Java 11 - Chapter 1

The `javac` program generates instructions in a special format that the `java` command can run called *bytecode*. Then `java` launches the *Java Virtual Machine* (JVM) before running the code. The JVM knows how to run bytecode on the actual machine it is on.

A Java program begins execution with its `main()` method. A `main()` method is the gateway between the startup of a Java process, which is managed by the Java Virtual Machine (JVM), and the beginning of the programmer's code. The JVM calls on the underlying system to allocate memory and CPU time, access files, and so on. In this section, you will learn how to create a `main()` method, pass a parameter, and run a program both with and without the `javac` step.

The `java` command starts a Java application. It does this by starting a Java runtime environment, loading a specified class, and calling that class's `main` method.

Starting in Java 11, you can run a program without compiling it first—well, without typing the `javac` command that is. Let's create a new class:

```
public class SingleFileZoo {
   public static void main(String[] args) {
      System.out.println("Single file: " + args[0]);
   }
}
```

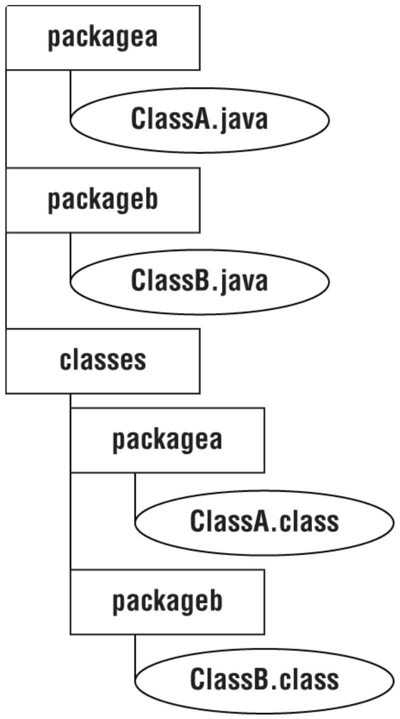
We can run our `SingleFileZoo` example without actually having to compile it.

```
java SingleFileZoo.java
```

This feature is called launching *single-file source-code* programs. The name cleverly tells you that it can be used only if your program is one file. This means if your program has two `.java` files, you still need to use `javac`.

Full command	Single-file source-code command
`javac HelloWorld.java` `java HelloWorld`	`java HelloWorld.java`
Produces a class file	Fully in memory
For any program	For programs with one file
Can import code in any available Java library	Can only import code that came with the JDK

`javac -d classes packagea/ClassA.java packageb/ClassB.java`



To run the program, you specify the classpath so Java knows where to find the classes. There are three options you can use. All three of these do the same thing:

```
java -cp classes packageb.ClassB
java -classpath classes packageb.ClassB
java --class-path classes packageb.ClassB
```

Option	Description
`-cp <classpath>` `-classpath <classpath>` `--class-path <classpath>`	Location of classes needed to compile the program
`-d <dir>`	Directory to place generated class files

Creating a JAR File

The simplest commands create a `jar` containing the files in the current directory. You can use the short or long form for each option.

```
jar -cvf myNewFile.jar .
jar --create --verbose --file myNewFile.jar .
```

Alternatively, you can specify a directory instead of using the current directory.

```
jar -cvf myNewFile.jar -C dir .
```

Running a Program in One Line with Packages

You can use single-file source-code programs from within a package as long as they rely only on classes supplied by the JDK. This code meets the criteria.

```
package singleFile;

import java.util.*;

public class Learning {
```

```java
   private ArrayList list;
   public static void main(String[] args) {
      System.out.println("This works!");
   }
}
```

You can run either of these commands:

```
java Learning.java            // from within the singleFile directory
java singleFile/Learning.java // from the directory above singleFile
```