Building Java Projects

Compiling Java code

Using Makefiles

Packages and dependencies

Using Gradle builds

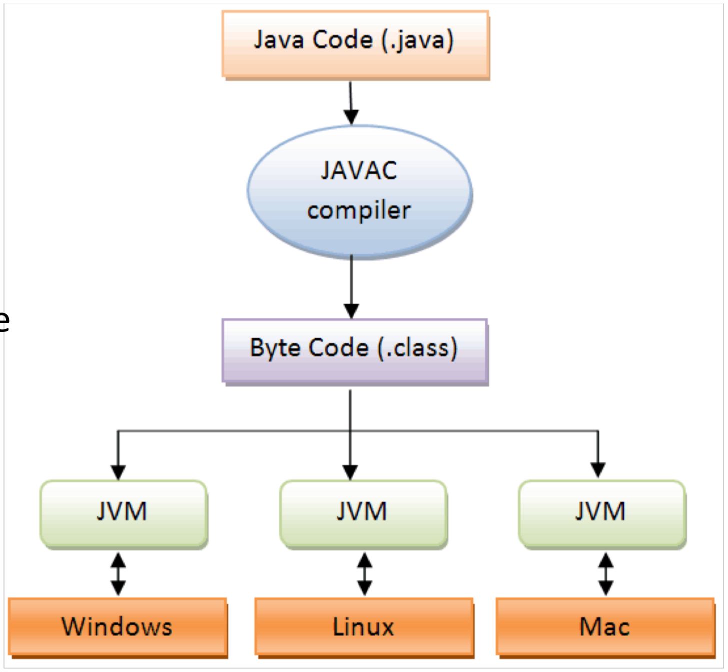
Portability through Virtualization

Native vs. Intermediate compilation

- C++ compilers produce native code.
- Some languages (e.g. Python) produce intermediate code which is interpreted at runtime.
- Java compiles to IR: the compiler produces bytecode (.class files), which are executed by a Java Virtual Machine (JVM)

Why is this useful?

- JVM can execute code produced by any language that emits Java IR.
  - e.g. Java, Scala, Kotlin
- Java IR code can runs on any platform that has a JVM (Mac, Linux, Windows, Raspberry Pi, Arduino…)

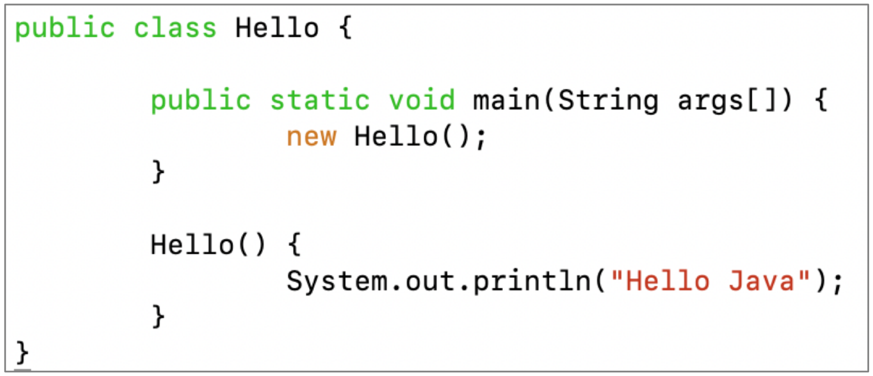


http://viralpatel.net/blogs/java-virtual-machine-an-inside-story/

Compiling Java

Java classes need to be contained in a file of the same name. e.g. class Hello needs to be in Hello.java.

Hello.java:



```java
public class Hello {

	public static void main(String args[]) {
		new Hello();
	}

	Hello() {
		System.out.println("Hello Java");
	}
}
```

```
$  javac Hello.java
$  ls
    Hello.class  Hello.java
$  java Hello
    Hello Java
```

Multiple Files

In Java, you typically have many source files (one class in each file).
We can build multiple Java files with wildcards, typically in a makefile.

Makefile:

```
NAME = "Hello"
all:
	@echo "Compiling..."
	javac $(NAME).java
run: all
	@echo "Running..."
	java $(NAME)
clean:
	rm -rf *.class
```

We typically don’t use makefiles for complex projects. Why?

Complex Builds

`Make’ doesn’t scale to large projects very well.

- Projects can easily be hundreds (thousands, tens of thousands) of files and classes.
- Dependencies are difficult to navigate and manage, and `make` doesn’t help. At all.
- Larger projects use tools that help resolve dependencies, support incremental builds. e.g. Ant, Maven, Gradle.

We’re going to use `**gradle’** for this course.

- Provides more functionality that `make’
- Faster to build and compile apps
- Supports Java (C++, Android)
- Works from the command-line but is also supported by major IDEs

  e.g. IntelliJ, Xcode, Visual Studio.

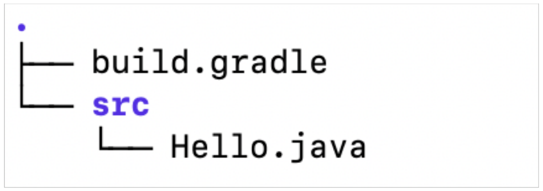
Gradle Build Tool

www.gradle.org

Hello Gradle!

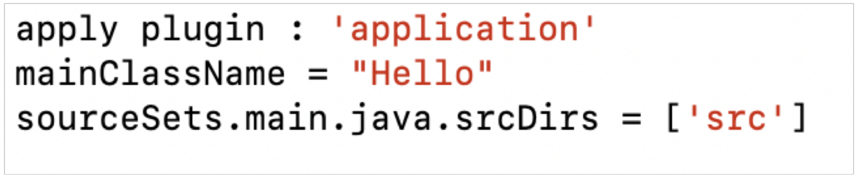
Let’s convert to Gradle!

1. Move the source code into a src directory.
2. Create a build.gradle file to replace the makefile.



```
.
├── build.gradle
└── src
    └── Hello.java
```

Build.gradle:



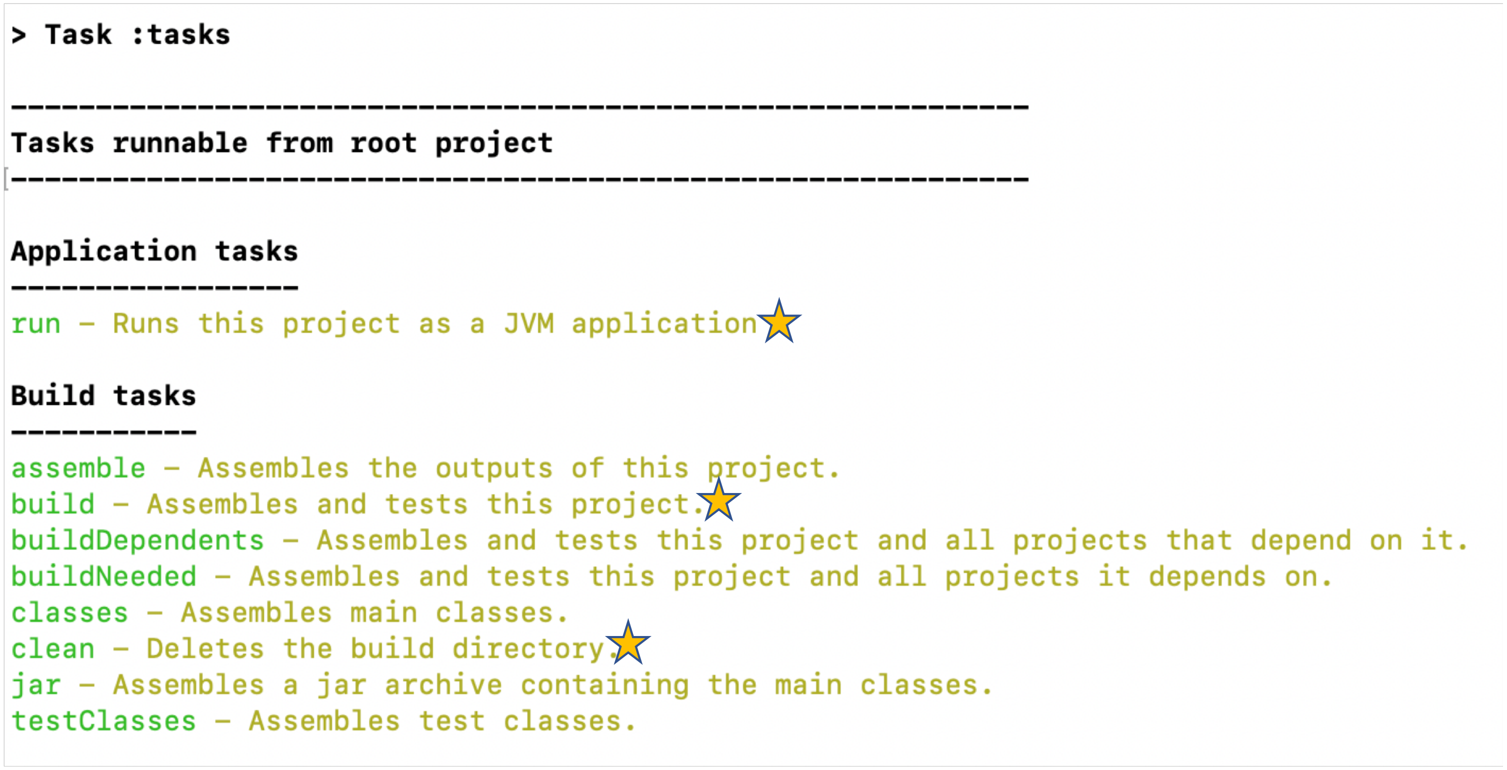
```
apply plugin : 'application'
mainClassName = "Hello"
sourceSets.main.java.srcDirs = ['src']
```

← this is a Java application
← the name of the class to run
← subdirectory containing code

```
$ gradle build
BUILD SUCCESSFUL in 0s
5 actionable tasks: 5 up-to-date
$ gradle run
> Task :run
Hello Gradle!
```

Gradle Tasks

$ gradle tasks



```
> Task :tasks

------------------------------------------------------------
Tasks runnable from root project
------------------------------------------------------------

Application tasks
-----------------
run - Runs this project as a JVM application★

Build tasks
-----------
assemble - Assembles the outputs of this project.
build - Assembles and tests this project.★
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
classes - Assembles main classes.
clean - Deletes the build directory.★
jar - Assembles a jar archive containing the main classes.
testClasses - Assembles test classes.
```

... and many more.

Java Packages

Name collisions are also very likely in large projects

- For example, classes with the same name, which can happen when mixing code from different sources (e.g. third party libraries and your code).
- C++ uses namespaces to logically group code and avoid name collisions.

Java groups classes into ”packages”, which serve the same purpose.

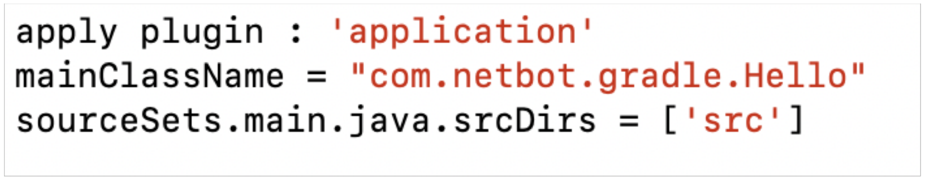
- Convention is to define a package name using company URL backwards (e.g. com.sun.awt).
- Package name needs to be unique to your code/project (e.g. com.uwaterloo.cs349.gradle)

- **`package`** keyword to assign source to a package
  - Typically, a package is a subdirectory that mirrors the package name (dot-separated).
  - e.g. “com.cs349.graphics” package is in directory structure com/cs349/graphics.
- **`import`** keyword to include a class from a different package
  - This is also how you include bundled Java libraries.

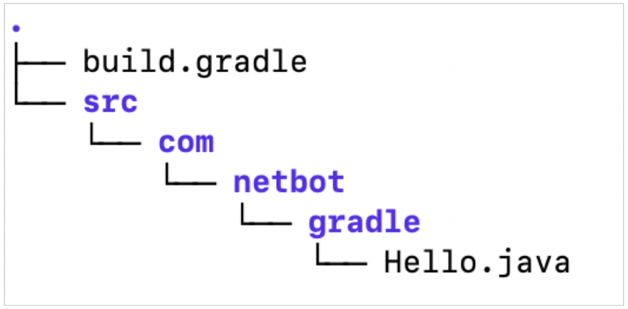
Using Gradle with Packages

- Put the package name in your course code (e.g. com.netbot.gradle).
- Create the directory structure to match the package name.
- build.gradle: change className to the full name (e.g. com.netbot.gradle.Hello)

Build.gradle:



```
apply plugin : 'application'
mainClassName = "com.netbot.gradle.Hello"
sourceSets.main.java.srcDirs = ['src']
```



```
.
├── build.gradle
└── src
    └── com
        └── netbot
            └── gradle
                └── Hello.java
```

```
$ gradle build
BUILD SUCCESSFUL in 0s
5 actionable tasks: 5 up-to-date

$ gradle run
> Task :run
Build and run with Gradle!
```

Installing Gradle

Gradle Build Tool

www.gradle.org

- Mac: brew install gradle
- Linux: apt-get install gradle

Build Java projects with Gradle
https://medium.com/@petehouston/build-java-projects-with-gradle-103247d4b2b3

Gradle Tutorial : How to build and run a Java Application
https://www.youtube.com/watch?v=RrVURuzcFhY