Working with IntelliJ IDE

Student Workbook 1c – Integrated Development Environments

Version 7.0 Y

Table of Contents

Module 1

IntelliJ Basics

Section 1–1

The IntelliJ IDE

Understanding IntelliJ IDE

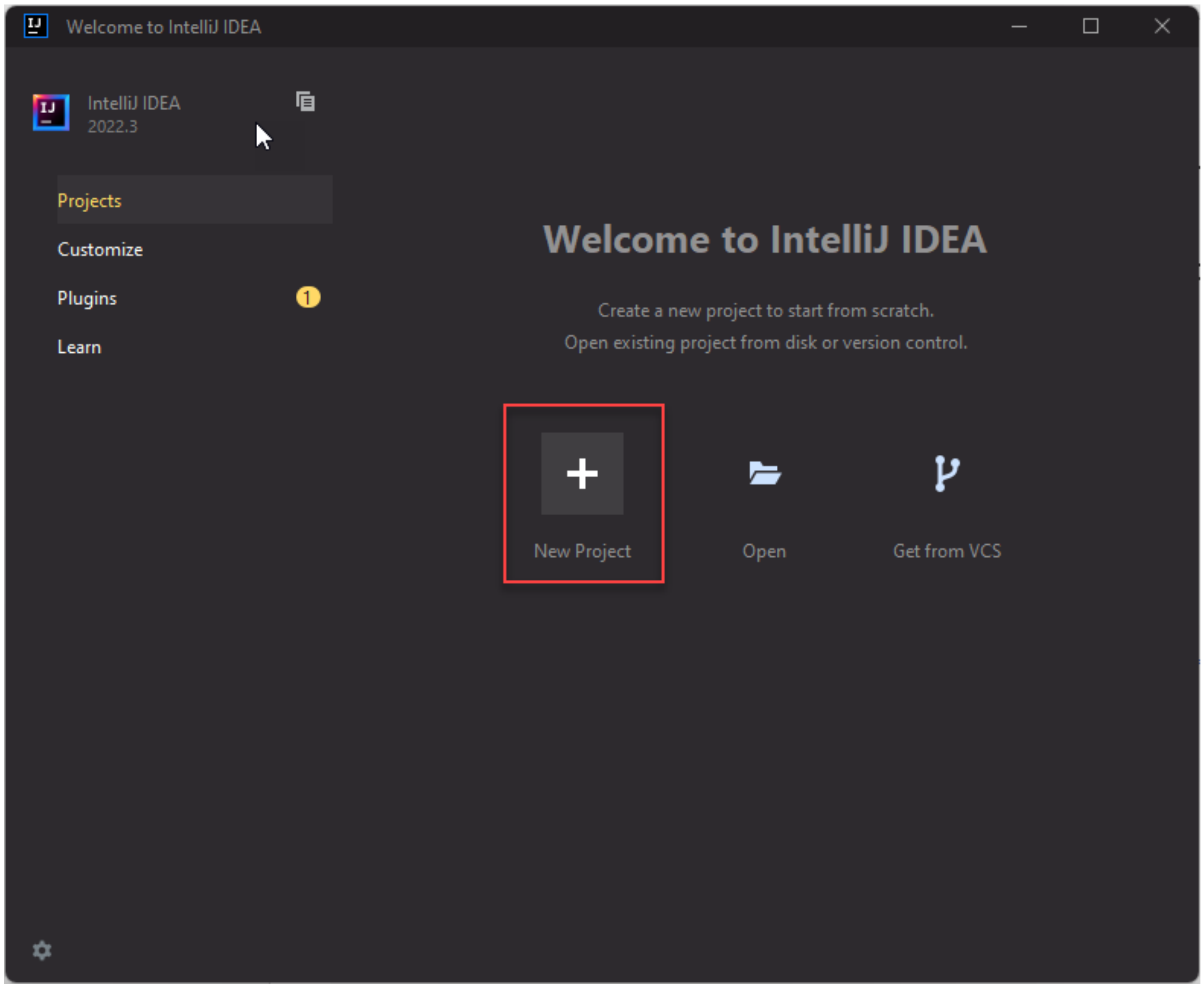
- **Integrated Development Environment - A program that contains comprehensive utilities to develop software**
- **The leading IDE for Java and Kotlin development**
  - Run, Test and Debug Java projects
  - Integrated with Git
- **Plugins available to customize and extend your environment**
- **A product of JetBrains**
  - Free Community Edition or paid Ultimate Edition

Java Projects

- **A Java Project is just a folder that contains all of the project files**
- **Java source code**
- **IntelliJ manages all types of Java projects and build tools**
  - There are a few different project build management tools
    - Maven – currently the most popular build manager for java
    - Gradle
    - Ant
- **Multiple ways to create projects**
  - Create the project directly from IntelliJ
    - IntelliJ will build the appropriate folder structure and starter files
  - Import a project from a VCS such as Git
  - Create a project manually, then open it in IntelliJ
    - Projects can also be created with tools like Maven

Creating a New Java Project

- Open IntelliJ and select Projects -> New Project

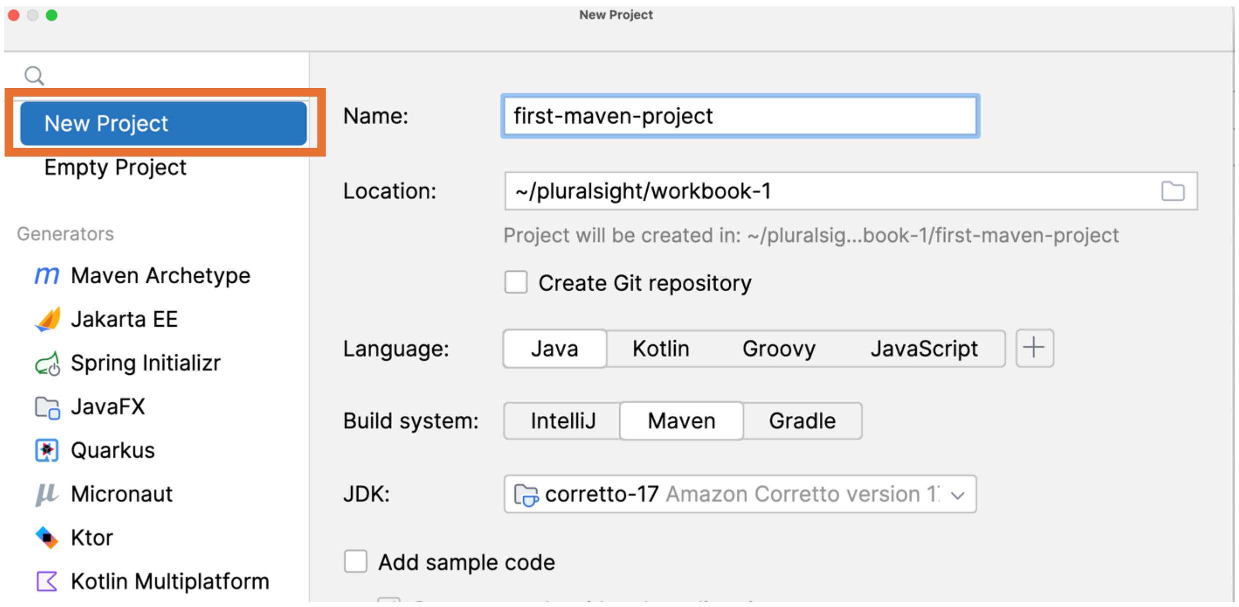


Welcome to IntelliJ IDEA

Create a new project to start from scratch.
Open existing project from disk or version control.

The New Project Dialog

- **The New Project dialog lets you choose from various ways to create and initialize your project**



Name: first-maven-project

Location: ~/pluralsight/workbook-1

Project will be created in: ~/pluralsig...book-1/first-maven-project

☐ Create Git repository

Language: [✓ Java] Kotlin Groovy JavaScript +

Build system: IntelliJ [✓ Maven] Gradle

JDK: corretto-17 Amazon Corretto version 1

☐ Add sample code

  - New Project creates a basic project with some minimal starter code

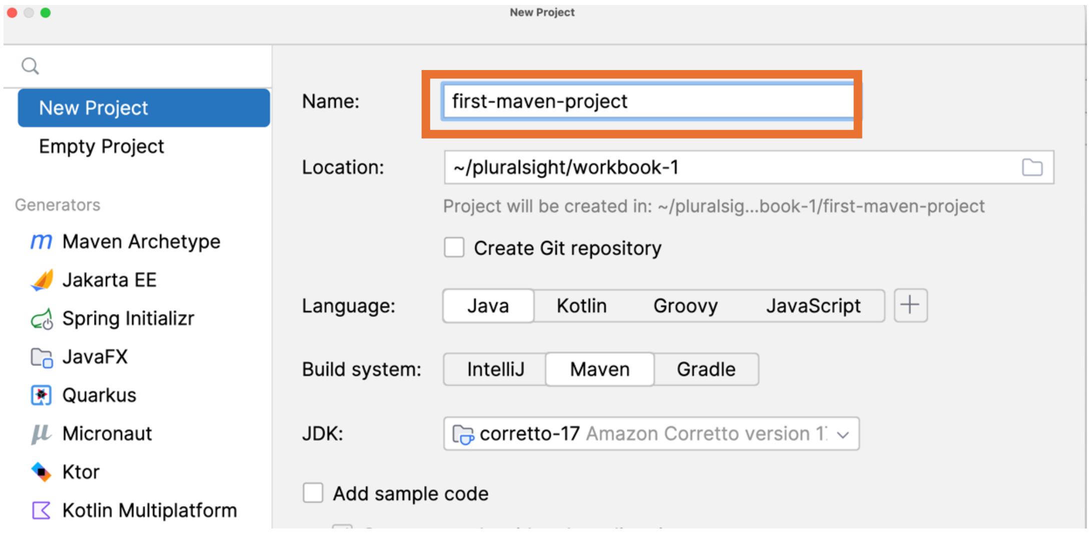
  - Empty Project creates the project folder only; all other configurations must be added manually later

  - Generators are specialized project templates

    * These allow you to create projects with significant pre-generated boilerplate (or starter) code

Project Name and Location

- Enter a Project Name

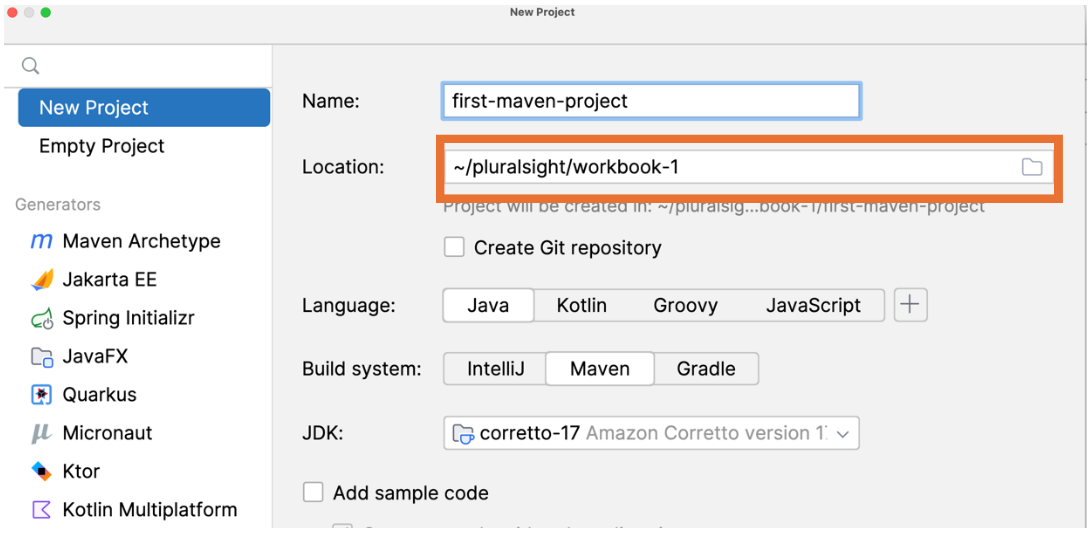

New Project

New Project
Empty Project

Generators
Maven Archetype
Jakarta EE
Spring Initializr
JavaFX
Quarkus
Micronaut
Ktor
Kotlin Multiplatform

Name: first-maven-project
Location: ~/pluralsight/workbook-1
Project will be created in: ~/pluralsig...book-1/first-maven-project
Create Git repository
Language: Java | Kotlin | Groovy | JavaScript
Build system: IntelliJ | Maven | Gradle
JDK: corretto-17 Amazon Corretto version 1
Add sample code

- Select the directory/folder where the project will be saved


New Project

New Project
Empty Project

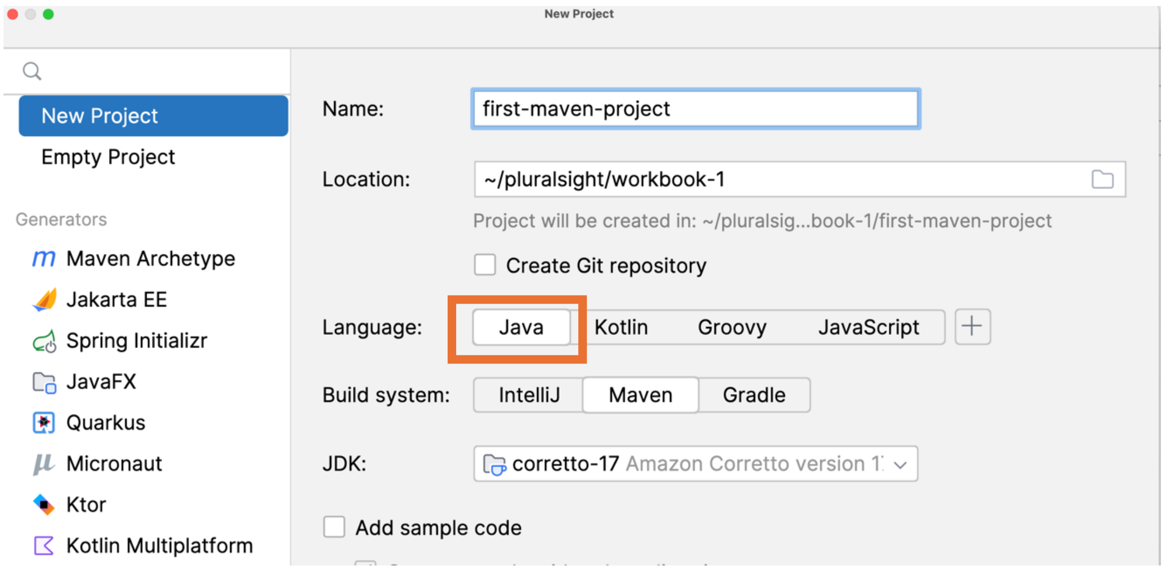
Generators
Maven Archetype
Jakarta EE
Spring Initializr
JavaFX
Quarkus
Micronaut
Ktor
Kotlin Multiplatform

Name: first-maven-project
Location: ~/pluralsight/workbook-1
Project will be created in: ~/pluralsig...book-1/first-maven-project
Create Git repository
Language: Java | Kotlin | Groovy | JavaScript
Build system: IntelliJ | Maven | Gradle
JDK: corretto-17 Amazon Corretto version 1
Add sample code

- You can choose whether to create a local Git repository to hold the project code

Project Language and Build System

- Select the project language



New Project

New Project
Empty Project

Generators
Maven Archetype
Jakarta EE
Spring Initializr
JavaFX
Quarkus
Micronaut
Ktor
Kotlin Multiplatform

Name: first-maven-project

Location: ~/pluralsight/workbook-1

Project will be created in: ~/pluralsig...book-1/first-maven-project

Create Git repository

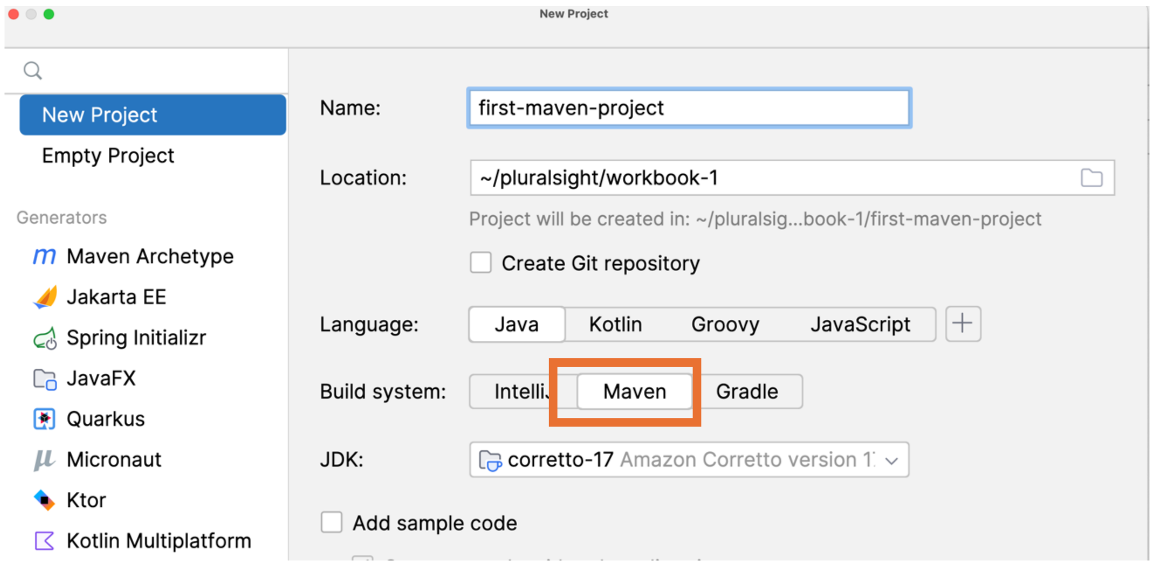
Language: Java | Kotlin | Groovy | JavaScript | +

Build system: IntelliJ | Maven | Gradle

JDK: corretto-17 Amazon Corretto version 1

Add sample code

- Select the project build system



New Project

New Project
Empty Project

Generators
Maven Archetype
Jakarta EE
Spring Initializr
JavaFX
Quarkus
Micronaut
Ktor
Kotlin Multiplatform

Name: first-maven-project

Location: ~/pluralsight/workbook-1

Project will be created in: ~/pluralsig...book-1/first-maven-project

Create Git repository

Language: Java | Kotlin | Groovy | JavaScript | +

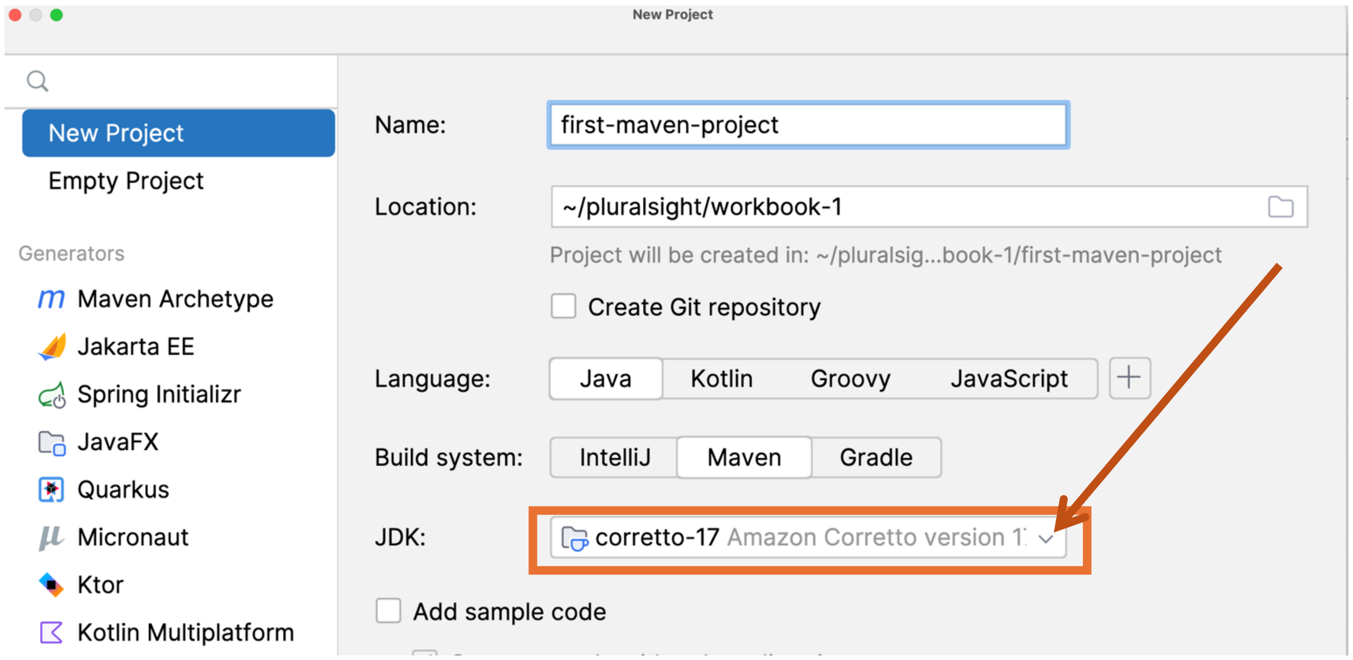
Build system: IntelliJ | Maven | Gradle

JDK: corretto-17 Amazon Corretto version 1

Add sample code

Select the Java Version

- **We will be using Java 17 during this course**
  - If you do not have the Java 17 JDK, IntelliJ will give you options to download and install it directly


New Project

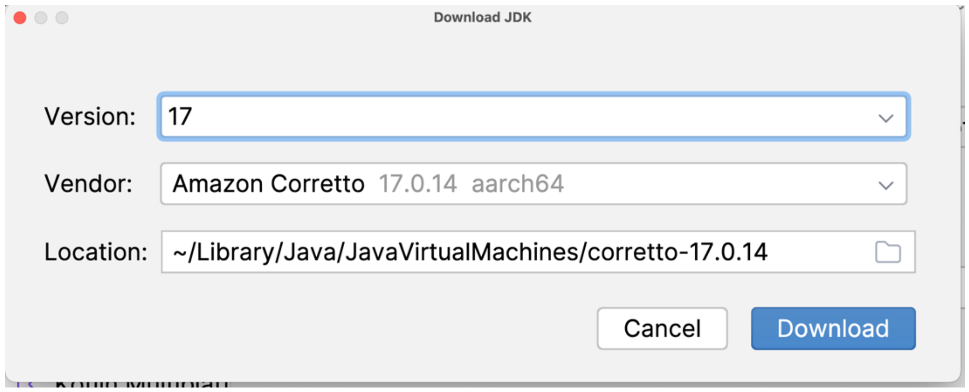
New Project
Empty Project

Generators
Maven Archetype
Jakarta EE
Spring Initializr
JavaFX
Quarkus
Micronaut
Ktor
Kotlin Multiplatform

Name: first-maven-project
Location: ~/pluralsight/workbook-1
Project will be created in: ~/pluralsig...book-1/first-maven-project
Create Git repository
Language: Java | Kotlin | Groovy | JavaScript
Build system: IntelliJ | Maven | Gradle
JDK: corretto-17 Amazon Corretto version 1
Add sample code

Installing JDK 17 (If Necessary)

- Select and download the Java 17 JDK



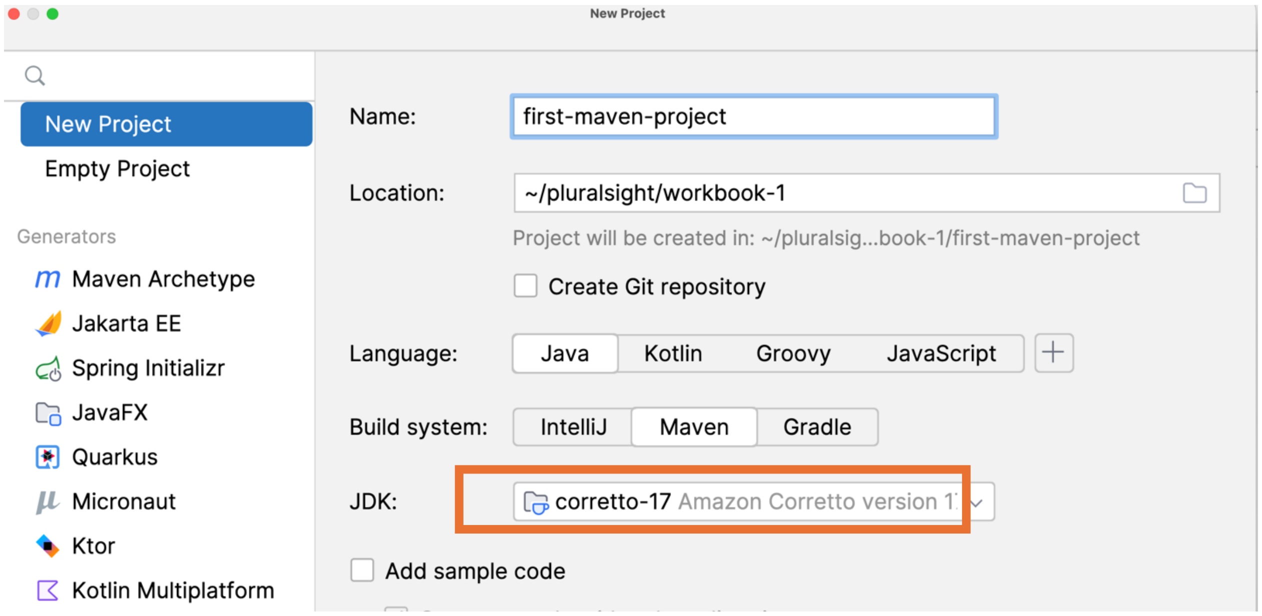
Download JDK

Version: 17

Vendor: Amazon Corretto 17.0.14 aarch64

Location: ~/Library/Java/JavaVirtualMachines/corretto-17.0.14

- Ensure that JDK 17 is selected



New Project

Name: first-maven-project

Location: ~/pluralsight/workbook-1

Project will be created in: ~/pluralsig...book-1/first-maven-project

Create Git repository

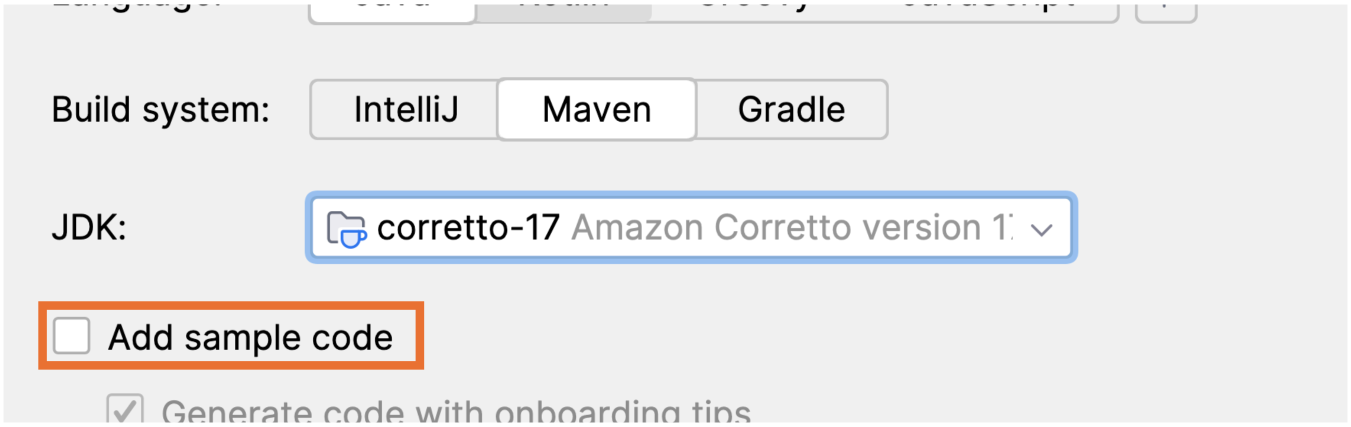
Language: Java

Build system: Maven

JDK: corretto-17 Amazon Corretto version 1

Add sample code

- Uncheck **Add sample code**



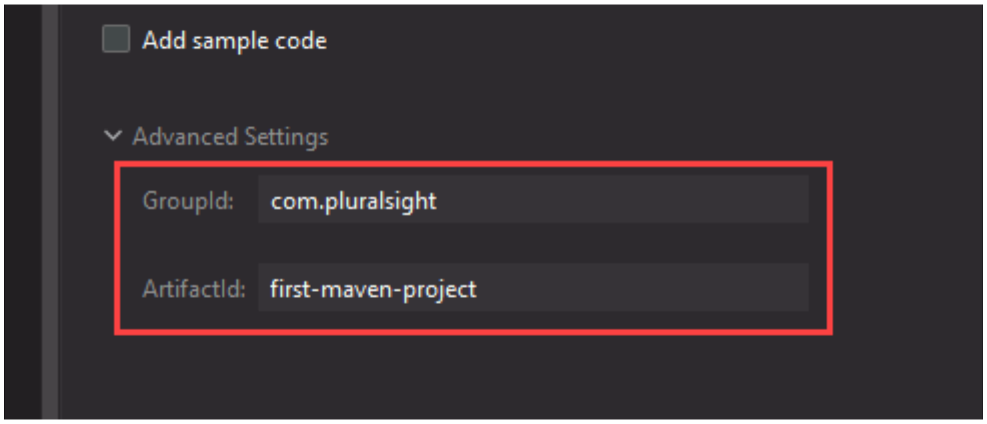
Build system: IntelliJ Maven Gradle

JDK: corretto-17 Amazon Corretto version 1

☐ Add sample code

☑ Generate code with onboarding tips

- Expand the Advanced Settings tab and update the **GroupId** and **ArtifactId**



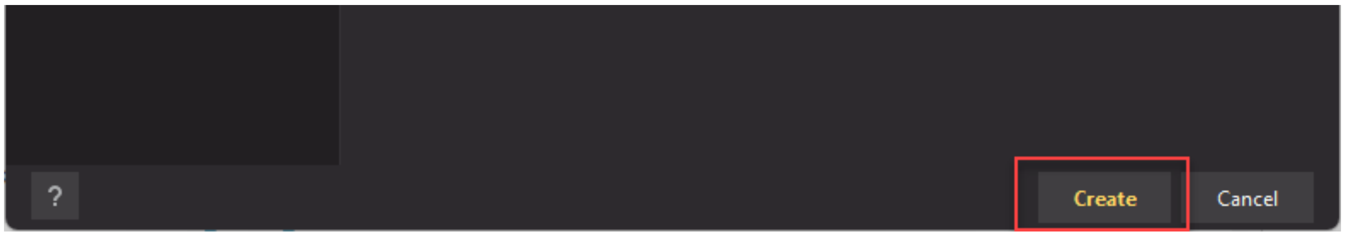
☐ Add sample code

Advanced Settings

GroupId: com.pluralsight

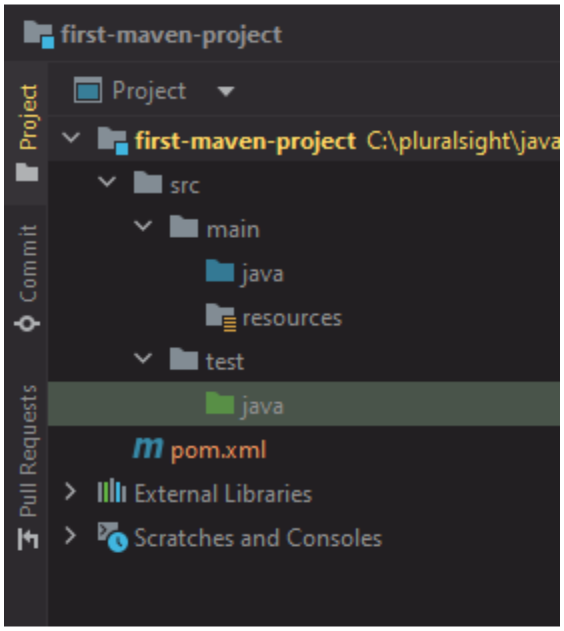
ArtifactId: first-maven-project

- Click Create



Explore the Project

- **After creating the project IntelliJ will open the project folder**

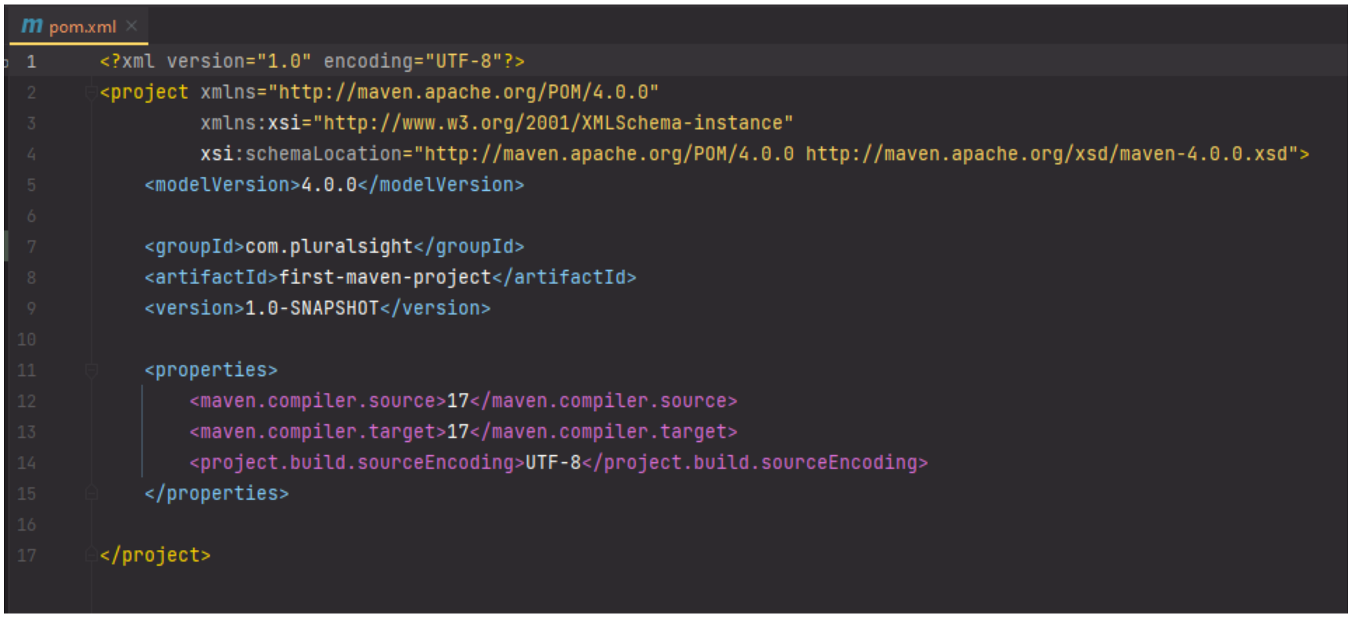


- **This folder structure is the standard structure for Maven Projects**

```
First-maven-project/
    ├── pom.xml
    └── src/
        └── main/
            └── java/
        └── test/
            └── java/
```

The `pom.xml` file

- **The `pom.xml` file is a Maven file that is used to define**
  - project configurations (name, version, jdk build version, etc.)
  - a list of external project dependencies



pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.pluralsight</groupId>
    <artifactId>first-maven-project</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

</project>
```

- **We will learn more about this file later in the cohort**

Maven project folder structure

- **All application code must be added to the** `src/main/java` **directory**

- **Unit tests are added to the** `src/test/java` **directory**
  - You will learn more about unit tests later in the cohort

Adding a package

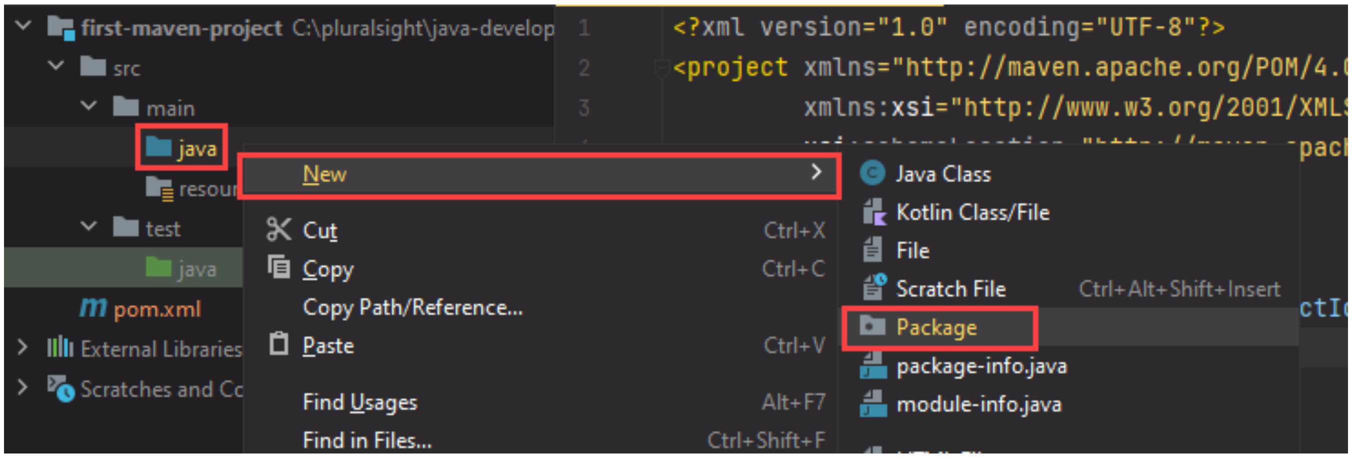
- **Packages appear as folders in a Java project**
  - They allow us to organize our code
  - Packages names are all lower case and follow the following convention

    `com.companyname.projectname`

- **Each dot in the package name implies a subdirectory in the java source tree**

- **Packages are added relative to the** `src/main/java` **folder**
  - All Java projects *should* have at least one package
    - i.e. we should not add a Java file directly into the `java` folder

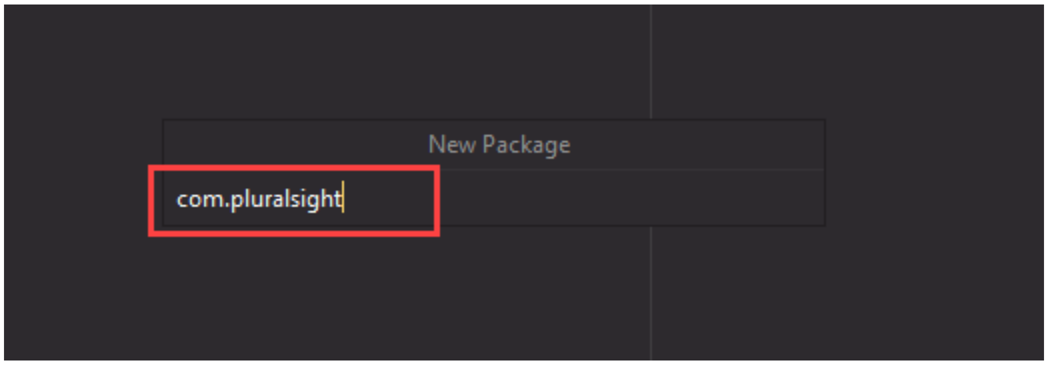
- **Create a package by right-clicking on the** **`main/java`** **folder and select New -> Package**



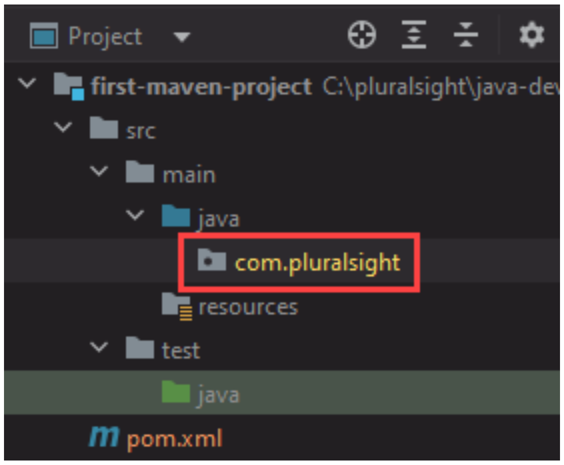
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.
         xmlns:xsi="http://www.w3.org/2001/XMLS
```

Set the package name

- Add a package name in the **New Package** window and hit **Enter**



- A new **Package** will have been created for you

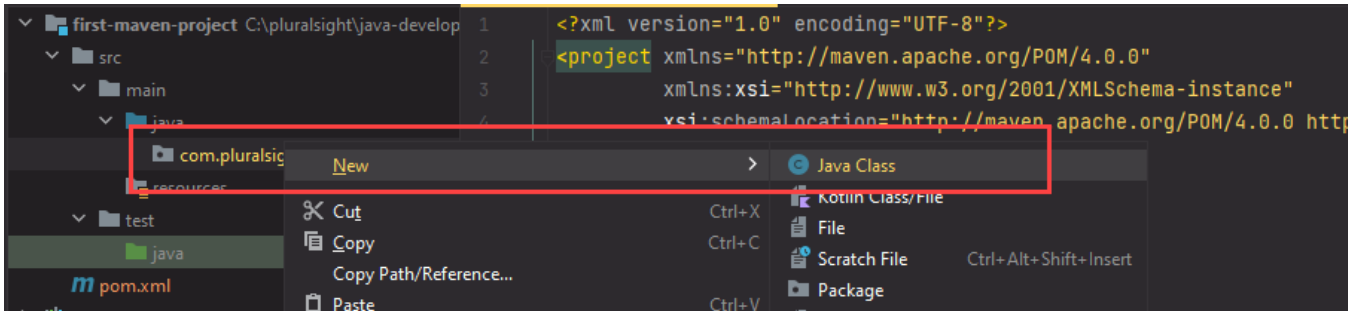


```
Project
first-maven-project C:\pluralsight\java-de
  src
    main
      java
        com.pluralsight
      resources
    test
      java
  pom.xml
```

Creating a class

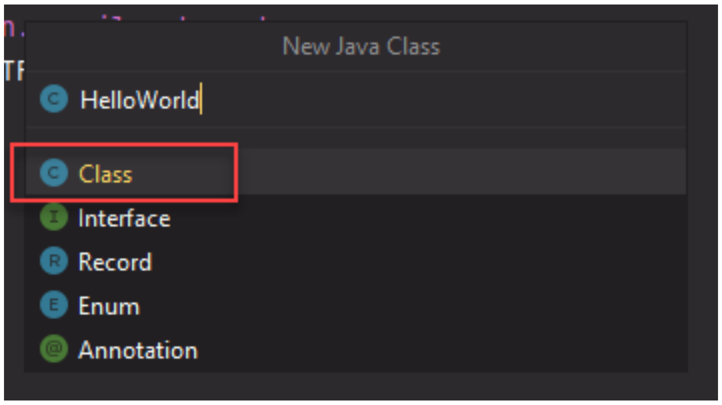
- Now that you have created the package, you can create a class in the package.

- **Right-click** on the package you just create in the **Project Explorer** on the left hand window. Select **New -> Java Class**



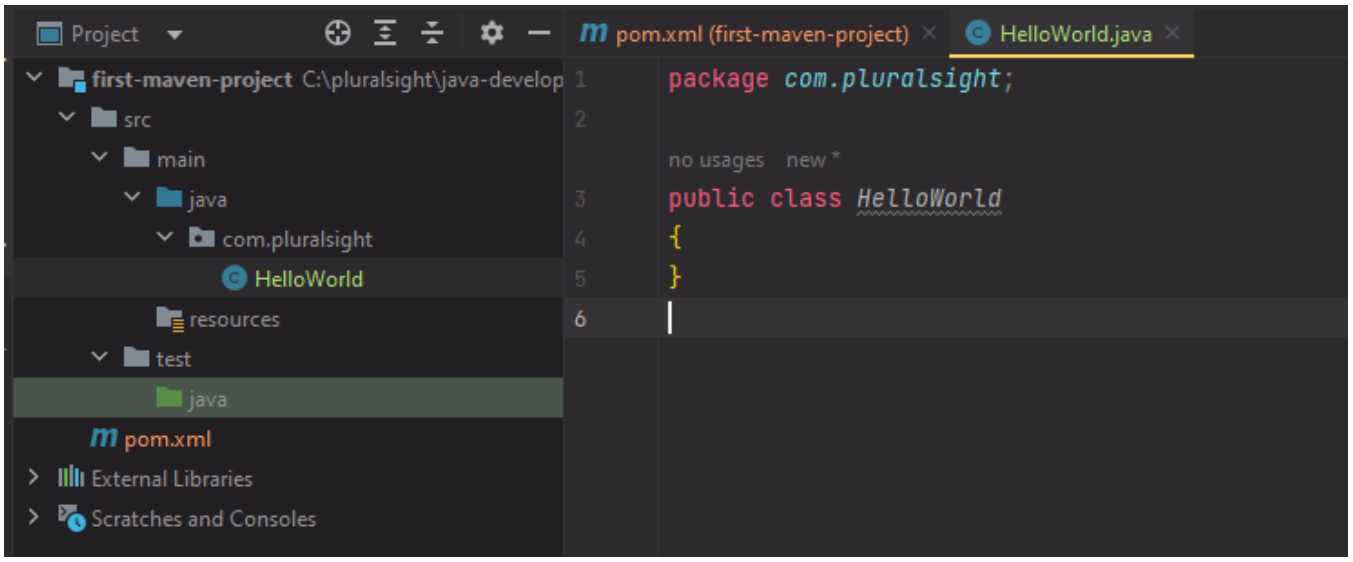
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http
```

- In the **New Java Class** window enter the name of your class and hit **Enter**
  - Class names always start with an uppercase letter



The Java source file

- Your new class will be created and open in IntelliJ



```java
package com.pluralsight;

public class HelloWorld
{
}

```

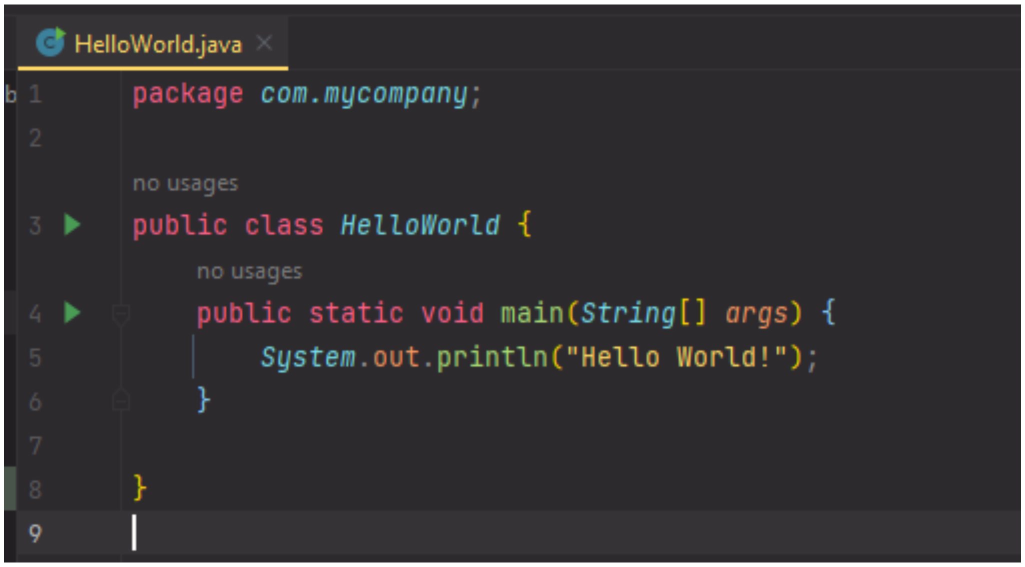
Finishing the Application

- A Java application must have an **Entry Point** into the application in order to run it

- The **Entry Point** is a function named main, defined like this

```
public static void main(String[] args) {
   // your code goes here
}
```

- Finish the HelloWorld project



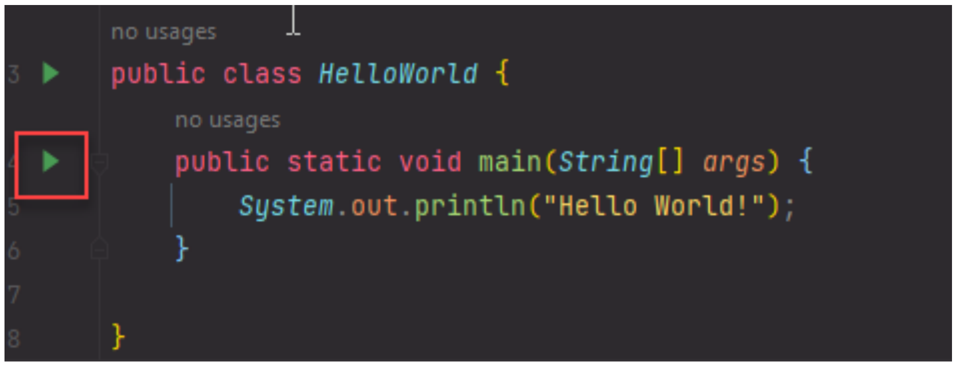
HelloWorld.java

```
package com.mycompany;

no usages
public class HelloWorld {
    no usages
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }

}
```

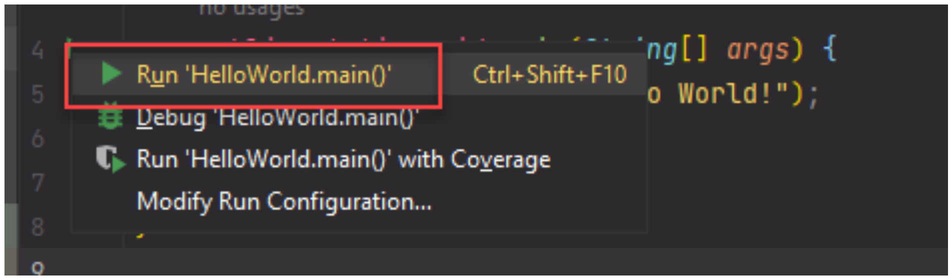
Running your application

- Once you have added the `static void main` function, a green arrow appears next to that function

- At any time, you can click on the green arrow, to run your application

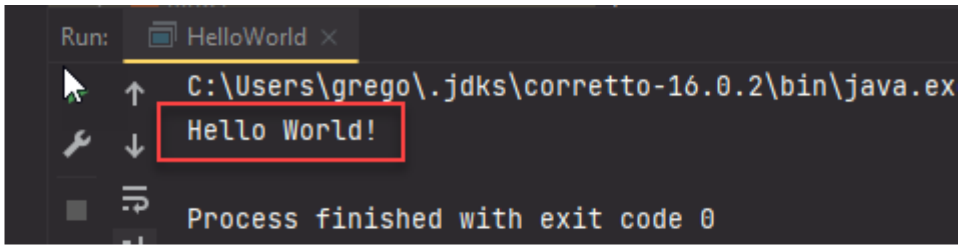


```
no usages
public class HelloWorld {
    no usages
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }

}
```



Run 'HelloWorld.main()'
Ctrl+Shift+F10
Debug 'HelloWorld.main()'
Run 'HelloWorld.main()' with Coverage
Modify Run Configuration...

- This should immediately print the results at the bottom of the window

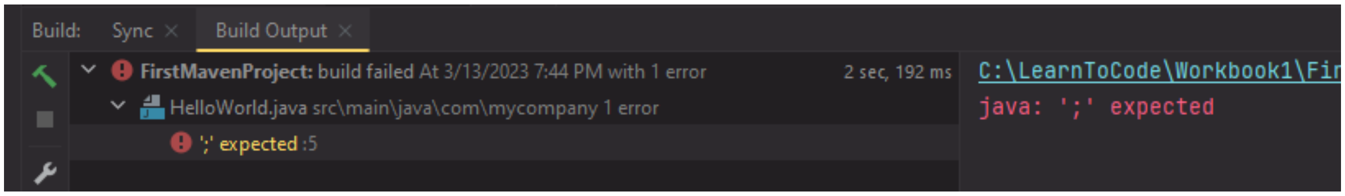


```
C:\Users\grego\.jdks\corretto-16.0.2\bin\java.ex
Hello World!

Process finished with exit code 0
```

Detecting and Fixing Errors

- **If your application does not run, it is most likely because of an error in your code**
  - IntelliJ can help find errors quickly

- **When you attempt to run the application, you may get a compile error message**

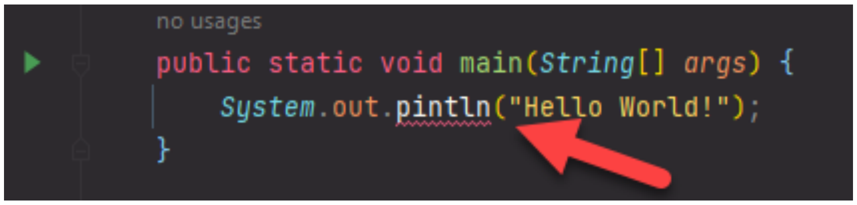


```
FirstMavenProject: build failed At 3/13/2023 7:44 PM with 1 error    2 sec, 192 ms
  HelloWorld.java src\main\java\com\mycompany 1 error
    ';' expected :5
```

```
C:\LearnToCode\Workbook1\Fir
java: ';' expected
```

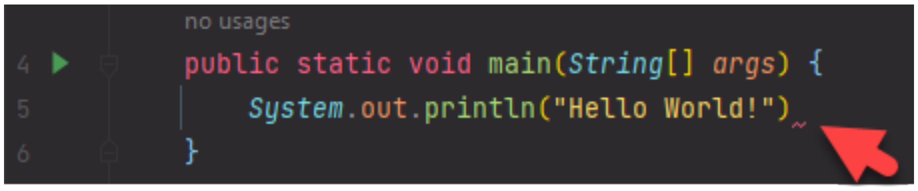
Common Errors

- **Misspelled functions or variables**
  - Missing “r” in `println()`



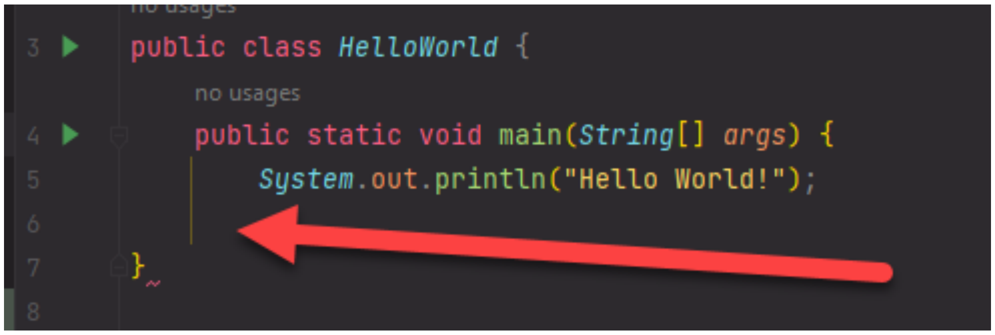
```
no usages
public static void main(String[] args) {
    System.out.pintln("Hello World!");
}
```

- **Missing semi-colon at the end of a line**



```
no usages
public static void main(String[] args) {
    System.out.println("Hello World!")
}
```

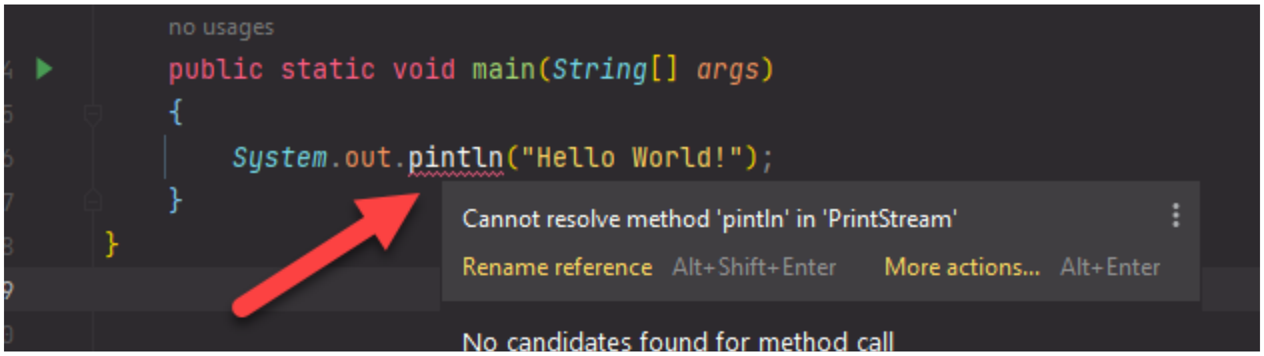
- **Missing close curly brace**
  - Here the compiler believes that the function has a close curly, but the class is missing one



```
no usages
public class HelloWorld {
    no usages
    public static void main(String[] args) {
        System.out.println("Hello World!");

}
```

Lean on Your Tools

- DO read the error messages; they really want to help!
- You can also hover over the **red squiggly** line to get information about the error



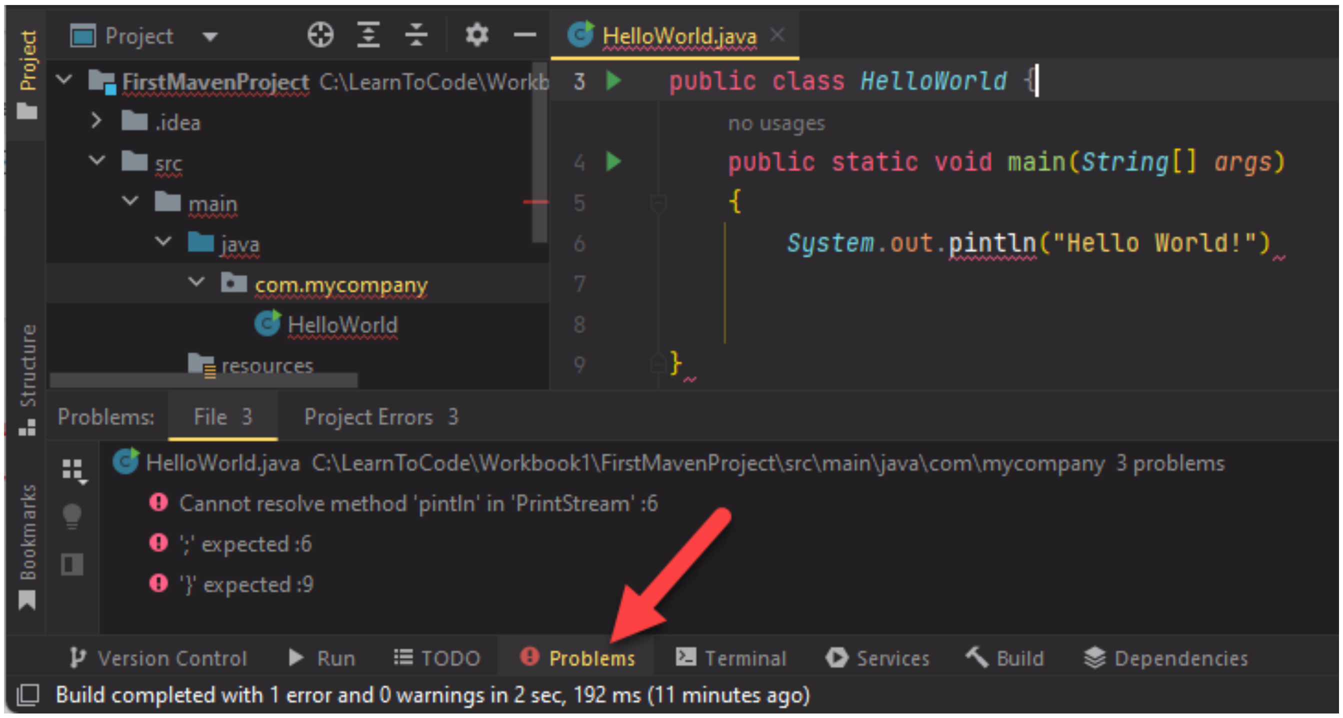
```
no usages
public static void main(String[] args)
{
    System.out.pintln("Hello World!");
}
}
```

Cannot resolve method 'pintln' in 'PrintStream'

Rename reference Alt+Shift+Enter More actions... Alt+Enter

No candidates found for method call

- Additionally, there is a **Problems** tab at the bottom of IntelliJ that will list all potential problems with your code



```
public class HelloWorld {
    no usages
    public static void main(String[] args)
    {
        System.out.pintln("Hello World!")
}
```

Problems: File 3 Project Errors 3

HelloWorld.java C:\LearnToCode\Workbook1\FirstMavenProject\src\main\java\com\mycompany 3 problems

Cannot resolve method 'pintln' in 'PrintStream' :6

';' expected :6

'}' expected :9

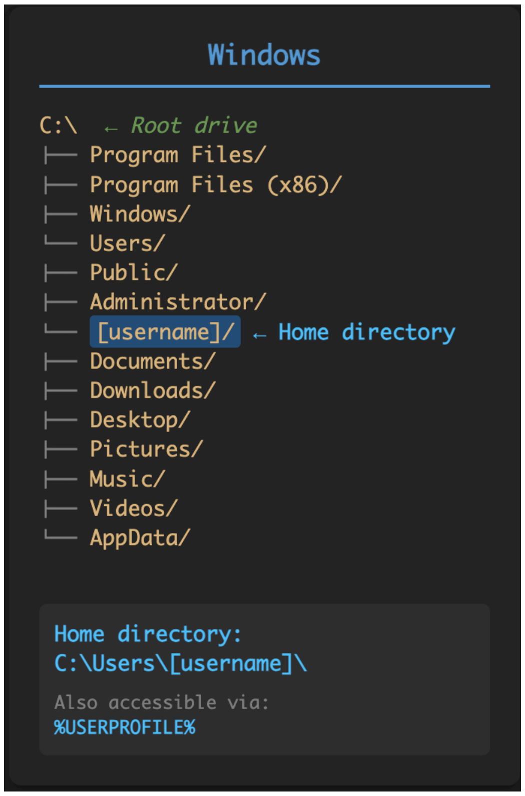
Build completed with 1 error and 0 warnings in 2 sec, 192 ms (11 minutes ago)

Exercises – Shopping List

Complete the following exercise by adding the new project into the workbook-1 folder

If you don't have a workbook-1 folder, create a new folder within the pluralsight directory. The path should look like:

On Mac/Linux: `~/pluralsight/workbook-1`
On Windows: `C:/Users/[username]/pluralsight/workbook-1`

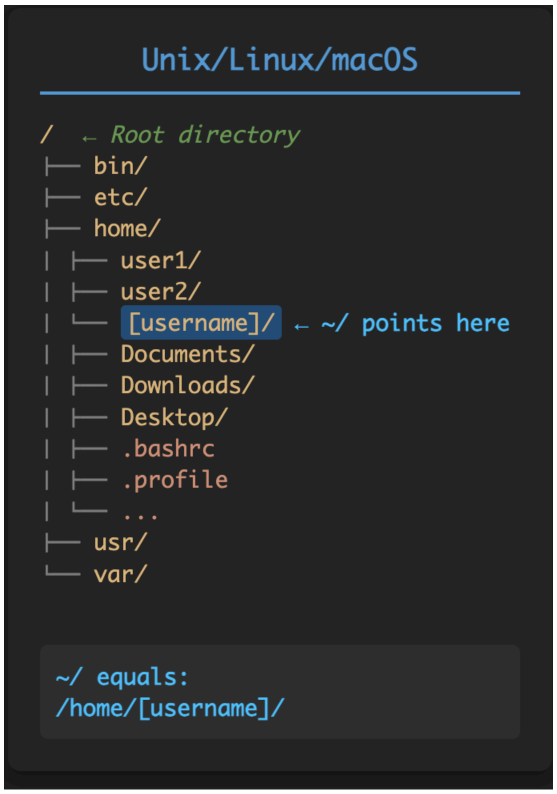


Windows

```
C:\  ← Root drive
├── Program Files/
├── Program Files (x86)/
├── Windows/
└── Users/
├── Public/
├── Administrator/
└── [username]/ ← Home directory
├── Documents/
├── Downloads/
├── Desktop/
├── Pictures/
├── Music/
├── Videos/
└── AppData/
```

Home directory:
C:\Users\[username]\

Also accessible via:
%USERPROFILE%

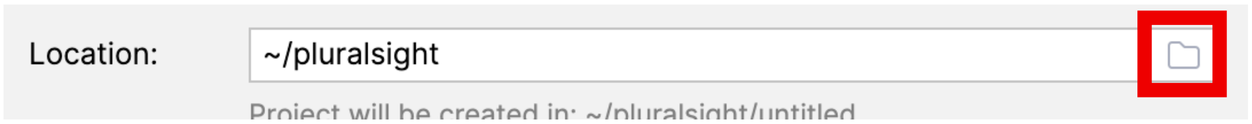


Unix/Linux/macOS

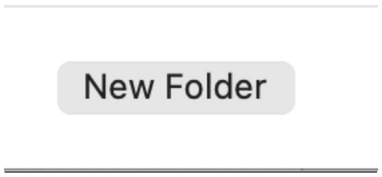
```
/  ← Root directory
├── bin/
├── etc/
├── home/
│  ├── user1/
│  ├── user2/
│  └── [username]/ ← ~/ points here
│  ├── Documents/
│  ├── Downloads/
│  ├── Desktop/
│  ├── .bashrc
│  ├── .profile
│  └── ...
├── usr/
└── var/
```

~/ equals:
/home/[username]/

Also, within Intellij's new project screen, the Location folder button will allow for you to navigate to your home directory and create a new folder!


Location: ~/pluralsight

Project will be created in: ~/pluralsight/untitled



EXERCISE 1

Using IntelliJ, create a new Java application that will list at least 10 items that should be on your shopping list.

1. Create a new package named `com.pluralsight`
2. In the `com.pluralsight` package create a new java class named `ShoppingList`. Remember it must be in a .java file of the same name.
3. Within the `ShoppingList` class, create a `main` method.
4. In the `main()` method use the `System.out.println()` to display a shopping list with at least 10 items. EX. `System.out.println("apricot");`
5. Run the program. If there are any errors, fix them and run it again.
6. Commit and Push your changes to GitHub using your IntelliJ IDE