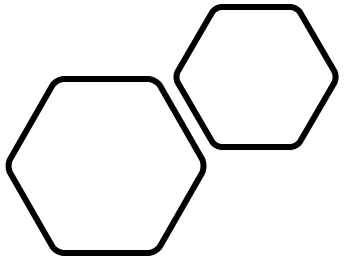
Introdução ao JUnit



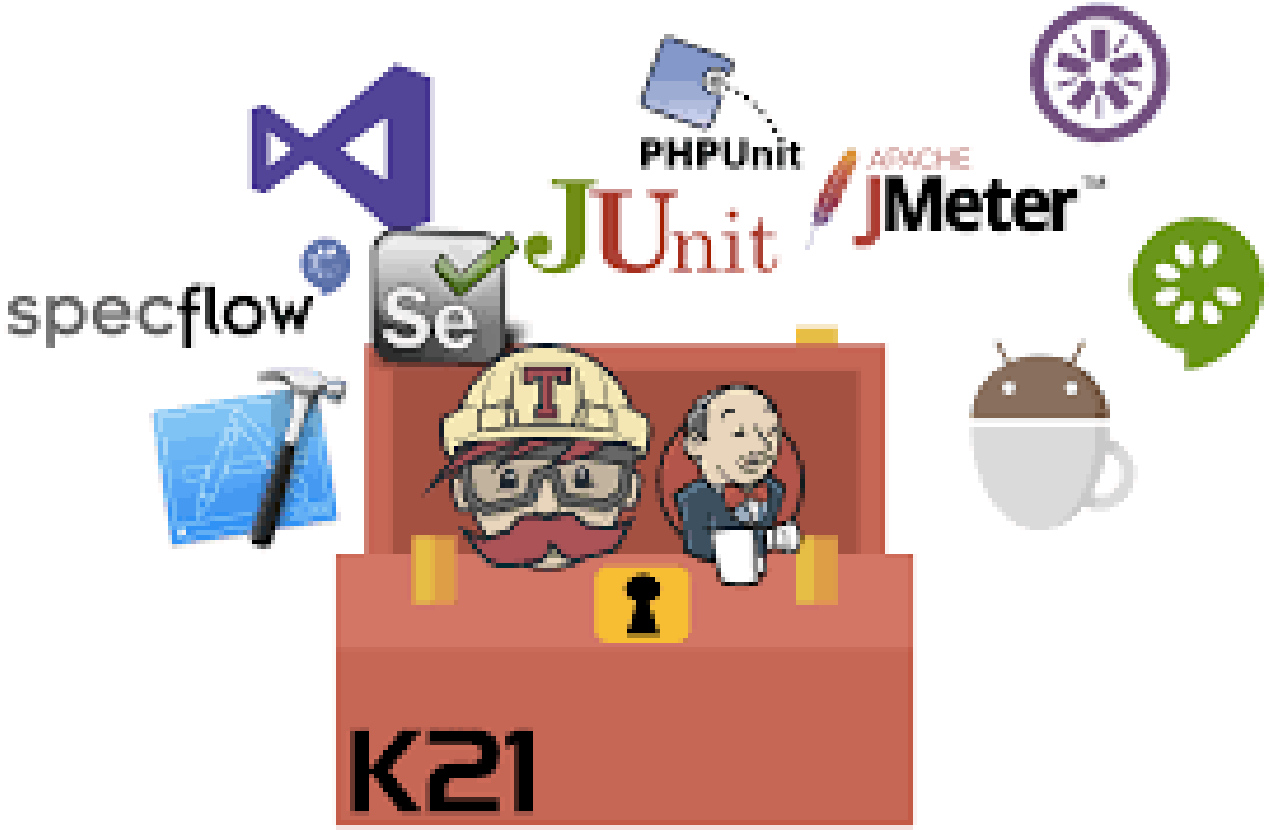
Prof. Bernardo Copstein

Baseado no livro “**Software Testing: From Theory to Practice**”

Maurice Aniche e Arie Van Deursen

Relembrando: como resolver as desvantagens do processo de teste unitário?

- Utilizar ferramentas de automação de testes
- Facilitam a execução dos *drivers* e a coleta de resultados



O Framework XUnit

Foi criado no contexto do surgimento do eXtreme Programming em 1998;

Permite a criação de testes unitários:

- Estruturados
- Eficientes
- Automatizados

Sua concepção adapta-se facilmente aos IDEs de desenvolvimento

JUnit: versão Java do framework (versão 5.0)

- http://junit.org/
- https://junit.org/junit5/docs/current/user-guide/

Características do JUnit

- Automatizam-se os teses de uma determinada classe criando um classe “driver”
  - Nome do driver = [nome da classe a ser testada] + “Test”
  - Os métodos do driver são void
  - Cada método procura testar um único aspecto
  - O nome do método pode ser qualquer um, mas é uma boa prática que o nome do método descreva o aspecto sendo testado
  - Métodos de teste são anotados com a anotação “@Test”
  - As verificações dentro dos métodos de teste são feitas usando asserções

Asserções

Nos métodos de teste exercita-se o código que se pretende testar

Depois de executar o que se pretende é necessário verificar o resultado

A verificação dos resultados é feita usando-se asserções

Asserções são declarações do que se pretende que seja verdadeiro em um determinado momento

Exemplos de asserções

assertEquals(valEsperado,valObtido)	Verdadeira se o valor esperado e o valor obtido são os mesmos
assertTrue(condição)	Verdadeira se a condição for avaliada como verdadeira
assertFalse(condição)	Verdadeira se a condição for avaliada como falsa

Para obter a lista completa de asserções acesse: JUnit's documentation

Ainda sobre asserções

Asserções são declarações do que acreditamos ser correto. Quando construímos classes drivers, usamos asserções para verificar as condições que desejamos testar.

O JUnit captura as exceções lançadas pela asserções. Quando uma asserção falha ela lança uma exceção que é capturada pelo JUnit que detecta a falha.

As falhas detectadas pelo JUnit são então compiladas em um relatório.

Exemplo de driver de teste

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class RomanNumeralTest {
  @Test
  public void singleNumber() {
     RomanNumeral roman = new RomanNumeral();
     int result = roman.convert("I");
     Assertions.assertEquals(1, result);
  }

  @Test
  public void numberWithManyDigits() {
     RomanNumeral roman = new RomanNumeral();
     int result = roman.convert("VIII");
     Assertions.assertEquals(8, result);
  }

  @Test
  public void numberWithSubtractiveNotation() {
     RomanNumeral roman = new RomanNumeral();
     int result = roman.convert("IV");
     Assertions.assertEquals(4, result);
  }

  @Test
  public void numberWithAndWithoutSubNotation(){
     RomanNumeral roman = new RomanNumeral();
     int result = roman.convert("XLIV");
     Assertions.assertEquals(44, result);
  }
}
```

Definindo “cenários”

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class RomanNumTestWithBeforeEach  {
   private RomanNumeral roman;

   @BeforeEach
   public void initialize() {
      this.roman = new RomanNumeral();
   }

   @Test
   public void singleNumber() {
      int result = roman.convert("I");
      Assertions.assertEquals(1,  result);
   }
...
```

- O método anotado com “BeforeEach” executa antes de cada método de teste
- Desta forma garante-se a definição do cenário antes da execução do teste

Projeto x Execução

Projeto

- Identificar os casos de teste
  - Usando diferentes técnicas e estratégias potencialmente reveladoras de defeitos
- Implementar os drivers de teste
  - Explorando os recursos das tecnologias existentes

Execução

- Executar os casos de teste
- Gerar os relatórios
- Analisar os relatórios

Recomendações

- Projete casos de teste independentes uns dos outros;
- Não teste apenas o “positivo”. Garanta que seu código responde adequadamente em todos os cenários;
- Crie um driver para cada classe;
- Inclua o nome do método em cada teste;
- Depure os testes quando for o caso. Não se esqueça de que os testes também são código!

Utilizando o JUnit

Utilize o JUnit em conjunto com o VSCode e o Maven

Utilize o arcabouço padrão do Maven: maven-archetype-quickstart

Utilize o arquivo “.pom” fornecido pelo professor

Veja o roteiro de configuração do ambiente e a lista de exercícios