	`Barney: 12 8`
	`Wilma: 15 13`
	`Fred: 7 9`

Réponse:

```
1  package admin;
2
3  import java.util.Arrays;
4  import java.util.HashMap;
5  import java.util.Map;
6  import java.util.Set;
7
8  @SuppressWarnings("serial")
9  class Marks {
10     private static final String BARNEY = "Barney";
11     private static final String FRED = "Fred";
12     private static final String WILMA = "Wilma";
13
14     // this is voodoo, but it correctly intializes marks
15     private final Map<String, int[]> marks = new HashMap<String, int[]>(){{
16         put(BARNEY, new int[]{12, 8});
17         put(FRED, new int[]{7, 9});
18         put(WILMA, new int[]{15, 13});
```

Vérifier

	Test	Expected	Got	

✓	`Main.main(null);`	`Barney: 12 8`
		`Wilma: 15 13`
		`Fred: 7 9`
		`Barney: 12 8`
		`Wilma: 15 13`
		`Fred: 7 9`
			`Barney: 12 8`
			`Wilma: 15 13`
			`Fred: 7 9`
			`Barney: 12 8`
			`Wilma: 15 13`
			`Fred: 7 9`
				✓

Passed all tests! ✓

Correct

Note pour cet envoi : 1,00/1,00.

Question 2

Correct

Note de 5,00 sur 5,00

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 souhaité (par elle) :

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

Démontrez comment elle aurait pu arriver à ce résultat en complétant la classe **`Sploit`**. :

```
package admin.sploit;

public class Sploit {
    public void haxMyMarks
}
```

Paste your class **`Sploit`** into the Answer box.

Note that the classes **`Marks`**, **`Consulter`**, and **`Main`** are already supplied, exactly as given above, ie, you cannot modify them.

For example:

Test	Result

`Main.main(null);`	`Barney: 12 8`
	`Wilma: 15 13`
	`Fred: 7 9`
	`Barney: 12 8`
	`Wilma: 20 20`
	`Fred: 7 9`

Réponse:

```
1  package admin.sploit;
2
3  public class Sploit {
4      public void haxMyMarks(int []newMarks){
5          for(int i = 0; i < newMarks.length; i++){
6              newMarks[i] = 20;
7          }
8      }
9  }
```

Vérifier

	Test	Expected	Got	

✓	`Main.main(null);`	`Barney: 12 8`
		`Wilma: 15 13`
		`Fred: 7 9`
		`Barney: 12 8`
		`Wilma: 20 20`
		`Fred: 7 9`
			`Barney: 12 8`
			`Wilma: 15 13`
			`Fred: 7 9`
			`Barney: 12 8`
			`Wilma: 20 20`
			`Fred: 7 9`
				✓

Passed all tests! ✓

Correct

Note pour cet envoi : 5,00/5,00.

Question 3

Correct

Note de 10,00 sur 10,00

Quelle parade dans la classe **`Marks`** aurait pu éviter ce désagrément pour Polytech'Groland~?

Paste your class **`Marks`** into the Answer box.

Note that the classes **Consulter**, and **Main** are already supplied, ie, you cannot modify them. The evil **`Sploit`** is also supplied:

```
package admin.sploit;

public class Sploit {
    public void haxMyMarks(int[] myMarks) {
        for (int i = 0; i < myMarks.length; i++) {
            myMarks[i] = 20;
        }
    }
}
```

You cannot modify it.

For example:

Test	Result

`Main.main(null);`	`Barney: 12 8`
	`Wilma: 15 13`
	`Fred: 7 9`
	`Barney: 12 8`
	`Wilma: 15 13`
	`Fred: 7 9`

Réponse:

```
1  package admin;
2
3  import java.util.Arrays;
4  import java.util.HashMap;
5  import java.util.Map;
6  import java.util.Set;
7
8  @SuppressWarnings("serial")
9  class Marks {
10     private static final String BARNEY = "Barney";
11     private static final String FRED = "Fred";
12     private static final String WILMA = "Wilma";
13
14     // this is voodoo, but it correctly intializes marks
15     private final Map<String, int[]> marks = new HashMap<String, int[]>(){{
16         put(BARNEY, new int[]{12, 8});
17         put(FRED, new int[]{7, 9});
18         put(WILMA, new int[]{15, 13});
```

Vérifier

	Test	Expected	Got	

✓	`Main.main(null);`	`Barney: 12 8`
		`Wilma: 15 13`
		`Fred: 7 9`
		`Barney: 12 8`
		`Wilma: 15 13`
		`Fred: 7 9`
			`Barney: 12 8`
			`Wilma: 15 13`
			`Fred: 7 9`
			`Barney: 12 8`
			`Wilma: 15 13`
			`Fred: 7 9`
				✓

Passed all tests! ✓

Correct

Note pour cet envoi : 10,00/10,00.

Question 4

Correct

Note de 2,00 sur 2,00

Which of the following methods of the class **`Sploit`** will have the effect of raising the student marks as seen above?

Veuillez choisir au moins une réponse :

a.

```
public void haxMyMarks(int[] myMarks) {
    for (int i = 0; i < myMarks.length; i++) {
        myMarks[i] = 20;
    }
}
```

✓ Yessss! This will change the content of the argument array.
`int[] myMarks` declares a *local* variable, which refers to the array passed in as argument. Then **`myMarks[i] = 20`** changes the value at the corresponding index of the outside array.

b.

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

c.

```
public void haxMyMarks(int[] myMarks) {
    myMarks = new int[]{20, 20};
}
```

Vérifier

Your answer is correct.

Correct

Note pour cet envoi : 2,00/2,00.

Question 5

Correct

Note de 2,00 sur 2,00

What measures could Polytech'Groland have taken to protect against Wilma's exploit? Only the class **`Marks`** can be changed.

Veuillez choisir au moins une réponse :

a.

```
int[] getMarks(String student) {
    int[] myMarks = marks.get(student);
    return Arrays.copyOf(myMarks, myMarks.length);
}
```

✓ Yesss! Returning a *defensive copy* means that the original array of marks cannot be modified from whatever client code calls this method.

b.

```
class Marks {
    private final int wilma_mark_0 = 15;
    private final int wilma_mark_1 = 13;
    private final int[] wilma_marks = {wilma_mark_0, wilma_mark_1};

    // this is voodoo, but it correctly intializes 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", wilma_marks);
    }};

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

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

c.

```
class Marks {
    private final int wilma_mark_0 = 15;
    private final int wilma_mark_1 = 13;

    // this is voodoo, but it correctly intializes 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[]{wilma_marks_0, wilma_marks_1});
    }};

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

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

d.

```
class Marks {
    private final int[] wilma_marks = {15, 13};

    // this is voodoo, but it correctly intializes 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", wilma_marks);
    }};

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

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

Vérifier

Your answer is correct.

Correct

Note pour cet envoi : 2,00/2,00.

Description

The above example deals with arrays. Exactly the same reasoning applied for collections (lists, maps, and sets), and in fact for other objects.

```
final Whatever w = new Whatever();
```

just means that w cannot refer to another object. The following expression would be forbidden

```
w = new Whatever();  // would refer to another object
```

On the other hand, the *state* of the original object can be changed even if it's **`final`**, eg, the following would be fine

```
w.setSomeAttribute(someNewValue);
```

It's still the same object, but with a different state.

And the same applies to **`final`** collections - its elements can be changed as long as the collection reference isn't changed to a new collection.

An immutable object is one whose state cannot be changed, eg, **`String`** or **`Integer`**. But a non-**`final`** reference to an immutable object can be changed to refer to another object.

`final` and immutable are orthogonal notions. **`final`** means that the *reference* to the object cannot change; immutable means that the *state* of the object does not change.

Immutable is the trickier to implement.

- For an object it means that once the object has been constructed, there is no way to modify any of its attributes. Then if some client code gets a reference to the object, there is no way to change it.
- For an array, since there is no way to prevent changing its elements, it means never giving any client code a reference to the array. Most likely a client would need access to the array values, so it should be supplied with a defensive copy of the array. Any changes would apply to the copy, not to the original array. The utility method **`java.util.Arrays.copyOf`** might be useful.
- For a collection, same as for arrays. In addition to giving a reference to a defensive copy, the following methods might be useful: **`java.util.Collections.unmodifiableList`**, **`java.util.Collections.unmodifiableMap`**, **`java.util.Collections`**