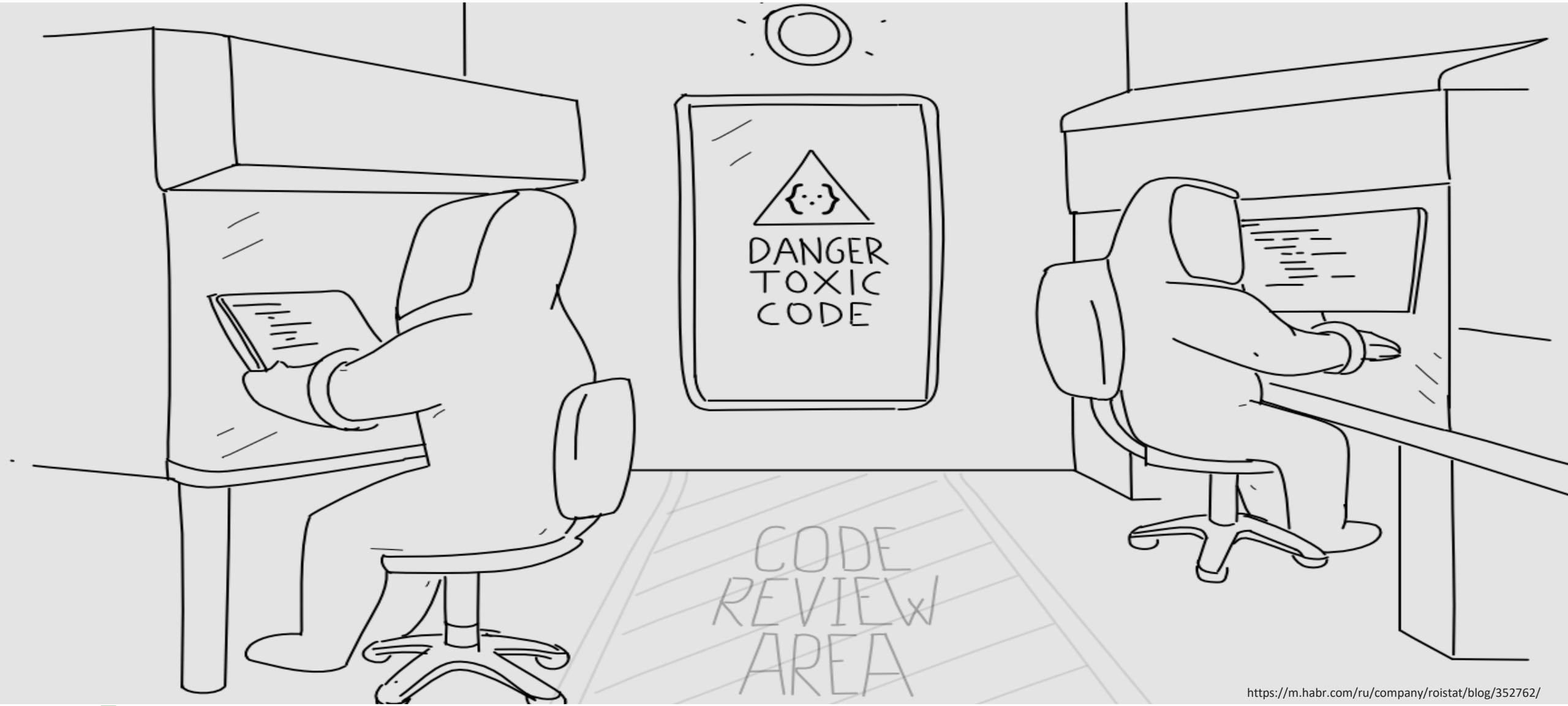
Java Training
Code Convention

Agenda

- Code convention
- Unit testing
- Test driven development
- Arrays

Why Have Code Conventions?



https://m.habr.com/ru/company/roistat/blog/352762/

Why Have Code Conventions?

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.

Naming Variables and Constants

Variables should be :

- *Short and meaningful*
- *camelCase*
- ***not** start with underscore "_" or dollar sign '$' characters*
- *One-character variable names should be avoided except for temporary variables*


```
double firstNumber = 2.3;
int i = 0;
String userName = "Elvis";
static final int MIN_WIDTH = 4;
static final String NAME = "Elvis";
```

Constants should be:

- ***All uppercase** with words separated by underscores "_"*


```
double Number1 = 2.3;
int a1 = 1;
String nameOfUser = "Elvis";
```

Naming Methods and Classes

Methods should be :

- *Verbs*
- *camelCase*
- *Clear meaning*


```
void changeGear(int value);
void speedUp(int increment);
interface Bicycle {
}
class MountainBike {
}
```

Classes and Interfaces should be:

- *Noun*
- *PascalCase*
- *A whole word, not abbreviation.*


```
void gear(int value);

interface IBicycle {
}
class mountainBike {
}
```

Replace Magic Number with Symbolic Constant

Problem

Your code uses a number that has a certain meaning to it.

Solution

Replace this number with a constant that has a human-readable name explaining the meaning of the number.

```
double potentialEnergy(double mass, double height) {
  return mass * height * 9.81;
}
```


```
static final double GRAVITATIONAL_CONSTANT = 9.81;

double calculateEnergy(double mass, double height) {
  return mass * height * GRAVITATIONAL_CONSTANT;
}
```

Formatting

`while (i < 5)`
`{`
`  // some code`
`}`
	`while (i < 5) {`
	`  // some code`
	`}`

`void changeGear(int value)`
`{`
`  // some code`
`}`
	`void changeGear(int value) {`
	`  // some code`
	`}`

`sample.getSomething().getData()`	`Something value = sample.getSomething();` `Data data = value.geData();`

`int count;`
`…`
`count = 0`
	`int count = 0;`

Deep nesting structure should be avoided

- *Not more than 3 nested statements or loops*
- In rare cases *max 5 nested statements or loops*

https://morioh.com/p/35b3207b558e

```
if (firstCondition) {
    if (secondCondition) {
        if (thirdCondition) {
            // do something
        }
    }
}
```

```
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; k++) {
        for (int k = 0; k < 5; k++) {
            // do something
        }
    }
}
```

Nested Conditional VS Guard Clauses

MORE NESTED

```
Double calculate(Double value) {
  Double result;
  if (value != null) {
    // do something
  } else {
    throw new UserException();
  }
  return result;
}
```

LESS NESTED

```
Double calculate(Double value) {
  if (value == null) {
    throw new UserException();
  }
  // do something
  return result;
}
```

https://www.pcsteps.com/508-nested-virtualization/

Extract Method

```
public void print() {
  printBanner();

  // Print details
  System.out.println("name: " + name);
  System.out.println("amount: " + getOutstanding());
}
```

Method **BEFORE**

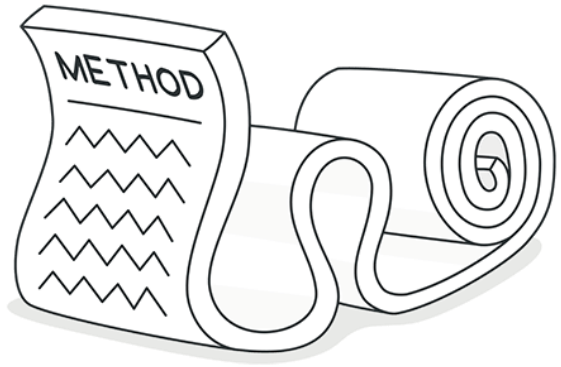
Method **AFTER**

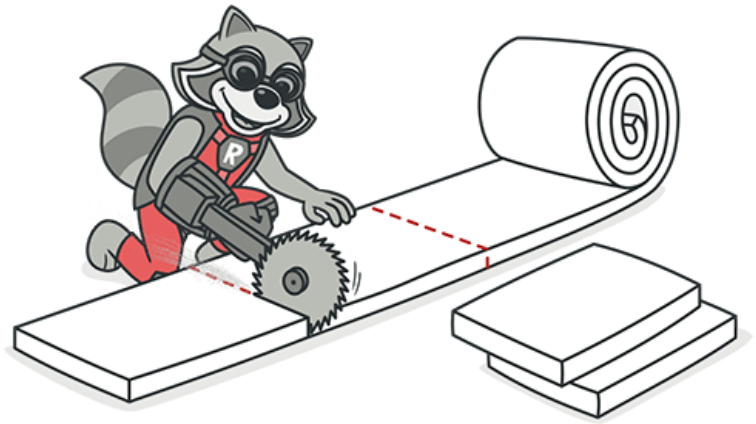
```
public void print() {
  printBanner();
  Details details = getOutstanding();
  printDetails(details);
}

private void printDetails(double outstanding) {
  System.out.println("name: " + name);
  System.out.println("amount: " + outstanding);
}
```

https://webformyself.com/krutoe-ispolzovanie-psevdoelementov-css-before-i-after/

Methods summary

- Method should be not more then 20 lines (50 lines maximum)
- Method should fit your screen
- Use guard clauses to reduce nesting
- Decompose method using private methods





https://refactoring.guru/smells/long-method


Unit testing

Why we need tests?

- Unfortunately, when a programmer writes a code, it does not always work as expected, although having tested it with our own hands, we can think so.

Reasons for code failure

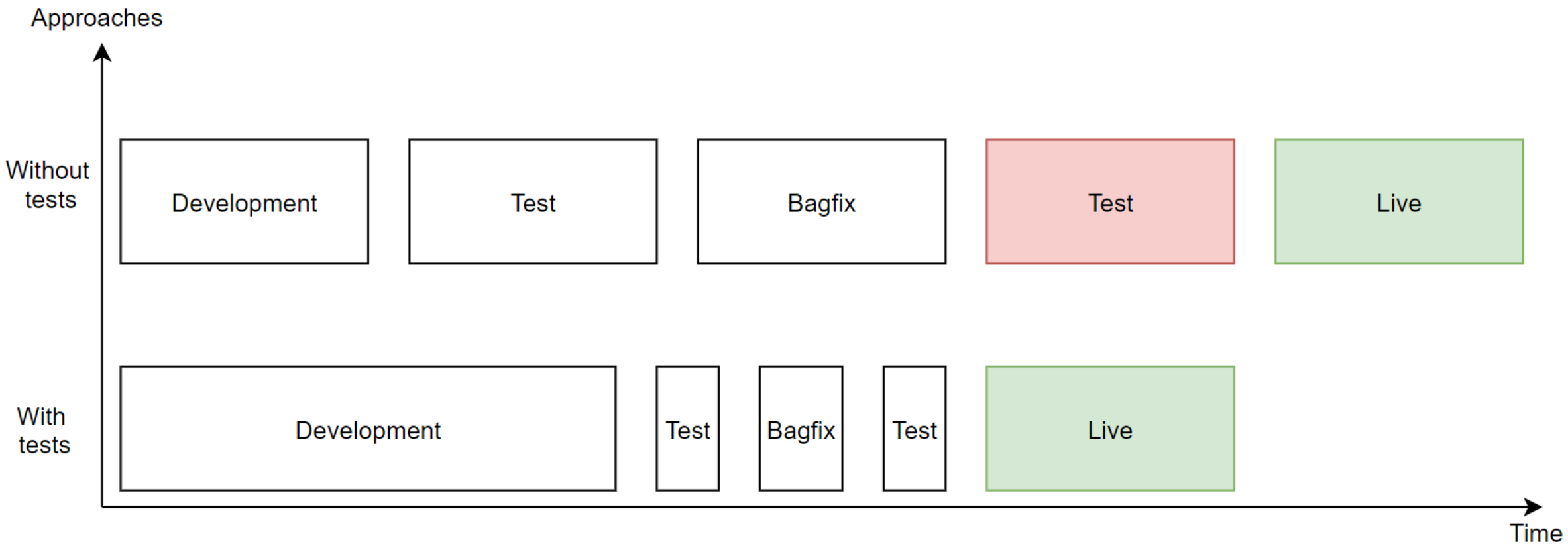
You forgot some little thing

Someone else forgot some little thing

Impressions of using tests

- You are 100% sure that your code is correct because you have tests.
- Tests are an example of using your code
- Saving time in perspective

Feature release



https://youtu.be/8u6_hctdhqI


JUnit

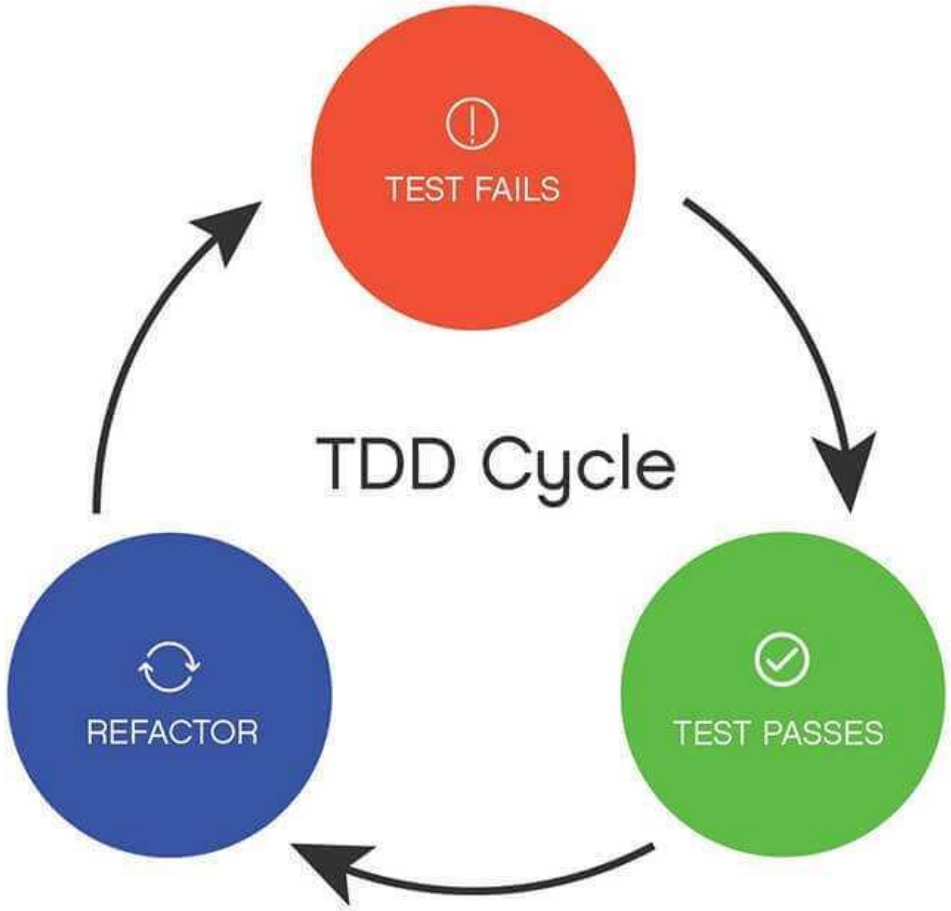

mockito

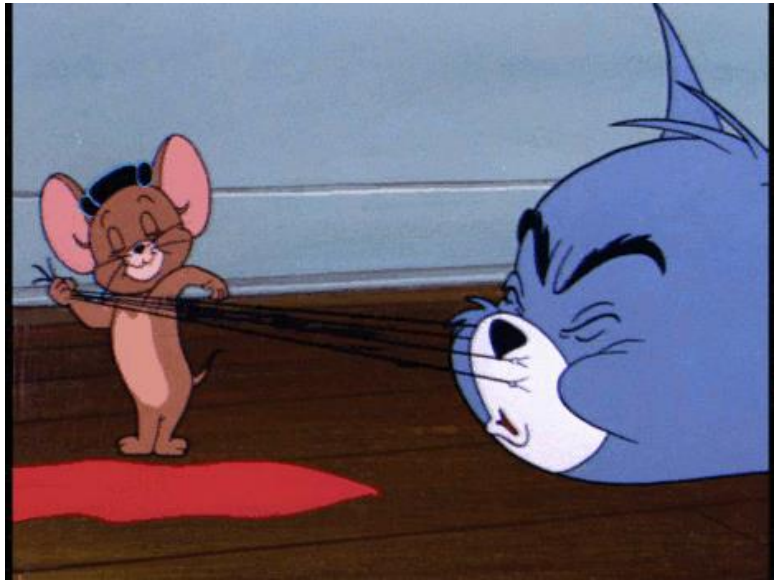

TestNG


PowerMock


What is Test Driven Development?

RED-GREEN-REFACTORING





Test First

```
public class CalculatorTest {

    @Test
    public void testAddShouldAddWhenNumbersPositive() {

        Calculator calculator = new Calculator();

        int result = calculator.add(2, 3);

        Assert.assertEquals(5, result);
    }
}
```

Minimal possible implementation

```
public class Calculator {
    public int add(int first, int second) {
        return 5;
    }
}
```


Test naming

Unit Test Naming Conventions

- *Test name should express a specific requirement*
- *Test name could include the expected input or state and the expected result for that input or state*
- Test name should be presented as a statement or fact of life that expresses workflows and outputs
- Test name could include the name of the tested method and what exactly it tests

```
@Test
public void testAddShouldAddWhenNumbersPositive() {
    Calculator calculator = new Calculator();
    int result = calculator.add(5, 9);
    assertEquals(14, result);
}
```

https://medium.com/@stefanovskyi/unit-test-naming-conventions-dd9208eadbea

https://www.codeaffine.com/2014/03/17/getting-junit-test-names-right/


Test structure

Test structure

```
@Test
public void testAddShouldAddWhenNumbersPositive() {
    // given (pre-conditions)
    Calculator calculator = new Calculator();
    // when (is always one line!)
    int result = calculator.add(5, 9, operation);
    // then (post-conditions)
    assertEquals(14, result);
}
```

Positive and negative scenarios

- Positive cases are when we are testing with valid data and await a success result
- Negative cases are when we are testing with invalid data and await a failure

https://techbeacon.com/app-dev-testing/why-negative-testing-matters-how-avoid-your-next-fail

Negative example

```
@Test (expected = IllegalArgumentException.class) // then
public void testAddShouldThrowExceptionWhenNumbersTooLarge() {
    // given
    int firstNumber = Integer.MAX_VALUE;
    int secondNumber = Integer.MAX_VALUE;
    // when
    calculator.add(firstNumber, secondNumber);
}
```

Negative example

```
@Test
public void testAddShouldThrowExceptionWhenNumbersTooLarge() {
    // given
    int firstNumber = Integer.MAX_VALUE;
    int secondNumber = Integer.MAX_VALUE;
    // when
    Executable executable = () -> calculator.add(firstNumber, secondNumber);
    // then
    assertThrows(IllegalArgumentException.class, executable);
}
```


Data providers

Data provider example

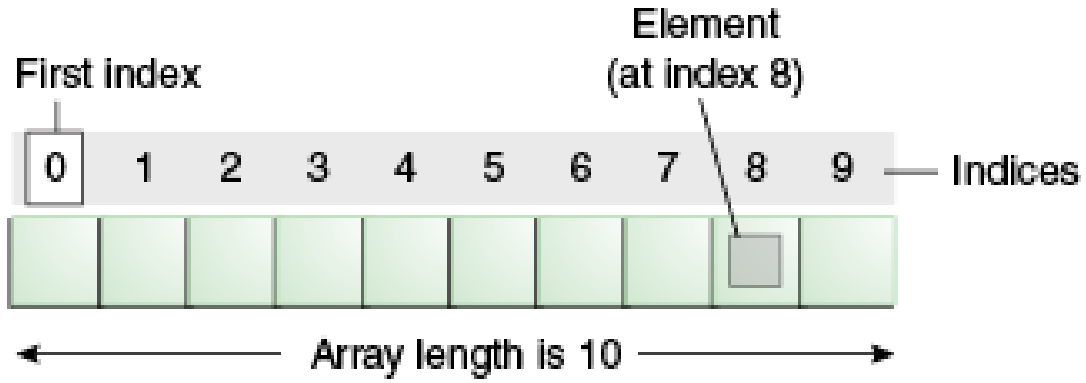
```
@ParameterizedTest
@MethodSource("sumTwoNumbersDataSource")
public void testAddShouldAddWhenNumbersPositive(int first, int second, int expected) {
    double result = calculator.add(first, second);
    assertEquals(expected, result);
}

 private static Stream<Arguments> sumTwoNumbersDataSource() {
     return Stream.of(
         Arguments.of(2, 3, 5),
         Arguments.of(214, 3, 217),
         Arguments.of(1000, 2000, 3000)
     );
 }
```

TestNG


Arrays

What is Array?

- Java provides a data structure, the array, which stores a fixed-size sequential collection of items of the same type.
- Array is index based, first item of the array is stored at 0 index.



Declaring array

- `int[] numbers = new int[10];`
- `int[] numbers = new int[] {1, 2, 3};`
- `int numbers[] = {1, 2, 3, 0, -1, -2, -3};`

The values of items of uninitialized arrays for which memory is allocated are set to zero.

Get array items

To access the i-th item of an array, use the [] operator.

- Get first element in array

```
int[] numbers = new int[] {1, 2, 3};    // declaring array
int first = numbers[0];                 // 0 – first index in array
System.out.println(first);              // print number 1 in console
```

- To sequentially obtain the items of an array, loops are used

```
int[] numbers = new int[] {1, 2, 3};          // declaring array
for (int i = 0; i < numbers.length; i++) {    // loop start
   System.out.println(numbers[i]);            // print array items
}
```


POJO

Plain Old Java Object (POJO)

```java
public class Point {

    private double x;
    private double y;

    public Point() {
    }

    public Point(final double x, final double y) {
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }

    public void setX(final double x) {
        this.x = x;
    }

    public double getY() {
        return y;
    }

    public void setY(final double y) {
        this.y = y;
    }
}
```

Homework

- Fork from repo 

- To task according to description and project structure
- Write tests

Hint: Array should be a POJO class (desirably immutable)

https://github.com/filippstankevich/array

https://refactoring.guru/smells/long-method

Next class

- Object and classes
- Inheritance
- Equals and hash code

https://refactoring.guru/smells/long-method

Thanks