Algo

Part 1 - Basics

CDA

Objectifs

Objectifs des cours d'algo

- Revenir sur les bases (homogénéiser)

- Découvrir la syntaxe de Java

- Décomposer un problème

- Réaliser des algos de niveau intermédiaire préparant aux tests techniques 

C'est quoi un algo ?

C'est quoi un algorithme ?

Des ingrédients

Des ustensiles

Un gâteau au yaourt

Un chef heureux

Une recette de cuisine de gateau au yaourt

C'est quoi un algorithme ?

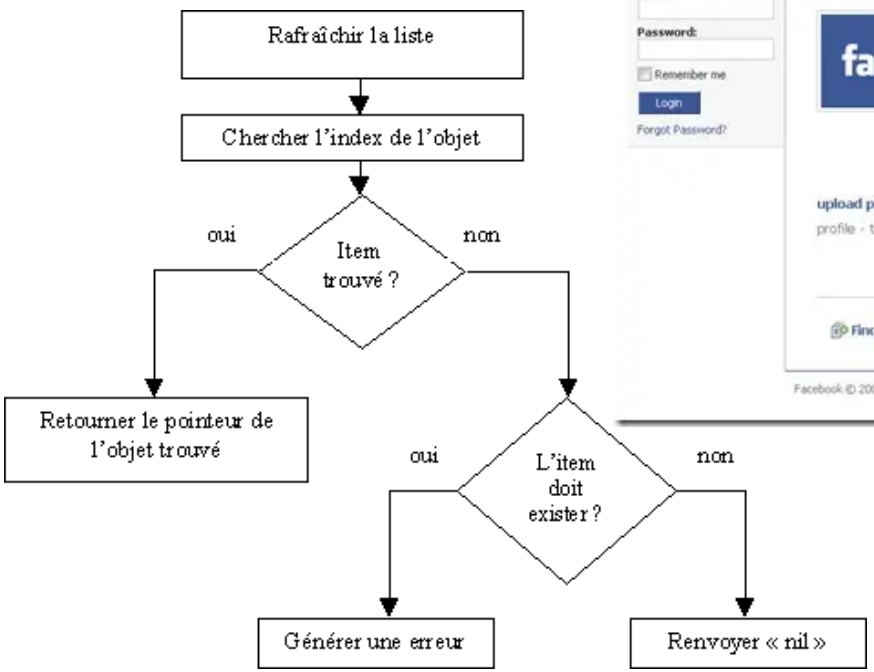
```
1 String message = "Hello world";
2 boolean jaiFaim = true;
3 int res = 42;
```

Données

```
1 if (jaiFaim) {
2   return "Miam, je mange";
3 } else {
4   return "Je bois un café";
5 }
```

Conditions / boucles / opérateurs...

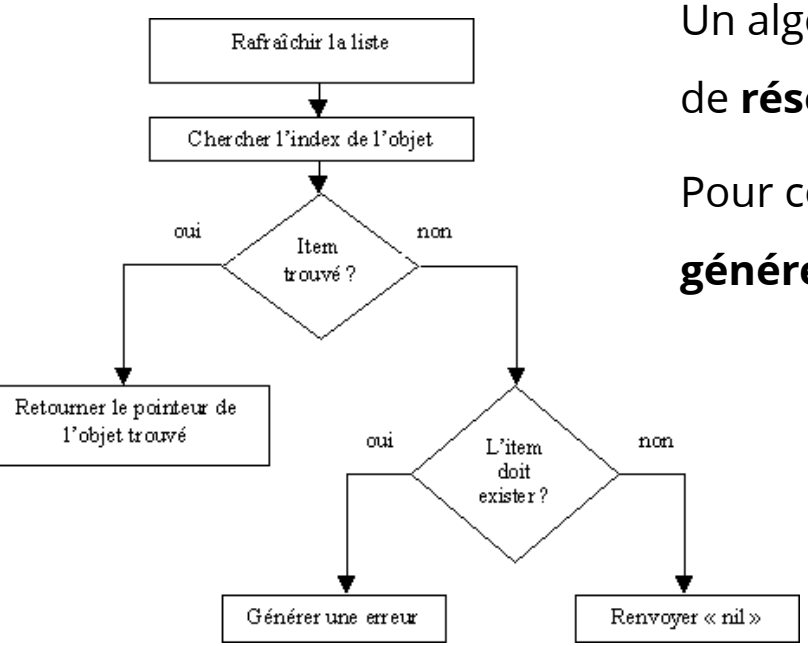
Un•e dev



Un algorithme

Un logiciel

C'est quoi un algorithme ?



Un algorithme est une **séquence d'instructions** qui permet de **résoudre un problème**.

Pour cela on **manipule souvent une donnée d'entrée** pour **générer une donnée retournée**.

```
1 // Pour faire du jus d'orange il faut :
2   // Instruction 1 : Découper l'orange
3   // Instruction 2 : Presser les deux parties de l'orange
4   // Instruction 3 : Répéter les instructions précédentes
5   //    autant de fois qu'il y a d'oranges
6
```

Algo intermédiaire ou complexe

Algo simple ou intermédiaire

⚠️ *L'ordre des instructions est important !*

Codewars !

Codewars !

Plateforme de challenges pour développeurs•euses

- Beaucoup de challenges (kata) et de membres

- +ieurs niveaux de difficultés

- +ieurs langages

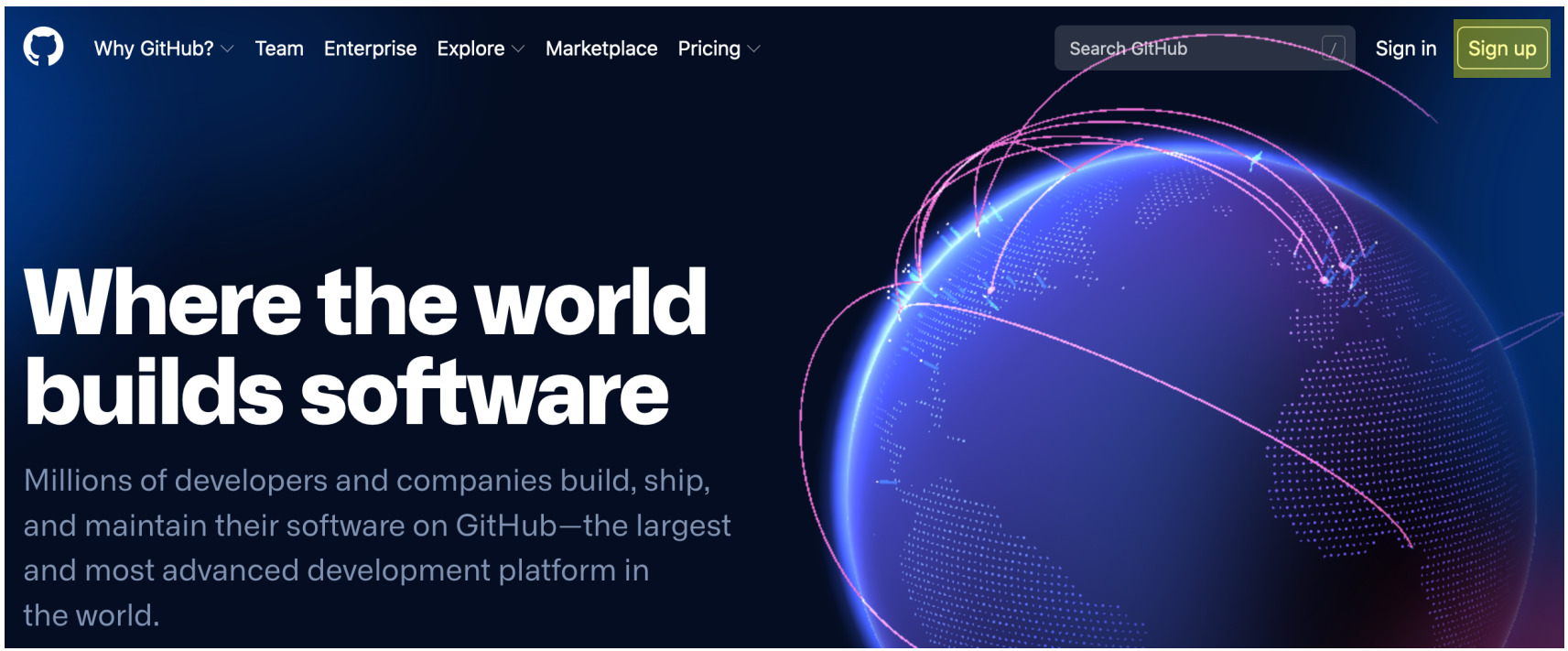
- "Best practices" et "clever"

- Monde pro (visibilité et entrainement)

- Fun

Codewars !

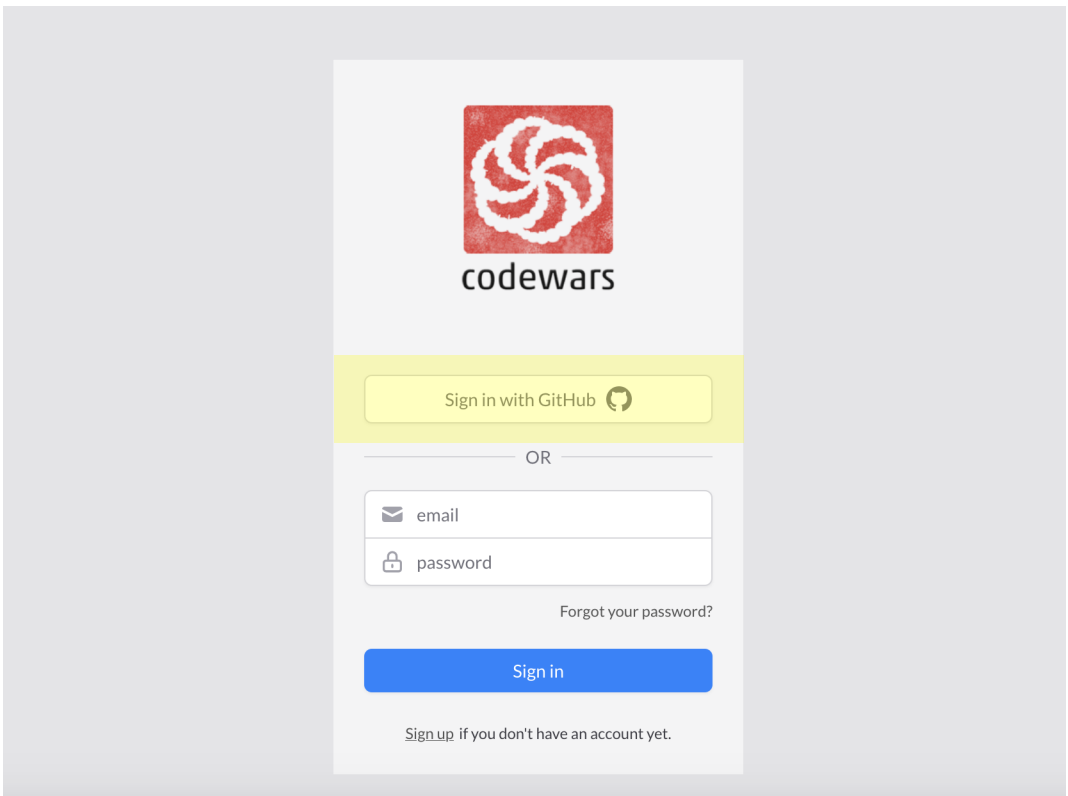
- Créer un compte Github (c'est quoi ?)



- Follow me (nicolaslechenic)

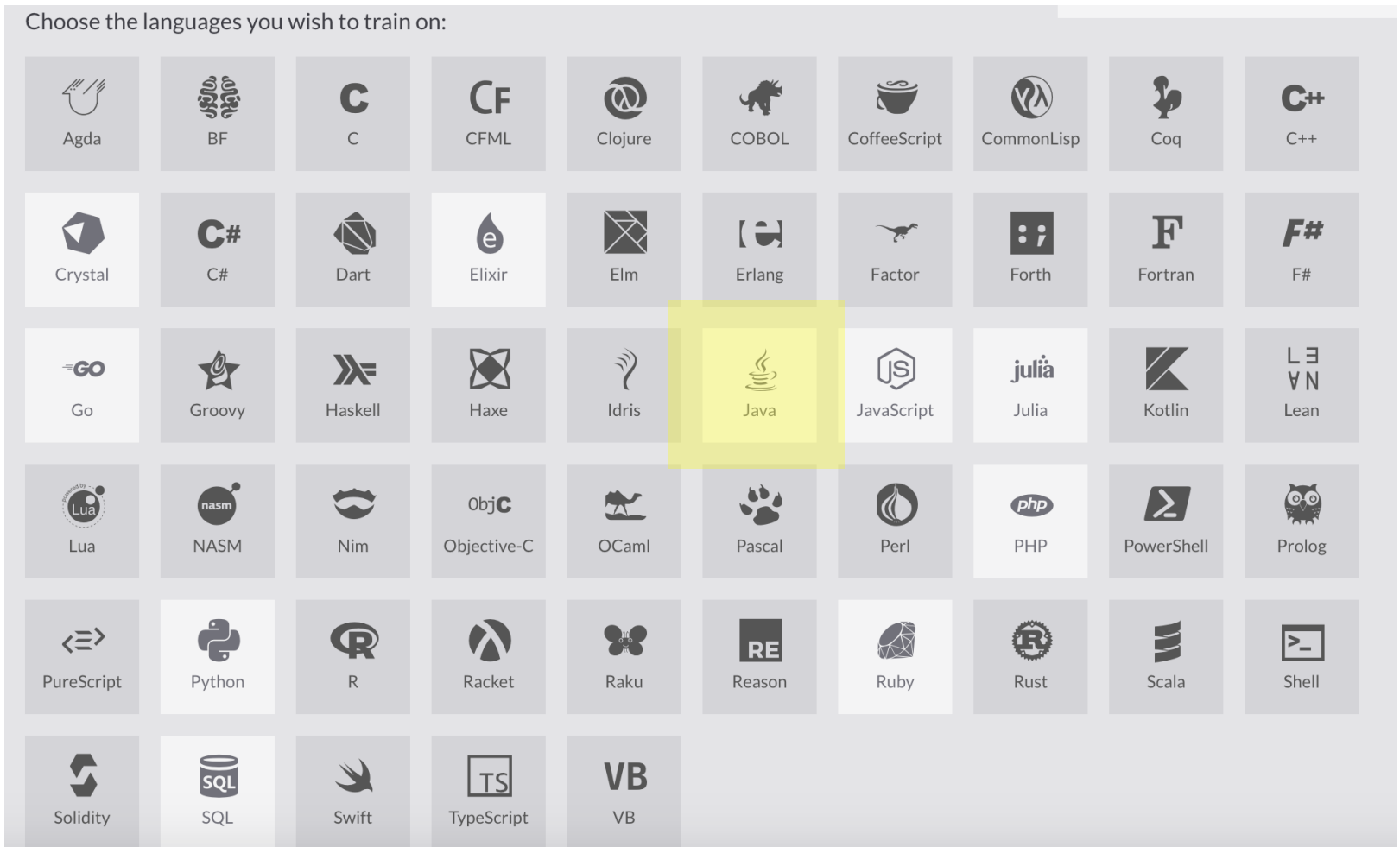
Codewars !

- Se connecter sur Codewars avec Github



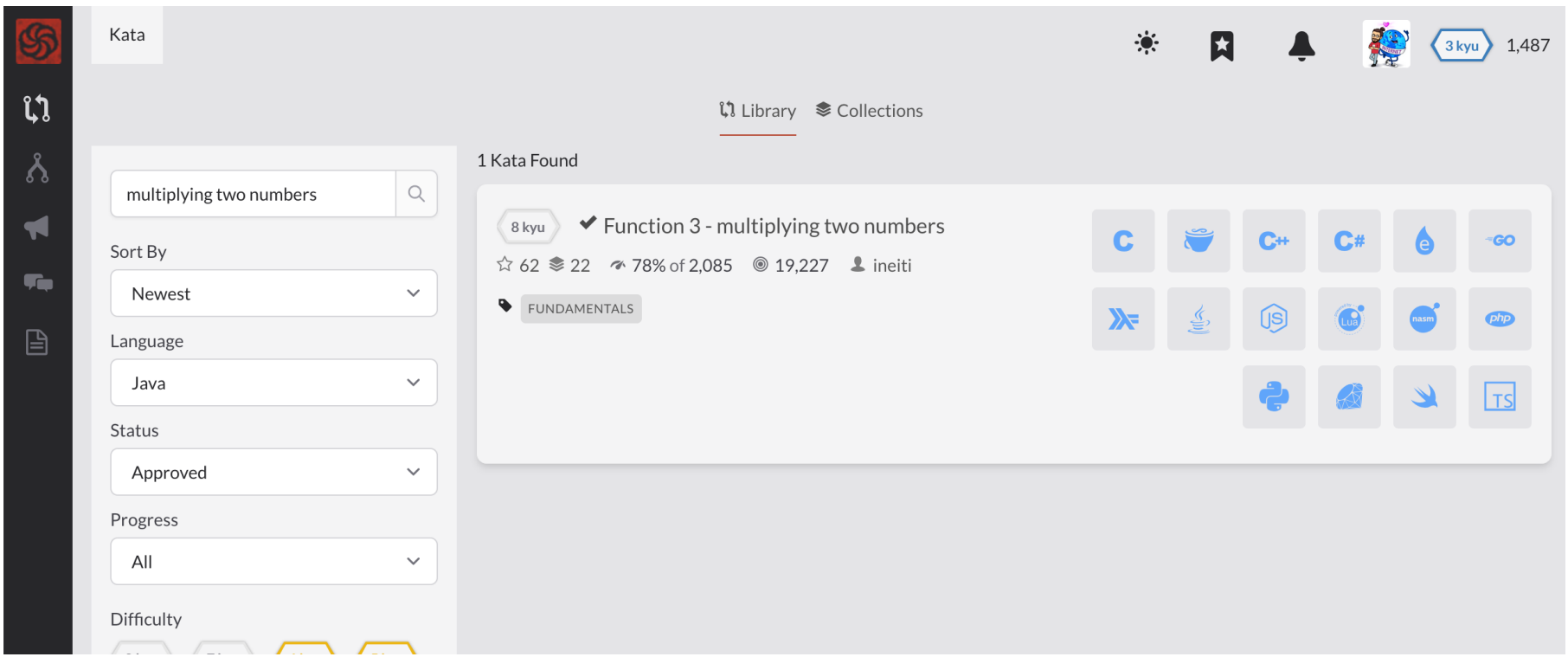
Codewars !

- Cliquer sur Java



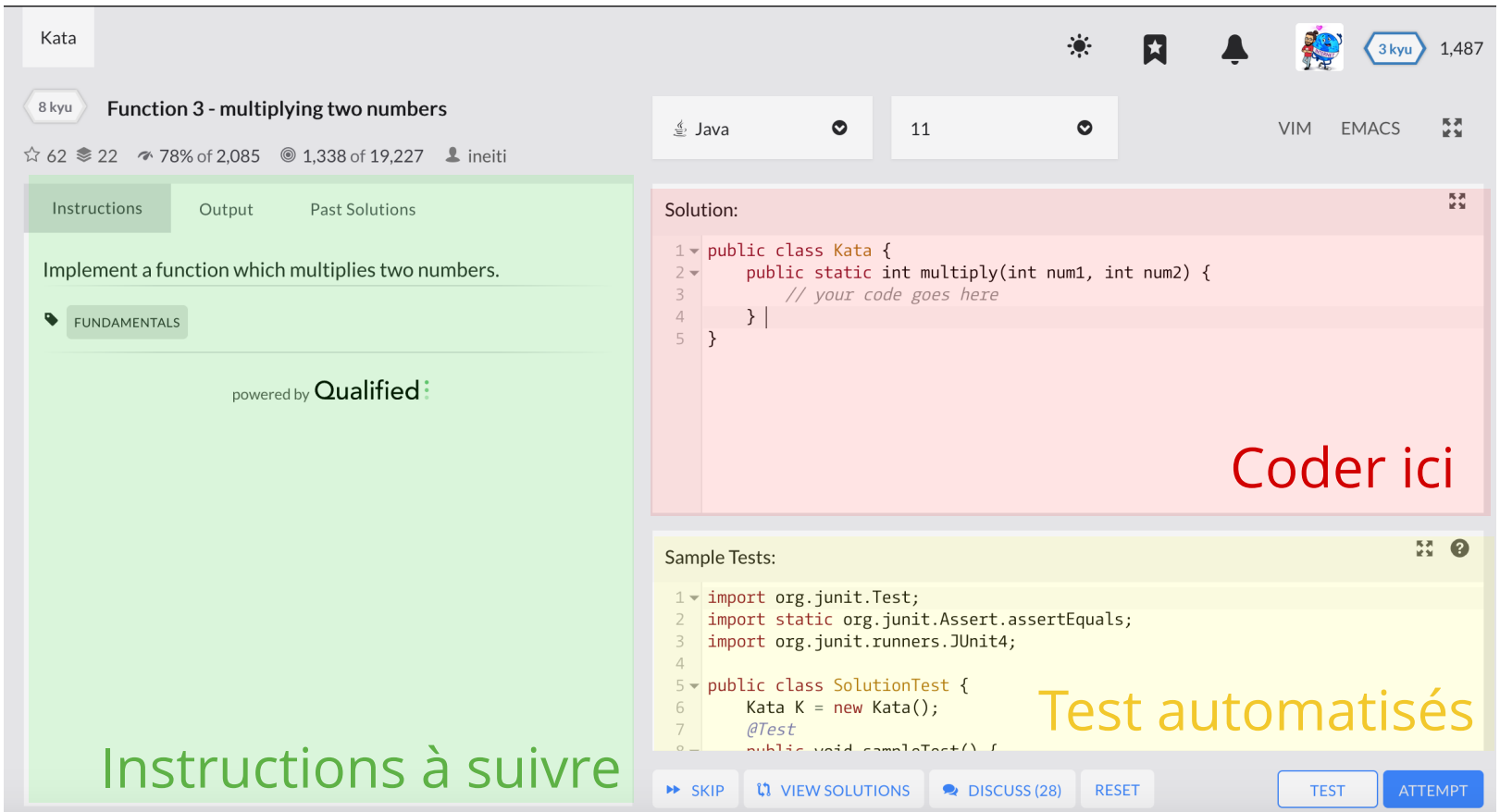
Codewars !

- Premier kata



Codewars !

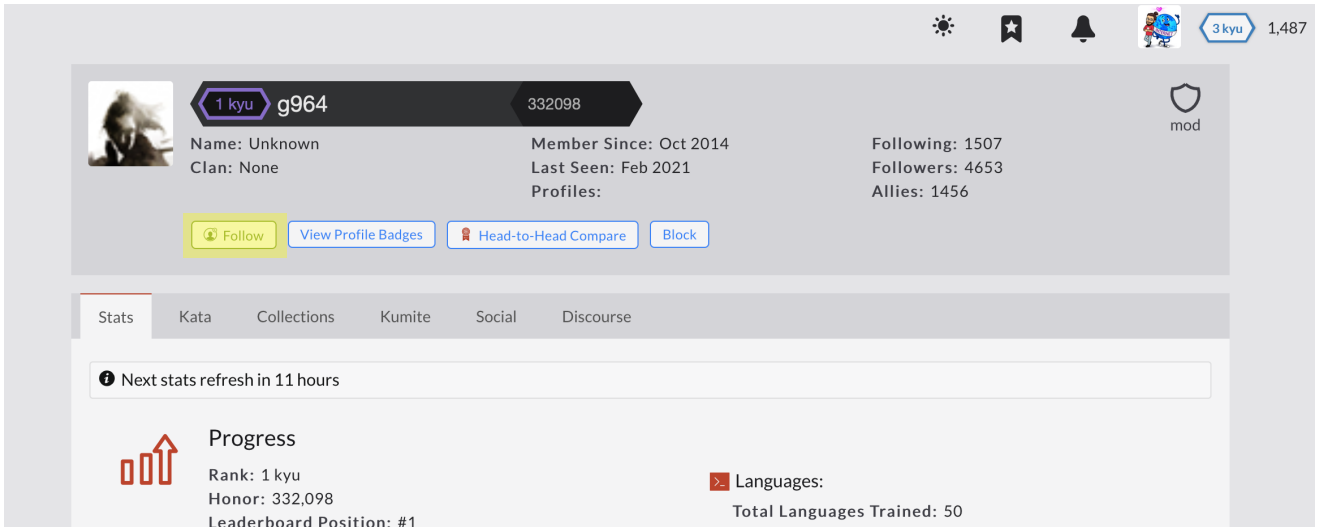
- Premier kata



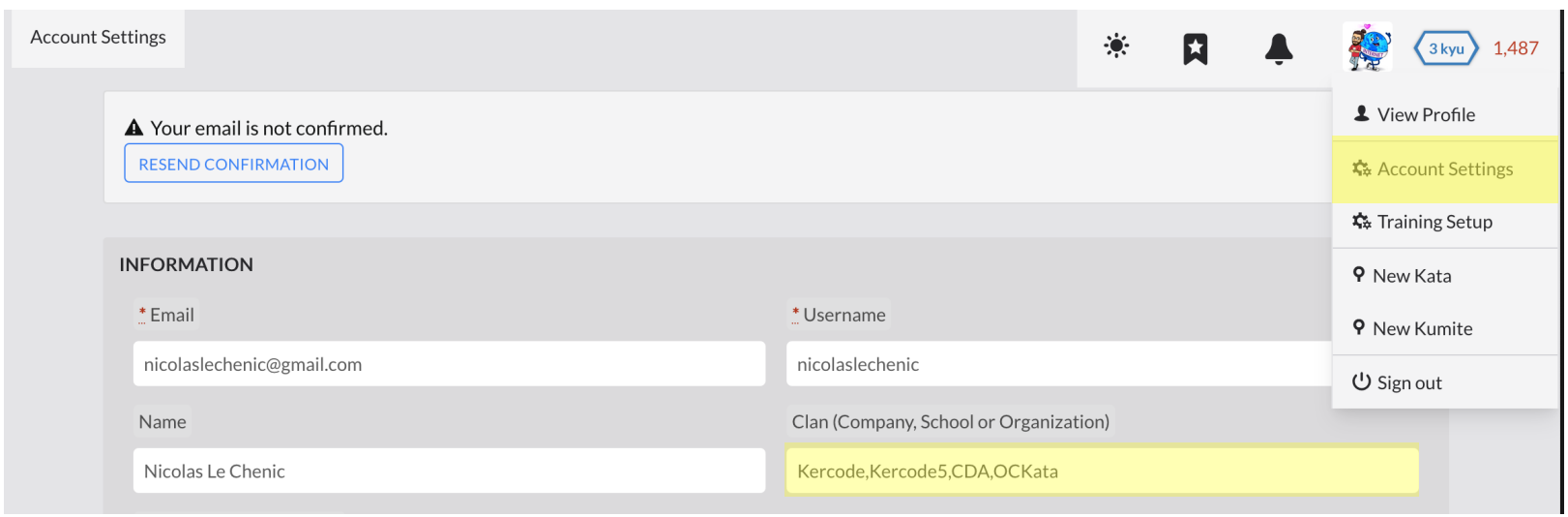
Codewars !

Premier kata (tests, réalisation et solutions)

Codewars !



- Follow me (/users/nicolaslechenic)



- Clan CDA2 (account settings)

Exos !

Codewars - Découverte

Credit Card Mask

Credit Card Mask

Solution étape par étape (Décomposer)

Commenter les tests sans honte ! (shameless green)

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5  public class SolutionTest {
6      @Test
7      public void testSolution() {
8          //assertEquals("############5616", Maskify.maskify("4556364607935616"));
9          //assertEquals("#######5616",      Maskify.maskify(     "64607935616"));
10         assertEquals("1",                Maskify.maskify(                "1"));
11         assertEquals("",                 Maskify.maskify(                 ""));
12
13         // "What was the name of your first pet?"
14         //assertEquals("##ippy",                                  Maskify.maskify("Skippy")
15         //assertEquals("##################################man!",  Maskify.maskify("Nanananananananananananananana
16     }
17 }
```

Credit Card Mask

Solution étape par étape (Résolution partielle)

```java
public class Maskify {
    public static String maskify(String str) {
        int rest = str.length() - 4;

        if(rest < 1) { return str; }

        return "";
    }
}
```

Test Results:

▼ SolutionTest

▶ testSolution

Completed in 12ms

Credit Card Mask

Premières remarques :

```java
public class Maskify {
    public static String maskify(String str) {
        int rest = str.length() - 4;

        if(rest < 1) { return str; }

        return "";
    }
}
```

- Java est un langage objet, on reviendra sur cette partie plus tard
- **str** est la donnée que l'on va traiter
- Ce qui nous intéresse sont **les instructions** à l'intérieur de cette partie

Credit Card Mask

Les variables :

Premières remarques (Les variables)

```
1 public class Maskify {
2     public static String maskify(String str) {
3         int rest = str.length() - 4;
4
5
6
7
8     }
9 }
```

Type Nom Valeur

Opérateur d'affectation

Credit Card Mask

Les variables

On peux se représenter une variable comme étant une boîte qui nous permet de stocker et de restituer de l'information

```
1 String maBoite = "la valeur à stocker...";
```

On accède à une variable en écrivant son nom

```
1 maBoite
2 // Retournera "la valeur à stocker..."
```

Information présente dans la boîte :

- D'un type (ici String)

- D'une valeur (ici "la valeur à stocker...")

Credit Card Mask

Les variables

```java
int res = 42;
```

Java

- Langage à typage statique explicite
- L'ordinateur connaît le type à la compilation (+ performant)
- Plus verbeux (- maintenable)

```ruby
res = 42
```

Ruby

- Langage à typage dynamique
- L'ordinateur détermine le type lors de l'execution (- performant)
- Moins verbeux (+ maintenable)

Credit Card Mask

Instruction conditionnelle *if*

```
1 public class Maskify {
2     public static String maskify(String str) {
3         int rest = str.length() - 4;
4 
5         if(rest < 1) { return str; }
6 
7         return "";
8     }
9 }
```

- Structure de contrôle
- Vérifie le respect de la condition
- **return** sortie de méthode

De manière littérale on pourrait traduire cette instruction comme ceci:

Si rest *est* ***inférieur à un*** *alors* ***retourne*** *la chaine de caractère* ***str***

Credit Card Mask

Instruction conditionnelle ***if, else if, else***

```
1  if(expression) { instruction à executer; }
2
3  if(expression) {
4    // instruction à executer;
5  } else {
6    // autre instruction
7  }
8
9  if(expression) {
10   // instruction(s) à executer;
11 } else if(autre_expression) {
12   // autre(s) instruction(s)
13 } else {
14   // encore d'autre(s) instruction(s)
15 }
```

La valeur de l'**expression** est un **boolean** (true ou false)

On écrit une condition à l'aide de l'instruction "if"

L'alternative "else" est facultative

Pour écrire d'autres options, on écrit "else if", et "else" pour la dernière option

On exécute les instructions d'un seul bloc (if ou else)

Credit Card Mask

Valeur de retour :

```
1 public class Maskify {
2     public static String maskify(String str) {
3         int rest = str.length() - 4;
4 
5         if(rest < 1) { return str; }
6 
7         return "";
8     }
9 }
```

En Java on précise le type de valeur retourné

Dans tous les cas, on doit retourner une valeur avec le type attendu

Credit Card Mask

On continue :

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5  public class SolutionTest {
6      @Test
7      public void testSolution() {
8          assertEquals("############5616", Maskify.maskify("4556364607935616"));
9          assertEquals("#######5616",      Maskify.maskify(     "64607935616"));
10         assertEquals("1",                Maskify.maskify(                "1"));
11         assertEquals("",                 Maskify.maskify(                 ""));
12
13         // "What was the name of your first pet?"
14         //assertEquals("##ippy",                                  Maskify.maskify("Skippy")
15         //assertEquals("##################################man!",  Maskify.maskify("Nanananananananananananananana
16     }
17 }
```

Credit Card Mask

Je liste les problèmes qui se posent :

- Comment faire pour récupérer les 4 derniers caractères en Java ?

- Comment faire pour remplacer par des # ceux d'avant ?

    - Répéter un caractère x fois ?

    - Coller deux chaînes de caractères ?

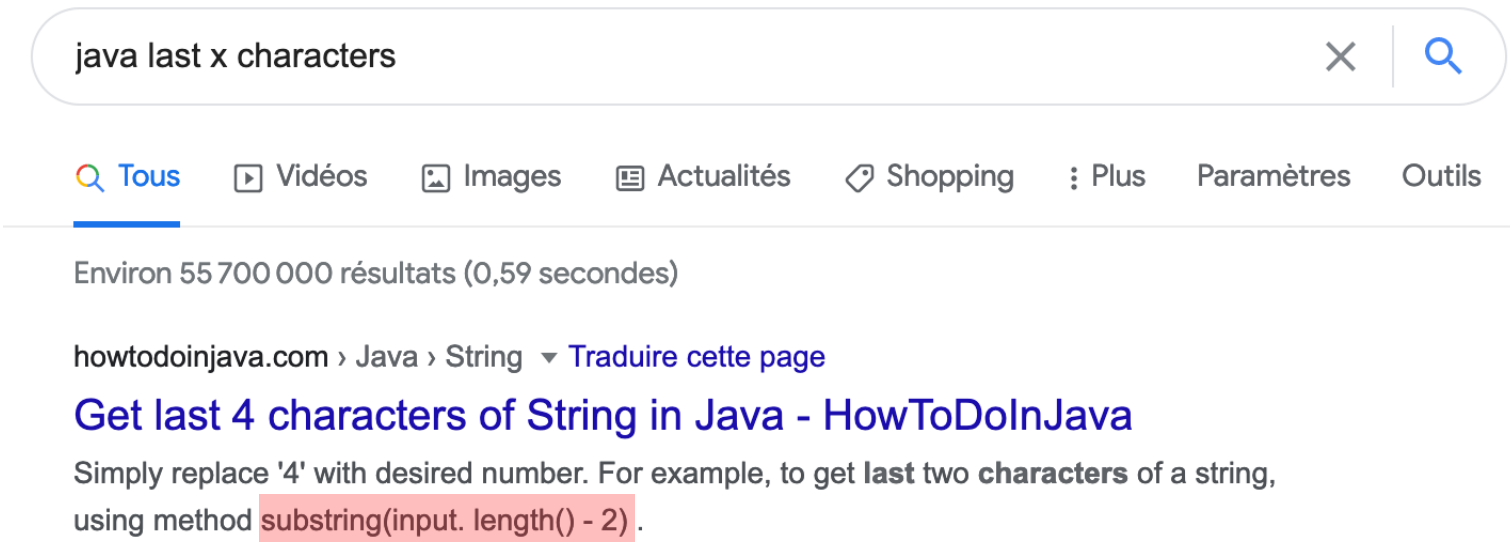
Credit Card Mask

Retour sur les tests :

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5  public class SolutionTest {
6      @Test
7      public void testSolution() {
8          //assertEquals("############5616", Maskify.maskify("4556364607935616"));
9          //assertEquals("#######5616",      Maskify.maskify(     "64607935616"));
10         assertEquals("1",                Maskify.maskify(                "1"));
11         assertEquals("",                 Maskify.maskify(                 ""));
12
13         assertEquals("5616",      Maskify.maskify(     "4556364607935616"));
14
15         // "What was the name of your first pet?"
16         //assertEquals("##ippy",                                  Maskify.maskify("Skippy")
17         //assertEquals("##################################man!",  Maskify.maskify("Nanananananananananananananana
18     }
19 }
```

Credit Card Mask

Chercher la solution (google)



Credit Card Mask

On continue :

```
1  public class Maskify {
2      public static String maskify(String str) {
3          int xChars = str.length() - 4;
4
5          if(xChars < 1) { return str; }
6
7          String lastFourDigits = str.substring(xChars);
8
9          return lastFourDigits;
10     }
11 }
```

Test Results:

▼ **SolutionTest**

▶ **testSolution**

Completed in 12ms

Credit Card Mask

Les types :

```
1  public class Maskify {
2      public static String maskify(String str) {
3          int xChars = str.length() - 4;
4
5          if(xChars < 1) { return str; }
6
7          String lastFourDigits = str.substring(xChars);
8
9          return lastFourDigits;
10     }
11 }
```

lowerCase = primitive

CamelCase = Wrapper (POO)

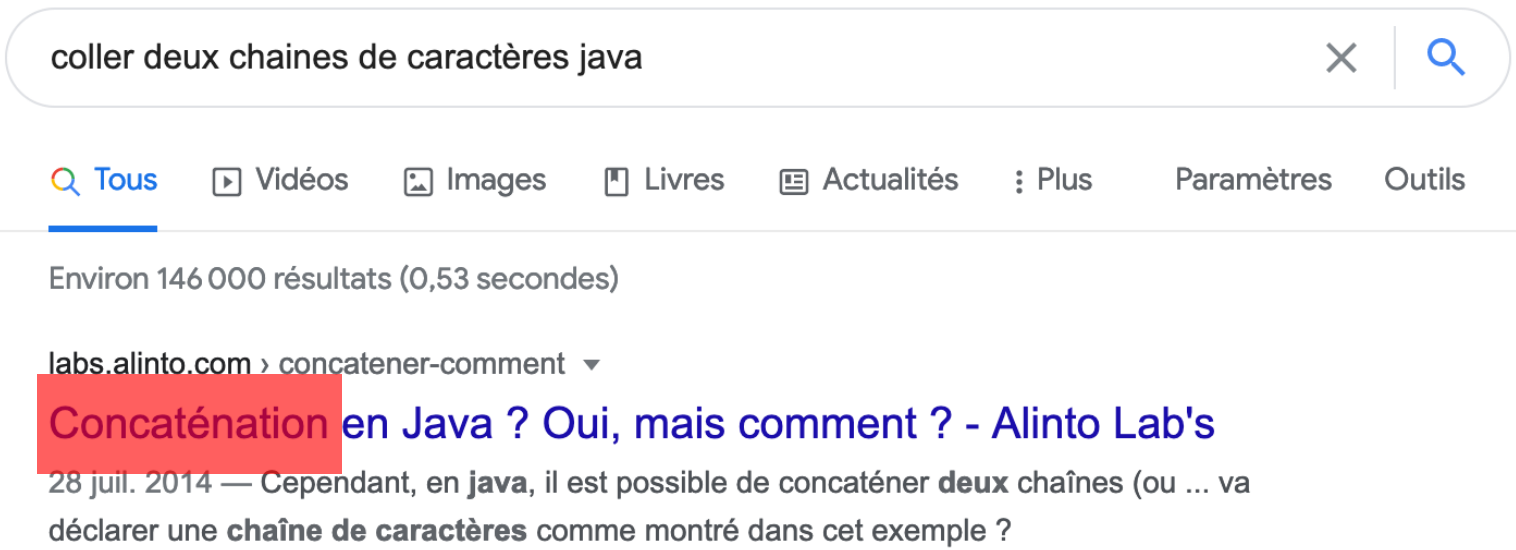
Credit Card Mask

Retour sur les tests :

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5  public class SolutionTest {
6      @Test
7      public void testSolution() {
8          assertEquals("############5616", Maskify.maskify("4556364607935616"));
9          //assertEquals("#######5616",      Maskify.maskify(     "64607935616"));
10         assertEquals("1",                 Maskify.maskify(                "1"));
11         assertEquals("",                  Maskify.maskify(                 ""));
12
13         //assertEquals("5616",      Maskify.maskify(     "4556364607935616"));
14
15         // "What was the name of your first pet?"
16         //assertEquals("##ippy",                                  Maskify.maskify("Skippy")
17         //assertEquals("##################################man!",  Maskify.maskify("Nanananananananananananananana
18     }
19 }
```

Credit Card Mask

Coller deux chaînes de caractères :



Credit Card Mask

Chercher la solution (rtfm)

~~Coller~~ Concaténer deux chaînes de caractères

Java 11

Solution:

```
1 public class Maskify {
2     public static String maskify(String str) {
3         throw new Uns
4     }
```

ORACLE

Java SE > Java SE Specifications > Java Language Specification

The Java® Language Specification

Next

The Java® Language Specification

Java SE 11 Edition

James Gosling

Bill Joy

Guy Steele

Gilad Bracha

Alex Buckley

Daniel Smith

2018-08-21

Credit Card Mask

Chercher la solution (rtfm)

~~Coller~~ Concaténer deux chaînes de caractères

Credit Card Mask

Chercher la solution (rtfm)

~~Coller~~ Concaténer deux chaînes de caractères

15.18.1. String Concatenation Operator +

If only one operand expression is of type `String`, then string conversion (§5.1.11) is performed on the other operand to produce a string at run time.

The result of string concatenation is a reference to a `String` object that is the concatenation of the two operand strings. The characters of the left-hand operand precede the characters of the right-hand operand in the newly created string.

The `String` object is newly created (§12.5) unless the expression is a constant expression (§15.28).

An implementation may choose to perform conversion and concatenation in one step to avoid creating and then discarding an intermediate `String` object. To increase the performance of repeated string concatenation, a Java compiler may use the `StringBuffer` class or a similar technique to reduce the number of intermediate `String` objects that are created by evaluation of an expression.

For primitive types, an implementation may also optimize away the creation of a wrapper object by converting directly from a primitive type to a string.

Example 15.18.1-1. String Concatenation

The example expression:

```
"The square root of 2 is " + Math.sqrt(2)
```

produces the result:

```
"The square root of 2 is 1.4142135623730952"
```

The + operator is syntactically left-associative, no matter whether it is determined by type analysis to represent string concatenation or numeric addition. In some cases care is required to get the desired result. For example, the expression:

```
a + b + c
```

is always regarded as meaning:

```
(a + b) + c
```

Credit Card Mask

```java
public class Maskify {
    public static String maskify(String str) {
        int xChars = str.length() - 4;

        if(xChars < 1) { return str; }

        String lastFourDigits = str.substring(xChars);

        return "############" + lastFourDigits;
    }
}
```

Test Results:

▼ **SolutionTest**

▶ **testSolution**

Completed in 12ms

Credit Card Mask

Les opérateurs :

```
1  public class Maskify {
2      public static String maskify(String str) {
3          int xChars = str.length() - 4;
4
5          if(xChars < 1) { return str; }
6
7          String lastFourDigits = str.substring(xChars);
8
9          return "############" + lastFourDigits;
10     }
11 }
```

Credit Card Mask

Les opérateurs courants :

```
1  // Addition
2  int a = 0;
3  int b = 0;
4
5  a + b;
6
7  // retourne 0
8
9
10 // Concaténation
11 String c = "La tête ";
12 String d = "à toto";
13
14 c + d;
15
16 // retourne La tête à toto
```

L'opérateur **+** permet l'**addition** lorsqu'on le positionne entre deux nombres

L'opérateur **+** permet la **concaténation** lorsqu'il un des deux n'est pas un nombre

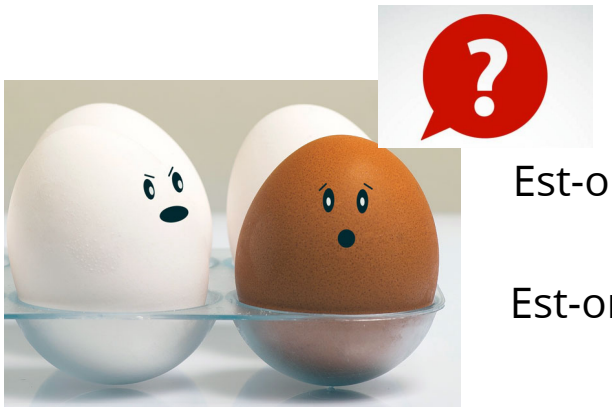
L'opérateur **-** permet la **soustraction**

L'opérateur ***** permet la **multiplication**

L'opérateur **/** permet la **division**

Credit Card Mask

Les opérateurs courants (==, !=, <, >)

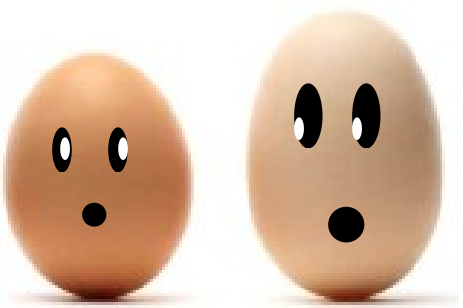


Est-on de la **même** couleur ? color_egg_a **==** color_egg_b

Est-on d'une couleur **différente** ? color_egg_a **!=** color_egg_b

OG est-il **plus grand que** OD ? left_egg **>** right_egg

OG est-il **plus petit que** OD ? left_egg **<** right_egg



*La réponse à ces questions est nécessairement oui **true** ou non **false** qui sont des valeurs de type Booléen **Boolean** !*

Credit Card Mask

Here we go ! (Le moment complexe arrive)

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5  public class SolutionTest {
6      @Test
7      public void testSolution() {
8          assertEquals("############5616", Maskify.maskify("4556364607935616"));
9          assertEquals("#######5616",      Maskify.maskify(     "64607935616"));
10         assertEquals("1",                Maskify.maskify(                "1"));
11         assertEquals("",                 Maskify.maskify(                 ""));
12
13         //assertEquals("5616",      Maskify.maskify(     "4556364607935616"));
14
15         // "What was the name of your first pet?"
16         //assertEquals("##ippy",                                    Maskify.maskify("Skippy")
17         //assertEquals("##################################man!",  Maskify.maskify("Nanananananananananananananana
18     }
19 }
```

Credit Card Mask

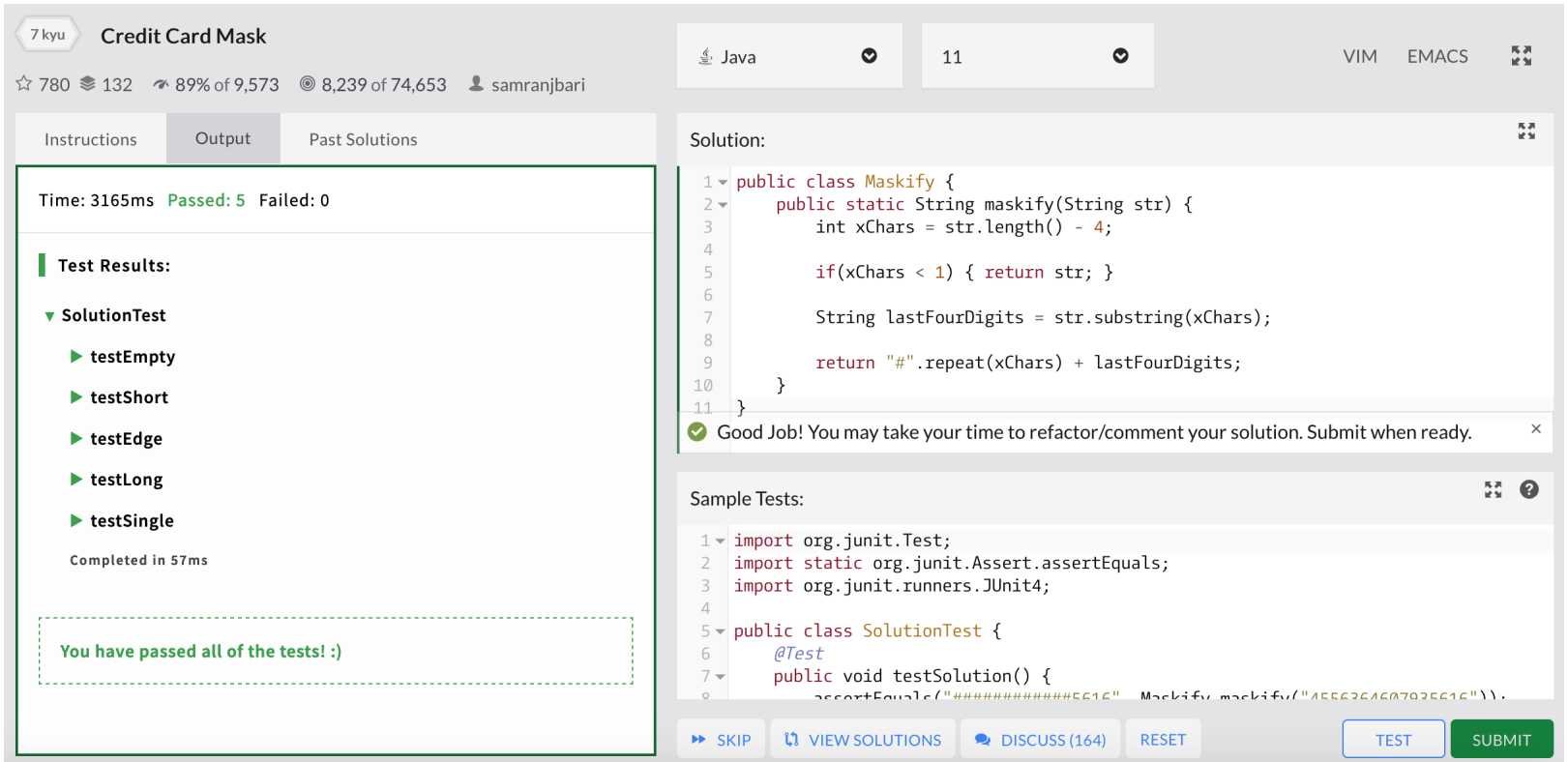
Here we go ! (ha ba en faite euu...)

```java
public class Maskify {
    public static String maskify(String str) {
        int xChars = str.length() - 4;

        if(xChars < 1) { return str; }

        String lastFourDigits = str.substring(xChars);

        return "#".repeat(xChars) + lastFourDigits;
    }
}
```

En décomposant le problème c'est plus simple !

- Felix

Credit Card Mask



GG !

À vous de jouer !

L'entraide est la bienvenue !

Do I get a bonus?

Fizz Buzz Cuckoo Clock

Colored triangle

Multiples of 3 or 5

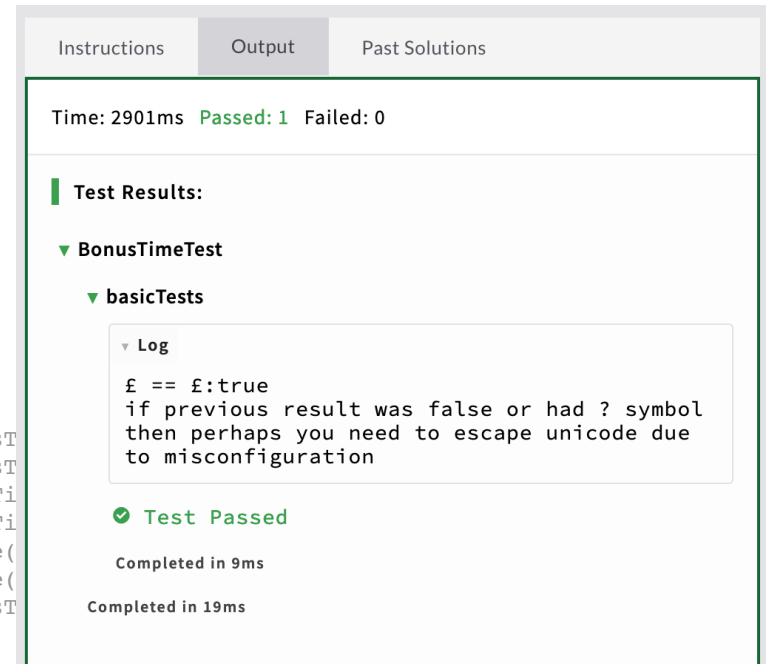
Do I get a bonus ?

Comprendre les tests sans être effrayé

Do I get a bonus ?

Comprendre les tests sans être effrayé

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5
6  public class BonusTimeTest {
7      @Test
8      public void basicTests() {
9        boolean wellConfigured=unicodeTest();
10       //assertEquals((wellConfigured?"£":"\u00A3")+"100000",Kata.bonusT
11       //assertEquals((wellConfigured?"£":"\u00A3")+"250000",Kata.bonusT
12       //assertEquals((wellConfigured?"£":"\u00A3")+"10000",Kata.bonusTi
13       //assertEquals((wellConfigured?"£":"\u00A3")+"60000",Kata.bonusTi
14       //assertEquals((wellConfigured?"£":"\u00A3")+"20",Kata.bonusTime(
15       //assertEquals((wellConfigured?"£":"\u00A3")+"78",Kata.bonusTime(
16       //assertEquals((wellConfigured?"£":"\u00A3")+"678900",Kata.bonusT
17     }
18
19     public boolean unicodeTest(){
20       System.out.println("\u00A3 == £:"+"\u00A3".equalsIgnoreCase("£"));
21       System.out.println("if previous result was false or had ? symbol then perhaps you need to escape unicode due to misconf
22       return "\u00A3".equalsIgnoreCase("£");
23     }
24 }
```



Do I get a bonus ?

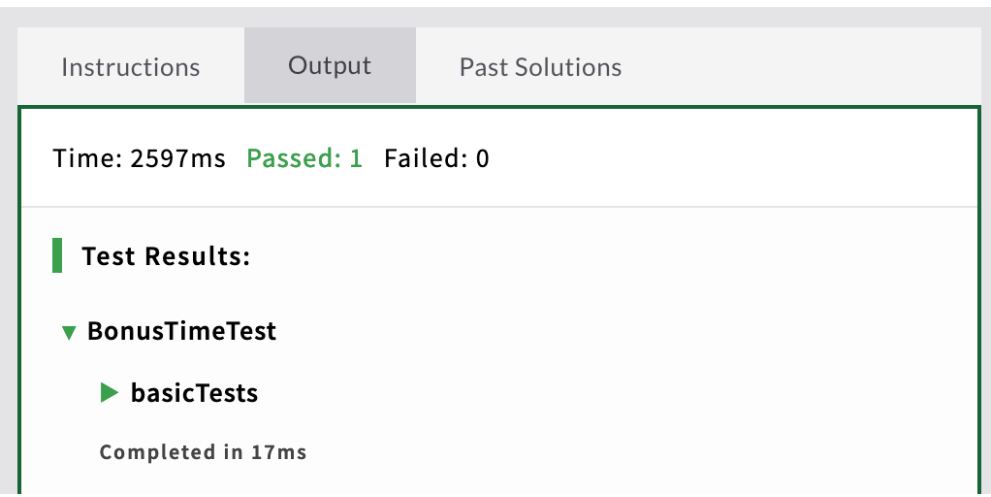
Mes tests

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5
6  public class BonusTimeTest {
7      @Test
8      public void basicTests() {
9        boolean wellConfigured=unicodeTest();
10       assertEquals("£10000",Kata.bonusTime(10000, false));
11       assertEquals((wellConfigured?"£":"\u00A3")+"100000",Kata.bonusTime(1
12
```

Do I get a bonus ?

Implémenter

```
1 public class Kata{
2   public static String bonusTime(final int salary, final boolean bonus) {
3     return "£10000";
4   }
5 }
```



Do I get a bonus ?

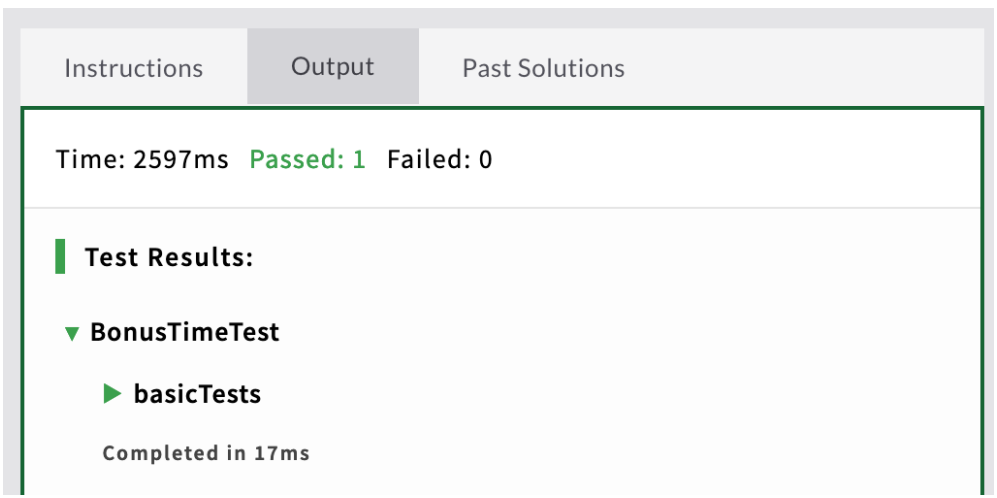
On continue :

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5
6  public class BonusTimeTest {
7      @Test
8      public void basicTests() {
9        boolean wellConfigured=unicodeTest();
10       assertEquals((wellConfigured?"£":"\u00A3")+"100000",Kata.bonusTime(10000, false));
11       assertEquals((wellConfigured?"£":"\u00A3")+"100000",Kata.bonusTime(10000, true));
12       assertEquals((wellConfigured?"£":"\u00A3")+"250000",Kata.bonusTime(25000, true));
13       assertEquals((wellConfigured?"£":"\u00A3")+"10000",Kata.bonusTime(10000, false));
14       assertEquals((wellConfigured?"£":"\u00A3")+"60000",Kata.bonusTime(60000, false));
15       assertEquals((wellConfigured?"£":"\u00A3")+"20",Kata.bonusTime(2, true));
16       assertEquals((wellConfigured?"£":"\u00A3")+"78",Kata.bonusTime(78, false));
17       assertEquals((wellConfigured?"£":"\u00A3")+"678900",Kata.bonusTime(67890, true));
18     }
19
20     public boolean unicodeTest(){
21       System.out.println("\u00A3 == £:"+"\u00A3".equalsIgnoreCase("£"));
22       System.out.println("if previous result was false or had ? symbol then perhaps you need
23       return "\u00A3".equalsIgnoreCase("£");
24     }
```

Do I get a bonus ?

On continue :

```java
public class Kata{
  public static String bonusTime(final int salary, final boolean bonus) {
    int newSalary;

    if(bonus) {
      newSalary = salary * 10;
    } else {
      newSalary = salary;
    }

    return "£" + newSalary;
  }
}
```



Do I get a bonus ?

Le scope :

```
1     int newSalary;
2
3     if(bonus) {
4       newSalary = salary * 10;
5     } else {
6       newSalary = salary;
7     }
8
9     // ici, j'ai accès à newSalary
10
11
12
13
14    // Différent de :
15
16
17    if(bonus) {
18      int newSalary = salary * 10;
19    } else {
20      int newSalary = salary;
21    }
22
23    // ici, je n'ai pas accès à newSalary
24
25
26
```

```
./src/main/java/Kata.java:11: error: cannot find
symbol
    return "£" + newSalary;
                 ^
  symbol:   variable newSalary
  location: class Kata
1 error
```

Do I get a bonus ?

Et si...

```
1 public class Kata{
2   public static String bonusTime(final int salary, final boolean bonus) {
3     if(bonus) {
4       salary = salary * 10;
5     }
6
7     return "£" + newSalary;
8   }
9 }
```

```
./src/main/java/Kata.java:4: error: final
parameter salary may not be assigned
      salary = salary * 10;
      ^
1 error
```

Do I get a bonus ?

Les constantes

```
1 public class Kata{
2   public static String bonusTime(final int salary, final boolean bonus) {
3     if(bonus) {
4       salary = salary * 10;
5     }
6
7     return "£" + newSalary;
8   }
9 }
```

Encore une boîte mais qui n'est accessible qu'en lecture

```
1 final String content = "Elite Clone Trooper";
```

Do I get a bonus ?

Alors, shameless ?!

```
1 public class Kata{
2   public static String bonusTime(int salary, boolean bonus) {
3     if(bonus) { salary *= 10; }
4     return "£" + salary;
5   }
6 }
```

- Pourquoi ça a été fait comme ça ?
- Les gens ne sont pas bêtes (majoritairement)
- Lire la documentation
- Interroger l'équipe !
- Regarder l'historique (git)

Do I get a bonus ?

Encore des opérateurs...

```
1 // Opérateurs d'assignation
2 int a = 0;
3
4 a += 12;
5 a /= 3;
6 a *= 3;
7 a -= 1;
8
9 // Quelle valeur ?
```

Do I get a bonus ?

Les opérateurs d'assignation

```
1  // Opérateurs d'assignation
2  int a = 0;
3
4  a += 12; // a = a + 12
5  a /= 4; // a = a / 4
6  a *= 3; // a = a * 3
7  a -= 1; // a = a - 1
8
9  // Quelle valeur ?
10
11 // résultat : 8
```

L'opérateur **=** permet d'**assigner** une valeur

L'opérateur **+=** permet de **réassigner** en **additionnant**

L'opérateur **-=** permet de **réassigner** en **soustrayant**

L'opérateur ***=** permet de **réassigner** en **multipliant**

L'opérateur **/=** permet de **réassigner** en **divisant**

Fizz Buzz Cuckoo Clock

Diviser les tests (cas simple)

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5  public class ExampleTestCases {
6      @Test
7      public void SomeBasicTimesTests () {
8        System.out.println("Testing with time " + "08:00");
9        assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:00"));
10       System.out.println("Testing with time " + "03:00");
11       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:00"));
12
13       // System.out.println("Testing with time " + "13:34");
14       // assertEquals("tick", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:34"));
15       // System.out.println("Testing with time " + "11:15");
16       // assertEquals("Fizz Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("11:15"));
17       // System.out.println("Testing with time " + "03:03");
18       // assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:03"));
19       // System.out.println("Testing with time " + "14:30");
20       // assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("14:30"));
21       // System.out.println("Testing with time " + "08:55");
22       // assertEquals("Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:55"));
23       // System.out.println("Testing with time " + "00:00");
24       // assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.
25       // System.out.println("Testing with time " + "12:00");
26       // assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.
27   }
28 }
```

Fizz Buzz Cuckoo Clock

Cas simple :

```
1 public class FizzBuzzCuckooClock {
2     public static String fizzBuzzCuckooClock(String time) {
3         int xCuckoo = Integer.parseInt(time.substring(0,2));
4 
5         return "Cuckoo ".repeat(xCuckoo - 1) + "Cuckoo";
6     }
7 }
```

Fizz Buzz Cuckoo Clock

Ajout des cas p.m :

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5  public class ExampleTestCases {
6      @Test
7      public void SomeBasicTimesTests () {
8        System.out.println("Testing with time " + "08:00");
9        assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:00"));
10       System.out.println("Testing with time " + "03:00");
11       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:00"));
12       System.out.println("Testing with time " + "13:00");
13       assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:00"));
14       System.out.println("Testing with time " + "15:00");
15       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("15:00"));
16
17       // System.out.println("Testing with time " + "13:34");
18       // assertEquals("tick", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:34"));
19       // System.out.println("Testing with time " + "11:15");
20       // assertEquals("Fizz Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("11:15"));
21       // System.out.println("Testing with time " + "03:03");
22       // assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:03"));
23       // System.out.println("Testing with time " + "14:30");
24       // assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("14:30"));
25       // System.out.println("Testing with time " + "08:55");
26       // assertEquals("Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:55"));
27       // System.out.println("Testing with time " + "00:00");
28       // assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.
29       // System.out.println("Testing with time " + "12:00");
30       // assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.
31   }
32 }
```

Fizz Buzz Cuckoo Clock

Ajout des p.m :

```
1 public class FizzBuzzCuckooClock {
2     public static String fizzBuzzCuckooClock(String time) {
3         int xCuckoo = Integer.parseInt(time.substring(0,2));
4         if(xCuckoo > 12) { xCuckoo -= 12; }
5 
6         return "Cuckoo ".repeat(xCuckoo - 1) + "Cuckoo";
7     }
8 }
```

Fizz Buzz Cuckoo Clock

Ha ce démon de minuit...

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5  public class ExampleTestCases {
6      @Test
7      public void SomeBasicTimesTests () {
8        System.out.println("Testing with time " + "08:00");
9        assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:00"));
10       System.out.println("Testing with time " + "03:00");
11       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:00"));
12       System.out.println("Testing with time " + "13:00");
13       assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:00"));
14       System.out.println("Testing with time " + "15:00");
15       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("15:00"));
16       System.out.println("Testing with time " + "00:00");
17       assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fiz
18
19       // System.out.println("Testing with time " + "13:34");
20       // assertEquals("tick", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:34"));
21       // System.out.println("Testing with time " + "11:15");
22       // assertEquals("Fizz Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("11:15"));
23       // System.out.println("Testing with time " + "03:03");
24       // assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:03"));
25       // System.out.println("Testing with time " + "14:30");
26       // assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("14:30"));
27       // System.out.println("Testing with time " + "08:55");
28       // assertEquals("Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:55"));
29       // System.out.println("Testing with time " + "00:00");
30       // assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.
31       // System.out.println("Testing with time " + "12:00");
32       // assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.
33   }
34 }
```

Fizz Buzz Cuckoo Clock

Haa ce démon de minuit...

```
1 public class FizzBuzzCuckooClock {
2     public static String fizzBuzzCuckooClock(String time) {
3         int xCuckoo = Integer.parseInt(time.substring(0,2));
4         if(xCuckoo > 12) { xCuckoo -= 12; }
5         if(xCuckoo == 0) { xCuckoo = 12; }
6 
7         return "Cuckoo ".repeat(xCuckoo - 1) + "Cuckoo";
8     }
9 }
```

Fizz Buzz Cuckoo Clock

Toujours des opérateurs

```
1 public class FizzBuzzCuckooClock {
2     public static String fizzBuzzCuckooClock(String time) {
3         int xCuckoo = Integer.parseInt(time.substring(0,2));
4         if(xCuckoo > 12) { xCuckoo -= 12; }
5         if(xCuckoo == 0) { xCuckoo = 12; }
6 
7         return "Cuckoo ".repeat(xCuckoo - 1) + "Cuckoo";
8     }
9 }
```

Fizz Buzz Cuckoo Clock

Encore des opérateurs de comparaison

```
1 // Opérateurs de comparaison
2 int a = 0;
3
4 a == 12; // retourne false
5 a != 4; // retourne true
6 a <= 3; // retourne true
7 a >= 1; // retourne false
```

L'opérateur **==** permet de **vérifier l'égalité**

L'opérateur **!=** permet de **vérifier l'inégalité**

L'opérateur **<=** permet de **vérifier l'infériorité**

L'opérateur **>=** permet de **vérifier la supériorité**

Fizz Buzz Cuckoo Clock

Ajoutons Fizz !

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5  public class ExampleTestCases {
6      @Test
7      public void SomeBasicTimesTests () {
8        System.out.println("Testing with time " + "08:00");
9        assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:00"));
10       System.out.println("Testing with time " + "03:00");
11       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:00"));
12       System.out.println("Testing with time " + "13:00");
13       assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:00"));
14       System.out.println("Testing with time " + "15:00");
15       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("15:00"));
16       System.out.println("Testing with time " + "00:00");
17       assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fiz
18       System.out.println("Testing with time " + "02:03");
19       assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("02:03"));
20
21
22       // System.out.println("Testing with time " + "13:34");
23       // assertEquals("tick", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:34"));
24       // System.out.println("Testing with time " + "11:15");
25       // assertEquals("Fizz Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("11:15"));
26       // System.out.println("Testing with time " + "03:03");
27       // assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:03"));
28       // System.out.println("Testing with time " + "14:30");
29       // assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("14:30"));
30       // System.out.println("Testing with time " + "08:55");
31       // assertEquals("Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:55"));
32       // System.out.println("Testing with time " + "00:00");
33       // assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.
34       // System.out.println("Testing with time " + "12:00");
35       // assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.
```

Fizz Buzz Cuckoo Clock

Ajoutons Fizz !

```
1  public class FizzBuzzCuckooClock {
2      public static String fizzBuzzCuckooClock(String time) {
3          int minutes = Integer.parseInt(time.substring(3,5));
4
5          if(minutes == 3) { return "Fizz"; }
6
7          int xCuckoo = Integer.parseInt(time.substring(0,2));
8          if(xCuckoo > 12) { xCuckoo -= 12; }
9          if(xCuckoo == 0) { xCuckoo = 12; }
10
11         return "Cuckoo ".repeat(xCuckoo - 1) + "Cuckoo";
12     }
13 }
```

Implémenter uniquement le test !

Fizz Buzz Cuckoo Clock

Plus de Fizz !

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5  public class ExampleTestCases {
6      @Test
7      public void SomeBasicTimesTests () {
8        System.out.println("Testing with time " + "08:00");
9        assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:00"));
10       System.out.println("Testing with time " + "03:00");
11       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:00"));
12       System.out.println("Testing with time " + "13:00");
13       assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:00"));
14       System.out.println("Testing with time " + "15:00");
15       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("15:00"));
16       System.out.println("Testing with time " + "00:00");
17       assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fiz
18       System.out.println("Testing with time " + "02:03");
19       assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("02:03"));
20       System.out.println("Testing with time " + "02:06");
21       assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("02:06"));
22
23       // System.out.println("Testing with time " + "13:34");
24       // assertEquals("tick", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:34"));
25       // System.out.println("Testing with time " + "11:15");
26       // assertEquals("Fizz Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("11:15"));
27       // System.out.println("Testing with time " + "03:03");
28       // assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:03"));
29       // System.out.println("Testing with time " + "14:30");
30       // assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("14:30"));
31       // System.out.println("Testing with time " + "08:55");
32       // assertEquals("Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:55"));
33       // System.out.println("Testing with time " + "00:00");
34       // assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.
35       // System.out.println("Testing with time " + "12:00");
```

Fizz Buzz Cuckoo Clock

Plus de Fizz ! (un peu trop même)

```
1  public class FizzBuzzCuckooClock {
2      public static String fizzBuzzCuckooClock(String time) {
3          int minutes = Integer.parseInt(time.substring(3,5));
4
5          if(minutes == 0) {
6            int xCuckoo = Integer.parseInt(time.substring(0,2));
7            if(xCuckoo > 12) { xCuckoo -= 12; }
8            if(xCuckoo == 0) { xCuckoo = 12; }
9
10           return "Cuckoo ".repeat(xCuckoo - 1) + "Cuckoo";
11         } else {
12           return "Fizz";
13         }
14     }
15 }
```

Ça passe ! (shameless green)

Fizz Buzz Cuckoo Clock

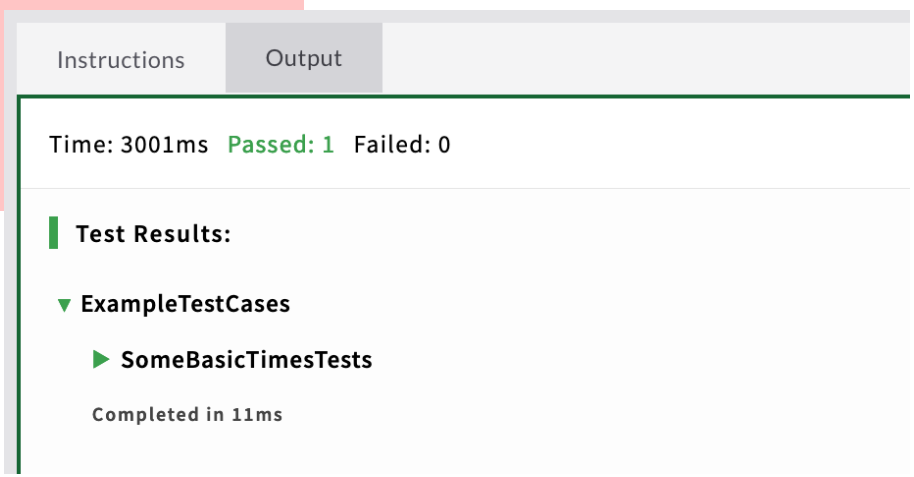
Avec Buzz maintenant !

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5  public class ExampleTestCases {
6      @Test
7      public void SomeBasicTimesTests () {
8        System.out.println("Testing with time " + "08:00");
9        assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:00"));
10       System.out.println("Testing with time " + "03:00");
11       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:00"));
12       System.out.println("Testing with time " + "13:00");
13       assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:00"));
14       System.out.println("Testing with time " + "15:00");
15       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("15:00"));
16       System.out.println("Testing with time " + "00:00");
17       assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fiz
18       System.out.println("Testing with time " + "02:03");
19       assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("02:03"));
20       System.out.println("Testing with time " + "02:06");
21       assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("02:06"));
22       System.out.println("Testing with time " + "02:05");
23       assertEquals("Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("02:05"));
24
25       // System.out.println("Testing with time " + "13:34");
26       // assertEquals("tick", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:34"));
27       // System.out.println("Testing with time " + "11:15");
28       // assertEquals("Fizz Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("11:15"));
29       // System.out.println("Testing with time " + "03:03");
30       // assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:03"));
31       // System.out.println("Testing with time " + "14:30");
32       // assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("14:30"));
33       // System.out.println("Testing with time " + "08:55");
34       // assertEquals("Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:55"));
35       // System.out.println("Testing with time " + "00:00");
```

Fizz Buzz Cuckoo Clock

Avec Buzz maintenant !

```
1  public class FizzBuzzCuckooClock {
2      public static String fizzBuzzCuckooClock(String time) {
3          int minutes = Integer.parseInt(time.substring(3,5));
4
5          if(minutes == 0) {
6            int xCuckoo = Integer.parseInt(time.substring(0,2));
7            if(xCuckoo > 12) { xCuckoo -= 12; }
8            if(xCuckoo == 0) { xCuckoo = 12; }
9
10           return "Cuckoo ".repeat(xCuckoo - 1) + "Cuckoo";
11         } else {
12           if(minutes % 3 == 0) {
13             return "Fizz";
14           } else {
15             return "Buzz";
16           }
17
18         }
19     }
20 }
```



Fizz Buzz Cuckoo Clock

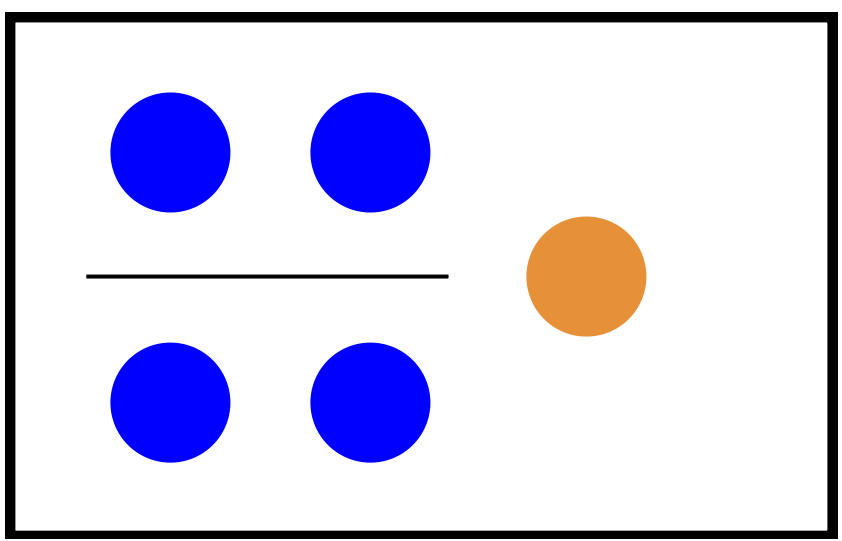
Modulo

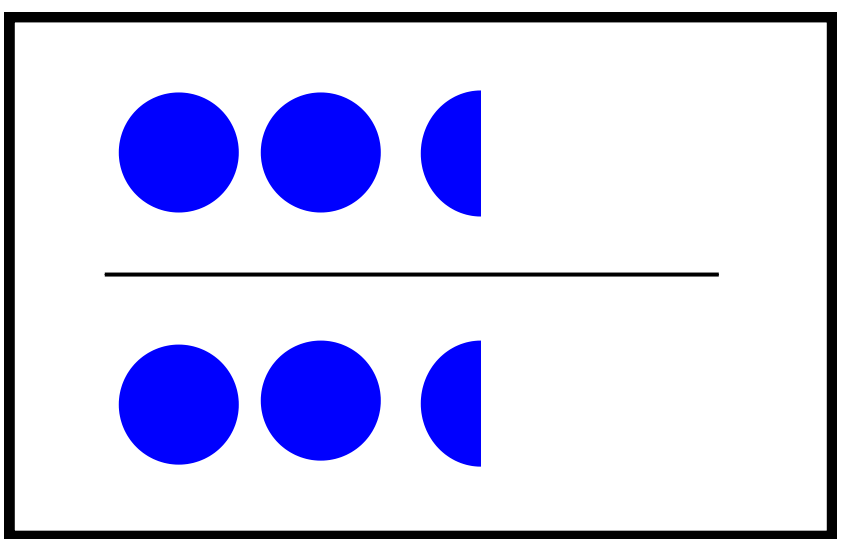
Modulo permet d'afficher le reste d'une division euclidienne

4 % 2 | 2 * 2 reste 0 modulo vaut 0

5 % 2 | 2 * 2 reste 1 modulo vaut 1

Division Euclidienne | Division décimale





Fizz Buzz Cuckoo Clock

Puis avec Fizz Buzz

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5  public class ExampleTestCases {
6      @Test
7      public void SomeBasicTimesTests () {
8        System.out.println("Testing with time " + "08:00");
9        assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:00")
10       System.out.println("Testing with time " + "03:00");
11       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:00"));
12       System.out.println("Testing with time " + "13:00");
13       assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:00"));
14       System.out.println("Testing with time " + "15:00");
15       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("15:00"));
16       System.out.println("Testing with time " + "00:00");
17       assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.
18
19       System.out.println("Testing with time " + "13:03");
20       assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:03"));
21
22       System.out.println("Testing with time " + "02:06");
23       assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("02:06"));
24
25       System.out.println("Testing with time " + "02:05");
26       assertEquals("Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("02:05"));
27
28       System.out.println("Testing with time " + "02:05");
29       assertEquals("Fizz Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("02:15"));
30       // System.out.println("Testing with time " + "11:15");
31       // assertEquals("Fizz Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("11:15"));
32       // System.out.println("Testing with time " + "03:03");
33       // assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:03"));
```

Fizz Buzz Cuckoo Clock

Puis avec Fizz Buzz

```java
public class FizzBuzzCuckooClock {
    public static String fizzBuzzCuckooClock(String time) {
        int minutes = Integer.parseInt(time.substring(3,5));

        if(minutes == 0) {
          int xCuckoo = Integer.parseInt(time.substring(0,2));
          if(xCuckoo > 12) { xCuckoo -= 12; }
          if(xCuckoo == 0) { xCuckoo = 12; }

          return "Cuckoo ".repeat(xCuckoo - 1) + "Cuckoo";
        } else {
          String res = "";

          if(minutes % 3 == 0) {
            res += "Fizz ";
          }
          if(minutes % 5 == 0) {
            res += "Buzz";
          }

          return res.trim();
        }
    }
}
```

Fizz Buzz Cuckoo Clock

Enfin avec tick

```
1  import org.junit.Test;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.runners.JUnit4;
4
5  public class ExampleTestCases {
6      @Test
7      public void SomeBasicTimesTests () {
8        System.out.println("Testing with time " + "08:00");
9        assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:00"));
10       System.out.println("Testing with time " + "03:00");
11       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:00"));
12       System.out.println("Testing with time " + "13:00");
13       assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:00"));
14       System.out.println("Testing with time " + "15:00");
15       assertEquals("Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("15:00"));
16       System.out.println("Testing with time " + "00:00");
17       assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("00:00"));
18       System.out.println("Testing with time " + "13:03");
19       assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("13:03"));
20       System.out.println("Testing with time " + "02:06");
21       assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("02:06"));
22       System.out.println("Testing with time " + "02:05");
23       assertEquals("Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("02:05"));
24       System.out.println("Testing with time " + "02:15");
25       assertEquals("Fizz Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("02:15"));
26       System.out.println("Testing with time " + "02:01");
27       assertEquals("tick", FizzBuzzCuckooClock.fizzBuzzCuckooClock("02:01"));
28       System.out.println("Testing with time " + "11:15");
29       assertEquals("Fizz Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("11:15"));
30       System.out.println("Testing with time " + "03:03");
31       assertEquals("Fizz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("03:03"));
32       System.out.println("Testing with time " + "14:30");
33       assertEquals("Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("14:30"));
34       System.out.println("Testing with time " + "08:55");
35       assertEquals("Buzz", FizzBuzzCuckooClock.fizzBuzzCuckooClock("08:55"));
36       System.out.println("Testing with time " + "00:00");
37       assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("00:00"));
38       System.out.println("Testing with time " + "12:00");
39       assertEquals("Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo Cuckoo", FizzBuzzCuckooClock.fizzBuzzCuckooClock("12:00"));
40   }
41 }
```

Fizz Buzz Cuckoo Clock

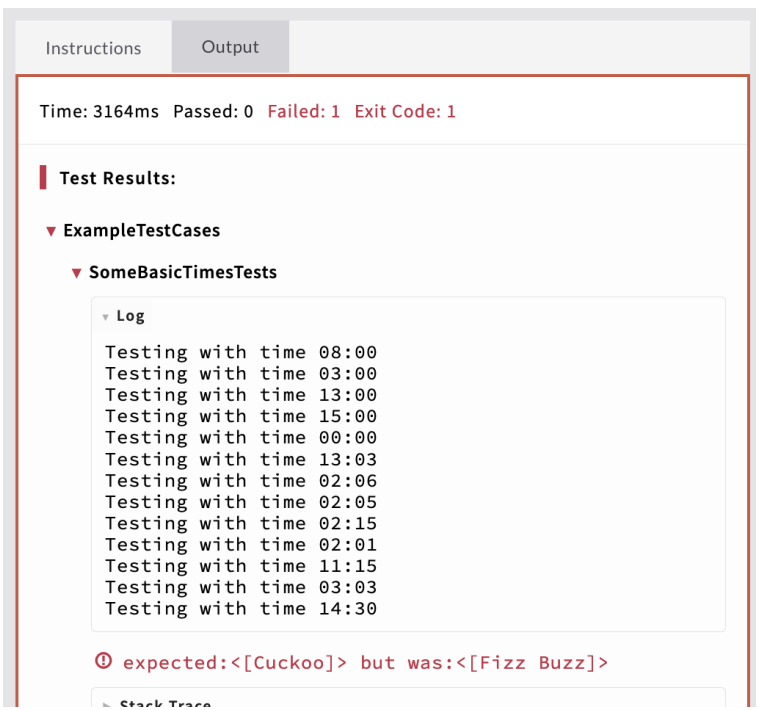
Enfin avec tick

```
1  public class FizzBuzzCuckooClock {
2      public static String fizzBuzzCuckooClock(String time) {
3          int minutes = Integer.parseInt(time.substring(3,5));
4
5          if(minutes == 0) {
6            int xCuckoo = Integer.parseInt(time.substring(0,2));
7            if(xCuckoo > 12) { xCuckoo -= 12; }
8            if(xCuckoo == 0) { xCuckoo = 12; }
9
10           return "Cuckoo ".repeat(xCuckoo - 1) + "Cuckoo";
11         } else {
12           String res = "";
13
14           if(minutes % 3 == 0) {
15             res += "Fizz ";
16           }
17           if(minutes % 5 == 0) {
18             res += "Buzz";
19           }
20
21           return res.length() > 0 ? res.trim() : "tick";
22         }
23     }
24 }
```

Fizz Buzz Cuckoo Clock

Enfin avec tick... Oups...

```java
public class FizzBuzzCuckooClock {
    public static String fizzBuzzCuckooClock(String time) {
        int minutes = Integer.parseInt(time.substring(3,5));

        if(minutes == 0) {
          int xCuckoo = Integer.parseInt(time.substring(0,2));
          if(xCuckoo > 12) { xCuckoo -= 12; }
          if(xCuckoo == 0) { xCuckoo = 12; }

          return "Cuckoo ".repeat(xCuckoo - 1) + "Cuckoo";
        } else {
          String res = "";

          if(minutes % 3 == 0) {
            res += "Fizz ";
          }
          if(minutes % 5 == 0) {
            res += "Buzz";
          }

          return res.length() > 0 ? res.trim() : "tick";
        }
    }
}
```



Fizz Buzz Cuckoo Clock

Enfin avec tick... Oups...

```java
public class FizzBuzzCuckooClock {
    public static String fizzBuzzCuckooClock(String time) {
        int minutes = Integer.parseInt(time.substring(3,5));

        if(minutes == 0) {
          int xCuckoo = Integer.parseInt(time.substring(0,2));
          if(xCuckoo > 12) { xCuckoo -= 12; }
          if(xCuckoo == 0) { xCuckoo = 12; }

          return "Cuckoo ".repeat(xCuckoo - 1) + "Cuckoo";
        } else {
          if(minutes == 30) { return "Cuckoo"; }

          String res = "";

          if(minutes % 3 == 0) {
            res += "Fizz ";
          }
          if(minutes % 5 == 0) {
            res += "Buzz";
          }

          return res.length() > 0 ? res.trim() : "tick";
        }
    }
}
```

Fizz Buzz Cuckoo Clock

Refacto ensemble !

```
1  public class FizzBuzzCuckooClock {
2      public static String fizzBuzzCuckooClock(String time) {
3          int minutes = Integer.parseInt(time.substring(3,5));
4
5          if(minutes == 0) {
6            int xCuckoo = Integer.parseInt(time.substring(0,2));
7            if(xCuckoo > 12) { xCuckoo -= 12; }
8            if(xCuckoo == 0) { xCuckoo = 12; }
9
10           return "Cuckoo ".repeat(xCuckoo - 1) + "Cuckoo";
11         } else {
12           if(minutes == 30) { return "Cuckoo"; }
13
14           String res = "";
15
16           if(minutes % 3 == 0) {
17             res += "Fizz ";
18           }
19           if(minutes % 5 == 0) {
20             res += "Buzz";
21           }
22
23           return res.length() > 0 ? res.trim() : "tick";
24         }
25     }
26 }
```

Fizz Buzz Cuckoo Clock

Refacto ensemble !

```
1  public class FizzBuzzCuckooClock {
2      public static String fizzBuzzCuckooClock(String time) {
3          int minutes = Integer.parseInt(time.substring(3));
4
5          if(minutes == 0) {
6            int xCuckoo = Integer.parseInt(time.substring(0,2));
7            if(xCuckoo > 12) xCuckoo -= 12;
8            if(xCuckoo == 0) xCuckoo = 12;
9
10           return "Cuckoo ".repeat(xCuckoo - 1) + "Cuckoo";
11         } else if(minutes == 30) {
12                 return "Cuckoo";
13         } else {
14           String res = "";
15
16           if(minutes % 3 == 0) res += "Fizz ";
17           if(minutes % 5 == 0) res += "Buzz";
18
19           return res.length() > 0 ? res.trim() : "tick";
20         }
21     }
22 }
```