

jUnit

Prof. Rodrigo Macedo

Escopo do Curso

- Conceituação Geral: qualidade de software x testes
- JUnit - conceitos, características e aplicabilidade.
- Estrutura jUnit.
- Anotações.
- Demais propriedades do jUnit.
- Questões de concursos



Qualidade

- De acordo com a norma ISO 9000, a definição de qualidade é:
'Qualidade é o grau no qual um conjunto de características inerentes, satisfaz a requisitos'.

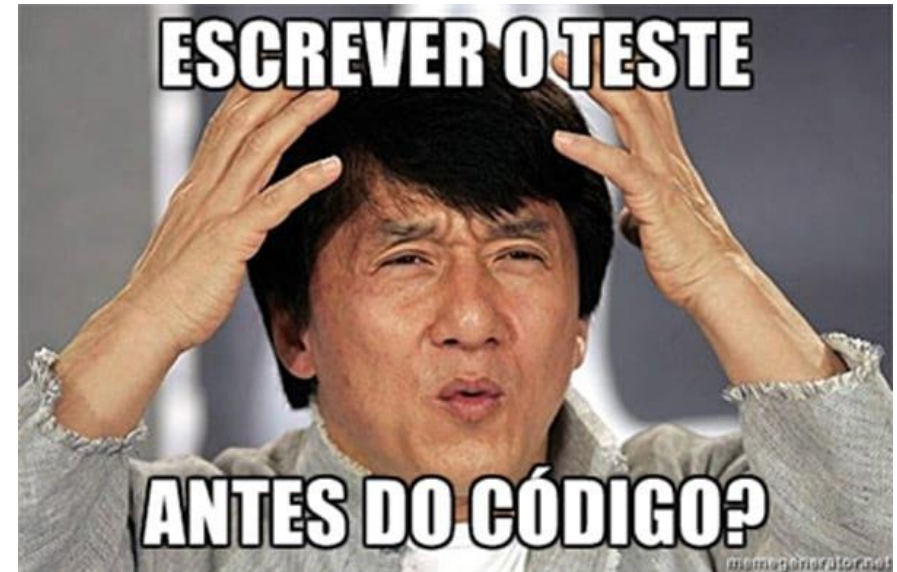
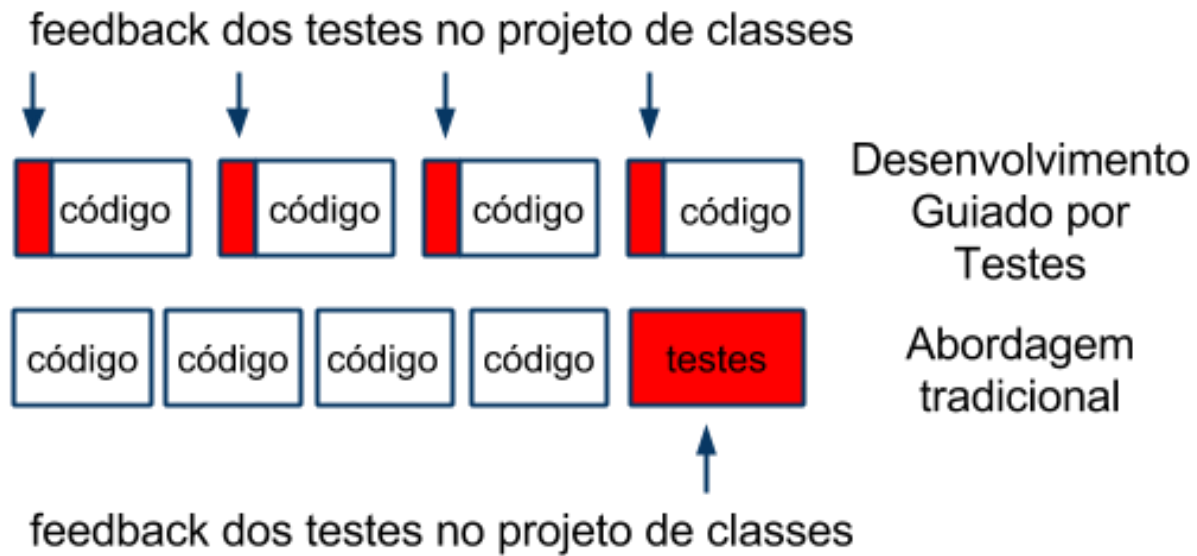
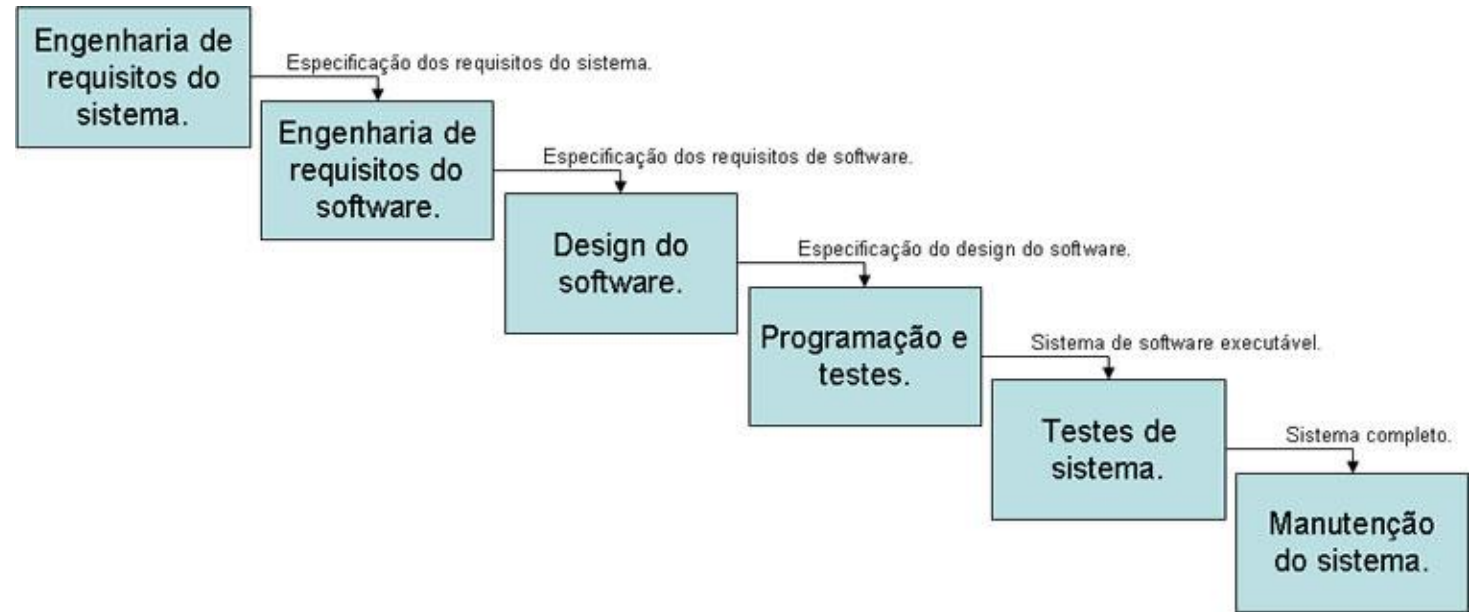
Em termos de software temos:

- **Garantia da Qualidade:** processo de auditoria dos requisitos de qualidade e de resultados de medições visando garantir que sejam usados padrões de qualidade e definição operacionais.
- **Controle da Qualidade:** processo de monitoramento e registro de resultados das atividades de qualidade para avaliar o desempenho.

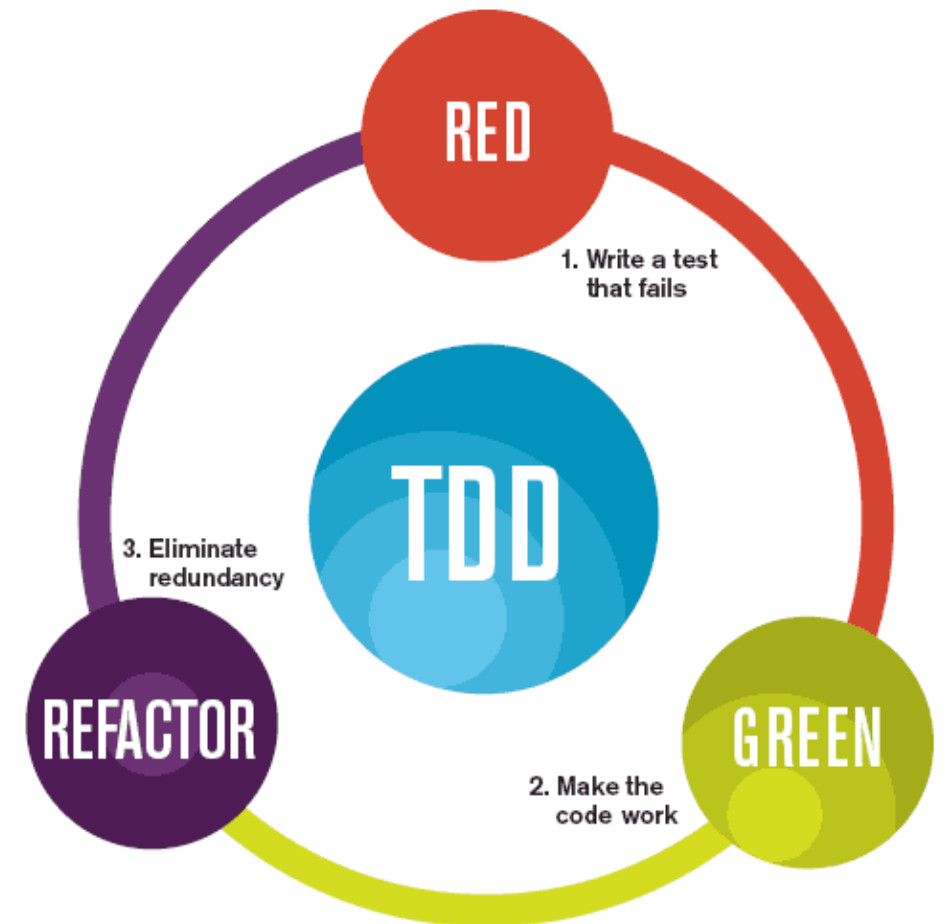
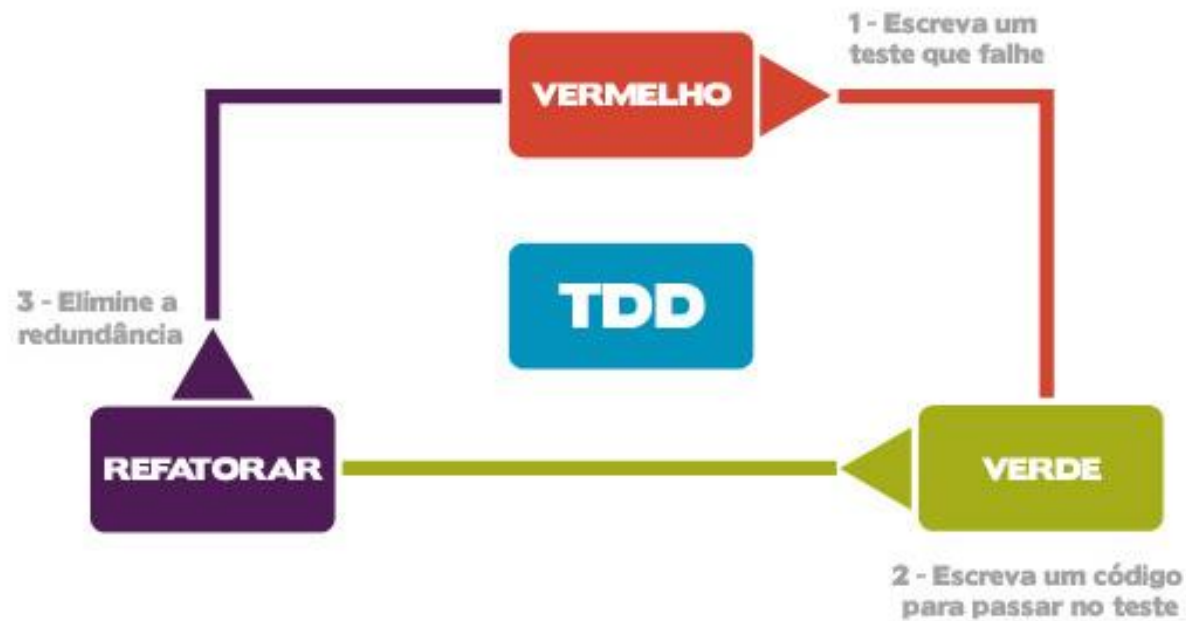
Custo x Investimento - Qualidade



TDD - Uma das boas práticas listadas na XP.

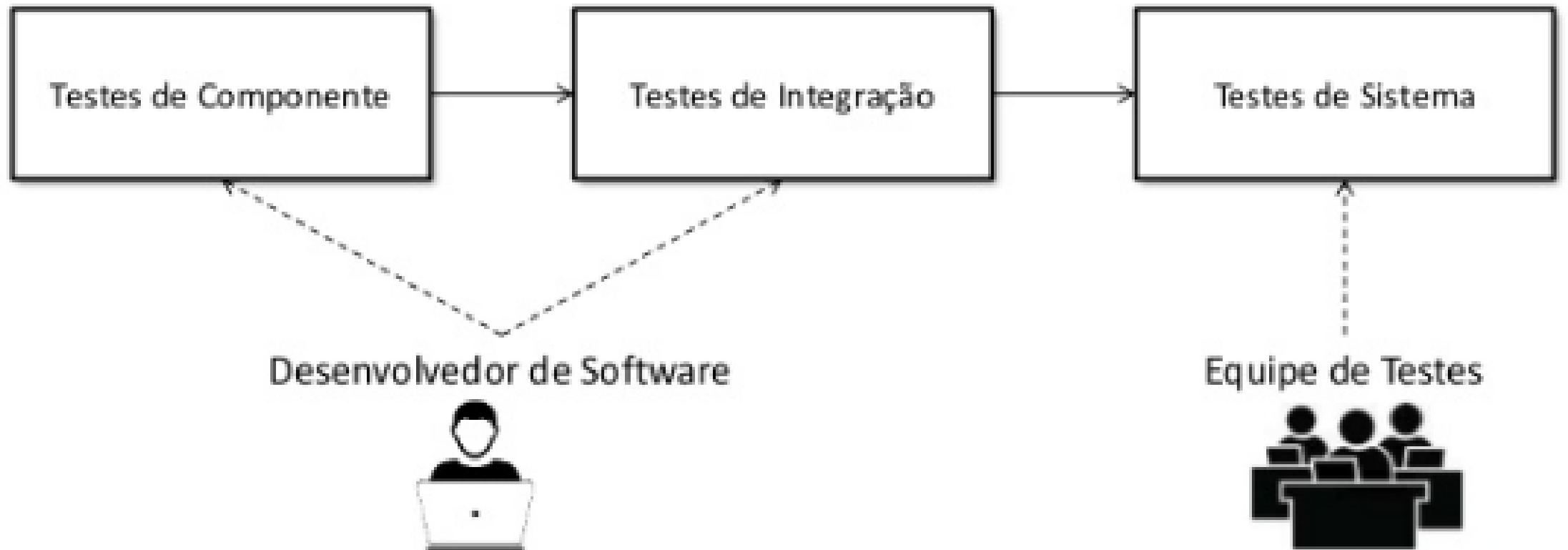


Ciclo de vida



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Fases do Processo de Teste

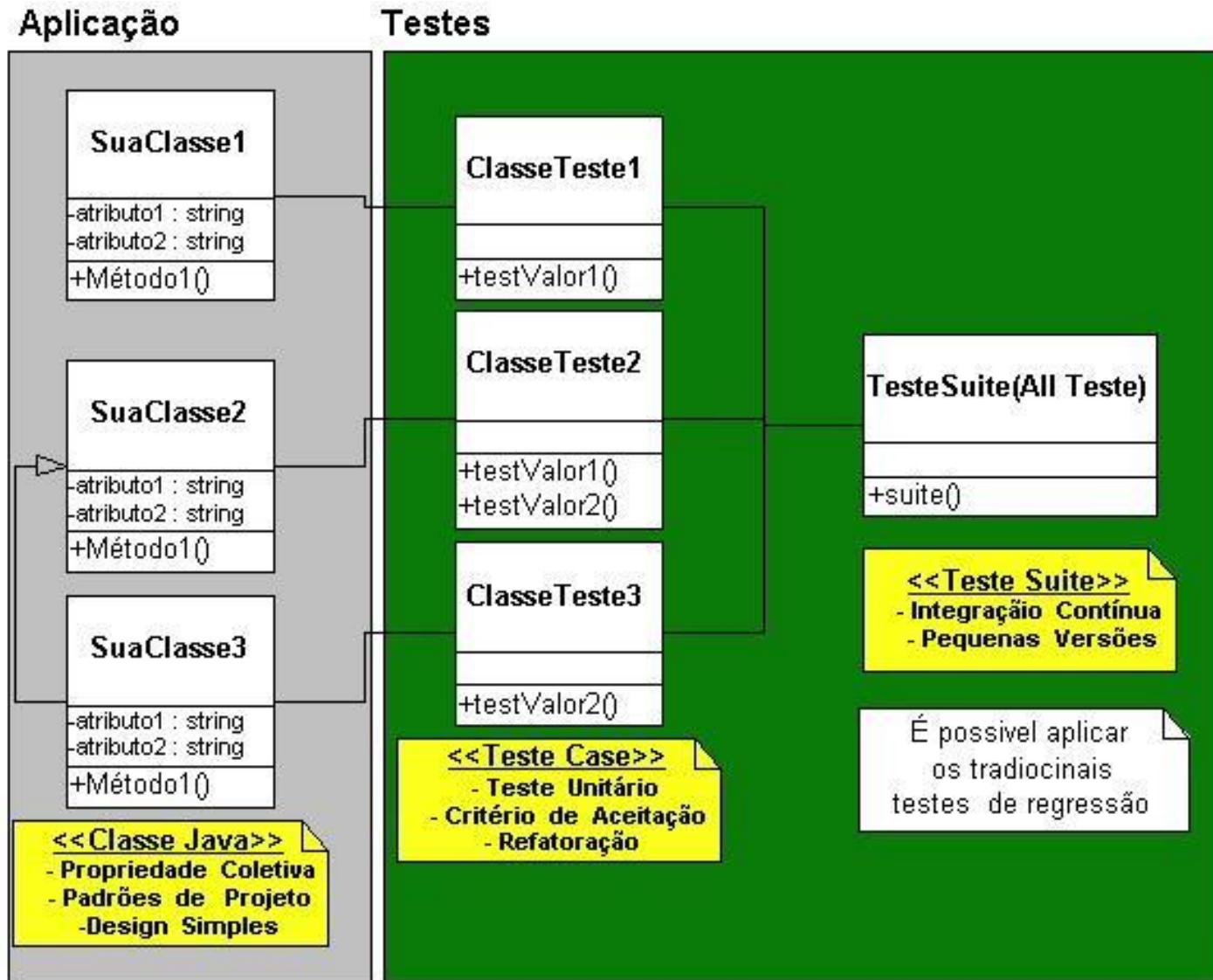


O que são Testes Unitários?

- É um nível de teste de software no qual os componentes individuais são testados.
- O propósito é validar se cada unidade do software executa como esperado.
- Está relacionado ao teste do tipo estrutural (caixa branca), realizado internamente no código.
- Nivel mais baixo dos tipos de teste realizado.



O que são Testes Unitários?



O que são Testes Unitários?

Os testes unitários tem como benefício:

- Garantir que problemas serão descobertos o quanto antes (regra de Myers).
- Facilitar a manutenção do código.
- Servir como documentação.
- Pré-requisito para a integração contínua.
- Ajudam a melhorar o design do seu código e torná-lo um melhor desenvolvedor.
- Resulta em outras práticas XP como: Código coletivo, refatoração, integração contínua;

Podemos contar com alguma ferramenta que nos ajude a criar testes automatizados?



jUnit

- Framework utilizado para facilitar o desenvolvimento e automatização de testes unitários em código **Java**.
- Framework open-source, criado por Erich Gamma e Kent Beck.
- Fornece uma API (conjunto de classes) para construir os testes tanto por meio de aplicação gráfica, quanto em modo console, para execução dos testes criados.



jUnit - Características

- JUnit pode verificar se cada unidade de código funciona da forma esperada.
- Facilita a criação, execução automática de testes e a apresentação dos resultados
- É Orientado a Objeto;
- É free e pode ser baixado em: www.junit.org
- Recentemente a versão oficial e estável foi concedida a versão 5 do JUnit.



jUnit - Características

- O jUnit funciona com base em **anotações** (Annotations) e essas anotações indicam se um método é de teste ou não, se um método deve ser executado antes da classe e/ou depois da classe.
- As anotações também indicam se o teste deve ou não ser ignorado e se a classe em questão é uma suíte de teste, ou seja, se a partir desta classe é disparada a execução das demais classes de teste, entre outras anotações menos utilizadas.



jUnit 5

- JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage.
- **JUnit Platform:** A Plataforma JUnit serve como base para o lançamento de estruturas de teste na JVM. Isso também define a API TestEngine para desenvolver uma estrutura de teste executada na plataforma.
- **JUnit Jupiter:** combinação do novo modelo de programação e modelo de extensão para gravar testes e extensões no JUnit 5. O subprojeto Jupiter fornece um TestEngine para executar testes baseados em Jupiter na plataforma
- **JUnit Vintage:** fornece um TestEngine para executar testes baseados no JUnit 3 e JUnit 4 na plataforma.

Como instalar o JUnit?

Há pelo menos duas formas:

- Caso você não tenha o JUnit instalado, faça o download do arquivo junit.jar em www.junit.org, após inclua-o no classpath para compilar e rodar os programas de teste.
- Porém o JUnit já vem configurado nas versões recentes de IDE's como Eclipse, NetBeans, JBuilder, BlueJ e outros.



Requisitos

- Toda classe de teste deve ser criada com a palavra **Test** no final.
- Já um teste é um método contido nessas classes que é usada apenas para teste e anotado com um **@Test**.
- O método que será testado deve ser **público** e sem retorno (**void**).
- Uma classe, para ser testada pelo Junit, deve se estender a partir de **TestCase**.
- As classes de teste **não** devem ser **abstratas** e devem ter um **único construtor**.



Exemplo - junit 3

```
public class DatabaseConnectionTest extends @TestCase{
    private static Connection testConnection;
    private static ConnectionDAO genericJDBCDao;
    @Test
    public void connectDatabase(){
        genericJDBCDao = new ConnectionDAO("root", "");
        genericJDBCDao.setDbms("mysql");
        genericJDBCDao.setDbName("teste");
        genericJDBCDao.setServerName("localhost");
        genericJDBCDao.setPortNumber("3306");
        testConnection = genericJDBCDao.createConnection();
        genericJDBCDao.setConnection(testConnection);
    }
}
```

Exemplo - junit 5

**Não é necessário
estender a classe
TestCase.**

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

import example.util.Calculator;

import org.junit.jupiter.api.Test;

class MyFirstJUnitJupiterTests {

    private final Calculator calculator = new Calculator();

    @Test
    void addition() {
        assertEquals(2, calculator.add(1, 1));
    }

}
```

Anotações

O JUnit Jupiter suporta as seguintes anotações para configurar testes e estender a estrutura:

- **@Test:** indica que um método é um método de teste. Diferentemente da anotação @Test da JUnit 4, essa anotação não declara nenhum atributo, pois as extensões de teste no JUnit Jupiter operam com base em suas próprias anotações dedicadas.
- **@ParameterizedTest:** Indica que um método é um teste parametrizado.



Anotações

O JUnit Jupiter suporta as seguintes anotações para configurar testes e estender a estrutura:

- **@RepeatedTest:** Indica que um método é um modelo de teste para um teste repetido
- **@TestFactory:** Indica que um método é uma fábrica de testes para testes dinâmicos.
- **@TestTemplate:** indica que um método é um modelo para casos de teste projetados para serem chamados várias vezes, dependendo do número de contextos de chamada retornados pelos provedores registrados.



Anotações

O JUnit Jupiter suporta as seguintes anotações para configurar testes e estender a estrutura:

- **@TestMethodOrder:** Usado para configurar a ordem de execução do método de teste para a classe de teste anotada; semelhante ao @FixMethodOrder da JUnit 4.
- **@TestInstance:** Usado para configurar o ciclo de vida da instância de teste para a classe de teste anotada.
- **@DisplayName:** Declara um nome de exibição personalizado para a classe ou método de teste.



Anotações

O JUnit Jupiter suporta as seguintes anotações para configurar testes e estender a estrutura:

- **@DisplayNameGeneration:** Declara um gerador de nome para exibição personalizado para a classe de teste.
- **@BeforeEach:** Indica que o método anotado deve ser executado antes de cada método `@Test`, `@RepeatedTest`, `@ParameterizedTest` ou `@TestFactory` na classe atual; análogo ao `@Before` da JUnit 4. - **Semelhante ao `setUp()` em versões anteriores.**
- **@AfterEach:** Indica que o método anotado deve ser executado após cada método `@Test`, `@RepeatedTest`, `@ParameterizedTest` ou `@TestFactory` na classe atual; análogo ao `@After` da JUnit 4. **Semelhante ao `tearDown()` em versões anteriores.**



Anotações

O JUnit Jupiter suporta as seguintes anotações para configurar testes e estender a estrutura:

- **@BeforeAll:** Indica que o método anotado deve ser executado antes de todos os métodos `@Test`, `@RepeatedTest`, `@ParameterizedTest` e `@TestFactory` na classe atual; análogo ao `@BeforeClass` da JUnit 4.
- **@AfterAll:** Indica que o método anotado deve ser executado depois de todos os métodos `@Test`, `@RepeatedTest`, `@ParameterizedTest` e `@TestFactory` na classe atual; análogo ao `@AfterClass` da JUnit 4.
- **@Nested:** Indica que a classe anotada é uma classe de teste aninhada não estática. Os métodos `@BeforeAll` e `@AfterAll` não podem ser usados diretamente em uma classe de teste `@Nested`, a menos que o ciclo de vida da instância de teste "por classe" seja usado.



Anotações

O JUnit Jupiter suporta as seguintes anotações para configurar testes e estender a estrutura:

- **@Tag:** Usado para declarar tags para testes de filtragem, no nível da classe ou do método; análogo aos grupos de teste no TestNG ou às categorias na JUnit 4.
- **@Disabled:** Usado para desativar uma classe ou método de teste; análogo ao @Ignore da JUnit 4.
- **@Timeout:** Usado para falhar em um teste, fábrica de teste, modelo de teste ou método de ciclo de vida se sua execução exceder uma determinada duração.



Anotações

O JUnit Jupiter suporta as seguintes anotações para configurar testes e estender a estrutura:

- **@ExtendWith:** Usado para registrar extensões declarativamente.
- **@RegisterExtension:** Usado para registrar extensões programaticamente por meio de campos.
- **@TempDir:** Usado para fornecer um diretório temporário via injeção de campo ou injeção de parâmetro em um método de ciclo de vida ou método de teste.



Anotações - junit 4

@Test	Identifica um método de testes
@Test(expected= Exception.class)	Falha se o método testado não retornar essa exceção
@Test(timeout=100)	Falha se o método demorar mais de 100milisegundos
@Before public void method()	Executa antes de cada teste . Usado para preparar o ambiente de teste
@After public void method()	Executa depois de cada teste . Usado para liberar recursos do teste
@BeforeClass public static void method()	Executa 1 vez , antes de todos os testes. Usado para preparar o ambiente de teste O método anotado deve ser estático
@AfterClass public static void method()	Executa 1 vez , depois de todos os testes. Usado para liberar recursos do teste O método anotado deve ser estático
@Ignore	Ignora algum método de teste

Teste de Classes e Métodos

- **Classe de Teste:** qualquer classe de nível superior, classe de membro estático ou classe `@Nested` que contenha pelo menos um método de teste. *As classes de teste não devem ser abstratas e devem ter um único construtor.*
- **Método de Teste:** qualquer método de instância anotado diretamente ou meta-anotado com `@Test`, `@RepeatedTest`, `@ParameterizedTest`, `@TestFactory` ou `@TestTemplate`.
- **Método do Ciclo de Vida:** qualquer método diretamente anotado ou meta-anotado com `@BeforeAll`, `@AfterAll`, `@BeforeEach` ou `@AfterEach`.

Teste de Classes e Métodos - Exemplo

```
class StandardTests {  
  
    @BeforeAll  
    static void initAll() {  
    }  
  
    @BeforeEach  
    void init() {  
    }  
  
    @Test  
    void succeedingTest() {  
    }  
  
    @Test  
    void failingTest() {  
        fail("a failing test");  
    }  
  
    @Test  
    @Disabled("for demonstration purposes")  
    void skippedTest() {  
        // not executed  
    }  
}
```

Nomes de Teste

Classes de teste e métodos de teste podem declarar nomes de exibição personalizados via `@DisplayName` - com espaços, caracteres especiais e até emojis - que serão exibidos nos relatórios de teste e pelos executores e IDEs

```
@DisplayName("A special test case")
class DisplayNameDemo {

    @Test
    @DisplayName("Custom test name containing spaces")
    void testWithDisplayNameContainingSpaces() {
    }

    @Test
    @DisplayName("J°□° J")
    void testWithDisplayNameContainingSpecialCharacters() {
    }
}
```

Asserções

- O JUnit Jupiter vem com muitos dos métodos de asserção que o JUnit 4 possui e inclui alguns que se prestam bem ao uso com lambdas do Java 8.
- Na versão 5, foi dado suporte para asserções em Kotlin.
- Todas as asserções do JUnit Jupiter são métodos estáticos na classe `org.junit.jupiter.api.Assertions`.
- Essa estrutura permite que especifiquemos uma mensagem de erro e comparar o resultado real do teste com o resultado esperado.

Asserções

- **assertTrue([message,] boolean condition):** verifica se a condição booleana é verdadeira.
- **assertSame([message,] expected, actual):** verifica se ambas as variáveis se referem ao mesmo objeto.
- **assertNotSame([message,] expected, actual):** verifica se ambas as variáveis se referem a objetos diferentes.



Outras Asserções

`fail(String)`

Faz o método falhar

`assertTrue(boolean, condition)`

Testa se a condição é verdadeira

`assertFalse(boolean, condition)`

Testa se a condição é falsa

`assertEquals(expected, actual)`

Teste se dois valores são iguais

`assertEquals(expected, actual,
tolerance)`

Para float ou double, a tolerância é o número de casas decimais a ser verificado

`assertNull(object)`

Verifica se o objeto é nulo

`assertNotNull(object)`

Verifica se o objeto não é nulo

Asserções - Exemplo

```
@Test
void standardAssertions() {
    assertEquals(2, calculator.add(1, 1));
    assertEquals(4, calculator.multiply(2, 2),
        "The optional failure message is now the last parameter");
    assertTrue('a' < 'b', () -> "Assertion messages can be lazily evaluated -- "
        + "to avoid constructing complex messages unnecessarily.");
}
```

Asserções de “terceiros”

- Para asserções adicionais, o JUnit recomenda o uso de bibliotecas de asserções de terceiros, como AssertJ, Hamcrest, Truth etc.
- A ideia é incorporar novas features de asserções, ou até mesmo desenvolver suas próprias asserções por meio da API `org.junit.jupiter.api.Assertions` disponibilizada pelo JUnit.

Asserções de “terceiros” - Hamcrest

```
import static org.hamcrest.CoreMatchers.equalTo;
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;

import example.util.Calculator;

import org.junit.jupiter.api.Test;

class HamcrestAssertionsDemo {

    private final Calculator calculator = new Calculator();

    @Test
    void assertWithHamcrestMatcher() {
        assertThat(calculator.subtract(4, 1), is(equalTo(3)));
    }

}
```

Suposições em JUnit

- O JUnit Jupiter vem com um subconjunto dos métodos de suposição que o JUnit 4 fornece e adiciona alguns que se prestam bem ao uso com expressões lambda e referências de método do Java 8.
- Todas as suposições JUnit Jupiter são métodos **estáticos** na classe `org.junit.jupiter.api.Assumptions`.
- A partir do JUnit Jupiter 5.4, também é possível usar métodos da classe `org.junit.Assume` da JUnit 4 para suposições.

Suposições em junit

```
class AssumptionsDemo {  
  
    private final Calculator calculator = new Calculator();  
  
    @Test  
    void testOnlyOnCiServer() {  
        assertTrue("CI".equals(System.getenv("ENV")));  
        // remainder of test  
    }  
  
    @Test  
    void testOnlyOnDeveloperWorkstation() {  
        assertTrue("DEV".equals(System.getenv("ENV")),  
            () -> "Aborting test: not on developer workstation");  
        // remainder of test  
    }  
}
```

Desabilitando Testes

- Classes de teste inteiras ou métodos de teste individuais podem ser desativados por meio da anotação @Disabled.

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

@Disabled("Disabled until bug #99 has been fixed")
class DisabledClassDemo {

    @Test
    void testWillBeSkipped() {
    }

}
```

Condições de Execução em Testes

- A API de extensão `ExecutionCondition` no JUnit Jupiter permite que os desenvolvedores habilitem ou desabilitem um contêiner ou teste com base em determinadas condições programaticamente.
- São as condições:
 1. Condições do SO.
 2. Condições do JRE (Java Runtime Environment).
 3. Condições de propriedades do sistema.
 4. Condições baseadas em scripts.

Condições do SO

- Um contêiner ou teste pode ser ativado ou desativado em um sistema operacional específico por meio das anotações `@EnabledOnOs` e `@DisabledOnOs`.

```
@Test
@EnabledOnOs(MAC)
void onlyOnMacOs() {
    // ...
}

@TestOnMac
void testOnMac() {
    // ...
}

@Test
@EnabledOnOs({ LINUX, MAC })
void onLinuxOrMac() {
    // ...
}
```


Condições de propriedades do sistema

- Um contêiner ou teste pode ser ativado ou desativado com base no valor da propriedade do sistema JVM nomeada por meio das anotações `@EnabledIfSystemProperty` e `@DisabledIfSystemProperty`.

```
@Test
@EnabledIfSystemProperty(named = "os.arch", matches = ".*64.*")
void onlyOn64BitArchitectures() {
    // ...
}

@Test
@DisabledIfSystemProperty(named = "ci-server", matches = "true")
void notOnCiServer() {
    // ...
}
```

Condições baseadas em script

- O jUnit Jupiter fornece a capacidade de ativar ou desativar um contêiner ou teste, dependendo da avaliação de um script configurado por meio da anotação `@EnabledIf` ou `@DisabledIf`.

```
@Test // Static JavaScript expression.  
@EnabledIf("2 * 3 == 6")  
void willBeExecuted() {  
    // ...  
}  
  
@RepeatedTest(10) // Dynamic JavaScript expression.  
@DisabledIf("Math.random() < 0.314159")  
void mightNotBeExecuted() {  
    // ...  
}
```

Marcando Testes

- As classes e métodos de teste podem ser marcados através da anotação @Tag. Essas tags podem ser usadas posteriormente para filtrar a descoberta e a execução de testes.
- Condições para nomear uma tag:
 1. Uma tag não deve ser nula ou em branco.
 2. Uma tag não deve conter espaços em branco.
 3. Uma tag aparada não deve conter caracteres reservados.

Marcando Testes - Exemplo

```
import org.junit.jupiter.api.Tag;  
import org.junit.jupiter.api.Test;  
  
@Tag("fast")  
@Tag("model")  
class TaggingDemo {  
  
    @Test  
    @Tag("taxes")  
    void testingTaxCalculation() {  
    }  
  
}
```

Ordem de Execução dos Testes

- Por padrão, os métodos de teste serão ordenados usando um algoritmo que é determinístico.
- Embora os testes de unidade verdadeiros normalmente não devam depender da ordem em que são executados, há momentos em que é necessário impor uma ordem de execução específica do método de teste - por exemplo, ao gravar testes de integração ou testes funcionais em que a sequência dos testes é importante.
- Para controlar a ordem em que os métodos de teste são executados, anote sua classe de teste ou interface de teste com `@TestMethodOrder` e especifique a implementação desejada do `MethodOrderer`.

Ordem de Execução dos Testes

```
@TestMethodOrder(OrderAnnotation.class)
class OrderedTestsDemo {

    @Test
    @Order(1)
    void nullValues() {
        // perform assertions against null values
    }

    @Test
    @Order(2)
    void emptyValues() {
        // perform assertions against empty values
    }
}
```

Repetição de Testes

- O JUnit Jupiter fornece a capacidade de repetir um teste um número especificado de vezes, anotando um método com `@RepeatedTest` e especificando o número total de repetições desejadas. Cada chamada de um teste repetido se comporta como a execução de um método regular `@Test` com suporte completo para os mesmos retornos de chamada e extensões do ciclo de vida.
- O exemplo a seguir demonstra como declarar um teste chamado `repeatTest ()` que será repetido automaticamente 10 vezes.

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

Repetição de Testes - Exemplo

```
class RepeatedTestsDemo {  
  
    private Logger logger = // ...  
  
    @BeforeEach  
    void beforeEach(TestInfo testInfo, RepetitionInfo repetitionInfo) {  
        int currentRepetition = repetitionInfo.getCurrentRepetition();  
        int totalRepetitions = repetitionInfo.getTotalRepetitions();  
        String methodName = testInfo.getTestMethod().get().getName();  
        logger.info(String.format("About to execute repetition %d of %d for %s",  
            currentRepetition, totalRepetitions, methodName));  
    }  
  
    @RepeatedTest(10)  
    void repeatedTest() {  
        // ...  
    }  
  
    @RepeatedTest(5)  
    void repeatedTestWithRepetitionInfo(RepetitionInfo repetitionInfo) {  
        assertEquals(5, repetitionInfo.getTotalRepetitions());  
    }  
}
```


Testes Parametrizados

- Testes parametrizados tornam possível executar um teste várias vezes com argumentos diferentes. Eles são declarados como métodos normais `@Test`, mas usam a anotação `@ParameterizedTest`. Além disso, você deve declarar pelo menos uma fonte que fornecerá os argumentos para cada chamada e, em seguida, consumirá os argumentos no método de teste.
- O exemplo a seguir demonstra um teste parametrizado que usa a anotação `@ValueSource` para especificar uma matriz `String` como a fonte dos argumentos.

Testes Parametrizados - Exemplo

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

```
palindromes(String) ✓
├─ [1] racecar ✓
├─ [2] radar ✓
└─ [3] able was I ere I saw elba ✓
```

Tempo Limite de um Teste

- A anotação `@Timeout` permite declarar que um teste, fábrica de teste, modelo de teste ou método de ciclo de vida deve falhar se o tempo de execução exceder uma determinada duração. A unidade de tempo para a duração é padronizada em segundos, mas é configurável.
- O exemplo a seguir mostra como o `@Timeout` é aplicado ao ciclo de vida e aos métodos de teste.

```
class TimeoutDemo {  
  
    @BeforeEach  
    @Timeout(5)  
    void setUp() {  
        // fails if execution time exceeds 5 seconds  
    }  
  
    @Test  
    @Timeout(value = 100, unit = TimeUnit.MILLISECONDS)  
    void failsIfExecutionTimeExceeds100Milliseconds() {  
        // fails if execution time exceeds 100 milliseconds  
    }  
  
}
```

Q1) [CESPE TRE-MT 2010] Assinale a opção correta a respeito de JUnit.

- a) JUnit é um framework open-source (arcabouço livre) para escrever e executar testes automaticamente, sem necessidade de escrever código adicional.
- b) Em JUnit 4.x, a anotação `@Before` permite inicializar variáveis antes de executar métodos de teste. É possível ter múltiplos métodos anotados com `@Before`.
- c) A classe `AssertEquals(a,b)` compara dois valores. O teste é executado com sucesso se `a.equals(b)`.
- d) Fixture significa um conjunto de dados de teste e objetos utilizados na execução de um e somente um teste.
- e) JUnit 4.x utiliza a anotação `@JavaTest` para identificar os métodos que são métodos de teste.

Q1) [CESPE TRE-MT 2010] Assinale a opção correta a respeito de JUnit.

- a) JUnit é um framework open-source (arcabouço livre) para escrever e executar testes automaticamente, sem necessidade de escrever código adicional.
- b) Em JUnit 4.x, a anotação `@Before` permite inicializar variáveis antes de executar métodos de teste. É possível ter múltiplos métodos anotados com `@Before`.
- c) A classe `AssertEquals(a,b)` compara dois valores. O teste é executado com sucesso se `a.equals(b)`.
- d) Fixture significa um conjunto de dados de teste e objetos utilizados na execução de um e somente um teste.
- e) JUnit 4.x utiliza a anotação `@JavaTest` para identificar os métodos que são métodos de teste.

Q2) [FCC TRT - 23ª REGIÃO (MT) 2016] Um Técnico do Tribunal, que utiliza JUnit, sabe que dentre os métodos da classe TestCase aquele que é chamado depois de cada método de teste, usado para desfazer o que setUp() fez como, por exemplo, fechar uma conexão de banco de dados, é o

- a) run()
- b) setOff()
- c) tearDown()
- d) runTest()
- e) command()

Q2) [FCC TRT - 23ª REGIÃO (MT) 2016] Um Técnico do Tribunal, que utiliza JUnit, sabe que dentre os métodos da classe TestCase aquele que é chamado depois de cada método de teste, usado para desfazer o que setUp() fez como, por exemplo, fechar uma conexão de banco de dados, é o

- a) run()
- b) setOff()
- c) tearDown()
- d) runTest()
- e) command()

Q3) [FCC TRT - 14ª Região (RO e AC) 2016] Para a criação de testes unitários utilizando o JUnit, um Técnico utilizou o método

- a) assertEquals() que testa a igualdade entre dois objetos (esperado × retornado).
- b) assertObjFalse() que testa se o valor de um objeto é falso.
- c) setUp() que inicia um teste unitário e abre uma sessão do JUnit.
- d) assertNotNull() que testa um retorno booleano não nulo.
- e) tearDown() que finaliza um teste unitário e fecha a sessão do JUnit.

Q4)[CESPE STJ 2018] Julgue o seguinte item, relativo a métrica de qualidade de software, JUnit, SQL, Delphi e desenvolvimento mobile.

Uma característica e limitação do JUnit é a impossibilidade de definição de parâmetros para construtores e métodos.

Q3) [FCC TRT - 14ª Região (RO e AC) 2016] Para a criação de testes unitários utilizando o JUnit, um Técnico utilizou o método

- a) `assertEquals()` que testa a igualdade entre dois objetos (esperado × retornado).
- b) `assertObjFalse()` que testa se o valor de um objeto é falso.
- c) `setUp()` que inicia um teste unitário e abre uma sessão do JUnit.
- d) `assertNotNull()` que testa um retorno booleano não nulo.
- e) `tearDown()` que finaliza um teste unitário e fecha a sessão do JUnit.

Q4)[CESPE STJ 2018] Julgue o seguinte item, relativo a métrica de qualidade de software, JUnit, SQL, Delphi e desenvolvimento mobile.

Uma característica e limitação do JUnit é a impossibilidade de definição de parâmetros para construtores e métodos.

ERRADO

Q5) [COSEAC UFF 2016] Para que um método seja considerado um teste na versão 4 do JUnit, o desenvolvedor deve

- a) iniciá-lo com o prefixo test.
- b) registrá-lo usando o método estático RegisterTests.
- c) anotá-lo com @Test, torná-lo público e sem retorno (void).
- d) apenas mantê-lo com visibilidade pública.

Q6) [CESPE ANATEL 2014] Com relação a testes de sistemas de software, julgue o item a seguir.

Por meio do JUnit, framework de teste que viabiliza a documentação e execução automática de testes de unidade em Java, é possível obter informações a respeito da cobertura obtida pelos casos de teste.

Q5) [COSEAC UFF 2016] Para que um método seja considerado um teste na versão 4 do JUnit, o desenvolvedor deve

- a) iniciá-lo com o prefixo test.
- b) registrá-lo usando o método estático RegisterTests.
- c) anotá-lo com `@Test`, torná-lo público e sem retorno (void).
- d) apenas mantê-lo com visibilidade pública.

Q6) [CESPE ANATEL 2014] Com relação a testes de sistemas de software, julgue o item a seguir.

Por meio do JUnit, framework de teste que viabiliza a documentação e execução automática de testes de unidade em Java, é possível obter informações a respeito da cobertura obtida pelos casos de teste.

ERRADO

Q7) [FCC TRT20 2016] Em um método de uma classe de teste JUnit deseja-se testar o método estático chamado `multiplica` da classe `Vetores`. Para verificar se a saída desse método será 39 quando ele receber como parâmetro os arrays de números inteiros {3,4} e {5,6} utiliza-se a instrução

- a) `multiplica(new int{ 3, 4}, new int{ 5, 6}).assertEquals(39);`
- b) `assertEquals(39, Vetores.multiplica(new int[] { 3, 4}, new int[] { 5, 6}));`
- c) `Vetores.multiplica.assertTrue(new int{ 3, 4}, new int{ 5, 6}).assertEquals(39);`
- d) `Vetores.multiplica.assertFalse(39, (new int[] { 3, 4}, new int[] { 5, 6}));`
- e) `multiplica.assertTrue(39, (new int[] { 3, 4}, new int[] { 5, 6}));`

Q7) [FCC TRT20 2016] Em um método de uma classe de teste JUnit deseja-se testar o método estático chamado `multiplica` da classe `Vetores`. Para verificar se a saída desse método será 39 quando ele receber como parâmetro os arrays de números inteiros {3,4} e {5,6} utiliza-se a instrução

a) `multiplica(new int{ 3, 4}, new int{ 5, 6}).assertEquals(39);`

b) `assertEquals(39, Vetores.multiplica(new int[] { 3, 4}, new int[] { 5, 6}));`

c) `Vetores.multiplica.assertTrue(new int{ 3, 4}, new int{ 5, 6}).assertEquals(39);`

d) `Vetores.multiplica.assertFalse(39, (new int[] { 3, 4}, new int[] { 5, 6}));`

e) `multiplica.assertTrue(39, (new int[] { 3, 4}, new int[] { 5, 6}));`

Q8) [CESPE STF 2013] Com relação a testes de sistemas de software, julgue o item a seguir.

O método setUp() é utilizado para sinalizar o início do processo de teste, ao passo que o método tearDown() sinaliza o final desse processo, desfazendo o que o setUp() fez.

Q9) [CESPE STJ 2015] Julgue o seguinte item, relativo a conceitos de bibliotecas, serviços e utilitários Java.

JUnit é um framework utilizado para facilitar a geração de testes a fim de se verificar se os resultados gerados pelos métodos escritos em Java são os esperados.

Q8) [CESPE ANATEL 2014] Com relação a testes de sistemas de software, julgue o item a seguir.

O método setUp() é utilizado para sinalizar o início do processo de teste, ao passo que o método tearDown() sinaliza o final desse processo, desfazendo o que o setUp() fez.

CERTO

Q9) [CESPE STJ 2015] Julgue o seguinte item, relativo a conceitos de bibliotecas, serviços e utilitários Java.

JUnit é um framework utilizado para facilitar a geração de testes a fim de se verificar se os resultados gerados pelos métodos escritos em Java são os esperados.

CERTO

Q10) [CESPE TRE-ES 2011] Considerando que a aplicação de testes em um programa possibilita verificar se ele atende à sua especificação e se realiza o que o cliente deseja, julgue os itens subsecutivos, relativos a testes de software.

O framework JUnit, embora tenha sido projetado para realizar testes de unidade em um programa, também é utilizado para realizar testes funcionais.

Q11) [CESPE MPU 2010] No processo de teste de software, uma das metas consiste em demonstrar ao desenvolvedor e ao cliente que o software atende aos requisitos, e outra, em descobrir falhas ou defeitos no software que apresenta comportamento incorreto. Quanto aos processos de teste de software, julgue o próximo item.

O Junit é um conjunto de classes em Java que pode ser estendido para se criar um ambiente de testes de regressão automatizado.

Q10) [CESPE TRE-ES 2011] Considerando que a aplicação de testes em um programa possibilita verificar se ele atende à sua especificação e se realiza o que o cliente deseja, julgue os itens subsecutivos, relativos a testes de software.

O framework JUnit, embora tenha sido projetado para realizar testes de unidade em um programa, também é utilizado para realizar testes funcionais.

CERTO

Q11) [CESPE MPU 2010] No processo de teste de software, uma das metas consiste em demonstrar ao desenvolvedor e ao cliente que o software atende aos requisitos, e outra, em descobrir falhas ou defeitos no software que apresenta comportamento incorreto. Quanto aos processos de teste de software, julgue o próximo item.

O Junit é um conjunto de classes em Java que pode ser estendido para se criar um ambiente de testes de regressão automatizado.

CERTO

Q12) [FCC TRE-RN 2011] Em relação ao JUnit, considere:

- I. É um framework que auxilia a criação e execução de testes unitários sobre classes Java.
- II. Possui integração com várias IDEs e é largamente utilizado em equipes de Extreme Programming.
- III. Permite testes de unidades, conhecidos como "caixa branca", facilitando assim a correção de métodos e objetos.
- IV. Permite criar hierarquia de testes para testar todo ou apenas parte do sistema.

Está correto o que se afirma em:

- a) II, III e IV, apenas.
- b) II e IV, apenas.
- c) I e III, apenas.
- d) I, II, III e IV.
- e) I e II, apenas.

Q12) [FCC TRE-RN 2011] Em relação ao JUnit, considere:

- I. É um framework que auxilia a criação e execução de testes unitários sobre classes Java.
- II. Possui integração com várias IDEs e é largamente utilizado em equipes de Extreme Programming.
- III. Permite testes de unidades, conhecidos como "caixa branca", facilitando assim a correção de métodos e objetos.
- IV. Permite criar hierarquia de testes para testar todo ou apenas parte do sistema.

Está correto o que se afirma em:

- a) II, III e IV, apenas.
- b) II e IV, apenas.
- c) I e III, apenas.
- d) I, II, III e IV.
- e) I e II, apenas.

Q13) [CESPE FUB 2013] No que se refere à engenharia de software, julgue o item a seguir.

O desenvolvimento de software direcionado a testes pode contar com o uso de ferramentas automatizadas para criação de testes, como, por exemplo, o JUnit, um framework para testes em Java.

Q14) [CESPE MEC 2015] Com relação à análise e à avaliação de riscos em projetos de teste de software, julgue o item a seguir.

A JUnit é uma biblioteca típica de testes de apoio e fornece suporte para a execução de testes, geração de logs e verificação de resultados. Por meio do conjunto de classes C# do JUnit, é possível a criação de ambiente de testes automatizado.

Q13) [CESPE FUB 2013] No que se refere à engenharia de software, julgue o item a seguir.

O desenvolvimento de software direcionado a testes pode contar com o uso de ferramentas automatizadas para criação de testes, como, por exemplo, o JUnit, um framework para testes em Java.

CERTO

Q14) [CESPE MEC 2015] Com relação à análise e à avaliação de riscos em projetos de teste de software, julgue o item a seguir.

A JUnit é uma biblioteca típica de testes de apoio e fornece suporte para a execução de testes, geração de logs e verificação de resultados. Por meio do conjunto de classes C# do JUnit, é possível a criação de ambiente de testes automatizado.

ERRADO

Q15) [CESPE BRB 2011] A respeito de teste de software, julgue os itens a seguir.

JUnit é um framework open source que realiza testes unitários funcionais e de integração em aplicações desenvolvidas em qualquer linguagem.

Q16) [CESPE FUNPRES P 2016] A respeito das tecnologias relacionadas ao desenvolvimento web em Java, julgue o item a seguir.

No JUnit, os testes são realizados em sequência, por isso eles mantêm uma relação de dependência entre si.

Q15) [CESPE BRB 2011] A respeito de teste de software, julgue os itens a seguir.

JUnit é um framework open source que realiza testes unitários funcionais e de integração em aplicações desenvolvidas em qualquer linguagem.

ERRADO

Q16) [CESPE FUNPRES P 2016] A respeito das tecnologias relacionadas ao desenvolvimento web em Java, julgue o item a seguir.

No JUnit, os testes são realizados em sequência, por isso eles mantêm uma relação de dependência entre si.

ERRADO

Q17) [CESPE TRE-BA 2017] Uma equipe de desenvolvimento de projeto de automação comercial que padroniza os testes de software e, para isso, utiliza o JUnit para testar as classes Java desse projeto deve definir um método

- a) que não receba parâmetros.
- b) público sem retorno de parâmetros.
- c) público com retorno booleano.
- d) privado com retorno booleano.
- e) privado sem retorno booleano.

Q17) [CESPE TRE-BA 2017] Uma equipe de desenvolvimento de projeto de automação comercial que padroniza os testes de software e, para isso, utiliza o JUnit para testar as classes Java desse projeto deve definir um método

a) que não receba parâmetros.

b) público sem retorno de parâmetros.

c) público com retorno booleano.

d) privado com retorno booleano.

e) privado sem retorno booleano.

Q18) [FCC TRT - 11ª Região (AM e RR) 2017] Realizar testes de unidade é uma prática fundamental no desenvolvimento de software. Em projetos de software criados com Java, essa prática pode ser implementada com a ajuda de um popular framework denominado

- a) Apex.
- b) JUnit.
- c) GitUnit.
- d) VersionUnit.
- e) JTestUnit.

Q19) [CESPE CGE CE 2019] Nos testes unitários, é possível testar uma classe ou até mesmo um objeto Java. Nesse contexto, uma classe, para ser testada pelo Junit, deve se estender a partir de

- a) Collection.
- b) TestExecution.
- c) TestSetUp.
- d) TestCase.
- e) TestReporter.

Q18) [FCC TRT - 11ª Região (AM e RR) 2017] Realizar testes de unidade é uma prática fundamental no desenvolvimento de software. Em projetos de software criados com Java, essa prática pode ser implementada com a ajuda de um popular framework denominado

a) Apex.

b) JUnit.

c) GitUnit.

d) VersionUnit.

e) JTestUnit.

Q19) [CESPE CGE CE 2019] Nos testes unitários, é possível testar uma classe ou até mesmo um objeto Java. Nesse contexto, uma classe, para ser testada pelo Junit, deve se estender a partir de

a) Collection.

b) TestExecution.

c) TestSetUp.

d) TestCase.

e) TestReporter.

Q20) [COMPERVE UFRN 2018] O JUnit é um framework de teste de unidade para aplicações Java.

A respeito do JUnit, analise as afirmativas abaixo.

I. Na versão 4 do JUnit, quando se utiliza o método `assertEquals()` do JUnit para comparar duas variáveis do tipo `double`, é possível passar um terceiro parâmetro que corresponde ao delta, que corresponde à diferença máxima que será tolerada entre os dois números comparados.

II. Um dos métodos pertencentes ao framework JUnit é o método `assertObject()`, que compara quaisquer duas variáveis do tipo `Object`.

III. A anotação `@Before` pode ser associada a um método de testes JUnit e garante que este será o primeiro método de teste a ser executado.

IV. A versão 4 do JUnit oferece o método `assertThat()`, que traz maior flexibilidade às comparações que podem ser realizadas no corpo de um método de testes.

Estão corretas as afirmativas:

- a) II e III.
- b) I e III.
- c) I e IV.
- d) II e IV.

Q20) [COMPERVE UFRN 2018] O JUnit é um framework de teste de unidade para aplicações Java.

A respeito do JUnit, analise as afirmativas abaixo.

I. Na versão 4 do JUnit, quando se utiliza o método `assertEquals()` do JUnit para comparar duas variáveis do tipo `double`, é possível passar um terceiro parâmetro que corresponde ao delta, que corresponde à diferença máxima que será tolerada entre os dois números comparados.

II. Um dos métodos pertencentes ao framework JUnit é o método `assertObject()`, que compara quaisquer duas variáveis do tipo `Object`.

III. A anotação `@Before` pode ser associada a um método de testes JUnit e garante que este será o primeiro método de teste a ser executado.

IV. A versão 4 do JUnit oferece o método `assertThat()`, que traz maior flexibilidade às comparações que podem ser realizadas no corpo de um método de testes.

Estão corretas as afirmativas:

a) II e III.

b) I e III.

c) I e IV.

d) II e IV.

Q21) [FCC CREMESP 2016] Considere que um Analista de Sistemas quer realizar um teste de unidade usando o JUnit para verificar a precisão de números em ponto flutuante. Em condições ideais de implementação e execução, o Analista escreveu o seguinte trecho de código em Java:

```
@Test
public void testeDePontoFlutuante() {
    float result;
    result = 5/2;

    I
    .....
}
```

Para verificar se o resultado está correto e que respeite uma diferença entre os parâmetros, a lacuna I deve ser preenchida com

- a) `assertNotSame(2.5, result);`
- b) `assertEquals(2,5; result; 0,01);`
- c) `assertNotEqual(2.5, result, fail());`
- d) `assertEquals(2.5, result, 0.1);`
- e) `assertEquals(result, 2.50);`

Q21) [FCC CREMESP 2016] Considere que um Analista de Sistemas quer realizar um teste de unidade usando o JUnit para verificar a precisão de números em ponto flutuante. Em condições ideais de implementação e execução, o Analista escreveu o seguinte trecho de código em Java:

```
@Test
public void testeDePontoFlutuante() {
    float result;
    result = 5/2;

    I
    .....
}
```

Para verificar se o resultado está correto e que respeite uma diferença entre os parâmetros, a lacuna I deve ser preenchida com

- a) `assertNotSame(2.5, result);`
- b) `assertEquals(2,5; result; 0,01);`
- c) `assertNotEqual(2.5, result, fail());`
- d) `assertEquals(2.5, result, 0.1);`
- e) `assertEquals(result, 2.50);`

Q22) [FCC TRT13 2014] A equipe de desenvolvimento do Tribunal Regional do Trabalho da 13a Região utiliza a plataforma Java e seus recursos para desenvolver sistemas de software. Em determinado momento, tiveram que testar se os métodos das classes estavam produzindo os resultados esperados. Fizeram tanto testes isolados como baterias de testes automatizados baseados em modelos de testes padrão. Para realizar estes testes, optaram por utilizar o framework open-source mais popular atualmente com suporte à criação de testes automatizados para aplicações construídas em Java.

O framework utilizado e o tipo de teste realizado foram, respectivamente,

- a) JTest e teste de carga.
- b) JMeter e teste de unidade.
- c) JRun e teste funcional.
- d) JUnit e teste de unidade.
- e) JMeter e teste de carga.

Q22) [FCC TRT13 2014] A equipe de desenvolvimento do Tribunal Regional do Trabalho da 13a Região utiliza a plataforma Java e seus recursos para desenvolver sistemas de software. Em determinado momento, tiveram que testar se os métodos das classes estavam produzindo os resultados esperados. Fizeram tanto testes isolados como baterias de testes automatizados baseados em modelos de testes padrão. Para realizar estes testes, optaram por utilizar o framework open-source mais popular atualmente com suporte à criação de testes automatizados para aplicações construídas em Java.

O framework utilizado e o tipo de teste realizado foram, respectivamente,

- a) JTest e teste de carga.
- b) JMeter e teste de unidade.
- c) JRun e teste funcional.
- d) JUnit e teste de unidade.
- e) JMeter e teste de carga.

Q23) [FCC SANASA 2019] Considere que uma aplicação Java possua a classe abaixo.

```
public class Calculadora {  
    public double raiz(double n1) {  
        return Math.sqrt(n1);  
    }  
}
```

Para testar o método raiz utilizando JUnit, foi criada a classe de teste a seguir:

```
import org.junit.Test;  
import static org.junit.Assert.*;  
public class CalculadoraTest {  
    private final Calculadora calc;  
    public CalculadoraTest() {  
        calc = new Calculadora();  
    }  
    @Test  
    public void testRaiz() {  
        I ..... ;  
    }  
}
```

Para que o teste seja aprovado, a lacuna I deverá ser preenchida por

- a) Assert.assertEquals(calc.raiz(16),4,true);
- b) assertEquals(4, calc.raiz(16), 0)
- c) Assert.assertTest(4, calc.raiz(16),1)
- d) assertTrue(4, calc.raiz(16))
- e) Assert.test(4, calc.raiz(16))

Q23) [FCC SANASA 2019] Considere que uma aplicação Java possua a classe abaixo.

```
public class Calculadora {  
    public double raiz(double n1) {  
        return Math.sqrt(n1);  
    }  
}
```

Para testar o método raiz utilizando JUnit, foi criada a classe de teste a seguir:

```
import org.junit.Test;  
import static org.junit.Assert.*;  
public class CalculadoraTest {  
    private final Calculadora calc;  
    public CalculadoraTest() {  
        calc = new Calculadora();  
    }  
    @Test  
    public void testRaiz() {  
        I ..... ;  
    }  
}
```

Para que o teste seja aprovado, a lacuna I deverá ser preenchida por

- a) Assert.assertEquals(calc.raiz(16),4,true);
- b) assertEquals(4, calc.raiz(16), 0)
- c) Assert.assertTest(4, calc.raiz(16),1)
- d) assertTrue(4, calc.raiz(16))
- e) Assert.test(4, calc.raiz(16))

Q24) [FCC SANASA 2019] Um analista de TI está participando do desenvolvimento de um software orientado a objetos utilizando a plataforma Java. Na abordagem de desenvolvimento adotada, o código é desenvolvido de forma incremental, em conjunto com o teste para esse incremento, de forma que só se passa para o próximo incremento quando o atual passar no teste. Como o código é desenvolvido em incrementos muito pequenos e são executados testes a cada vez que uma funcionalidade é adicionada ou que o programa é refatorado, foi necessário definir um ambiente de testes automatizados utilizando um framework popular que suporta o teste de programas Java.

A abordagem de desenvolvimento adotada e o framework de suporte à criação de testes automatizados são, respectivamente,

- a) Behavior-Driven Development e JTest.
- b) Extreme Programming e Selenium.
- c) Test-Driven Development e Jenkins.
- d) Data-Driven Development and Test e JUnit.
- e) Test-Driven Development e JUnit.

Q24) [FCC SANASA 2019] Um analista de TI está participando do desenvolvimento de um software orientado a objetos utilizando a plataforma Java. Na abordagem de desenvolvimento adotada, o código é desenvolvido de forma incremental, em conjunto com o teste para esse incremento, de forma que só se passa para o próximo incremento quando o atual passar no teste. Como o código é desenvolvido em incrementos muito pequenos e são executados testes a cada vez que uma funcionalidade é adicionada ou que o programa é refatorado, foi necessário definir um ambiente de testes automatizados utilizando um framework popular que suporta o teste de programas Java.

A abordagem de desenvolvimento adotada e o framework de suporte à criação de testes automatizados são, respectivamente,

- a) Behavior-Driven Development e JTest.
- b) Extreme Programming e Selenium.
- c) Test-Driven Development e Jenkins.
- d) Data-Driven Development and Test e JUnit.
- e) Test-Driven Development e JUnit.

Q25) [FCC TRT14 2016] Considere o código em Java que realiza testes unitários com o framework JUnit:

As lacunas I e II devem ser, correta e respectivamente, preenchidas com

- a) extends TestCase e MatematicaTest.class
- b) extends junit e MatematicaTest.
- c) protected e assertEquals (MatematicaTest.class).
- d) protected TestCase e assertEquals(MatematicaTest)
- e) throws Exception e assertSame (MatematicaTest)

```
import junit.framework.TestCase;

public class MatematicaTest I {

    private Matematica math=new Matematica(18, 3);

    public void testadd() {
        this.assertTrue(this.math.add()==21);
    }
    public void testsubtract() {
        this.assertTrue(this.math.subtract()==15);
    }
    public void testmultiply() {
        this.assertTrue(this.math.multiply()==54);
    }
    public void testdivide() {
        this.assertTrue(this.math.divide()==6);
    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run( II );
    }
}
```

Q25) [FCC TRT14 2016] Considere o código em Java que realiza testes unitários com o framework JUnit:

As lacunas I e II devem ser, correta e respectivamente, preenchidas com

a) `extends TestCase` e `MatematicaTest.class`

b) `extends junit` e `MatematicaTest`.

c) `protected` e `assertEquals` (`MatematicaTest.class`).

d) `protected TestCase` e `assertEquals(MatematicaTest)`

e) `throws Exception` e `assertSame` (`MatematicaTest`)

```
import junit.framework.TestCase;

public class MatematicaTest I {

    private Matematica math=new Matematica(18, 3);

    public void testadd() {
        this.assertTrue(this.math.add()==21);
    }
    public void testsubtract() {
        this.assertTrue(this.math.subtract()==15);
    }
    public void testmultiply() {
        this.assertTrue(this.math.multiply()==54);
    }
    public void testdivide() {
        this.assertTrue(this.math.divide()==6);
    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run( II );
    }
}
```

Q26) [FCC TRT24 2017] Considere que foi criado um teste de funcionalidade com o Selenium e o JUnit, no qual foram usados uma aplicação web, o Selenium server e o Selenium test client com o JUnit, em condições ideais. O objetivo é testar o tamanho de um combo box em uma página jsp. O Selenium abrirá um browser, chamará a página e testará a combo box. A página é acessível pela url: <http://localhost:8080/teste/pagina.jsp>. Com a aplicação web e o servidor Selenium executando adequadamente, o teste com JUnit é mostrado no trecho de código abaixo.

Neste cenário, as lacunas I, II e III são correta e respectivamente preenchidas com as anotações

- a) @Before – @After – @TestRun
- b) @BeforeClass – @AfterClass – @Test
- c) @BeforeClass – @AfterClass – @Test(timeout==5000)
- d) @BeforeObject– @AfterObject – @Test(expected.Exception.class)
- e) @Before – @After – @Test(expected.IllegalArgumentException.class)

```
private static DefaultSelenium selenium;

    I
    .....

    public static void setup() {
        String url = "http://localhost:8080";
        selenium = new DefaultSelenium("localhost", 4444,
                                        "*firefox /usr/lib/firefox/firefox-bin", url);

        selenium.start();
    }

    II
    .....

    public static void tearDown() {
        selenium.stop();
    }

    III
    .....

    public void testSelectedIdOfSizeComboBox() {
        selenium.open("/teste/pagina.jsp");
        assertEquals("medium", selenium.getSelectedId("size"));
    }
```


Q26) [FCC TRT24 2017] Considere que foi criado um teste de funcionalidade com o Selenium e o JUnit, no qual foram usados uma aplicação web, o Selenium server e o Selenium test client com o JUnit, em condições ideais. O objetivo é testar o tamanho de um combo box em uma página jsp. O Selenium abrirá um browser, chamará a página e testará a combo box. A página é acessível pela url: `http://localhost:8080/teste/pagina.jsp`. Com a aplicação web e o servidor Selenium executando adequadamente, o teste com JUnit é mostrado no trecho de código abaixo.

Neste cenário, as lacunas I, II e III são correta e respectivamente preenchidas com as anotações

a) `@Before` – `@After` – `@TestRun`

b) `@BeforeClass` – `@AfterClass` – `@Test`

c) `@BeforeClass` – `@AfterClass` – `@Test(timeout==5000)`

d) `@BeforeObject`– `@AfterObject` – `@Test(expected.Exception.class)`

e) `@Before` – `@After` – `@Test(expected.IllegalArgumentException.class)`

```
private static DefaultSelenium selenium;

    I
    .....

    public static void setup() {
        String url = "http://localhost:8080";
        selenium = new DefaultSelenium("localhost", 4444,
                                        "*firefox /usr/lib/firefox/firefox-bin", url);

        selenium.start();
    }

    II
    .....

    public static void tearDown() {
        selenium.stop();
    }

    III
    .....

    public void testSelectedIdOfSizeComboBox() {
        selenium.open("/teste/pagina.jsp");
        assertEquals("medium", selenium.getSelectedId("size"));
    }
```

Q27) [CCV-UFC 2019] Sobre JUnit 5, assinale a alternativa correta.

- a) JUnit é um servidor de automação de código.
- b) As classes de teste (Test Class) não devem ser abstratas (abstract) e podem ter mais de um construtor.
- c) JUnit 5 é apresentada como a nova geração do framework. A versão JUnit 5 requer uma versão Java a partir de Java 5.
- d) JUnit 5 é composto por vários módulos diferentes de três diferentes subprojetos: JUnit Base + JUnit Jupiter + JUnit Regular.
- e) A anotação `@TestMethodOrder` em uma classe de testes, juntamente com a anotação `@Order` nos métodos de teste, possibilita controlar a ordem na qual os métodos de teste são executados.

Q27) [CCV-UFC 2019] Sobre JUnit 5, assinale a alternativa correta.

- a) JUnit é um servidor de automação de código.
- b) As classes de teste (Test Class) não devem ser abstratas (abstract) e podem ter mais de um construtor.
- c) JUnit 5 é apresentada como a nova geração do framework. A versão JUnit 5 requer uma versão Java a partir de Java 5.
- d) JUnit 5 é composto por vários módulos diferentes de três diferentes subprojetos: JUnit Base + JUnit Jupiter + JUnit Regular.
- e) A anotação `@TestMethodOrder` em uma classe de testes, juntamente com a anotação `@Order` nos métodos de teste, possibilita controlar a ordem na qual os métodos de teste são executados.

Q28) [COMPERVE UFRN 2018] A respeito do JUnit, analise as afirmativas abaixo.

I Na versão 4 do JUnit, quando se utiliza o método `assertEquals()` do JUnit para comparar duas variáveis do tipo `double`, é possível passar um terceiro parâmetro que corresponde ao delta, que corresponde à diferença máxima que será tolerada entre os dois números comparados.

II Um dos métodos pertencentes ao framework JUnit é o método `assertObject()`, que compara quaisquer duas variáveis do tipo `Object`.

III A anotação `@Before` pode ser associada a um método de testes JUnit e garante que este será o primeiro método de teste a ser executado.

IV A versão 4 do JUnit oferece o método `assertThat()`, que traz maior flexibilidade às comparações que podem ser realizadas no corpo de um método de testes.

Estão corretas as afirmativas

a) II e III.

b) I e III.

c) I e IV.

d) II e IV.

Q28) [COMPERVE UFRN 2018] A respeito do JUnit, analise as afirmativas abaixo.

I Na versão 4 do JUnit, quando se utiliza o método `assertEquals()` do JUnit para comparar duas variáveis do tipo `double`, é possível passar um terceiro parâmetro que corresponde ao delta, que corresponde à diferença máxima que será tolerada entre os dois números comparados.

II Um dos métodos pertencentes ao framework JUnit é o método `assertObject()`, que compara quaisquer duas variáveis do tipo `Object`.

III A anotação `@Before` pode ser associada a um método de testes JUnit e garante que este será o primeiro método de teste a ser executado.

IV A versão 4 do JUnit oferece o método `assertThat()`, que traz maior flexibilidade às comparações que podem ser realizadas no corpo de um método de testes.

Estão corretas as afirmativas

a) II e III.

b) I e III.

c) I e IV.

d) II e IV.

Q29) [COMPERVE UFRN 2018]

I Até a versão 3.8.1 do JUnit, todas as classes de testes precisavam herdar da classe TestCase do framework JUnit.

II A partir da versão 4 do JUnit, para se construir uma classe de teste, precisa-se apenas associar a anotação @Test à declaração de qualquer classe pública.

III Com a anotação @Test(timeout=), é possível definir o tempo de duração do teste em milissegundos. Se a execução ultrapassar o tempo definido, o teste irá acusar a falha.

IV Para que um determinado objeto seja compartilhado entre vários métodos de testes JUnit, deve-se colocar a inicialização do objeto no construtor da classe.

Estão corretas as afirmativas

- a) II e IV.
- b) I e IV.
- c) II e III.
- d) I e III.

Q29) [COMPERVE UFRN 2018] Considere as seguintes afirmativas a respeito do framework JUnit.

I Até a versão 3.8.1 do JUnit, todas as classes de testes precisavam herdar da classe TestCase do framework JUnit.

II A partir da versão 4 do JUnit, para se construir uma classe de teste, precisa-se apenas associar a anotação @Test à declaração de qualquer classe pública.

III Com a anotação @Test(timeout=), é possível definir o tempo de duração do teste em milissegundos. Se a execução ultrapassar o tempo definido, o teste irá acusar a falha.

IV Para que um determinado objeto seja compartilhado entre vários métodos de testes JUnit, deve-se colocar a inicialização do objeto no construtor da classe.

Estão corretas as afirmativas

a) II e IV.

b) I e IV.

c) II e III.

d) I e III.

Q30) [FCC TCE RS 2014] O JUnit é um framework para testes de unidades automatizados na plataforma Java. Em uma classe de teste criada no JUnit versão 4

- a) os métodos de teste são identificados com a anotação `@JUnitTest`.
- b) é necessário iniciar os nomes dos métodos de teste com a palavra `test`.
- c) é necessário utilizar o método `setUp()` para configurar o ambiente de teste antes da execução de cada caso de teste.
- d) as anotações `@BeforeClass` e `@AfterClass` são usadas para marcar métodos que devem ser executados, respectivamente, antes e depois da execução da classe de teste.
- e) é necessário utilizar o método `tearDown()` para configurar o ambiente de teste depois da execução de cada caso de teste.

Q30) [FCC TCE RS 2014] O JUnit é um framework para testes de unidades automatizados na plataforma Java. Em uma classe de teste criada no JUnit versão 4

- a) os métodos de teste são identificados com a anotação `@JUnitTest`.
- b) é necessário iniciar os nomes dos métodos de teste com a palavra `test`.
- c) é necessário utilizar o método `setUp()` para configurar o ambiente de teste antes da execução de cada caso de teste.
- d) as anotações `@BeforeClass` e `@AfterClass` são usadas para marcar métodos que devem ser executados, respectivamente, antes e depois da execução da classe de teste.
- e) é necessário utilizar o método `tearDown()` para configurar o ambiente de teste depois da execução de cada caso de teste.

GABARITO

Q1 – LETRA B. Q14 – ERRADO.

Q2 - LETRA C. Q15 – ERRADO.

Q3 – LETRA A. Q16 – ERRADO.

Q4 - ERRADO. Q17 – LETRA B.

Q5 - LETRA C. Q18 – LETRA B.

Q6 - ERRADO. Q19 - LETRA C.

Q7 – LETRA B. Q20 - LETRA C.

Q8 – CERTO. Q21 - LETRA D.

Q9 - CERTO. Q22 - LETRA D.

Q10 – CERTO. Q23 - LETRA B.

Q11 – CERTO. Q24 – LETRA E.

Q12 - LETRA D. Q25 - LETRA A

Q13 - CERTO. Q26 - LETRA B

Q27 - LETRA E Q28 - LETRA C

Q29 - LETRA D Q30 - LETRA D