Eclipse

1. File > Import... > Maven > Existing Maven Projects > Next
2. Selecione a pasta que contem o arquivo **pom.xml**
3. Finish

Para compilar, testar e empacotar:

1. Botão direito no projeto > Run As > maven build...
2. Digite package como goal
3. Run

Para Rodar os testes de dentro da IDE:

1. Instale o plugin do TestNG: https://marketplace.eclipse.org/content/testng-eclipse
2. Botão direito no projeto -- Run As -- TestNG Test

No IntelliJ IDEA

1. Import project
2. Escolha a pasta contendo o arquivo pom.xml
3. Selecione Import project from external model e escolha Maven
4. Next > Next > ... > Finish

Para Compilar, rodar os testes e empacotar: Shift Shift > maven goal > package

Para rodar os teste de dentro da IDE:

1. Vá até uma classe de testes
2. Posicione o cursor no nome da classe
3. Aperte Alt+Shift+F10

No NetBeans

Abra o projeto Maven com o menu Arquivo > Abrir Projeto... ou com o botão Abrir Projeto...

Para rodar os teste de dentro da IDE, clique no Projeto com o botão direito do mouse e selecione a opção 'Testar'.

Na Linha de Comando

Todos os exercícios da disciplina incluem um **Maven wrapper**. O maven wrapper é um script utilitário que permite compilar projetos maven sem a necessidade de ter o maven instalado no sistema. O script existe em duas versões:

- **`./mvnw`** (linux)
- **`./mvnw.cmd`** (windows)

O maven wrapper depende de um diretório oculto chamado **.mvn/** que fica junto com os scripts. Se esse diretório for perdido, os scripts irão irão apenas apresentar um erro ao invés de compilar o projeto.

Uma alternativa ao maven wrapper é usar o maven instalado no sistema. Utilize o gerenciador de pacotes (apt-get e similares) do seu sistema para isso. Virtualmente todas as distribuições possuem um pacote do maven nos seus repositórios oficiais. O comando a ser usado para o maven instalado no sistema é **`mvn`** (note a ausência do w!).

Enquanto o make recebe um target como argumento, o maven recebe um goal. Goals que serão interessantes nessa disciplina:

- **`mvn compile`** : Compila todos os arquivos .java em src/main/java
- **`mvn test-compile`** : Faz o `mvn compile` e compila todos os arquivos em src/test/java
- **`mvn verify`** : Faz o `mvn tests-compile` e roda os testes
- **`mvn package`** : Faz o `mvn verify` e gera um arquivo .jar

- Nessa disciplina, quando necessário, os pom.xml estão programados para gerar um arquivo .jar que inclui todas as dependências: não é necessário setar um classpath, apenas rode com **`java -jar target/meu-projeto-1.0-SNAPSHOT.jar`**
- Se os testes falham, mvn package não irá produzir o jar. Nesse caso pode ser útil pular os testes: **`mvn -DskipTests=true package`**

Finalmente, por conveniência e força de hábito, os Makefiles de exercícios além do alvo submission possuem alvos correspondentes aos goals do maven. Logo, make verify chama `./mvnw verify` (ou `mvn verify` se o wrapper foi perdido).

Mas o que é o Maven afinal?

O Apache Maven é um software "gerenciador de projetos de software". Ele assume três principais papeis:

1. Meta-informação do projeto: autor, repositório de fontes, licensa, etc;
2. Compilação: correspondente ao Make.O maven compila um programa produzindo artefatos (arquivos .class, .jar, ou .war). Assim como o Make, o Maven controla as depedências entre arquivos e determina modificação dos arquivos o que precisa ser re-compilado;
3. Gerenciamento de dependências: Para utiliza uma classe definida em outro projeto, como por exemplo JUnit ou Apache commons-lang, basta declarar uma dependência no arquivo pom.xml. Durante a compilação o Maven irá baixar a versão solicitada da dependência de uma forma que não existirão conflitos de versões no sistema (multiplas versões do mesmo software podem co-existir sem causar problemas em outros projetos).

A compilação no Maven segue estritamente uma convenção. Todo programa tem dois builds: main (o programa/biblioteca própriamente dito) e test (classes que testam o funcionamento do programa/biblioteca). A estrutra de um projeto Maven sempre é a seguinte:

- **`src/`** (todo o código fonte)
  - **`main/`** (código que implementa a funcionalidade do programa)
    - **`java/`** (arquivos java, numa hierarquia de pacotes)
    - **`resources/`** (arquivos embutidos no jar e acessados como resources)
  - **`test/`** (código que **testa** a funcionalidade do programa)
    - **`java/`** (arquivos java, numa hierarquia de pacotes)
    - **`resources/`** (arquivos a serem usados durante os testes como resources)
- **`target/`** (destino dos arquivos compilados e dos jars produzidos)

A definição de uma dependência e feita no pom.xml. Nessa disciplina o nunca será necessário editar o pom.xml. Se tiver curiosidade olhe a seção project/dependencies para exemplos. O registro público de projetos de software gerenciados pelo Maven pode ser consultado no https://search.maven.org/, que também tem os snippets XML com a informação de dependência.