GESTÃO E QUALIDADE DE SOFTWARE

TESTES UNITÁRIOS EM JAVA

OPERAÇÕES COM TEMPERATURAS

Aluna: Lara Luísa Ayrolla Abreu

1. O QUE SÃO TESTES UNITÁRIOS E PARA QUE SERVEM?

Testes unitários são testes realizados com a intenção de testar individualmente pequenos pedaço de um código, que tendem a ser funções ou métodos.

Por mais que possam ser realizados manualmente, na maioria das vezes são criados dentro do próprio projeto a ser testado e executados automaticamente.

São muito importantes para verificar a funcionalidade de cada parte do código e ajudam a identificar problemas de forma mais rápida, pois, a falha de um teste unitário é um grande indicativo de que o trecho de código referente a esse teste está com algum erro ou *bug*.

2. TESTES UNITÁRIOS NA LINGUAGEM JAVA

Na linguagem Java, que será utilizada para demonstrar os testes unitários deste relatório, testes unitários são comumente feitos utilizando jUnit, que é uma biblioteca e framework com diversas notações e funcionalidades que facilitam a criação dos testes unitários e a automatização da execução deles.

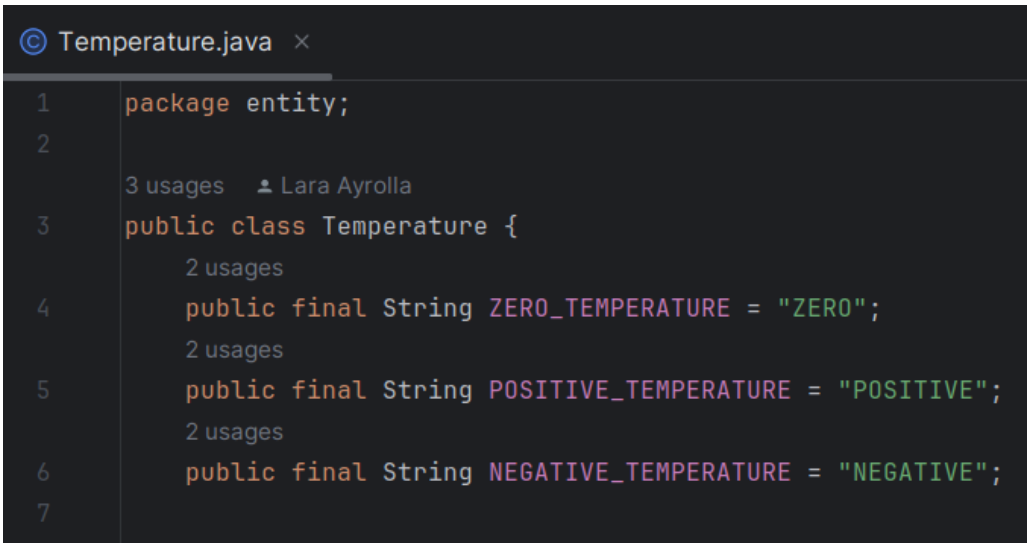
3. FUNCIONALIDADES DO CÓDIGO: OPERAÇÕES COM TEMPERATURA

O código a ser testado foi desenvolvido em Java, no mesmo projeto em que os testes unitários foram desenvolvidos. Para acessar o repositório do código fonte, basta clicar neste link.

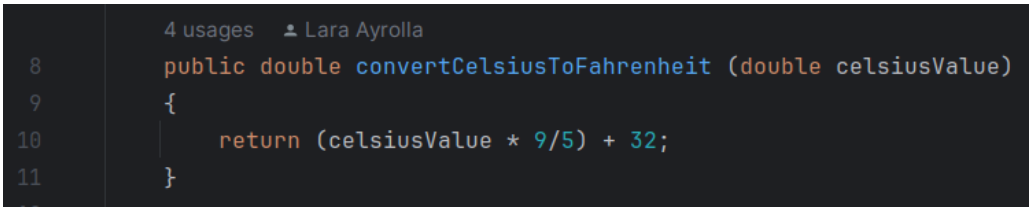
No pacote “entity”, foi criada a classe “Temperature”, na qual estão contidas diversas funções que fazem operações com valores númericos de temperaturas e retornam o resultado.

Ao começo da classe, foram criadas constantes, ou variáveis do tipo “final”, que serão utilizadas ao longo do código pelos métodos da classe.

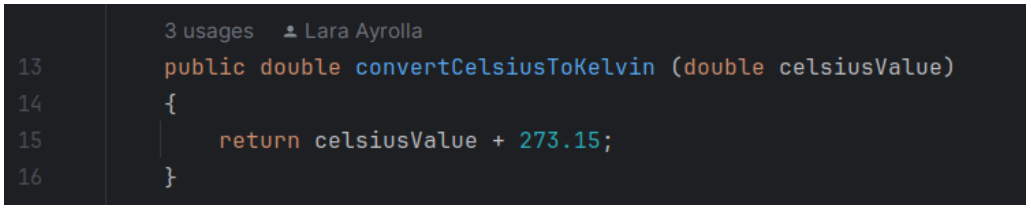
Quanto aos métodos, eles têm as funcionalidades de conversão de temperaturas (Celsius para Fahrenheit e Celsius para Kelvin), classificação de temperaturas (positiva, negativa ou zero), que utiliza as constantes previamente definidas, e verificação da assertividade da conversão de Celsius para Fahrenheit.



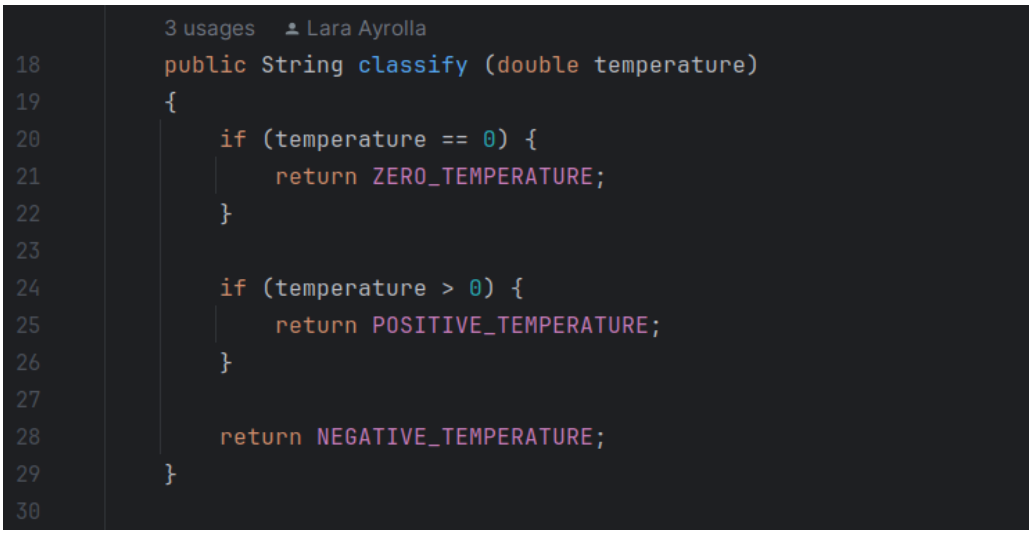
Declaração da classe “Temperature” e de constantes



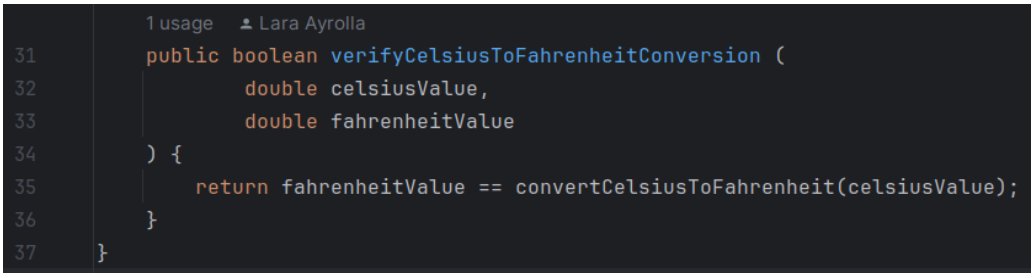
Função que converte uma temperatura em Celsius para Fahrenheit



Função que converte uma temperatura em Celsius para Kelvin



Função que classifica uma temperatura como positiva, negativa ou zero, utilizando constantes



Função que verifica se a conversão de uma temperatura em Celsius para Fahrenheit está correta

4. TESTES DESENVOLVIDOS

No pacote “tests”, foi criada a classe “TemperatureTests”, na qual estão contidos os métodos utilizados para testar as funções da classe “Temperature”.

Ao começo da classe, são feitas as importações da classe que será testada (“Temperature”), da biblioteca que será utilizada para o desenvolvimento dos testes (“jUnit”) e é instanciado um objeto de nome “temperature” do tipo “Temperature”, que será utilizado para chamar as funções da classe testada.

Foram desenvolvidos quatro métodos, com dez casos de teste, que verificam se o funcionamento das funções da classe “Temperature” está correto. Isso é feito através de comparações dos retornos das funções testadas com valores previamente definidos pelos testes como corretos ou incorretos.

Um teste passa somente se o resultado retornado pela função testada, quando recebe os valores predefinidos como parâmetro, coincide com o resultado esperado. Todos os casos de teste de um método de teste devem passar para que esse método passe.

```java
package tests;

import entity.Temperature;
import org.junit.*;

Lara Ayrolla
public class TemperatureTests {
    13 usages
    Temperature temperature = new Temperature();
```

Declaração da classe “TemperatureTests”, importações e instância da classe a ser testada

```java
    Lara Ayrolla
    @Test
    public void shouldConvertCelsiusToFahrenheit ()
    {
        Assert.assertEquals(
                expected: 23,
                temperature.convertCelsiusToFahrenheit( celsiusValue: -5),
                delta: 0
        );

        Assert.assertEquals(
                expected: 32,
                temperature.convertCelsiusToFahrenheit( celsiusValue: 0),
                delta: 0
        );

        Assert.assertEquals(
                expected: 64.4,
                temperature.convertCelsiusToFahrenheit( celsiusValue: 18),
                delta: 0
        );
    }
```

Método que testa a conversão de temperaturas em Celsius para Fahrenheit

```
Lara Ayrolla
@Test
public void shouldConvertCelsiusToKelvin ()
{
    Assert.assertEquals(
            expected: 268.15,
            temperature.convertCelsiusToKelvin( celsiusValue: -5),
            delta: 0
    );

    Assert.assertEquals(
            expected: 273.15,
            temperature.convertCelsiusToKelvin( celsiusValue: 0),
            delta: 0
    );

    Assert.assertEquals(
            expected: 291.15,
            temperature.convertCelsiusToKelvin( celsiusValue: 18),
            delta: 0
    );
}
```

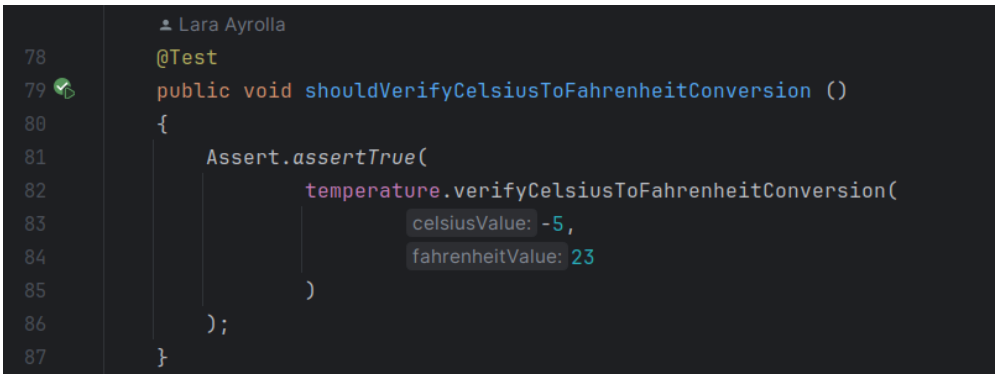
Método que testa a conversão de temperaturas em Celsius para Kelvin

```
Lara Ayrolla
@Test
public void shouldClassifyTemperature ()
{
    Assert.assertTrue(
            temperature.classify( temperature: -5)
            .equals(
                    temperature.NEGATIVE_TEMPERATURE
            )
    );

    Assert.assertTrue(
            temperature.classify( temperature: 0)
            .equals(
                    temperature.ZERO_TEMPERATURE
            )
    );

    Assert.assertTrue(
            temperature.classify( temperature: 18)
            .equals(
                    temperature.POSITIVE_TEMPERATURE
            )
    );
}
```

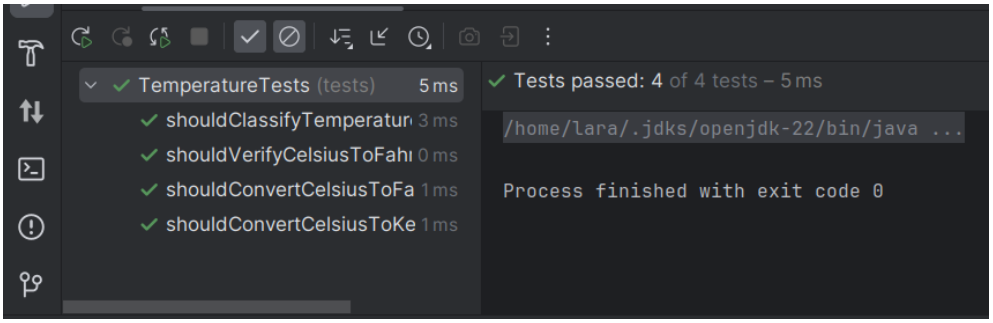
Método que testa a classificação de temperaturas, utilizando constantes da classe “Temperature”



Método que testa a verificação da conversão de temperaturas em Celsius para Fahrenheit

5. RESULTADOS E CONCLUSÃO

Após a execução dos testes da classe “TemperatureTests”, todos os dez casos de teste e, consequentemente, todos os quatro métodos, passaram.



Resultado da execução dos testes unitários

O resultado foi satisfatório e atingiu as expectativas. Isso ocorreu, porque os testes foram feitos de maneira cautelosa, utilizando valores adequados para a comparação com os retornos das funções, e a classe testada cumpriu o objetivo de criar códigos de qualidade que produzem as soluções esperadas.

É importante ressaltar que, mesmo que todas as funções da classe “Temperature” tenham sido testadas e todos os testes tenham passado, não é possível garantir a ausência de *bugs*. Algum caso específico pode não ter sido explorado.

Em adição, é possível que mudanças futuras no código criem *bugs* ou façam com que os testes deixem de passar. Nesse caso, o *bug* deve ser corrigido ou o teste ajustado para funcionar com a nova versão do código.

Por conta dessas incertezas, é importante criar uma base grande de testes, para identificar erros em todos os trechos do código o mais rápido possível, executar os testes com frequência e dar manutenção no código e nos testes.

6. REFERÊNCIAS

DevMedia. **E aí? Como você testa seus códigos?** Disponível em: https://www.devmedia.com.br/e-ai-como-voce-testa-seus-codigos/39478. Acesso em: 2 abr. 2024.

jUnit 5. Disponível em: https://junit.org/junit5/. Acesso em: 2 abr. 2024.

Testing Company. **Testes Automatizados e Unitários: Entenda as suas características e diferenças.** Disponível em: https://testingcompany.com.br/blog/testes-automatizados-e-unitarios-entenda-as-suas-caracteristicas-e-diferencas#:~:text=Comumente%2C%20testes%20unit%C3%A1rios%20s%C3%A3o%20desenvolvidos,nas%20fases%20iniciais%20do%20projeto. Acesso em: 2 abr. 2024.