Classes and Packages

Class Syntax

modifiers class *name attributes* {
    contents
}

For example:

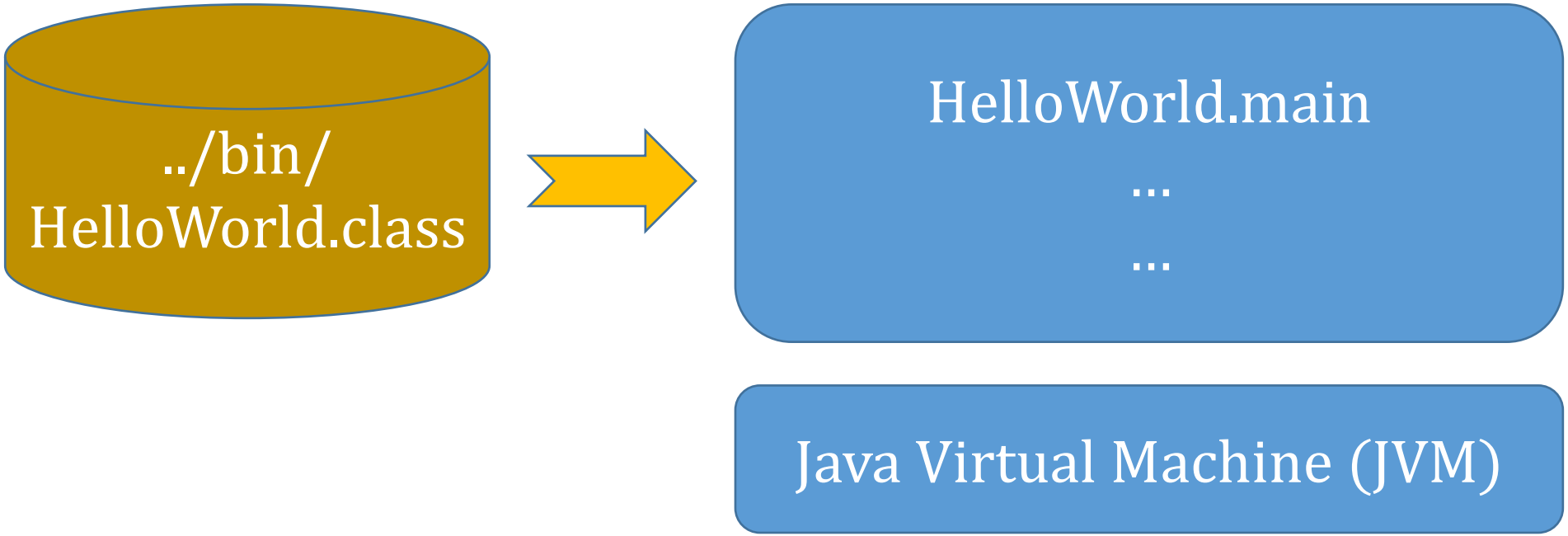
```
class HelloWorld {
    // class HelloWorld contents…
}
```

Compiling Java Code

PWD=~/cs140/src

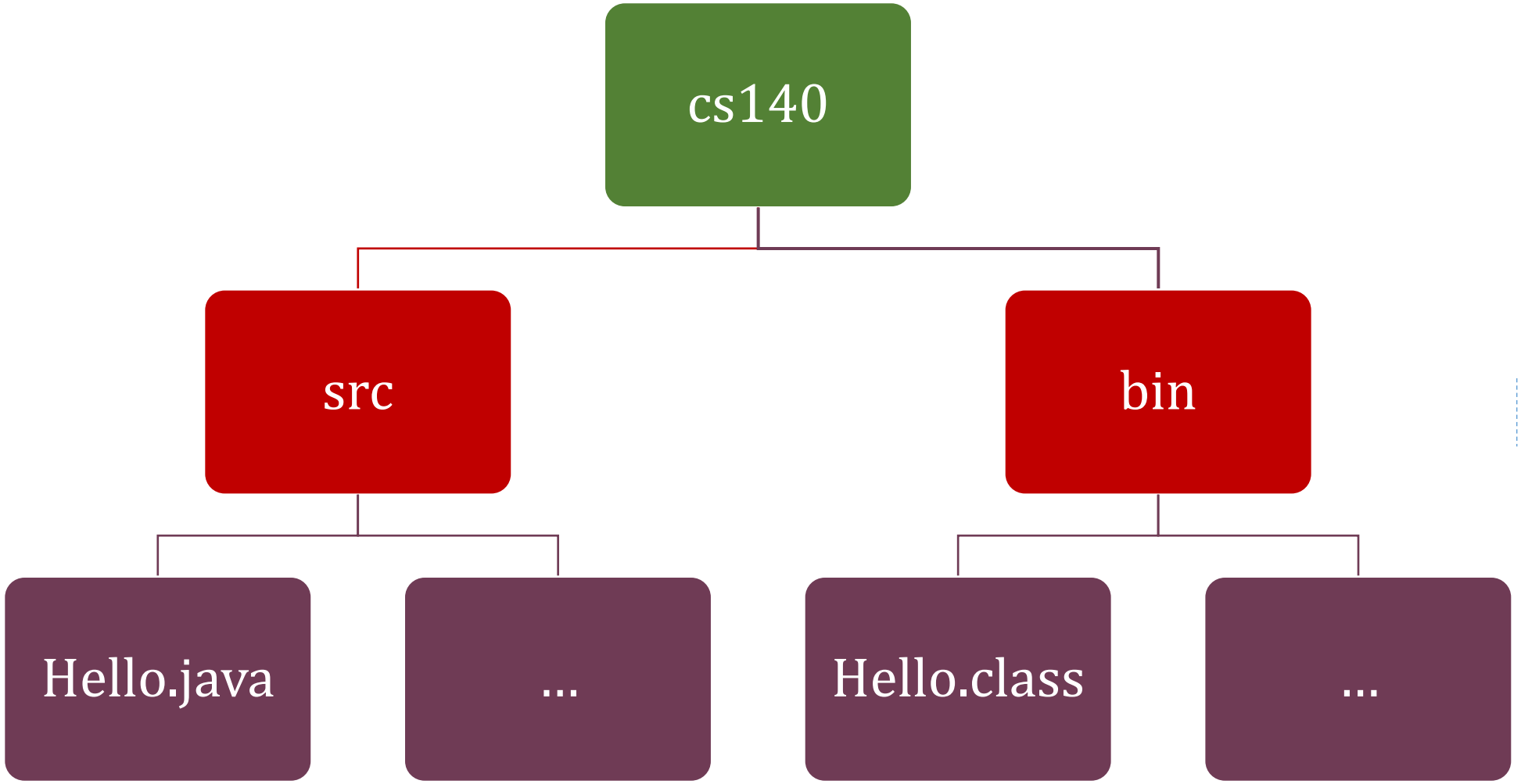
javac –d ../bin HelloWorld.java

Running Java Code



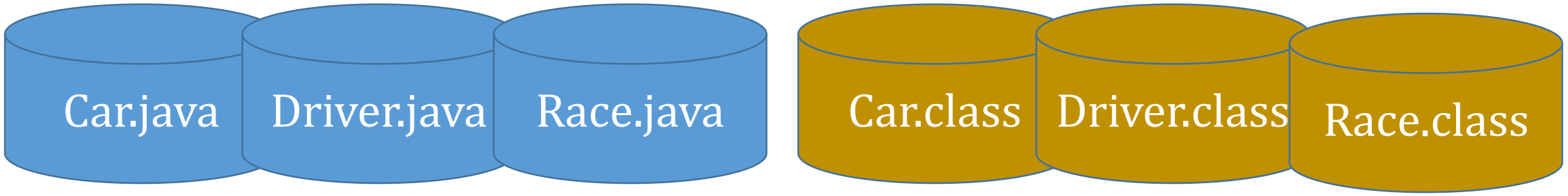
java –cp ../bin HelloWorld

Resulting Directory Structure



Java Package

- A Java program often contains multiple classes
  - Enable objects of different classes to interact with each other
  - For instance, cars and drivers
  - But cars may be in Cars class, drivers may be in Drivers class



- Packages in Java enable combining interacting classes to make a single, integrated set of programs

Typical Packages

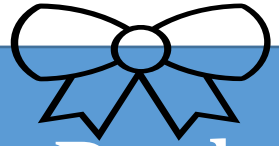
- Package identified in source file via a "package" statement
  - Syntax: `package` *name*;
  - For instance: `package hello;`
- Source (.java) files kept in a package sub-directory
- Bytecode (.class) files in a package sub-directory by compiler
- By convention, package names start with a lower case letter

Class Syntax

```
package package_name;

modifiers class name attributes {
    contents
}
```

Example Package



race Package

Car.java

```
package race;

class Car {
        // Car fields
        // Car methods
}
```

Driver.java

```
package race;

class Driver {
        // Driver fields
        // Driver methods
}
```

java compiler (javac) package rules

- When you compile your code without –d,
  the .class bytecode file is written to the current directory
  whether your code is in a package or not

- If you compile with the -d *classdir* flag,
  - If the code **is not** in a package;
    the .class file is written to *classdir*
  - If the code **is** in a package;
    the .class file is written to a package sub-directory of *classdir*
    (javac makes the sub-directory if it is not there.)

javac package examples

Current Directory	package	command	…. writes
~/cs140/src	---	javac HelloWorld.java	~/cs140/src/HelloWord.class
		javac –d . HelloWorld.java	~/cs140/src/HelloWorld.class
		javac –d ../bin HelloWorld.java	~/cs140/bin/HelloWorld.class
~/cs140/src/race	package race;	javac Car.java	~/cs140/src/race/Car.class
		javac –d . Car.java	~/cs140/src/race/race/Car.class
		javac –d ../../bin Car.java	~/cs140/bin/race/Car.class

pwd: ~/cs140/src/race
..: ~/cs140/src
../.. : ~/cs140
../../bin : ~/cs140/bin

Compiling Java Code

PWD=~/cs140/src/race

```
javac –d ../../bin *.java
```

CLASSPATH (-cp) and Packages

- Java utilities look for bytecode .class files using the "class path"
  - Compiler finds definitions for other classes using the class path
  - Java Interpreter finds referenced classes using the class path
  - By default, the class path is the current directory, but parameters (typically –cp pathname) allow you to specify a class path

- **If a class is in a package Java utilities look for class file in a sub-directory of the class path**
  - Subdirectory name is the package name

Java Execution (java) and Packages

- When running the "java" command, specify the package name as well as the class which contains the main method
  - e.g. "java race.Car"
    looks for main method in ./race/Car.class

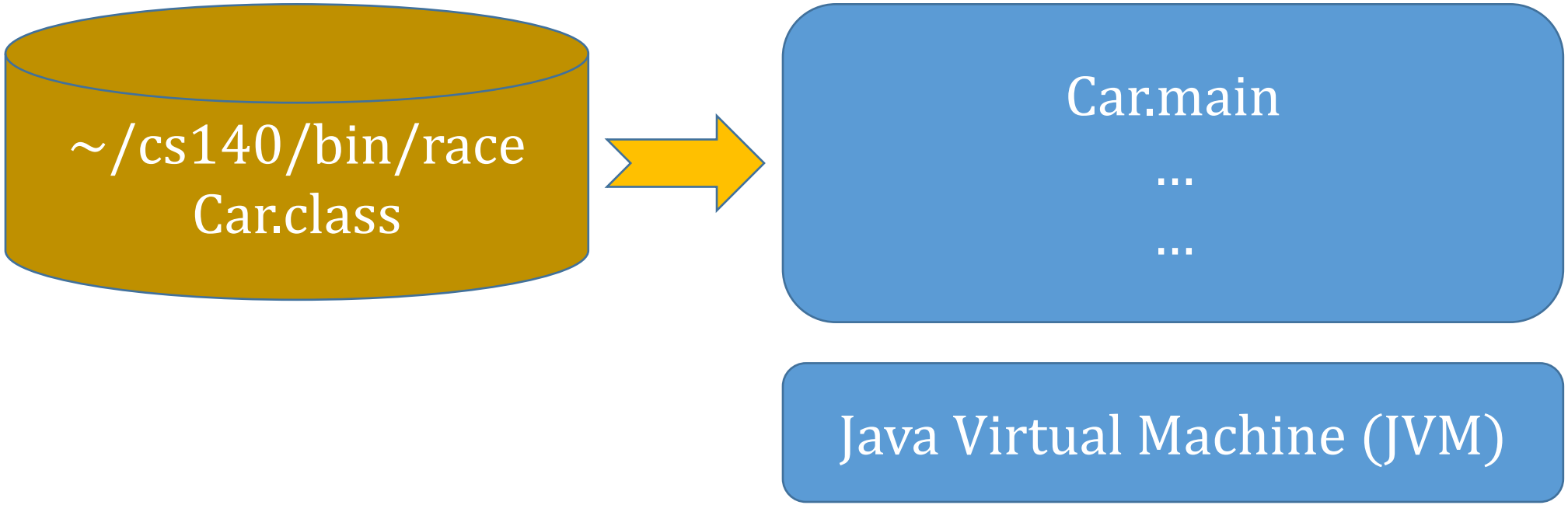
Separator is a dot, not a slash!

- When running the "java" command, use the "-cp" flag to specify where to look for the package sub-directory
  - e.g. "java –cp ~/cs140/bin race.Car"
    looks for main method in ~/cs140/bin/race/Car.class

I could have used -cp ../../bin

- If you make a mistake, you will get:
  Error: Could not find or load main class HelloWorld

Running Java Code



java –cp ../../bin race.Car

Resulting Directory Structure