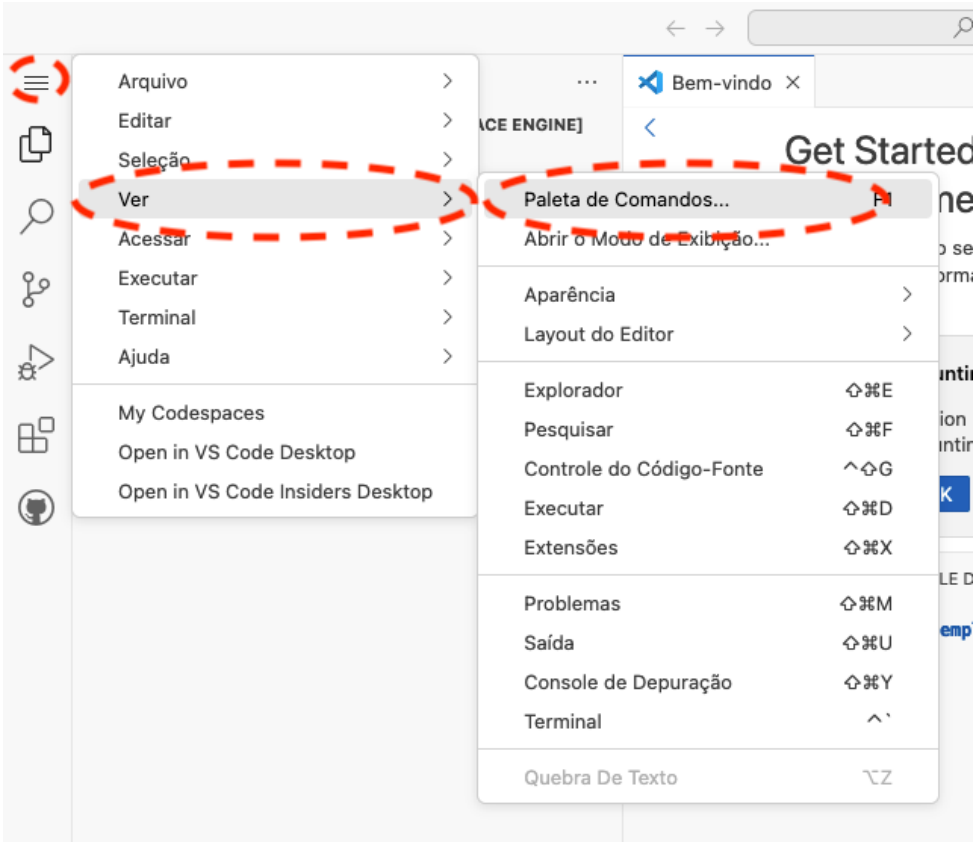
Projeto e Arquitetura de Software
Prof. Marcelo H. Yamaguti

EXERCÍCIO DE USO DO MAVEN

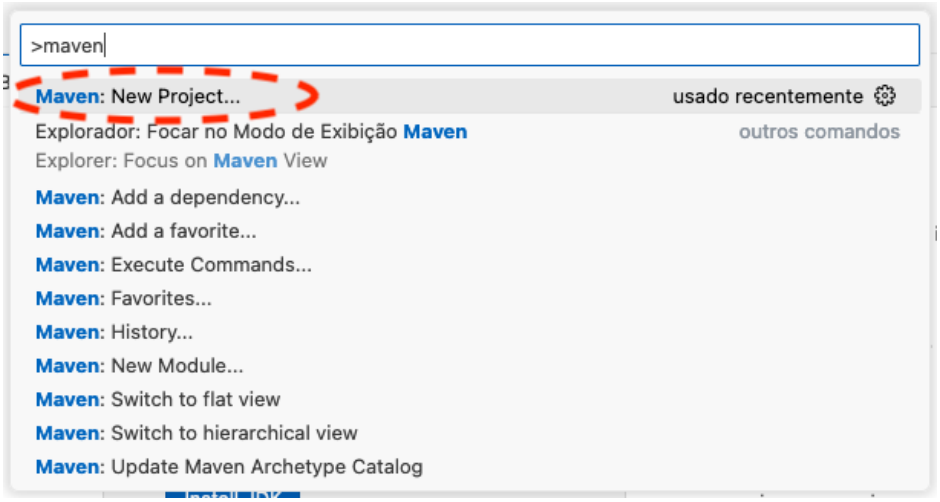
Configurando e usando o Maven

Este exercício aborda a configuração e uso do Maven.
Siga e execute os passos do roteiro.

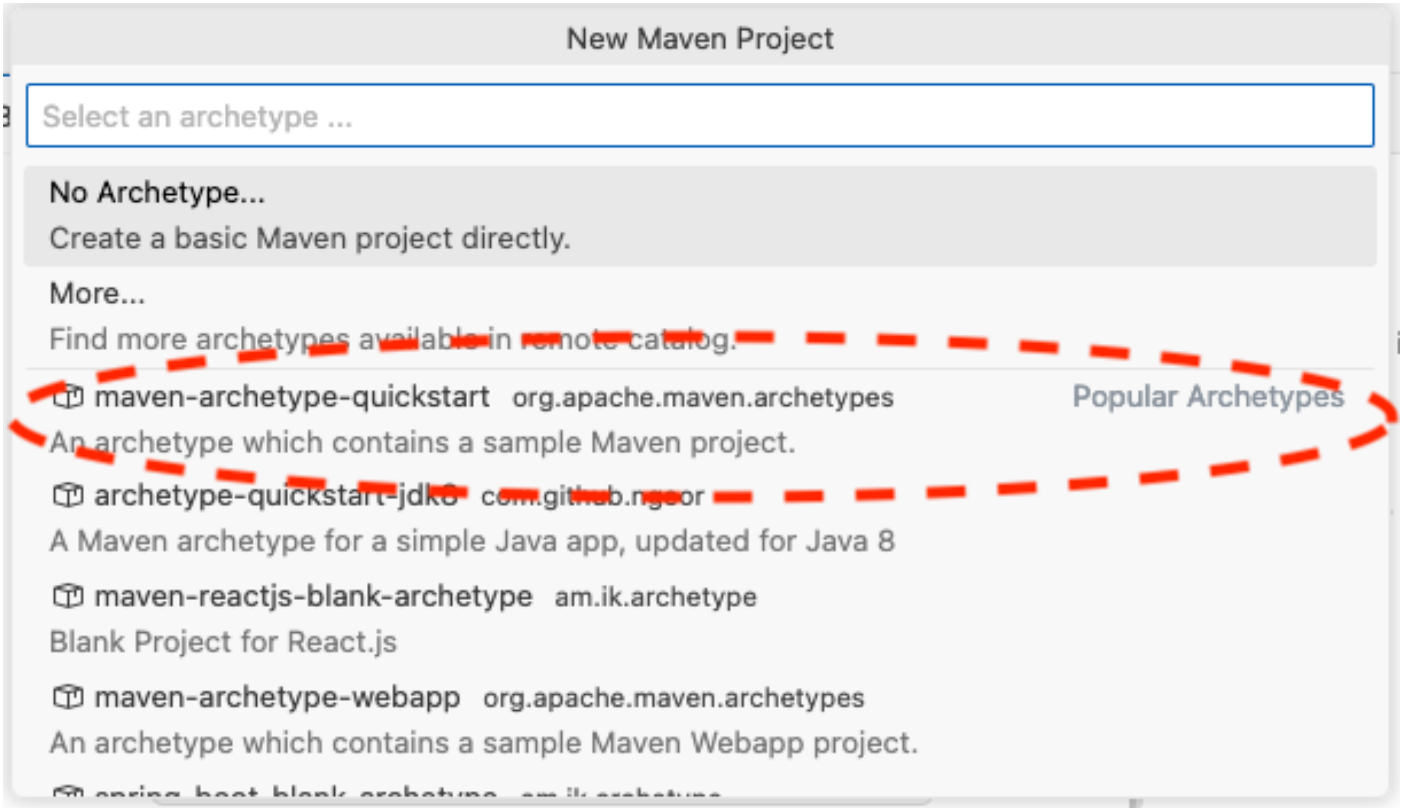
1. Crie um repositório (privado, Java) no GitHub para este exercício.
2. Inicie um Codespace a partir deste repositório.
3. No Codespace adicione o *plugin* 'Extension Pack for Java'
    a. Por *default*, o 'Maven for Java' é instalado. Entretanto, se não for instalado, procure este *plugin* e o instale.
4. Crie um projeto Maven:
    a. Inicie a paleta de comandos:



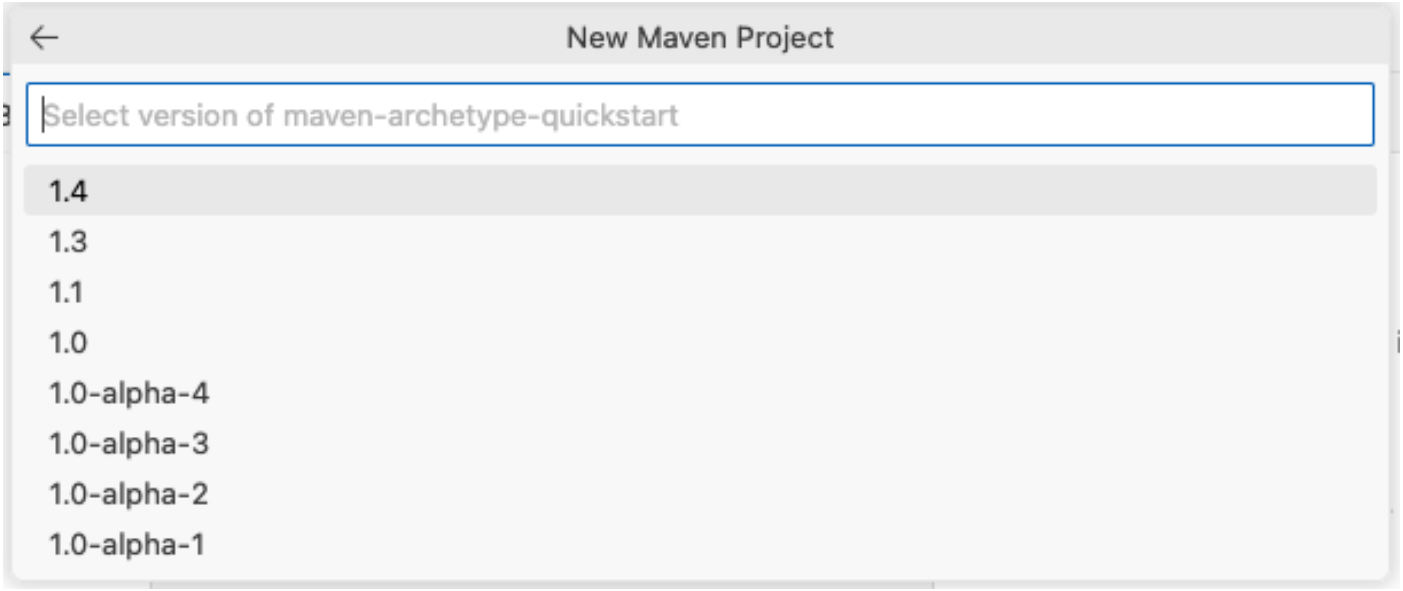
    b. Execute o comando de criar um novo projeto Maven:



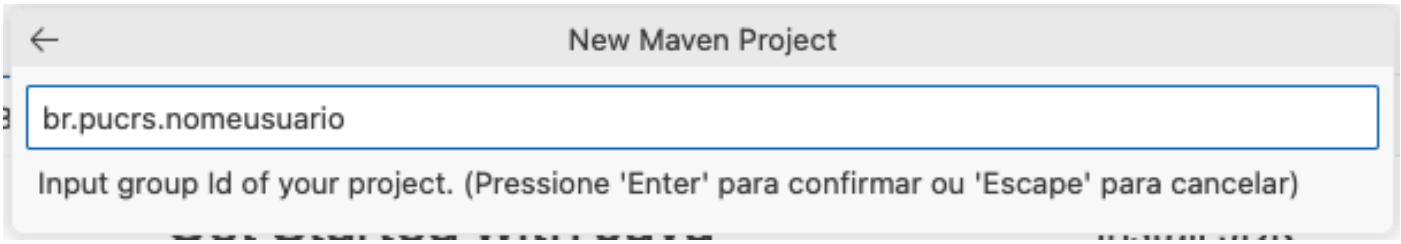
c. Selecione um arquétipo (modelo). Para este roteiro use ‘maven-archetype-quickstart’:



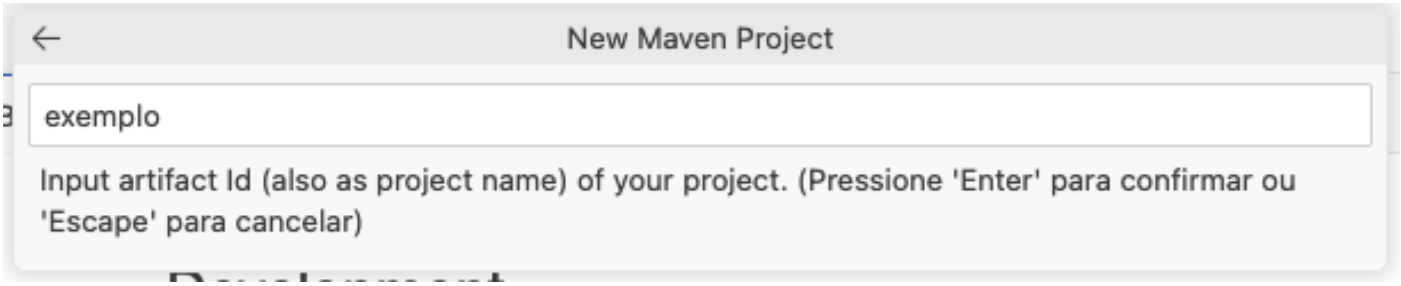
d. Selecione a versão mais recente:



e. Nomeie o identificador do grupo com um domínio. Use o seu usuário de e-mail da PUCRS invertido.



f. Nomeie o projeto:



g. Confirme a pasta do projeto e aguarde o Maven realizar as configurações iniciais.
h. No Terminal, indique a versão desejada para o projeto ou apenas confirme com ENTER.

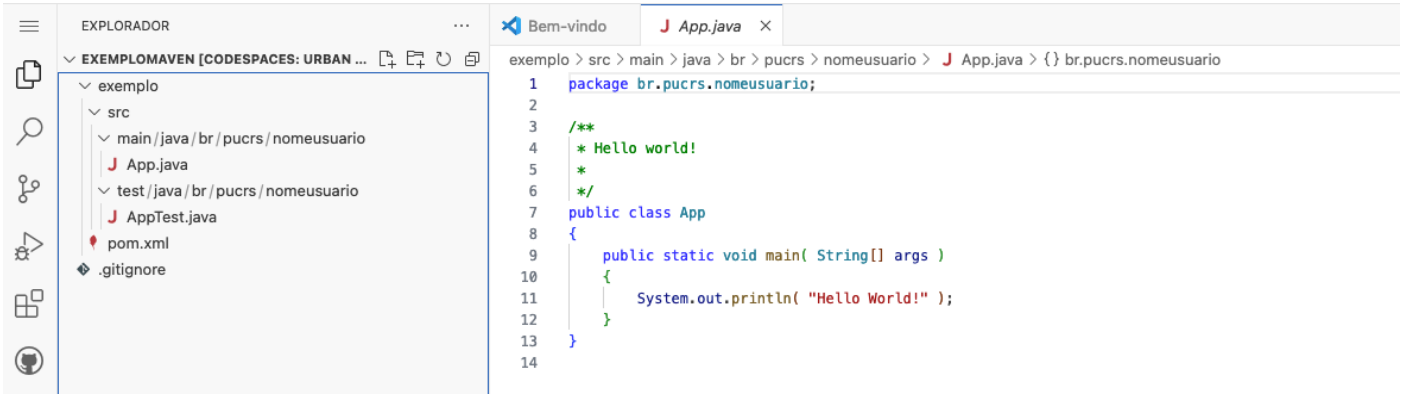
```
[INFO] Using property: groupId = br.pucrs.nomeusuario
[INFO] Using property: artifactId = exemplo
Define value for property 'version' 1.0-SNAPSHOT: :
```

i. No Terminal, revise as configurações e confirme com ENTER:

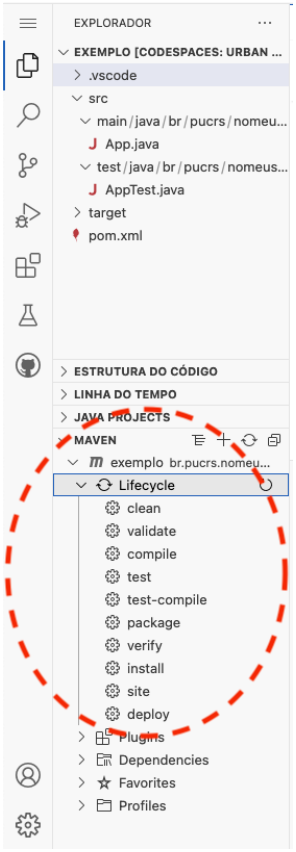
```
[INFO] Using property: package = br.pucrs.nomeusuario
Confirm properties configuration:
groupId: br.pucrs.nomeusuario
artifactId: exemplo
version: 1.0-SNAPSHOT
package: br.pucrs.nomeusuario
 Y: :
```

j. O Maven fará configurações finais e apresentará uma mensagem confirmação que o projeto foi criado.

k. Após a criação do projeto Maven explore a sua estrutura:



- Note que dentro da pasta src (códigos-fonte) há 2 pastas:
  - **main** ...: para implementar os códigos-fonte do sistema.
  - **test** ...: para implementar os testes do sistema.
- Na pasta target são gerados as compilações e montagens do código.
- Existe também o arquivo **POM.XML (Project Object Model)** que possui informações de configuração do projeto. Exemplos:
  - Dados de identificação do projeto
  - Properties: propriedades do projeto. Ex.: codificação de caracteres, versões do compilador e JVM.
  - Dependencies: dependências de outros softwares ou APIs para fazer a *build* do sistema. Ex.: API de teste unitário.
  - Build: dados o ciclo de montagem do sistema.
- O ciclo de montagem do sistema também pode ser visto em:



l. No arquivo POM.XML ajuste a versão do compilador e JVM para os disponíveis no Codespace.

```
<maven.compiler.source>21</maven.compiler.source>
<maven.compiler.target>21</maven.compiler.target>
```

m. Após este ajuste compile e execute o App.java
n. No arquivo POM.XML localize a seção de dependencies:

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

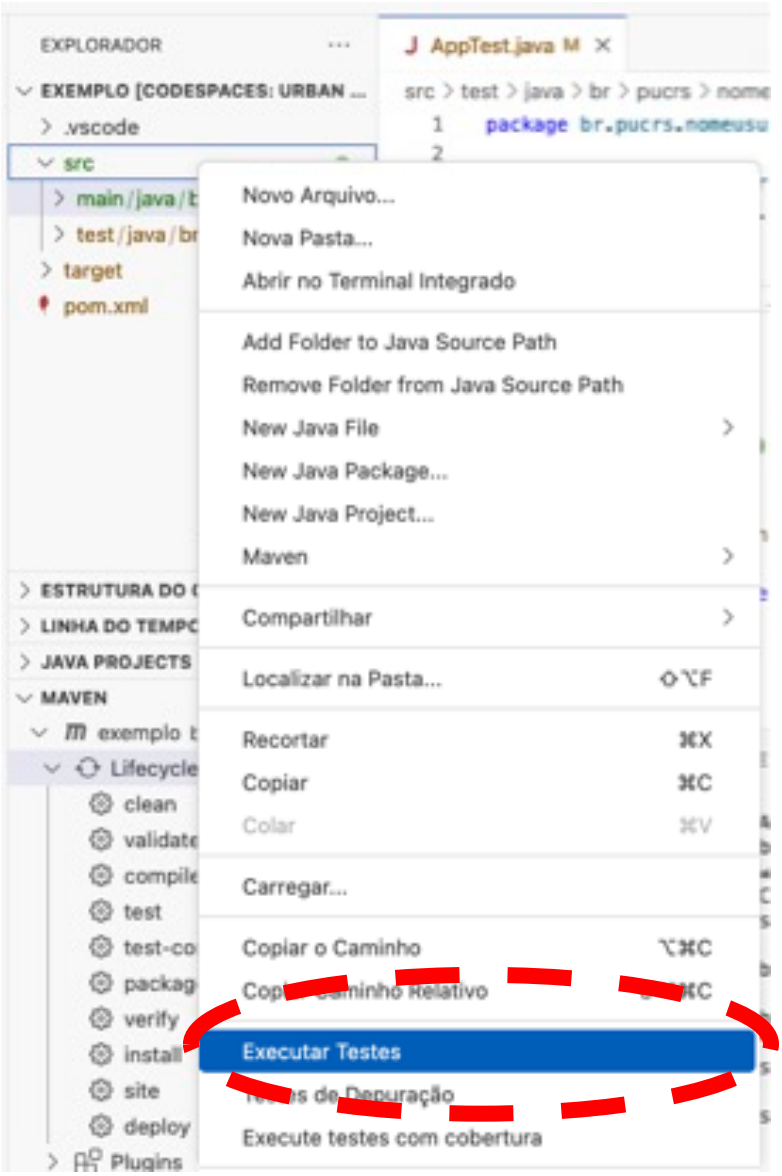
o. No arquivo POM.XML altere a seção de dependencies:

```
<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.11.4</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

p. Note que o Maven irá baixar e atualizar o projeto com as novas definições, inclusive indicando eventuais erros devido às modificações.
q. Veja que a classe AppTest.java está com indicação de erro, pois ela foi codificada para a versão 4 da API JUnit e agora o projeto está utilizando outra API de testes na versão 5.
r. Modifique os *imports* da classe AppTest.java para corrigir a indicação de erro:

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

s. Para executar testes, você pode clicar com o botão direito do *mouse* sobre a pasta 'src' e selecionar a opção Executar Testes.



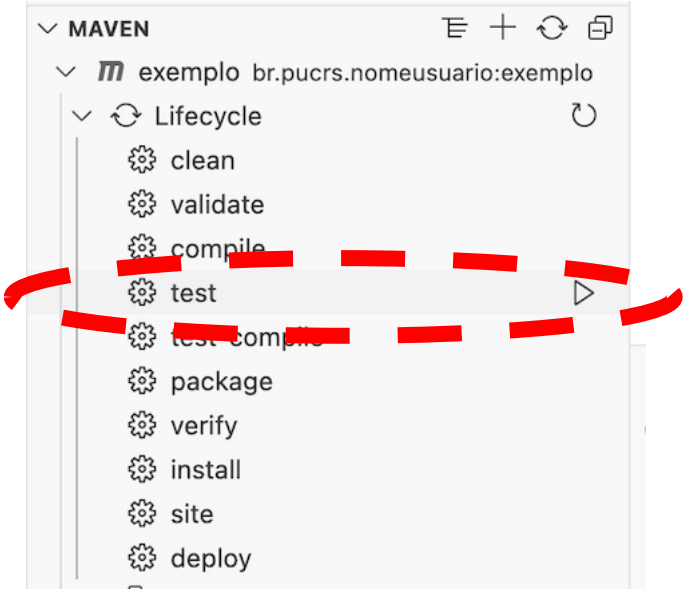
t. Note que se os testes tiveram sucesso, o relatório indicará todos os métodos de teste em verde (testes OK).
u. Na subpasta que foi criada para o projeto src/main/..., crie uma nova classe Java (com nome Conta) e coloque o código a seguir. Verifique o 'package' está correto para o seu projeto.

```
public class Conta {
   private double saldo;

   public double getSaldo() {
      return saldo;
   }

   public void depositar(double valor) {
      saldo = saldo + valor + 10;
   }

   public void sacar(double valor) {
      if(valor <= saldo)
         saldo = saldo - valor;
   }
}
```

v. Na subpasta src/test/..., crie uma nova classe Java (com nome ContaTest) e coloque o código a seguir. Verifique se o package está correto para o seu projeto.

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

public class ContaTest {
    @Test
    public void depositarTest() {
        Conta c = new Conta();  // Cria um objeto de teste Conta
        c.depositar(1000.00);  // Deposita 1000 na conta
        assertEquals(1000.00, c.getSaldo());   // Verifica se o saldo da conta é 1000.00
    }
}
```

w. Execute um novo teste.
x. Note que houve indicações de erro no teste em relação ao valor esperado e o valor retornado. É porque o método 'depositar' da classe Conta foi codificado com um erro. Corrija-o e execute novamente o teste para que o relatório indique que todos os testes passaram.
y. Você também pode executar o teste pelo ciclo de vida do Maven:



z. ***Passo opcional***: tente implementar e executar um método de teste para testar o método 'sacar' da classe Conta.

O roteiro descrito é uma introdução para o uso de Maven em um projeto Java dentro do Codespace. Entretanto, apresenta os passos para a compilação, testes e montagem automatizadas.

Lembretes:
- Como o Codespaces executa de forma virtual, lembre-se que continuamente fazer *commit* e *push* para o GitHub.
- Ao final deste roteiro, não se esqueça de parar o Codespace com o comando 'Stop Current Codespace'