Java

- Java is free and open-source and is widely used in developing Desktop, Mobile and Web Applications.

JDK vs JRE

- Java Development Kit
    - Develop
        - jdb
        - javadoc
    - Build
        - javac
        - jar
    - Run
        - JRE (Java Runtime Environment)
        - java
- Above tools are available in `bin` directory of the path where Java is installed.

Java Installation and Configuration

- Java Installation Steps

```
export PATH=$PATH:/opt/jdk-13.0.2/bin
```

Java – Build & Packaging

Compile

1. Develop Code

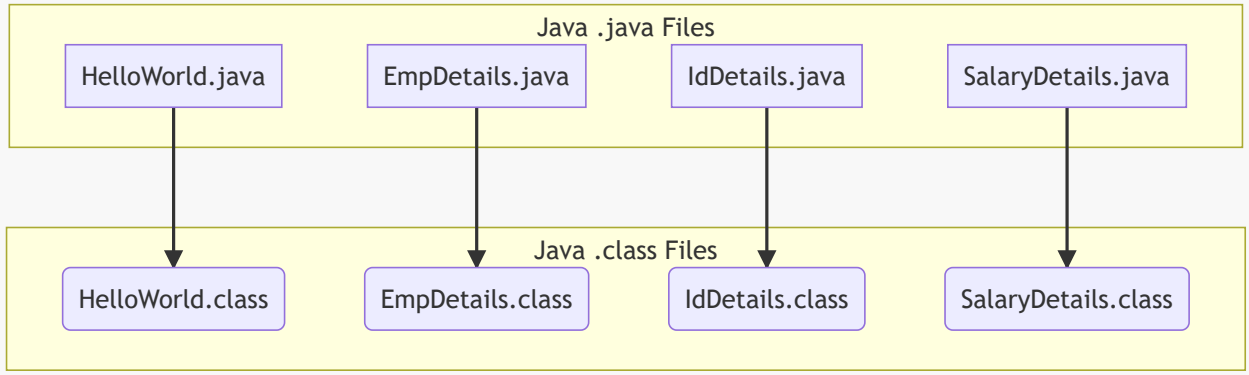
```
// Your First Program
class HelloWorld {
    public static void main(String[] args) {
    System.out.println("Hello, World inside main function");
}
    System.out.println("Hello, World! outside main function");
}
```

2. Compile Code

```
javac HelloWorld.java
# HelloWorld.class file is created
```

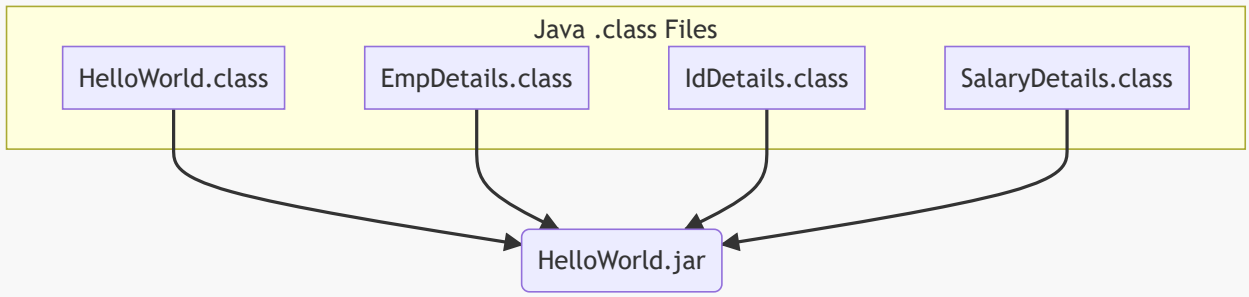
3. Execute the program

```
java HelloWorld
# HelloWorld is class name
```

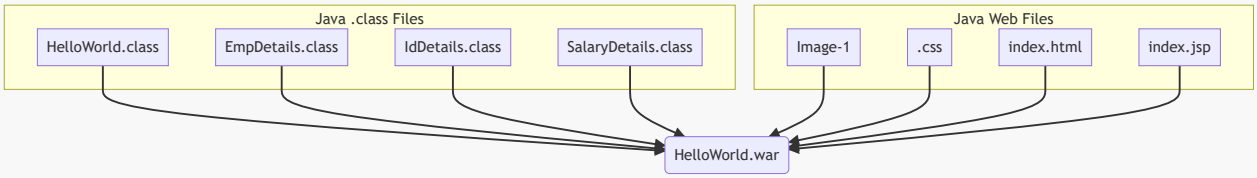


Package

- A Java application will have many `.java` code files. Each of this code file will have its own `.class` files.
- These files may be dependent on each other or it may have dependencies on external libraries.
- To distribute application to end users, these files needs to be packaged with an archive file like `Jar`.
- `Jar` stands for **Java archive** and is useful to compress and combine multiple `.class` files and libraries into a single package.



- In case of web application, there could be static HTML files or image files, for Web App, all files are packaged into a WAR (`Web Archive`) File.



- Create Jar File

```
jar cf HelloWorld.jar HelloWorld.class EmpDetails.class IdDetails.class
# above command will create HelloWorld.jar output file
# When this .jar file is created , a META-INF/MANIFEST.MF file is generated, this
file contains
```

- Content of MANIFEST.MF file

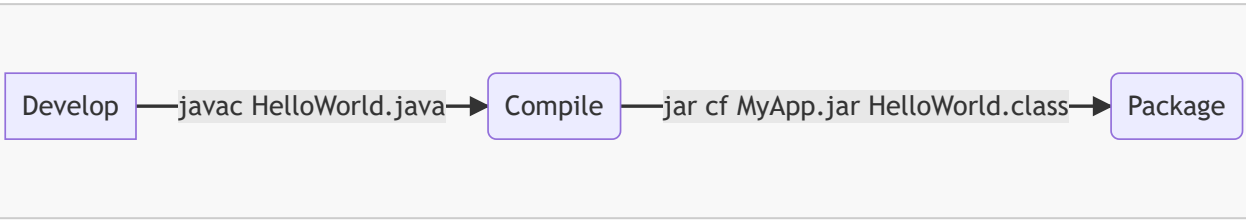
```
Manifest-Version: 1.0
Created-By: 1.8.0_241 (Jar Build Process)
Main-Class: HelloWorld
```

- Here **Main-Class** will be the starting i.e entry point of the application.
- Here, this Jar file can be executed on any system that contains JRE.
- Execute Jar file using java command

```
java -jar HelloWorld.jar
```

Document - javadoc

- To generate a document for the code written in **.java**, use `javadoc -d doc HelloWorld.java`



- Here **javac** command, **jar** command and **javadoc** command are a part of JDK.
- Ideally, when there will be multiple developers writing multiple **.java** files, the package and build process can be complex.

Build Tools for Java

- The Build Tools use configuration files where we can specify steps that are to be executed by build tool for build process
    - Build Steps
        - Compile
        - Package
        - Document
- The build tool will execute the steps in the order of definition of the build process.

Maven

- Maven is a Java tool, so Java should be installed in order execute maven commands.
- A standard directory structure that is followed for any Java Project.

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

Standard Directory Layout

Directory	HTML
`src/main/java`	Application/Library sources
`src/main/webapp`	Web application sources
`src/test/java`	Test sources
`src/test/resources`	Test resources

- At the top level, files descriptive of the project: a `pom.xml` file
- The `src/main/java` directory contains the project source code, and the `pom.xml` file is the project's Project Object Model, or POM.

The POM

- The pom.xml file is the core of a project's configuration in Maven. It is a single configuration file that contains the majority of information required to build a project in just the way you want. The POM is huge and can be daunting in its complexity, but it is not necessary to understand all of the intricacies just yet to use it effectively. This project's POM is:

```
<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>
    <artifactId>java-tomcat-sample</artifactId>
    <packaging>war</packaging>
    <groupId>com.mycompany.app</groupId>
    <version>1.0-SNAPSHOT</version>

    <properties>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
    </properties>

    <dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>
    </dependencies>
</project>
```

Maven Phases

- **validate**: validate the project is correct and all necessary information is available

- **compile**: compile the source code of the project
- **test**: test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- **package**: take the compiled code and package it in its distributable format, such as a JAR.
- **install**: install the package into the local repository, for use as a dependency in other projects locally.
- **deploy**: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.
- There are two other Maven lifecycles as below:
    - **clean**: cleans up artifacts created by prior builds
    - **site**: generates site documentation for this project