

cesde
digital

Centro para o Desenvolvimento
de Competências Digitais

Engenharia de Software II

Version
V1.0

Introdução JUnit – Testes unitários

Summary | [Noções Básicas](#) • [Demonstração Prática](#)

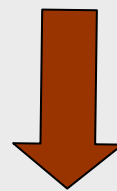
- Vantagens da realização de testes unitários
- Introdução aos testes com JUnit
- JUnit: criação de testes
- JUnit: execução de testes
- Obter e instalar o JUnit
- Demo

Porquê fazer testes unitários

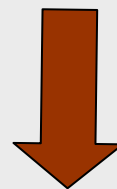
- Cobre especialmente condições “estranhas”;
- Facilita o trabalho de equipa. Perda do *síndrome de posse do Código*;
- Aumenta a confiança no trabalho produzido;
- Possibilita *refactoring*;
- API para self-documentation;
- Melhora o desenho.

Como utilizar testes

- O ciclo comum funciona da seguinte forma:
- É escrito novo código OU é resolvido um “bug”
- Execução de testes para verificar que as recentes mudanças no código funcionam.
- Introduzir o novo código de forma definitiva no sistema



Refactoring



...is the restructuring the software by applying a series of internal changes that do not affect its observable behaviour.

(Fowler 1999)

Um teste de software é um software que executa outro software, validando se os resultados são os esperados (teste de estado) ou se executa a sequência de eventos esperado (teste de comportamento);

Resumidamente, os testes de software permitem aos programadores verificar se a lógica do programa desenvolvido está de acordo com os requisitos;

A execução automática de testes permite identificar “bugs” resultantes de mudanças no código fonte.

São utilizadas pré-condições para definir um estado fixo (por exemplo uma string estática) que é utilizado como input, verificando se o código desenvolvido executa conforme o esperado;

Um teste unitário é um trecho de código que executa uma funcionalidade específica no código a testar e verifica um determinado estado ou comportamento;

Um teste unitário é aplicado a um trecho de código pequeno (método ou classe);

Os teste unitários não são adequados para testar interfaces gráficas ou interação de componentes;

- Framework open-source para testar código Java
- Permite a execução de testes de forma:
 - Fácil
 - Regular
 - Fiável
- Pode facilmente ser integrado num IDE como Eclipse, NetBeans, etc...
- Contém várias funcionalidades para *testing*.
- A principal permissa por trás do JUnit é a capacidade de testar cada componente de um programa de forma independente do resto do programa

O Junit integra diferentes módulos de 3 diferentes subprojectos:

JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage

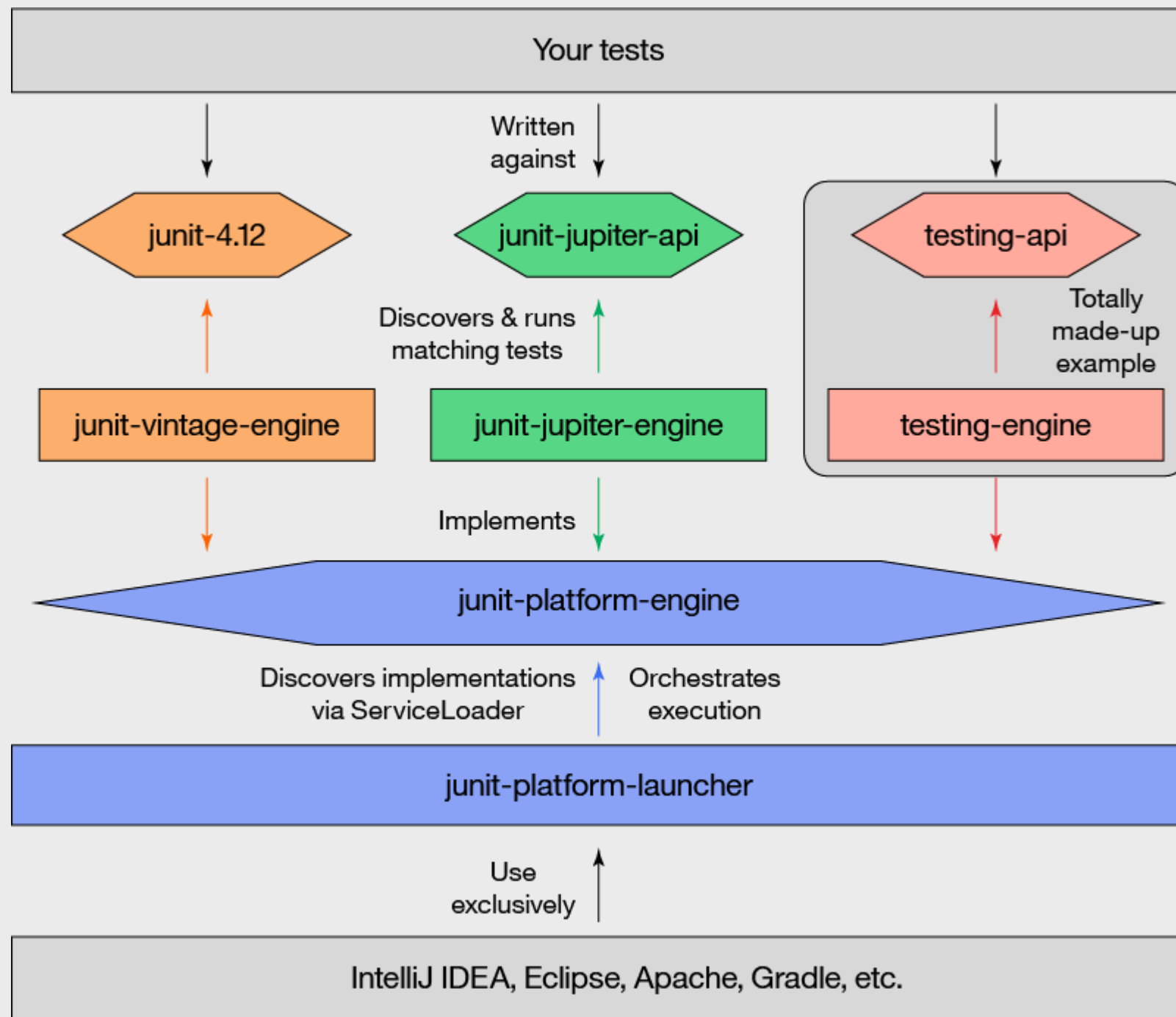
A plataforma JUnit define o motor de testes para o desenvolvimento da framework de testes;

O módulo JUnit Jupiter define um modelo de programação e extensão para o desenvolvimento de testes e extensões no JUnit 5.

O módulo JUnit Vintage disponibiliza um motor de testes compatível com JUnit 3 e JUnit 4;

É necessário possuir o JDK 8 (ou superior) instalado.

Visão geral da arquitetura



- A execução de testes unitários está dividida em duas partes:
 - Descoberta de testes e a criação do plano de testes;
 - Executar do plano de testes e reportar os resultados

Exemplo

Um teste é representado por um método que é apenas utilizado para testes, devendo estar anotado com `@Test`;

É utilizado o método `assert` (`assertEquals`) para verificar se o resultado esperado é igual ao resultado alcançado:

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class TestCases {

    @Test
    void testSoma() {
        assertEquals(2, 1 + 1);
    }

}
```

- TestCases (Java Classes)
- O nome começa por **test** ou termina com **test**
- Tests (Java Methods)
- Testa apenas um método
 - testXxx testa o método xxx
- Quando são executados múltiplos testes a um método utiliza-se a seguinte nomenclatura:
 - testXxxYxx, onde Yxx descreve uma condição em particular. (e.g. testEndsWithEmpty)
- Apenas um teste por método de teste ou test method
- setUp - inicialização de condições necessárias ao teste
- tearDown - eliminação das condições de inicialização

Exemplo

Os testes devem possuir mensagens representativas do resultado do teste. Exemplo:

```
import static org.junit.jupiter.api.Assertions.assertEquals;
```

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

```
public class TestCases {
```

Utilizar nomes que “expliquem” o objetivo
do teste

```
@Test
```

```
public void testMultiplicationOfZeroShouldReturnZero() {
```

```
    MyClass tester = new MyClass(); // MyClass is tested
```

```
    // assert statements
```

```
    assertEquals(0, tester.multiply(10, 0), "10 x 0 should be 0");
```

```
    assertEquals(0, tester.multiply(0, 10), "0 x 10 should be 0");
```

```
    assertEquals(0, tester.multiply(0, 0), "0 x 0 should be 0");
```

```
}
```

```
}
```

Uma classe de testes pode ter várias configurações que têm de cumprir com as seguintes condições:

- Não devem retornar nenhum valor (retorno void);
- Métodos não podem ser privados;

Métodos anotados com `@Test` representam métodos de teste.

- Existem outras anotações que representam métodos de teste:

- @ParameterizedTest :

```
@ParameterizedTest
@ValueSource(strings = { "racecar", "radar", "able was I ere I saw elba" })
void palindromes(String candidate) {
    assertTrue(isPalindrome(candidate));
}
```

- @RepeatedTest

```
@RepeatedTest(10)
void repeatedTest() {
    // ...
}
```

- A anotação @Disable pode ser utilizada para desativar métodos de teste e excluí-los da suite de execução;

Métodos – ciclo de vida

Métodos anotados com `@BeforeAll` devem ser estáticos e são `inicialização` executados uma única vez antes dos restantes métodos de teste;

Métodos anotados com `@BeforeEach` são invocados antes de cada método de teste; `setup`

Métodos anotados com `@AfterEach` são invocados após cada método de teste; `teardown`

Métodos anotados com `@AfterAll` devem ser estáticos e são `done` executados uma única vez depois dos restantes métodos de teste;

A anotação `@DisplayName` define uma descrição para a classe ou métodos.

Assertions são utilizadas para verificar uma condição que deverá ser avaliada em verdade de forma a que o teste possa continuar a executar;

Se uma assertion falhar, o teste irá ser suspenso e uma falha é reportada;

Método: Assertion	Descrição
<code>assertEquals(expected, actual)</code>	The assertion fails if <i>expected</i> does not equal <i>actual</i> .
<code>assertFalse(booleanExpression)</code>	The assertion fails if <i>booleanExpression</i> is not false.
<code>assertNull(actual)</code>	The assertion fails if <i>actual</i> is not null.
<code>assertNotNull(actual)</code>	The assertion fails if <i>actual</i> is null.
<code>assertTrue(booleanExpression)</code>	The assertion fails if <i>booleanExpression</i> is not true.

Podemos colocar várias assertions utilizando o método: `@assertAll`:

```
(...)  
assertAll("person",  
          () -> assertEquals("John",  
person.getFirstName()),  
          () -> assertEquals("Doe", person.getLastName())  
        );  
(...)
```

Método `@assertThrows` é utilizado quando a classe que está a ser testada retorna uma exceção esperada, o que pode ser considerado como uma condição que deverá ser avaliada (Exception Testing):

```
assertThrows (IllegalArgumentException.class, () -> classUnderTest.add(numbersToSum) ) ;
```

Assumptions

Assumptions são similares a Assertions com a exceção que as assumptions retornam true (sucesso) ou false (insucesso);

As Assumptions são úteis quando um método de teste só deve ser executado sob determinadas condições;

Suponhamos que queremos executar um teste apenas às sextas-feiras;

Se a Assumption não for verdade, o teste não é executado:

```
@Test
@DisplayName("This test is only run on Fridays")
public void testAdd_OnlyOnFriday() {
    LocalDateTime ldt = LocalDateTime.now();
    assumeTrue(ldt.getDayOfWeek().getValue() == 5);
}
```

Devemos utilizar assertions para verificar os resultados de um método de teste;

Devemos utilizar assumptions para determinar se o teste irá ser executado ou não;

A não execução do teste não é reportado como falha, o que não resulta na quebra do processo de “build”;

Os testes de software devem ser colocados separadamente do código da aplicação;

A convenção do Gradle refere-se:

- à colocação do código fonte do programa a testar na pasta: `src/main/java`
- à colocação do código relacionado com os testes de software na pasta: `src/test/java`

A framework JUnit (<https://junit.org/junit5/>) é uma framework de testes que utiliza anotações para identificar métodos que especificam um teste.

Introdução JUnit – Testes unitários [ESII] | Gradle

```
plugins {  
    id 'java'  
}  
  
group 'JUnitProject'  
version '1.0-SNAPSHOT'  
  
sourceCompatibility = 1.8
```

```
repositories {  
    mavenCentral()  
}
```

```
dependencies {  
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.3.1'  
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.3.1'  
}
```

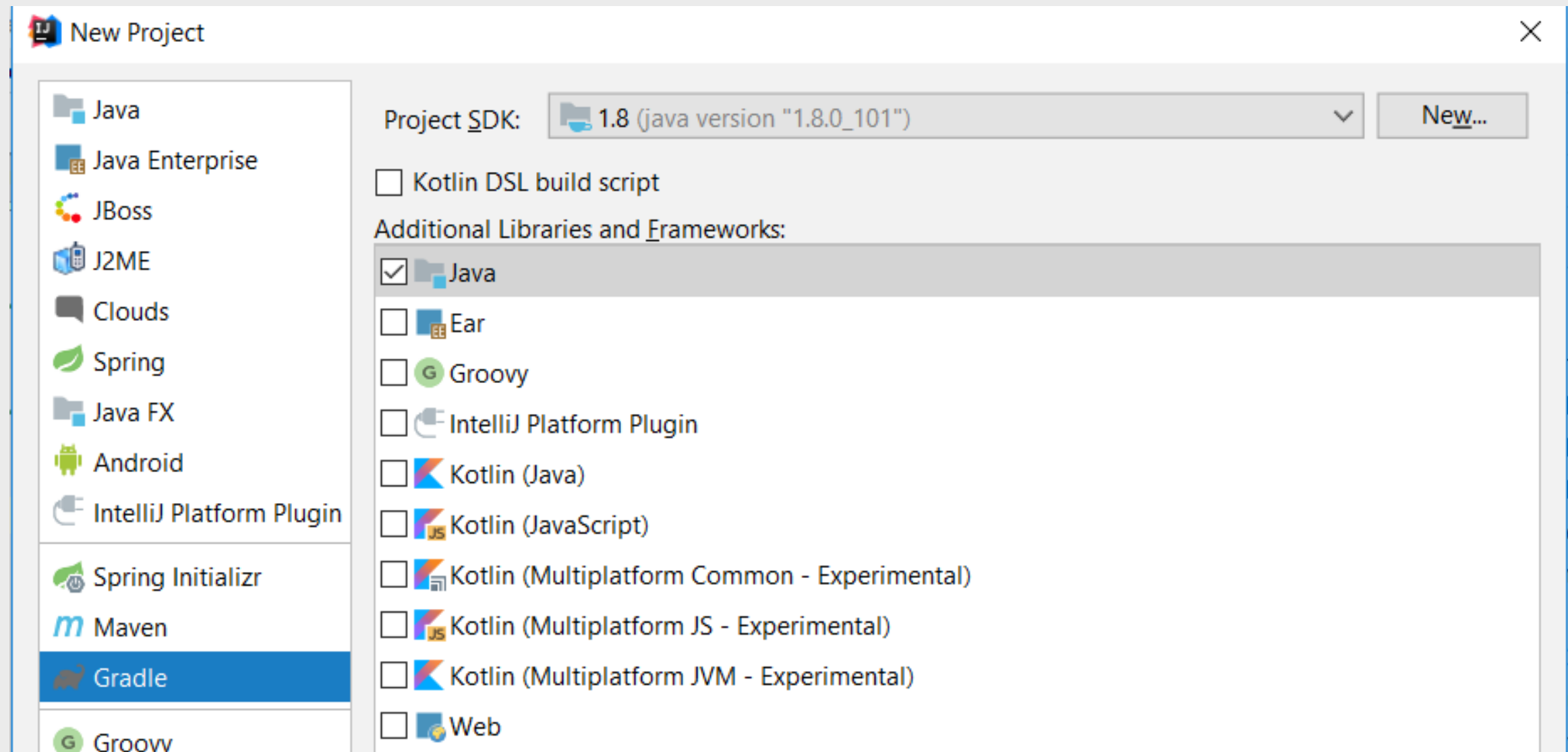
<https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine>

```
test {  
    useJUnitPlatform()  
    testLogging {  
        events "passed", "skipped", "failed", "standardOut", "standardError"  
    }  
}
```

Nível de detalhe dos log gerados

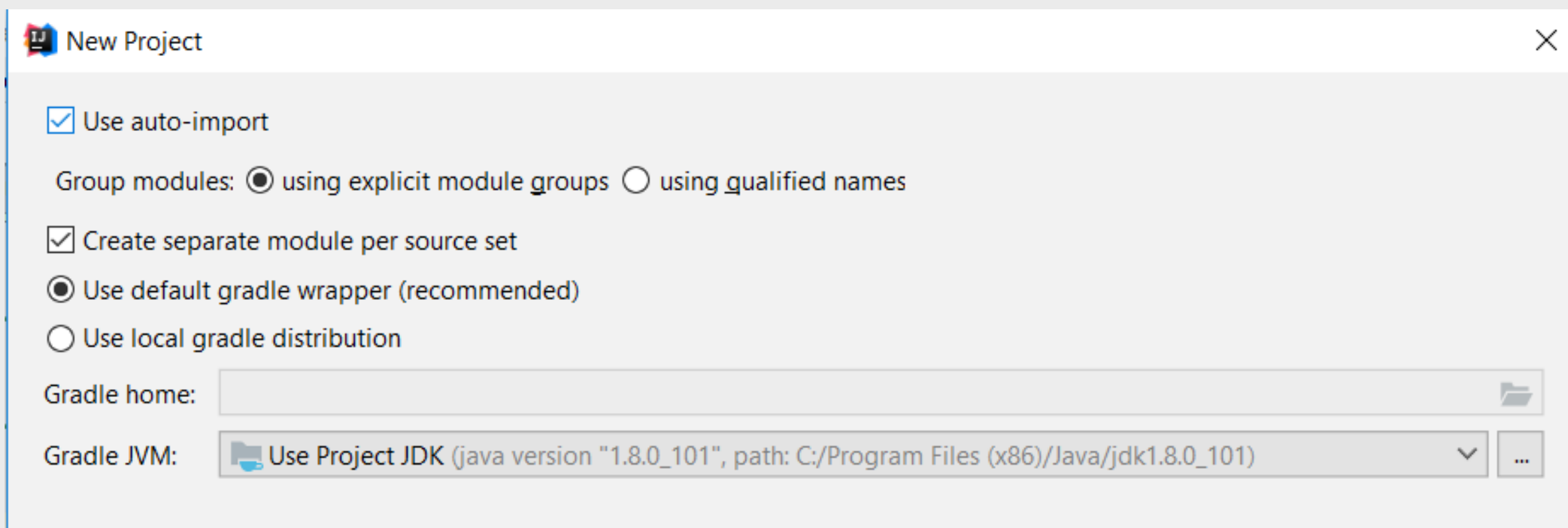
Demonstração – Gradle + IntelliJ

File->New->Project:



Demonstração – Gradle + IntelliJ

Configurar: auto-import:



The image shows the 'New Project' dialog box in IntelliJ IDEA. The 'Use auto-import' checkbox is checked. Under 'Group modules', 'using explicit module groups' is selected. 'Create separate module per source set' is also checked. For 'Use default gradle wrapper (recommended)', it is selected. The 'Gradle home' field is empty. The 'Gradle JVM' dropdown is set to 'Use Project JDK (java version "1.8.0_101", path: C:/Program Files (x86)/Java/jdk1.8.0_101)'.

New Project

☒ Use auto-import

Group modules: ☒ using explicit module groups ☐ using qualified names

☒ Create separate module per source set

☒ Use default gradle wrapper (recommended)

☐ Use local gradle distribution

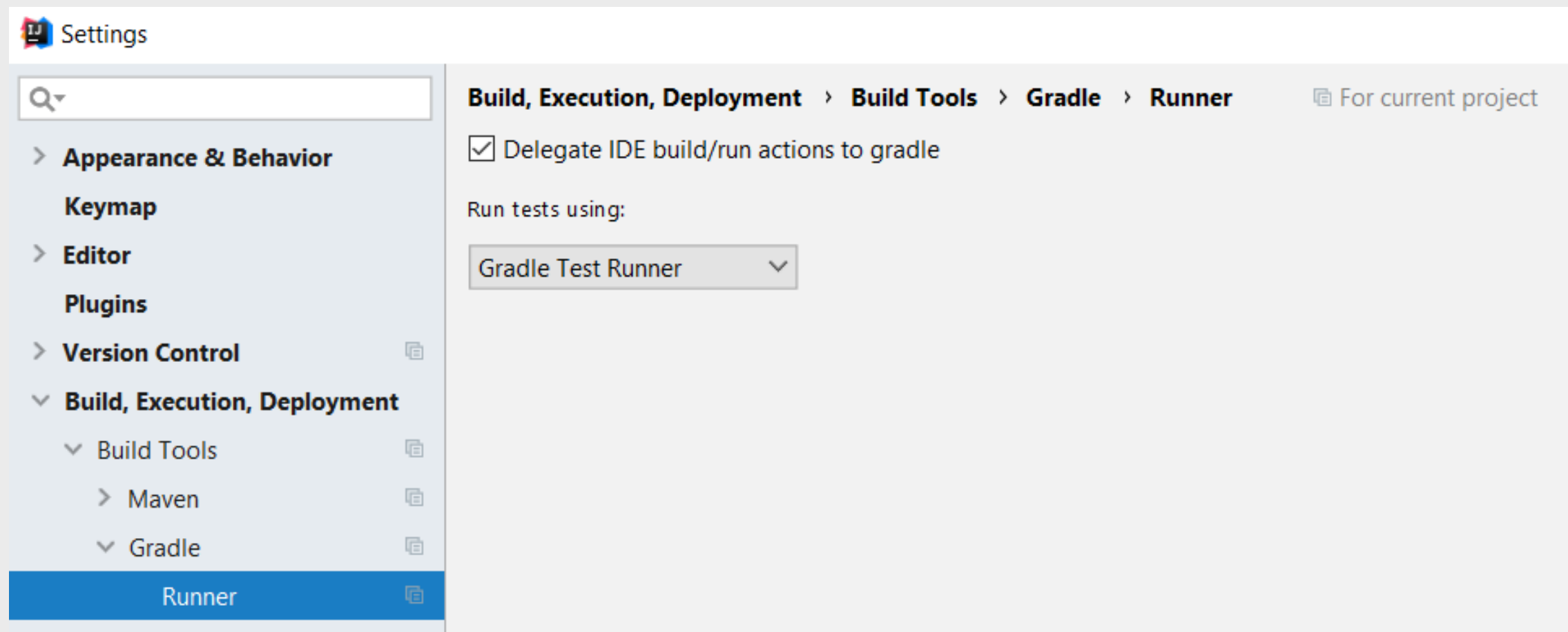
Gradle home:

Gradle JVM:

Demonstração – Gradle + IntelliJ

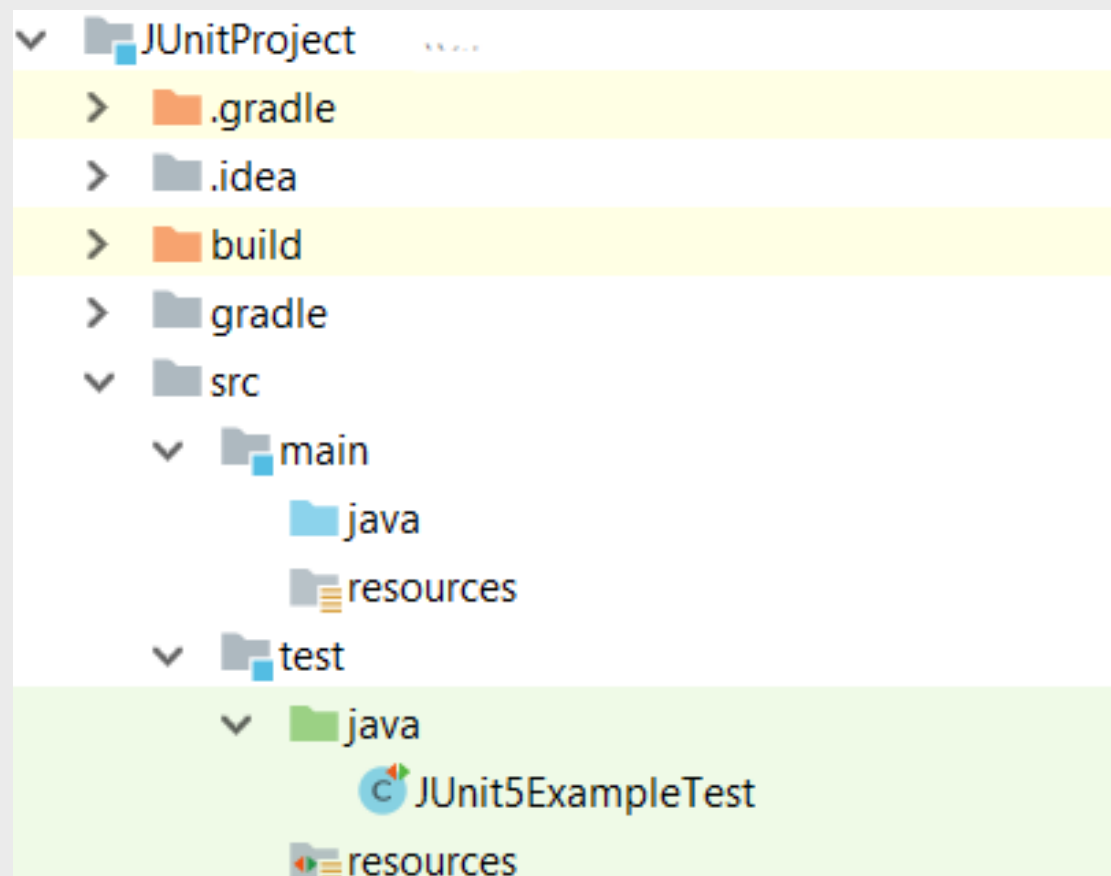
Delegar as builds/runs do IDE para o Gradle;

CTRL+ALT+S para aceder às propriedades do projeto:



Demonstração – Gradle + IntelliJ

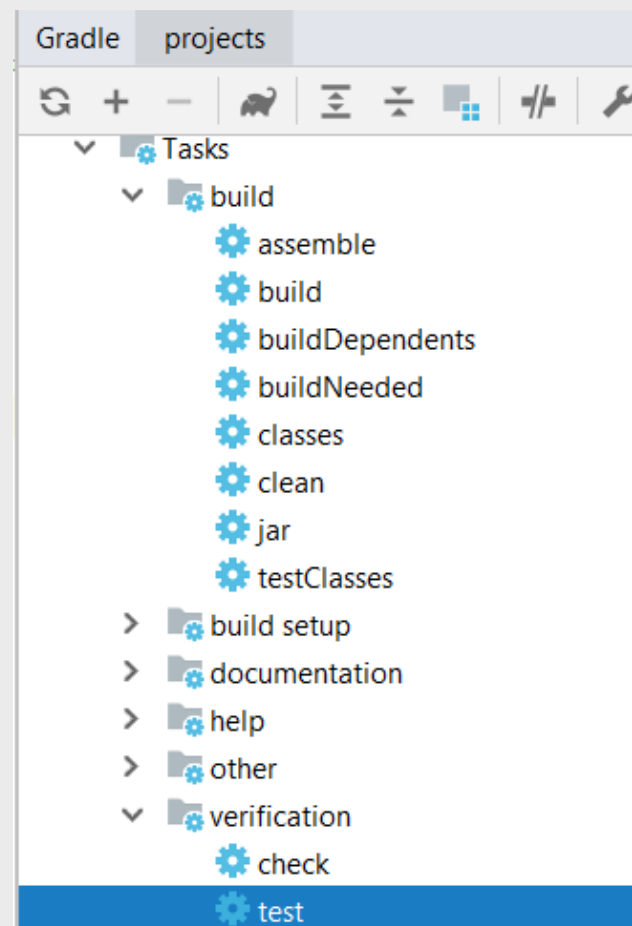
Criar a classe do slide 11 na pasta: src/main/test:



Demonstração – Gradle + IntelliJ

Executar o teste:

Utilizando a tarefa Gradle: Test:





[ESII]
Referências

Referências

- <https://docs.gradle.org/>
- <https://junit.org/junit5/>