

Projetos Java com Gradle

CST em Análise e Desenvolvimento de Sistemas

Prof. Emerson Ribeiro de Mello

mello@ifsc.edu.br

Licenciamento



Slides licenciados sob [Creative Commons "Atribuição 4.0 Internacional"](https://creativecommons.org/licenses/by/4.0/)

Sumário

- 1 Gradle
- 2 Como criar um projeto Java
 - Pela linha de comando
 - Usando as IDEs
- 3 Tarefas gradle
- 4 Executar uma aplicação Java
- 5 Incluir dependências de bibliotecas
- 6 Como empacotar aplicação Java

Gradle

Gradle, ferramenta de construção de projetos

Qual a necessidade para projetos Java?

- Para pequenos projetos de software, sejam esses escritos em C, C++ ou Java, não é necessário fazer uso de ferramentas de construção e automatização
- Para projetos complexos, com várias classes e com dependências de bibliotecas externas, é essencial usar ferramentas de automatização
 - Exemplos para Java: maven ou gradle
 - Exemplos para C++: CMake, bazel ou gradle

Algumas facilidades do gradle para projetos Java

<https://docs.gradle.org/>

- Compilação incremental somente dos arquivos alterados
- Empacotamento e distribuição da aplicação em JAR, WAR e EAR
- Execução automática de testes de unidade durante a construção
- Integração contínua com Jenkins, Travis CI e GitHub Actions



A vantagem de criar um projeto com o Gradle é que esse poderá ser aberto por qualquer IDE que tenha suporte ao gradle, ou mesmo, poderá ser compilado, executado, etc no terminal sem a necessidade de qualquer IDE

Como instalar o gradle

É necessário que tenha o JDK!

- Windows
 - Baixar instalador no site oficial¹
- Linux ou macOS
 - Via SDKman²: `sdk install gradle`



O gradle no apt-get do Linux está com uma versão defasada, opte por instalar via SDKMan

¹ <https://docs.gradle.org/current/userguide/installation.html>

² <https://sdkman.io>

IDE tem suporte ao gradle?



Jetbrains IntelliJ IDEA

- Versão *Community* (gratuita) já vem com JDK, gradle, git, etc.



Microsoft Visual Studio Code

- É necessário ter JDK, gradle instalados no sistema operacional
- Extensões necessárias
 - Extension Pack for Java

Como criar um projeto Java

Como criar um projeto Java

Pela linha de comando

Pela linha de comando: gradle init

```
mello@mbpruzh:~/tmp/primeiro-exemplo

→ primeiro-exemplo gradle init ←

Select type of project to generate:
1: basic
2: application
3: library
4: Gradle plugin
Enter selection (default: basic) [1..4] 2 ←

Select implementation language:
1: C++
2: Groovy
3: Java
4: Kotlin
5: Scala
6: Swift
Enter selection (default: Java) [1..6] 3 ←

Generate multiple subprojects for application? (default: no) [yes, no] no ←

Select build script DSL:
1: Kotlin
2: Groovy
Enter selection (default: Kotlin) [1..2] 2 ←

Select test framework:
1: JUnit 4
2: TestNG
3: Spock
4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4] 4 ←

Project name (default: primeiro-exemplo): ←

Source package (default: primeiro.exemplo): poo ←

Enter target version of Java (min. 7) (default: 21): ←

Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no] no

→ primeiro-exemplo
```

Pela linha de comando: `gradle init`

```
gradle init
```

```
Select type of project to generate:
```

```
Enter selection (default: basic) [1..4] 2
```

```
Select implementation language:
```

```
Enter selection (default: Java) [1..6] 3
```

```
Generate multiple subprojects for application? (default: no) [yes, no] no
```

```
Select build script DSL:
```

```
1: Kotlin
```

```
2: Groovy
```

```
Enter selection (default: Kotlin) [1..2] 2
```

```
Select test framework:
```

```
Enter selection (default: JUnit Jupiter) [1..4] 4
```

```
Project name (default: primeiro-exemplo): primeiro-exemplo
```

```
Source package (default: primeiro.exemplo): ads.poo
```

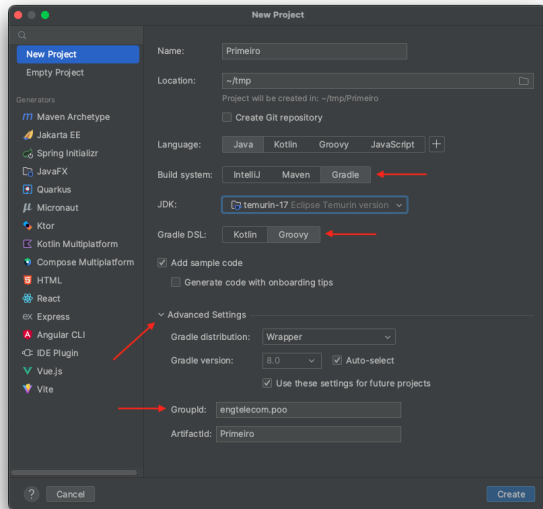
```
Enter target version of Java (min. 7) (default: 21):
```

```
Generate build using new APIs and behavior (some features may change in the next minor  
release)? (default: no) [yes, no] no
```

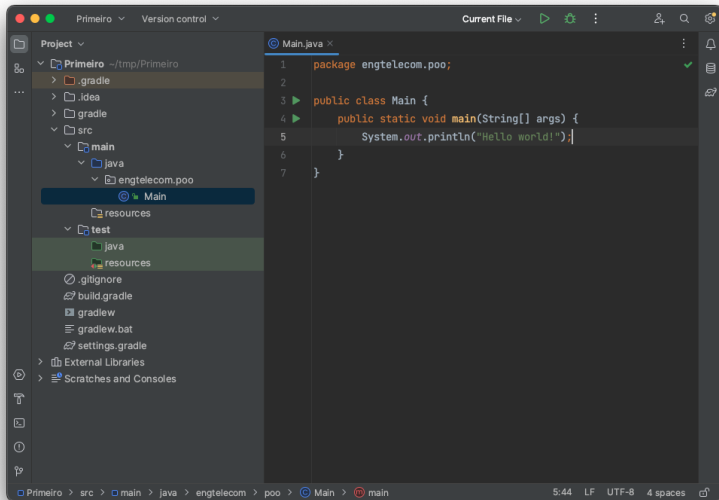
Como criar um projeto Java

Usando as IDEs

IntelliJ IDEA






IntelliJ IDEA



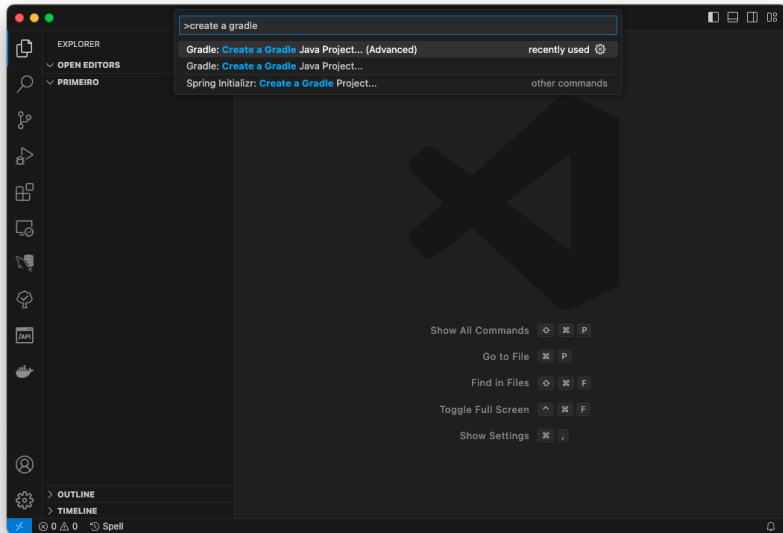
Visual Studio Code

- Crie um diretório e abra o diretório com o VSCode

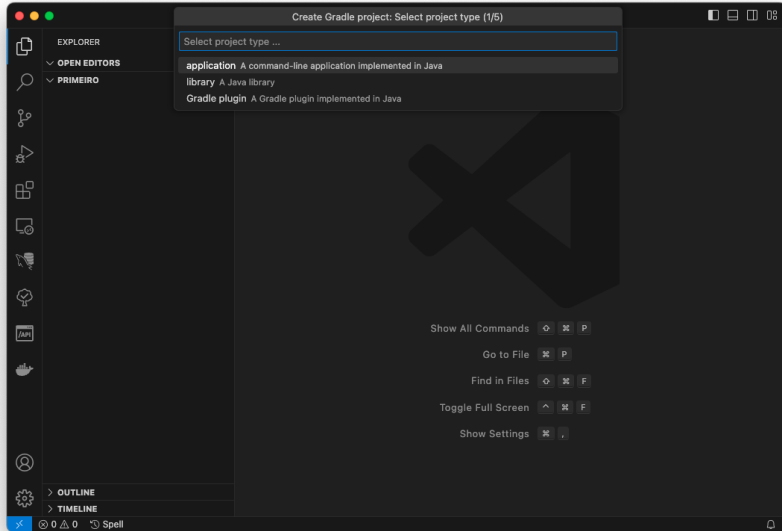
```
mkdir primeiro  
cd primeiro  
code .
```

- Abra o *prompt* de comando do VSCode (i.e.  +  + )
 - Digite: create a gradle java project (advanced)
 - Escolha: application
 - Escolha: groovy
 - Escolha: JUnit Jupiter
 - Informe o nome do projeto
 - Informe o nome do pacote Java para o projeto

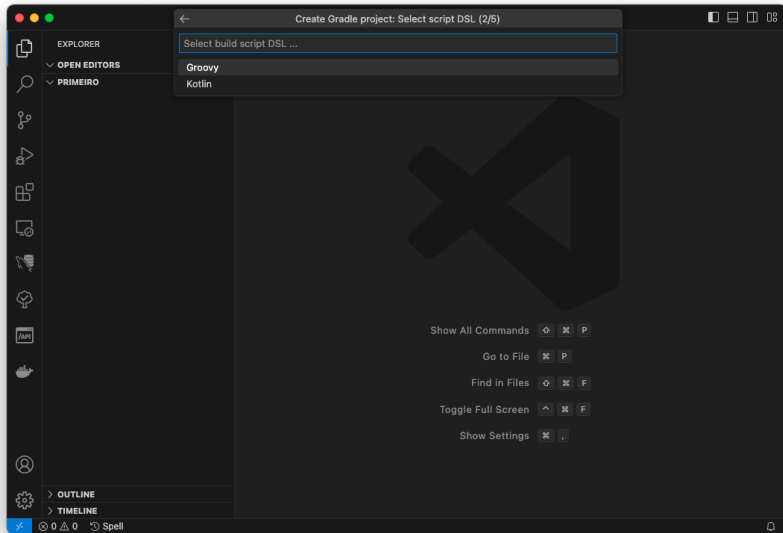
Visual Studio Code



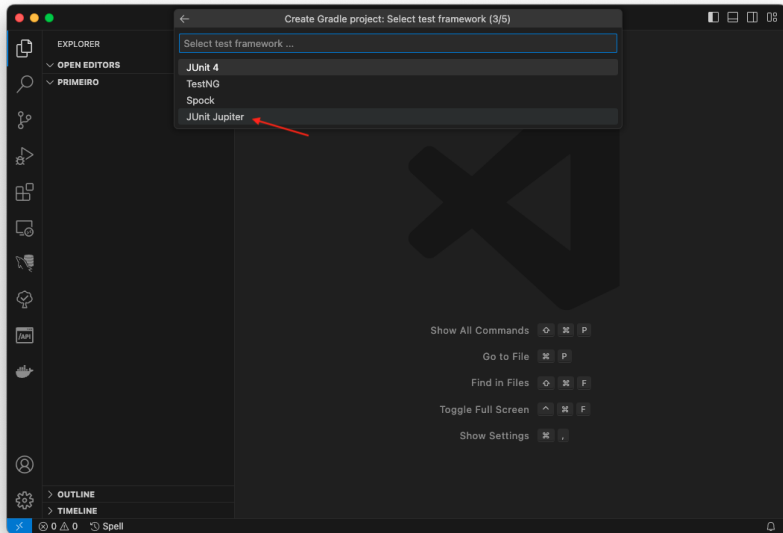
Visual Studio Code



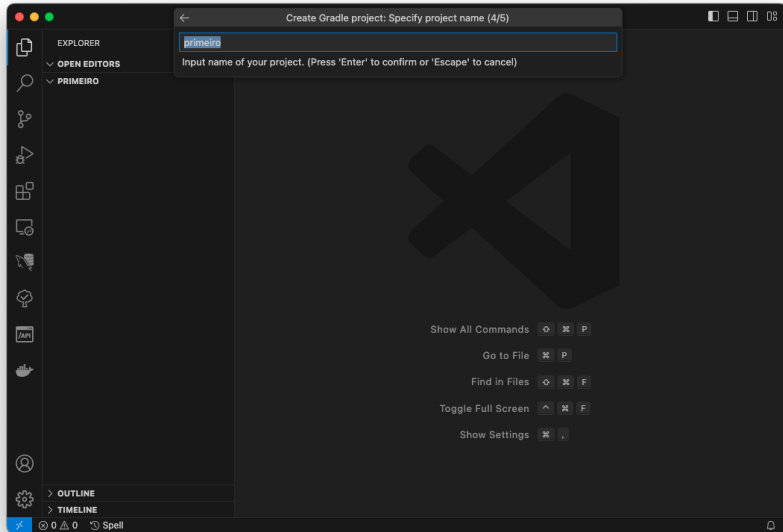
Visual Studio Code



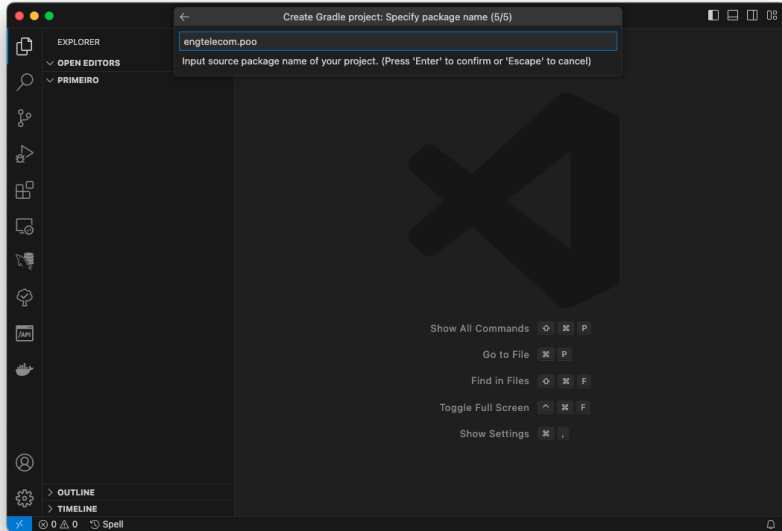
Visual Studio Code



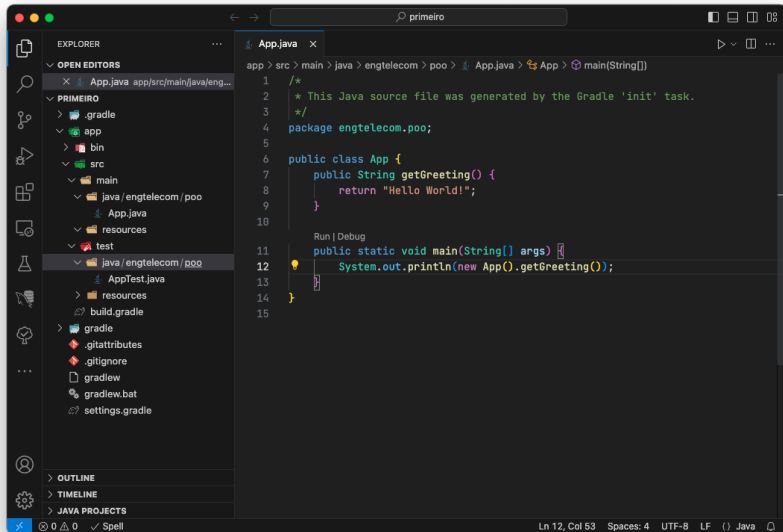
Visual Studio Code



Visual Studio Code



Visual Studio Code



Diferenças entre projetos criados com IntelliJ e gradle init

gradle init ou VSCode

```
mello@mbprush:~/tmp/primeiro
→ primeiro tree --charset=ascii
.
|-- app
|   |-- build.gradle
|   |-- src
|       |-- main
|           |-- java
|               |-- engtelecom
|                   |-- poo
|                       |-- App.java
|               |-- resources
|       |-- test
|           |-- java
|               |-- engtelecom
|                   |-- poo
|                       |-- AppTest.java
|               |-- resources
|-- gradle
|   |-- wrapper
|       |-- gradle-wrapper.jar
|       |-- gradle-wrapper.properties
|-- gradlew
|-- gradlew.bat
|-- settings.gradle
```

IntelliJ

```
mello@mbprush:~/tmp/a/primeiro
→ primeiro tree --charset=ascii
.
|-- build.gradle
|-- gradle
|   |-- wrapper
|       |-- gradle-wrapper.jar
|       |-- gradle-wrapper.properties
|-- gradlew
|-- gradlew.bat
|-- settings.gradle
|-- src
|   |-- main
|       |-- java
|           |-- engtelecom
|               |-- poo
|                   |-- Main.java
|       |-- resources
|-- test
|   |-- java
|   |-- resources
```

12 directories, 7 files

```
→ primeiro
```


Diferenças entre projetos criados com IntelliJ e gradle init

■ gradle init ou VSCode

- Cria um módulo `app` e o projeto Java fica dentro de um subdiretório de mesmo nome
- Cria um exemplo de teste de unidade (classe `AppTest.java`)
- Adiciona o plugin `application` no arquivo `build.gradle`

■ IntelliJ

- O projeto Java fica na raiz do diretório do projeto
- Adiciona o plugin `java` no arquivo `build.gradle`



O *plugin application*^a adiciona tarefas que facilitam a execução de aplicações Java pelo terminal e implicitamente já invoca o *plugin java*

^ahttps://docs.gradle.org/current/userguide/application_plugin.html

Diferenças no *build.gradle* com IntelliJ e gradle init

gradle init ou VSCode

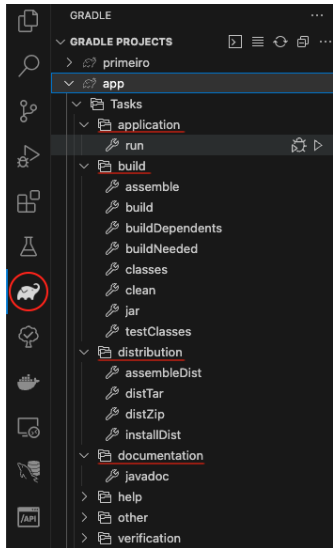
```
app > build.gradle > application
1  plugins {
2      id 'application'
3  }
4
5  repositories {
6      mavenCentral()
7  }
8
9  dependencies {
10     // Use JUnit Jupiter for testing.
11     testImplementation 'org.junit.jupiter:junit-jupiter:5.9.1'
12
13     // This dependency is used by the application.
14     implementation 'com.google.guava:guava:31.1-jre'
15 }
16
17 java {
18     toolchain {
19         languageVersion = JavaLanguageVersion.of(17)
20     }
21 }
22
23 application {
24     // Define the main class for the application.
25     mainClass = 'engtelecom.poo.App'
26 }
```

IntelliJ

```
build.gradle (primeiro) x
1  plugins {
2      id 'java'
3  }
4
5  group = 'engtelecom.poo'
6  version = '1.0-SNAPSHOT'
7
8  repositories {
9      mavenCentral()
10 }
11
12 dependencies {
13     testImplementation platform('org.junit:junit-bom:5.9.1')
14     testImplementation 'org.junit.jupiter:junit-jupiter'
15 }
16
17 test {
18     useJUnitPlatform()
19 }
```

Tarefas gradle

Tarefas gradle



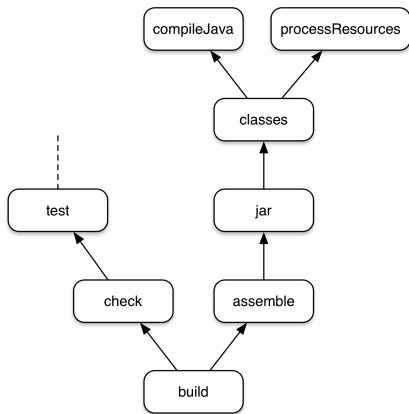
- Toda ação que o gradle pode fazer no projeto é por meio de tarefas
- As tarefas estão agrupadas (i.e build) e são usadas para compilar, empacotar, executar testes de unidade, etc
- Plugins gradle podem conter tarefas (i.e. plugin application traz a tarefa run)
- Você pode criar suas próprias tarefas^a
- Podem ser executadas pela linha de comando ou por atalhos na IDE (veja figura ao lado)

^ahttps://docs.gradle.org/current/userguide/tutorial_using_tasks.html

Ciclo de vida da tarefa build

https://docs.gradle.org/current/userguide/build_lifecycle.html

Partial task graph for a
standard Java build



- Tarefa `build` é usada para compilar um projeto, mas ela depende de outras tarefas (i.e. baixar dependências, execução dos testes de unidade)

Como executar as tarefas gradle

- Na IDE abra o painel do gradle e clique sobre a tarefa desejada
- Execute o comando `gradle`
- Execute o *script* `gradlew`³ (Linux e macOS) ou `gradlew.bat` (Windows)

Sempre opte pelo *script* `gradlew`! (a IDE usa ele)

- Não depende em ter o gradle instalado no sistema operacional
- Faz uso da versão do gradle indicada no projeto

```
./gradlew build
```

³https://docs.gradle.org/current/userguide/gradle_wrapper.html

Como atualizar a versão do gradle usada pelo projeto⁴

```
./gradlew wrapper --gradle-version latest
```

- O comando acima atualiza o *wrapper* para a última versão estável do gradle

⁴https://docs.gradle.org/current/userguide/gradle_wrapper.html#sec:upgrading_wrapper

Executar uma aplicação Java

Faremos uso do plugin application

- Garanta que o arquivo build.gradle contenha as linhas abaixo
 - *plugin application* irá fornecer a tarefa run
 - Seção *run* permite redirecionar entrada para a aplicação

```
plugins{  
    // Indicar o plugin e incluir a seção application  
    id 'application'  
}  
  
application{  
    // informar o nome do pacote e classe Java que tem o método main  
    mainClass = 'ads.poo.App'  
}  
  
// Para permitir o redirecionamento de entrada para sua aplicação  
run {  
    standardInput = System.in  
}
```

Projeto gradle com Kotlin como DSL

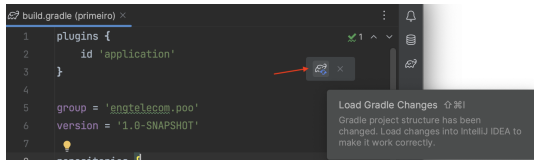
- Ao invés de usar Groovy como DSL, você pode usar Kotlin, assim o arquivo build.gradle terá a extensão .kts, por exemplo build.gradle.kts

```
plugins {  
    application  
}  
dependencies {  
    testImplementation(libs.junit.jupiter)  
    testRuntimeOnly("org.junit.platform:junit-platform-launcher")  
    implementation(libs.guava)  
}  
  
application {  
    mainClass = "ads.poo.App"  
}  
  
// Para permitir o redirecionamento de entrada para sua aplicação.  
// Atenção: O `in` está dentro de crases  
tasks.run.configure {  
    standardInput = System.`in`  
}
```

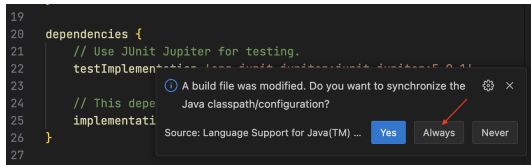
Alterações no arquivo build.gradle



Sempre que alterar o arquivo build.gradle será necessário que a IDE o releia para conhecer as alterações



IntelliJ



VSCode

Tarefa run fornecida pelo plugin application

```
# Executará a classe informada na seção application no build.gradle
./gradlew -q run
```

```
● → primeiro git:(master) × ./gradlew run
```

```
> Task :app:run
```

```
Hello World!
```

```
BUILD SUCCESSFUL in 450ms
```

```
2 actionable tasks: 2 executed
```

```
○ → primeiro git:(master) × █
```

Como minimizar as mensagens geradas pelo gradle?

- O gradle imprime algumas informações sobre o processo ao executar a aplicação
- Caso queira omitir essas informações, você pode:
 - Executar a tarefa `run` com a opção `-q` (quiet)
 - Adicionar as linhas abaixo dentro do arquivo chamado `gradle.properties` que pode estar dentro do diretório do projeto ou em `$HOME/.gradle`

```
org.gradle.console=plain  
org.gradle.logging.level=quiet
```

```
● → primeiro git:(master) × ./gradlew run  
Hello World!  
○ → primeiro git:(master) × █
```

Uso da tarefa `run`

Argumentos de linha de comando e redirecionamento de entrada

- Fornecendo argumentos de linha de comando para a aplicação Java

```
./gradlew -q run --args "Argumento1 Argumento2 Argumento3"
```

- Fazendo redirecionamento de entrada para a aplicação Java

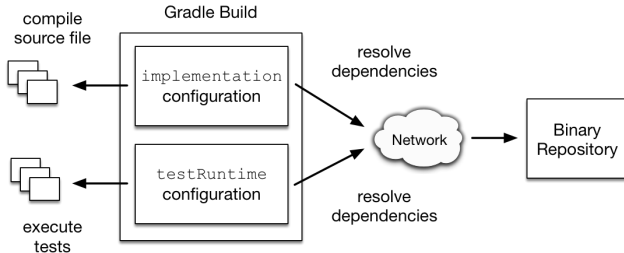
```
./gradlew -q run < arquivo.txt
```

Incluir dependências de bibliotecas

Fazendo uso de bibliotecas de terceiros

As bibliotecas Java são distribuídas em arquivos .jar

- Dependências de bibliotecas devem ser incluídas na seção dependencies do arquivo build.gradle
- Podem ser arquivos .jar presentes no diretório do projeto
- Podem ser obtidas de repositórios online (<https://mvnrepository.com>)



Fazendo uso de bibliotecas de terceiros

Inclusões no arquivo build.gradle

```
dependencies {  
    // importando arquivo JAR que está dentro do diretório libs  
    // https://github.com/poo29004/java-qr-code-barcode-jar/blob/master/app/libs/barcode.jar  
    implementation files('libs/barcode.jar')  
  
    // importando biblioteca que será baixada da internet  
    implementation 'com.google.zxing:core:3.5.3'  
    implementation 'com.google.zxing:javase:3.5.3'  
}
```

⚠ Atenção

No arquivo .gitignore é necessário garantir que os arquivos .jar contidos no diretório libs não sejam ignorados. Isso pode ser feito adicionando a linha: `!**/libs/*.jar`

Fazendo uso de bibliotecas de terceiros

Inclusões no arquivo `build.gradle.kts`

```
dependencies {  
    // importando arquivo JAR que está dentro do diretório libs  
    implementation(files("libs/barcode.jar"))  
  
    // importando biblioteca que será baixada da internet  
    implementation("com.google.zxing:core:3.5.3")  
    implementation("com.google.zxing:javase:3.5.3")  
}
```

Fazendo uso de bibliotecas de terceiros

Exemplo de uso da biblioteca `barcode.jar` e `zxing`

```
import barcode.CodigoDeBarra;
import java.nio.file.Path;
import java.nio.file.Paths;
import com.google.zxing.BarcodeFormat;
import com.google.zxing.qrcode.QRCodeWriter;
import com.google.zxing.common.BitMatrix;
import com.google.zxing.client.j2se.MatrixToImageWriter;

public class App {
    public static void main(String[] args) throws Exception {
        // Uso da classe CodigoDeBarra presente na biblioteca barcode.jar
        CodigoDeBarra.gerarCodigoDeBarra(123456, "saida.png");

        // Uso da classe BarcodeWriter presente na biblioteca zxing
        Path filePath = Paths.get("qrcode.png");
        QRCodeWriter qrCodeWriter = new QRCodeWriter();
        BitMatrix bitMatrix = qrCodeWriter.encode(valor, BarcodeFormat.QR_CODE, 300, 300);
        MatrixToImageWriter.writeToPath(bitMatrix, "PNG", filePath);
    }
}
```

Como empacotar aplicação Java

Como empacotar aplicação Java

- Uma aplicação Java compilada geralmente é entregue em um pacote `.jar`
- Para executar a aplicação é necessário ter o ambiente de execução Java (JRE) – o JDK já contém o JRE
- O plugin `application` provê duas tarefas e gera scripts (shell e bat) para facilitar a execução da aplicação
 - `distZip` – Cria um `app.zip` no subdiretório `build/distributions` o qual conterá scripts e JRE para executar a aplicação
 - `installDist` – Gera scripts em `app/build/install/app` para permitir executar a aplicação

Tarefas installDist e distZip do plugin application

```
# Para gerar scripts para executar sua aplicação
# Será criado em app/build/install/app
./gradlew installDist

# Execute a aplicação com o shell script ou arquivo .bat
# disponível no subdiretório bin
./app/build/install/app/bin/app

# Para empacotar a aplicação para ser distribuída
# Será gerado um app.zip no subdiretório build/distributions
./gradlew distZip
```

Arquivo JAR

- Java ARchive (JAR) é um **arquivo compactado com um conjunto de classes** e outros recursos (imagens, áudio, etc) e usado para distribuir aplicativos ou bibliotecas Java
- **Para executar um aplicativo** no formato JAR e que possua um arquivo manifesto^a indicando qual classe tem o método `main`

```
java -jar aplicativo.jar
```

- Indicando a classe (quando não tem manifesto)

```
java -cp:aplicativo.jar ads.poo.App
```



^a<https://docs.oracle.com/javase/tutorial/deployment/jar/manifestindex.html>

Para projetos que não possuem dependências de bibliotecas externas

- Incluir seção no arquivo build.gradle

```
jar {  
    manifest {  
        // Classe principal da aplicação  
        attributes "Main-Class": "ads.poo.App"  
    }  
}
```

- Execute a tarefa `./gradlew jar` para gerar o arquivo JAR no subdiretório `build/libs`
 - O arquivo gerado pode ser distribuído. Trata-se da sua aplicação empacotada em um JAR
- Execute a aplicação com o comando: `java -jar app.jar`

Quando o projeto possui dependências externas

- O JAR gerado no slide anterior não contém os JARs das bibliotecas informadas na seção dependencies do build.gradle e assim não será possível executar sua aplicação
- O plugin gradle *shadowJar*⁵ permite gerar um JAR contendo todas as bibliotecas de terceiros
- Alterações no arquivo build.gradle

```
plugins {  
    id 'com.github.johnrengelman.shadow' version '8.3.0'  
    id 'application'  
}  
application{  
    // informar o nome do pacote e classe Java que tem o método main  
    mainClass = 'ads.poo.Principal'  
}
```

⁵<https://imperceptiblethoughts.com/shadow/>

Gerando, executando e distribuindo com *shadowJar*

```
# Gerando o FAT JAR contendo todas as bibliotecas de terceiros
./gradlew shadowJar

# Executando o JAR gerado em build/libs/nome-do-projeto-all.jar
java -jar build/libs/nome-do-projeto-all.jar

# Gerando o ZIP contendo a aplicação. Ficará em build/distributions
./gradlew shadowDistZip
```

Referências

- <https://github.com/poo29004/projeto-java-gradle>
- <https://github.com/poo29004/biblioteca-de-terceiros>
- <https://github.com/johnrengelman/shadow>
- https://docs.gradle.org/current/userguide/application_plugin.html