Getting Started: Android Studio

Hello World in Java Programming

Android Studio

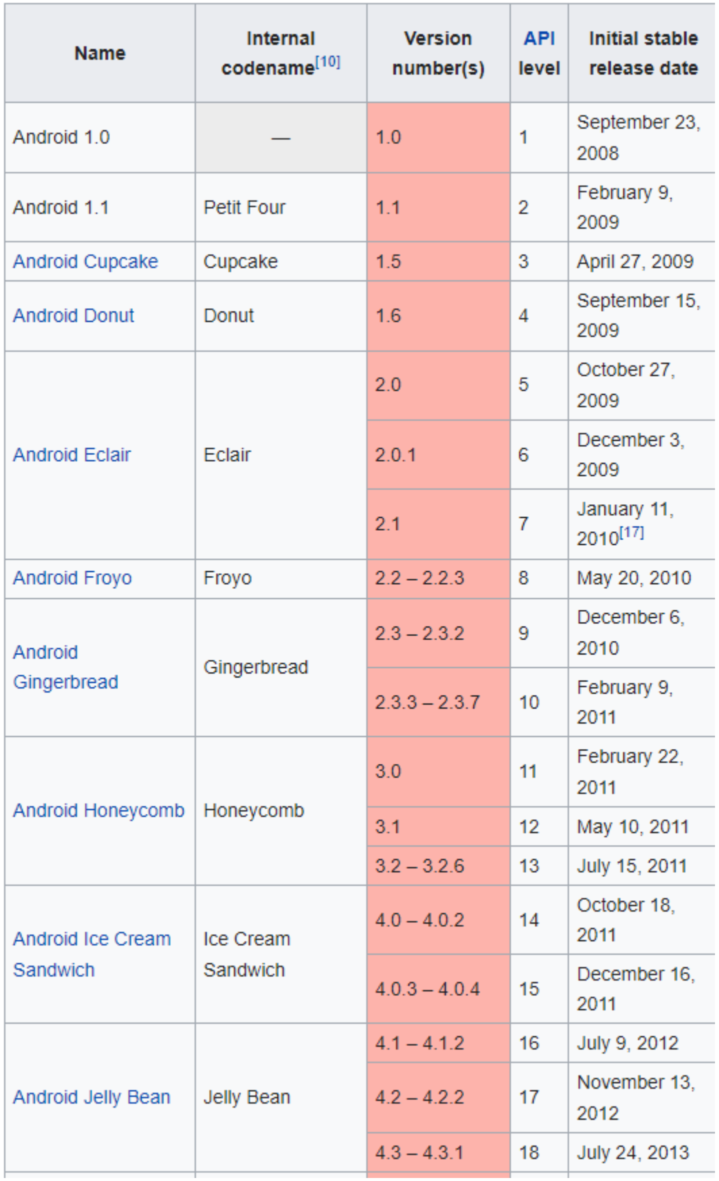
- In this course, we use **Android Studio** IDE to program JAVA and develop android application

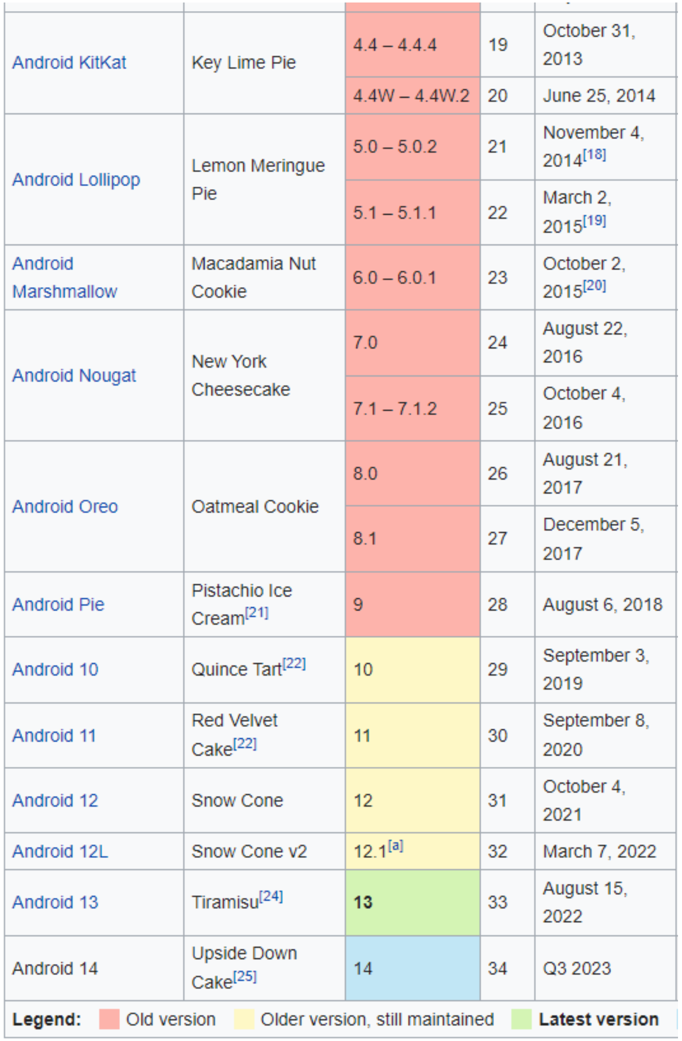
- You can use other IDE if you know what you are doing.

Android Studio: Create Project

- File -> New -> New Project
- Select the form factors: Phone and Tablet
- Select ‘No Activity’ > Next
- Application name: JavaPrograms
- Language: Java

Android Version History

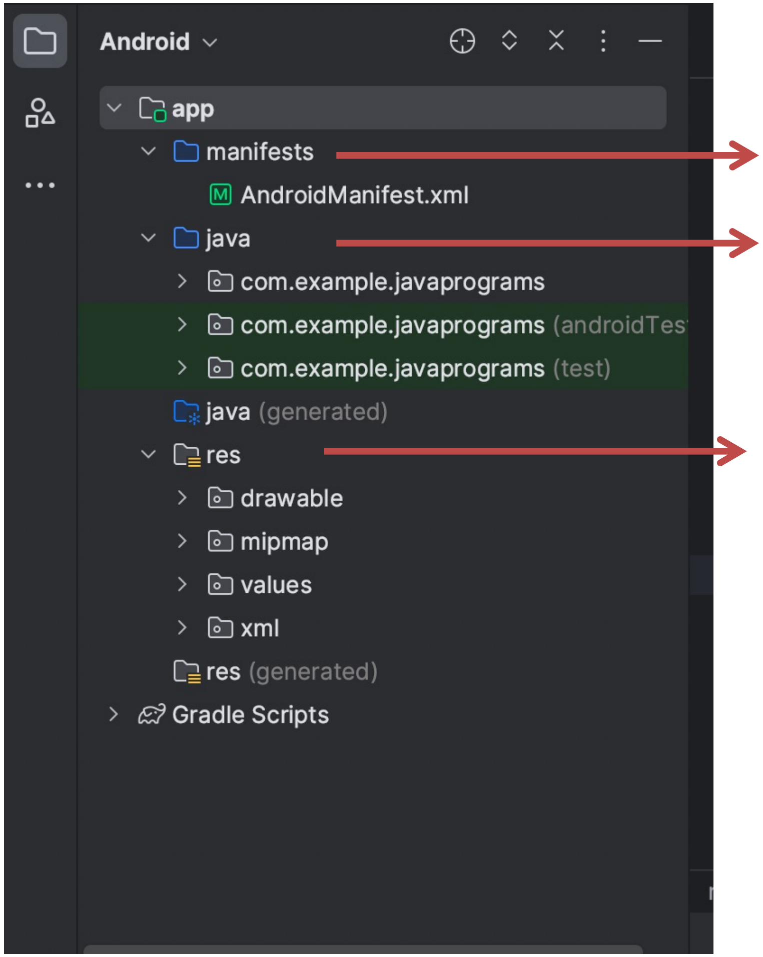




Name	Internal codename[10]	Version number(s)	API level	Initial stable release date
Android 1.0	—	1.0	1	September 23, 2008
Android 1.1	Petit Four	1.1	2	February 9, 2009
Android Cupcake	Cupcake	1.5	3	April 27, 2009
Android Donut	Donut	1.6	4	September 15, 2009
Android Eclair	Eclair	2.0	5	October 27, 2009
		2.0.1	6	December 3, 2009
		2.1	7	January 11, 2010[17]
Android Froyo	Froyo	2.2 – 2.2.3	8	May 20, 2010
Android Gingerbread	Gingerbread	2.3 – 2.3.2	9	December 6, 2010
		2.3.3 – 2.3.7	10	February 9, 2011
Android Honeycomb	Honeycomb	3.0	11	February 22, 2011
		3.1	12	May 10, 2011
		3.2 – 3.2.6	13	July 15, 2011
Android Ice Cream Sandwich	Ice Cream Sandwich	4.0 – 4.0.2	14	October 18, 2011
		4.0.3 – 4.0.4	15	December 16, 2011
Android Jelly Bean	Jelly Bean	4.1 – 4.1.2	16	July 9, 2012
		4.2 – 4.2.2	17	November 13, 2012
		4.3 – 4.3.1	18	July 24, 2013
Android KitKat	Key Lime Pie	4.4 – 4.4.4	19	October 31, 2013
		4.4W – 4.4W.2	20	June 25, 2014
Android Lollipop	Lemon Meringue Pie	5.0 – 5.0.2	21	November 4, 2014[18]
		5.1 – 5.1.1	22	March 2, 2015[19]
Android Marshmallow	Macadamia Nut Cookie	6.0 – 6.0.1	23	October 2, 2015[20]
Android Nougat	New York Cheesecake	7.0	24	August 22, 2016
		7.1 – 7.1.2	25	October 4, 2016
Android Oreo	Oatmeal Cookie	8.0	26	August 21, 2017
		8.1	27	December 5, 2017
Android Pie	Pistachio Ice Cream[21]	9	28	August 6, 2018
Android 10	Quince Tart[22]	10	29	September 3, 2019
Android 11	Red Velvet Cake[22]	11	30	September 8, 2020
Android 12	Snow Cone	12	31	October 4, 2021
Android 12L	Snow Cone v2	12.1[a]	32	March 7, 2022
Android 13	Tiramisu[24]	**13**	33	August 15, 2022
Android 14	Upside Down Cake[25]	14	34	Q3 2023

Legend: 🟥 Old version 🟨 Older version, still maintained 🟩 **Latest version**

Android Project Structure



```
Android
app
  manifests
    AndroidManifest.xml
  java
    com.example.javaprograms
    com.example.javaprograms (androidTes
    com.example.javaprograms (test)
  java (generated)
  res
    drawable
    mipmap
    values
    xml
  res (generated)
Gradle Scripts
```

Manifest file contains hardware and software features your Android app requires and its compatibility specification

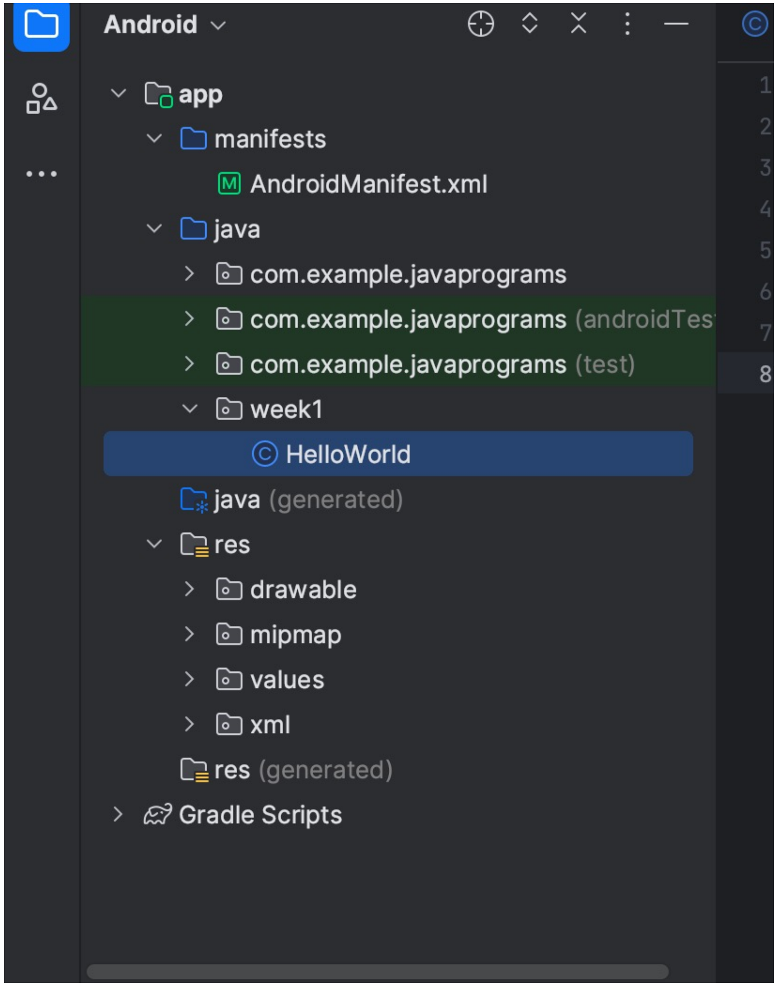
java folder contain java **packages.** Each package contain **java files**.

res folder contains non-code resources for your Android app

We will not use or modify most of these folders for now since Android programming starts on week 8 onwards.

For Java programming, we can work inside the **java** folder.

Creating Package and Java Files

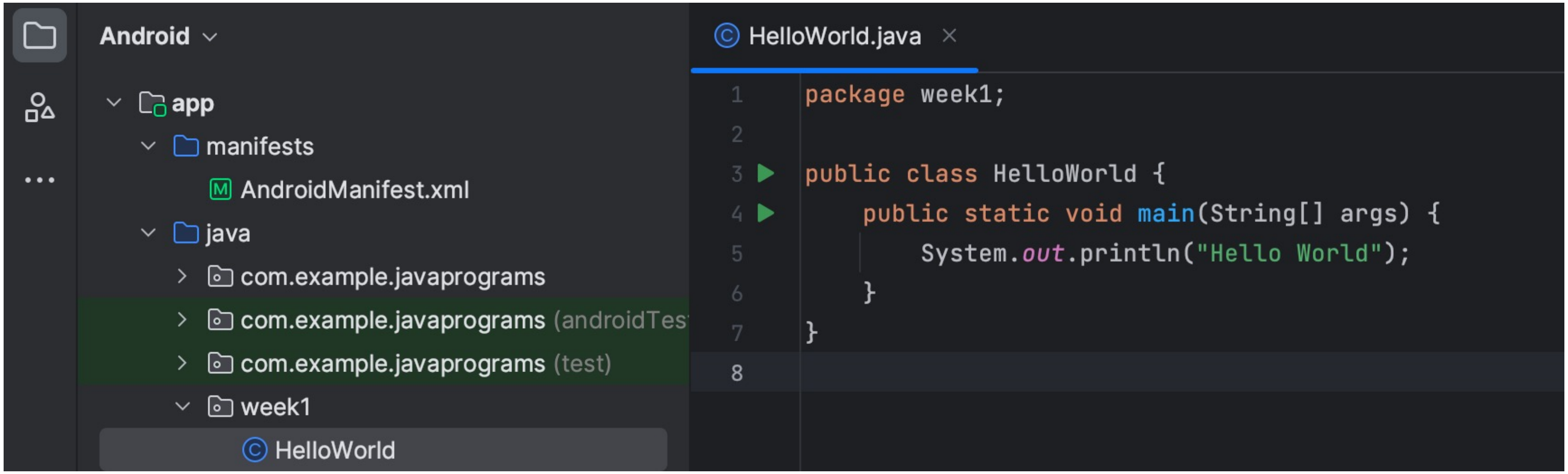


Step 1
Right click on the java folder -> New -> Package
Name the package: ”week1”

Step 2
Right click on the week1 package -> New -> Java Class
Name the file: “HelloWorld”

This is how it looks like now

Write Hello World Program



```java
package week1;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}

```

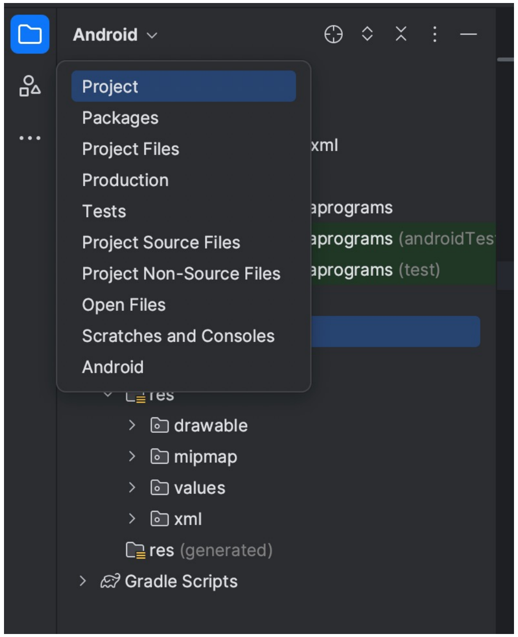
Setting Before Running a Java Code

- Running pure java code is slightly different with running Android app since it does not require simulator and build tool to compile and link all files into an app.

Setting Before Running a Java Code

Do the following setting before running our code:

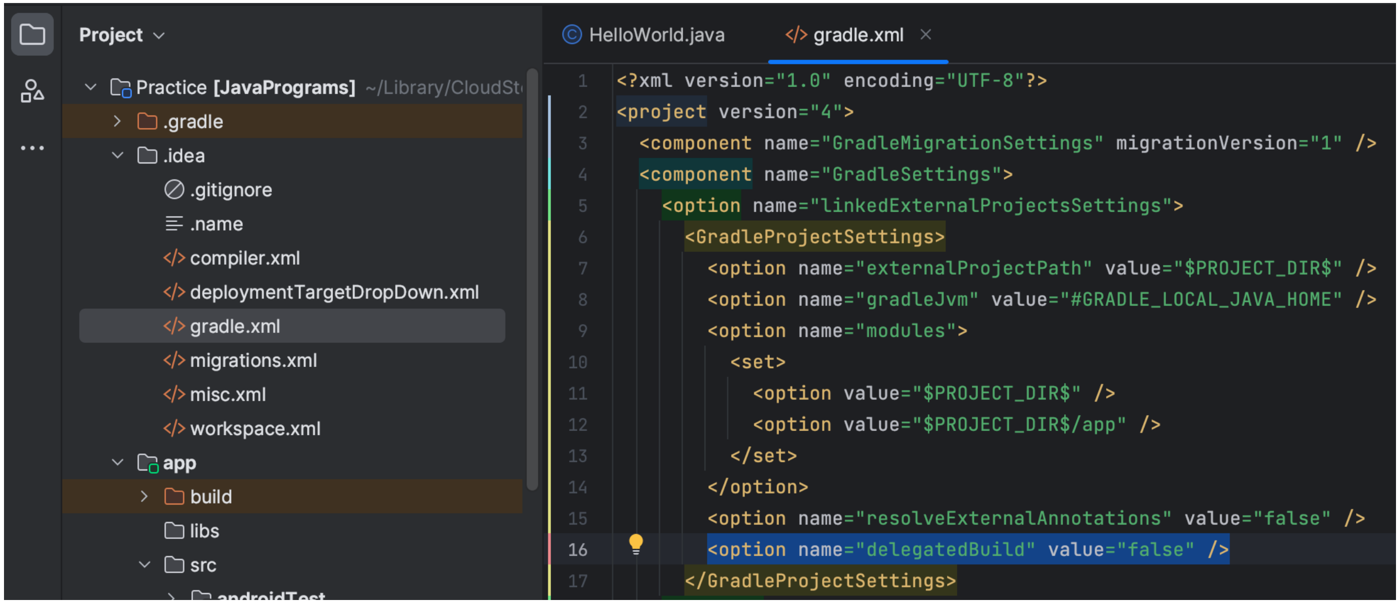
- Change Android view to Project view



Setting Before Running a Java Code

- Expand .idea -> Open gradle.xml file
- Type the following code before </GradleProjectSettings>:

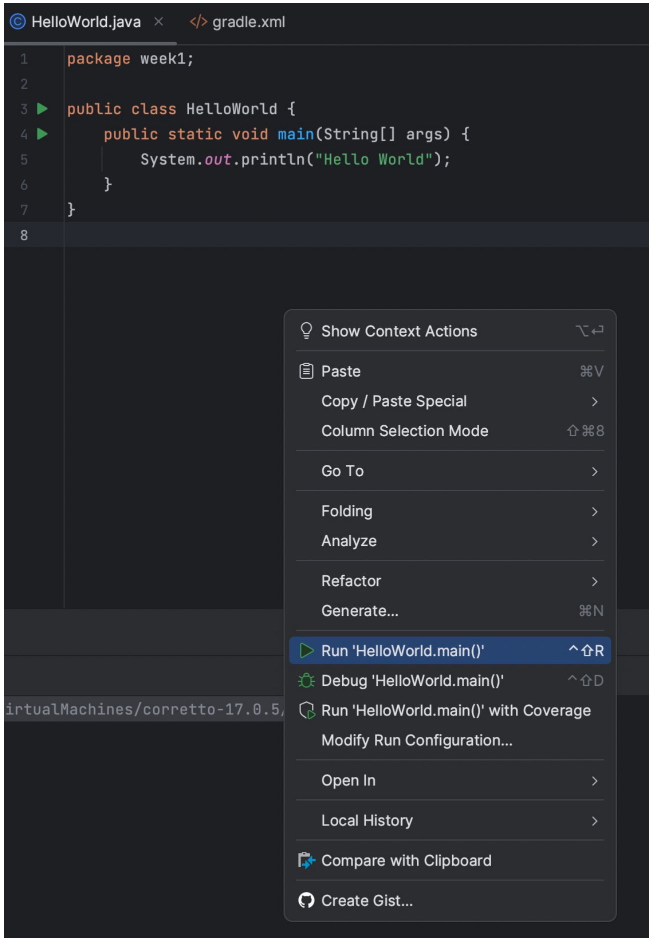
```
<option name="delegatedBuild" value="false" />
```



```
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="GradleMigrationSettings" migrationVersion="1" />
  <component name="GradleSettings">
    <option name="linkedExternalProjectsSettings">
      <GradleProjectSettings>
        <option name="externalProjectPath" value="$PROJECT_DIR$" />
        <option name="gradleJvm" value="#GRADLE_LOCAL_JAVA_HOME" />
        <option name="modules">
          <set>
            <option value="$PROJECT_DIR$" />
            <option value="$PROJECT_DIR$/app" />
          </set>
        </option>
        <option name="resolveExternalAnnotations" value="false" />
        <option name="delegatedBuild" value="false" />
      </GradleProjectSettings>
```

Running the Java Code

- Right Click anywhere on the editor
- Run ‘HelloWorld.main()’

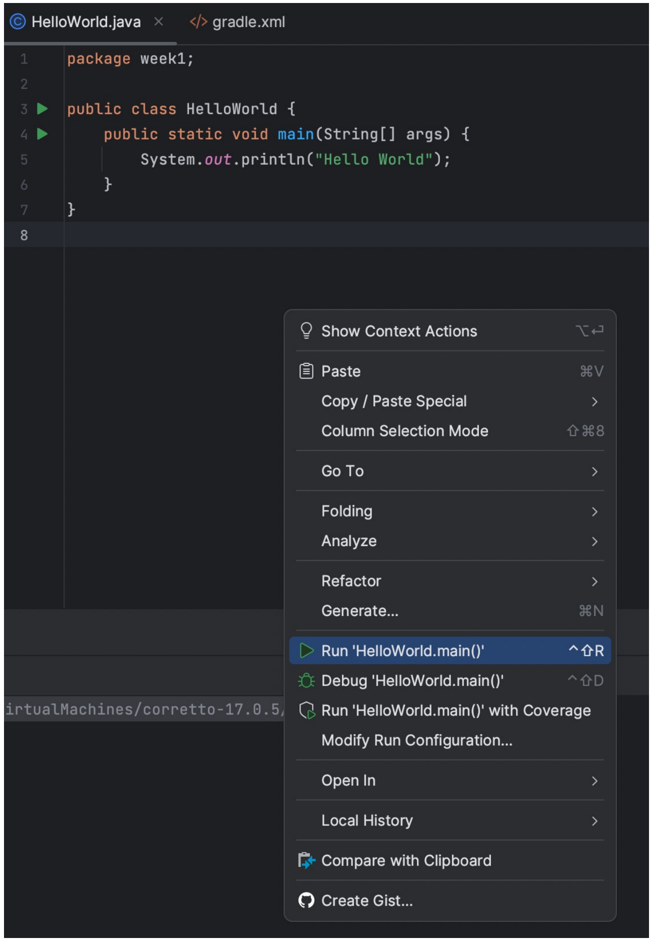


```
package week1;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}

```

```
irtualMachines/corretto-17.0.5/
```

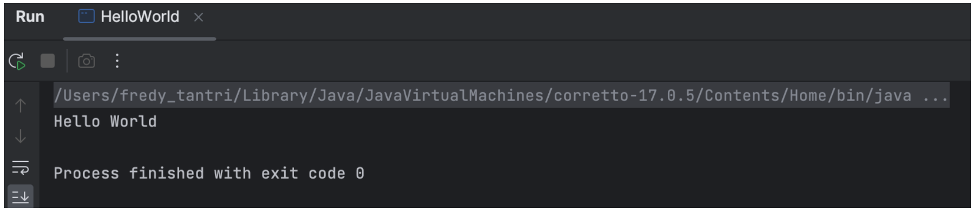
Running the Java Code

- Right Click anywhere on the editor
- Run ‘HelloWorld.main()’
- Now you should be able to see the output in the console



```
package week1;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

```
irtualMachines/corretto-17.0.5/
```



```
/Users/fredy_tantri/Library/Java/JavaVirtualMachines/corretto-17.0.5/Contents/Home/bin/java ...
Hello World

Process finished with exit code 0
```

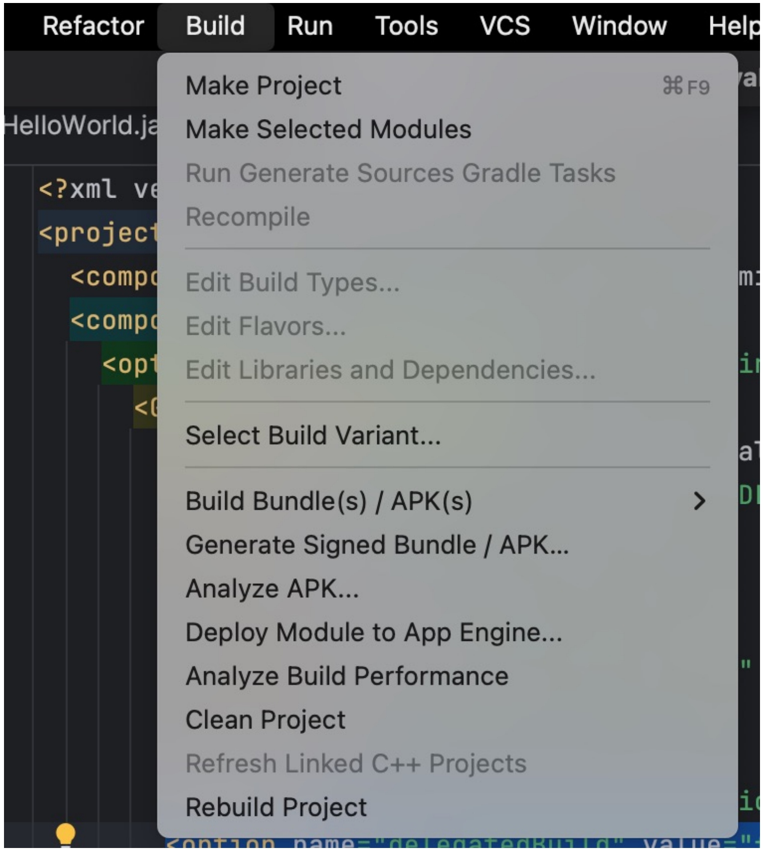
Possible Error

If you get error message "Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.”

Try the following:

on the toolbar, Click Build -> Clean Project

And then, Click Build again -> Rebuild Project



Library, Module, Package, and Class

- Each **.java** file must contain the same public **class** name. For example, **DemoClass.java** must have *public class DemoClass* declared.
- **Package** contains multiple java classes (**.java** files)
- **Module** is introduced in Java 9. Module is a collection of packages
- **Library** is just a general term in Java. Sometimes used interchangeably with module or package, sometimes refers to collection of modules.

Library, Module, Package, and Class

- We are mostly dealing with packages and classes.
- For example, when we need to import the in-built ArrayList data structure, we need to import **java.util.ArrayList**.
  In this case, **java** is a package, **util** is a sub-package, and **ArrayList** is a class.