Classes and Packages

Hello World Demo

Writing the Code

- Code in the file HelloWorld.java is the ***source code*** of the program.
- It must be written with an editor that produces <u>plain text.</u> Use Notepad, or some Java editing tool like Eclipse on Windows and gedit, vi or emacs or Eclipse on Linux
- All java code must be in a ***class***
- Class names start with a capital letter (not checked by the compiler, but good rule anyway)
- The file name matches the class name of the class defined in the file (not checked by the compiler, but good rule anyway)

Class Syntax

```
modifiers class name attributes {
    contents
}
```

For example:

```
class HelloWorld {
    // class HelloWorld body…
}
```

Object Orientation – Why classes?

- "Object" dictionary definition:
  - a material thing that can be seen and touched
  - a person or thing to which a specified action or feeling is directed
- Group similar objects into a "class"
  - a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality

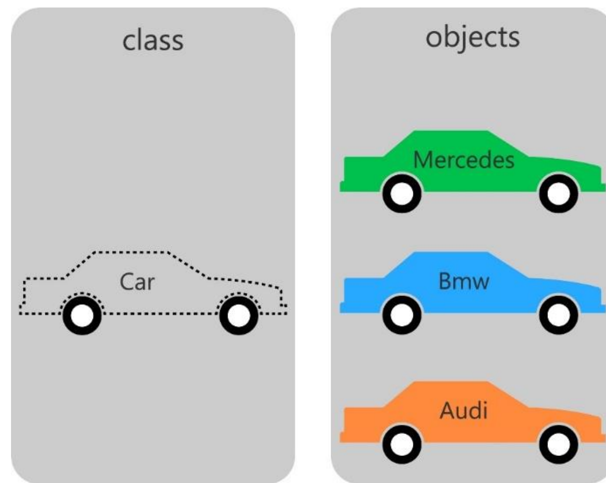
View the World as Classes of Objects

- An “object” is the computer model of a “real” object
- Group the objects our program is concerned about into classes
- Every object in a class shares:
  - How it is described – what data is used to describe it
  - How it can be manipulated – what actions can be performed on any object in the class

Example Class: Car

How Cars are Described

- Make
- Model
- Year
- Color
- Owner
- Location
- Mileage



Actions that can be applied to cars

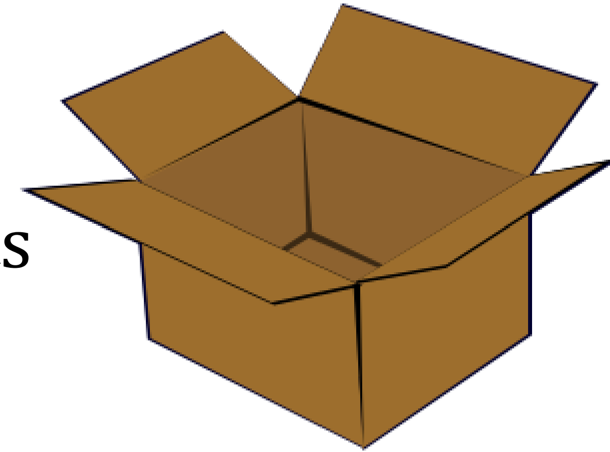
- Create a new car
- Transfer ownership
- Move to a new location
- Repaint
- Delete a car

What's in a class?

- Fields – The data used to describe an object in the class

- Methods – The functions used to manipulate an object in the class

Object Oriented Encapsulation



- Concept: Leave dealing with cars up to the car experts
- If you aren't a car expert, don't go under the hood!
- Make one place the "auto-mechanic" place
- That place has the only code that modifies car objects!
- If any one outside of that place wants to interact with cars, it has to invoke a service provided by the expert

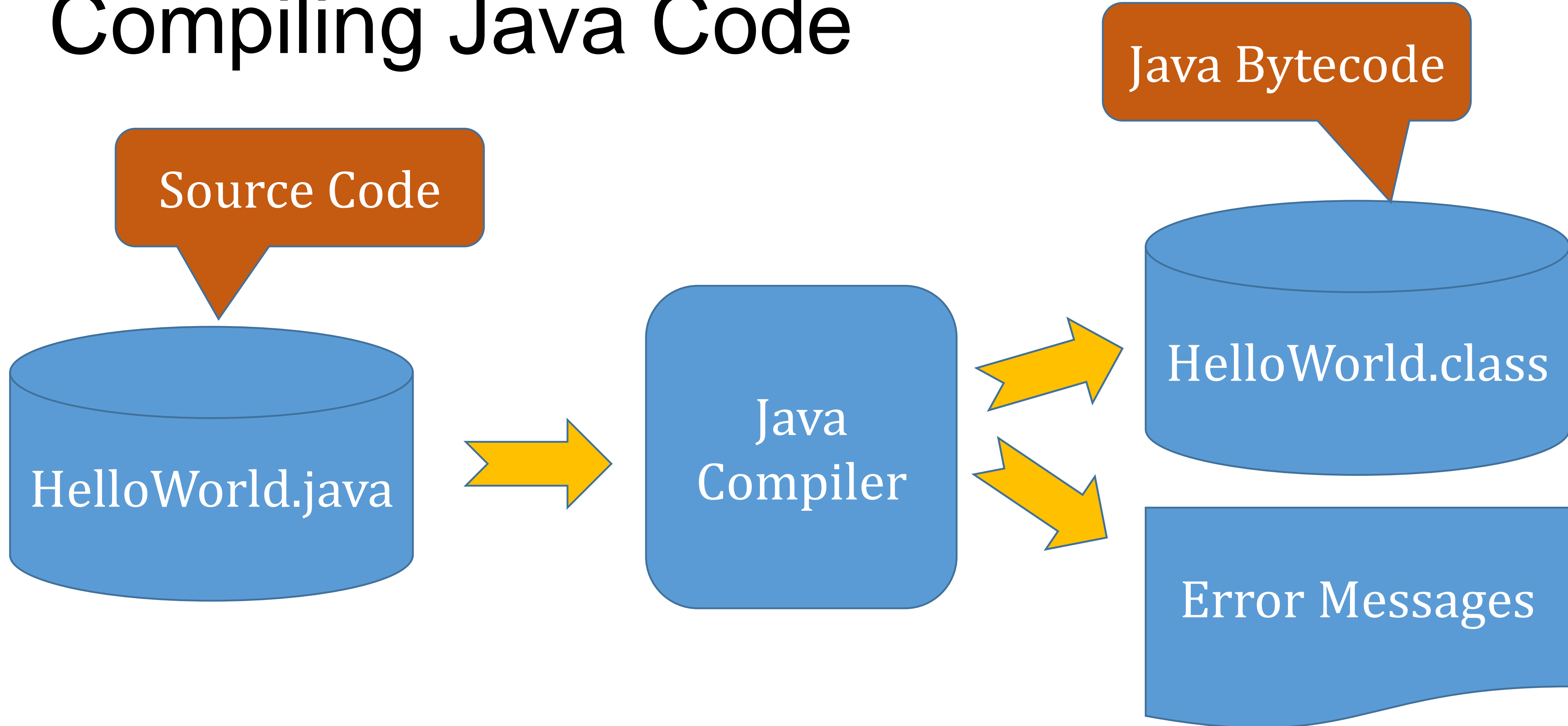
Why Object Orientation?

- Imposes structure on design
  - Forces everything into an object/action way of thinking
  - Reduces the number of choices we need to consider
- Establishes Responsibility / Traceability
  - If a “car” object has inconsistent values, there is a bug in the car class… it can’t be anywhere else!
- Establishes Areas of Expertise
  - Go to the car mechanic to get our car fixed… she knows how to fix it
- Re-Use
  - I can use the same classes in different programs

HelloWorld Class

- The HelloWorld class is not very good object oriented design!
- Since Java requires all code to be in classes, we had to make up a "class" to contain our "main" method
- In the future, classes will be more meaningful

Compiling Java Code

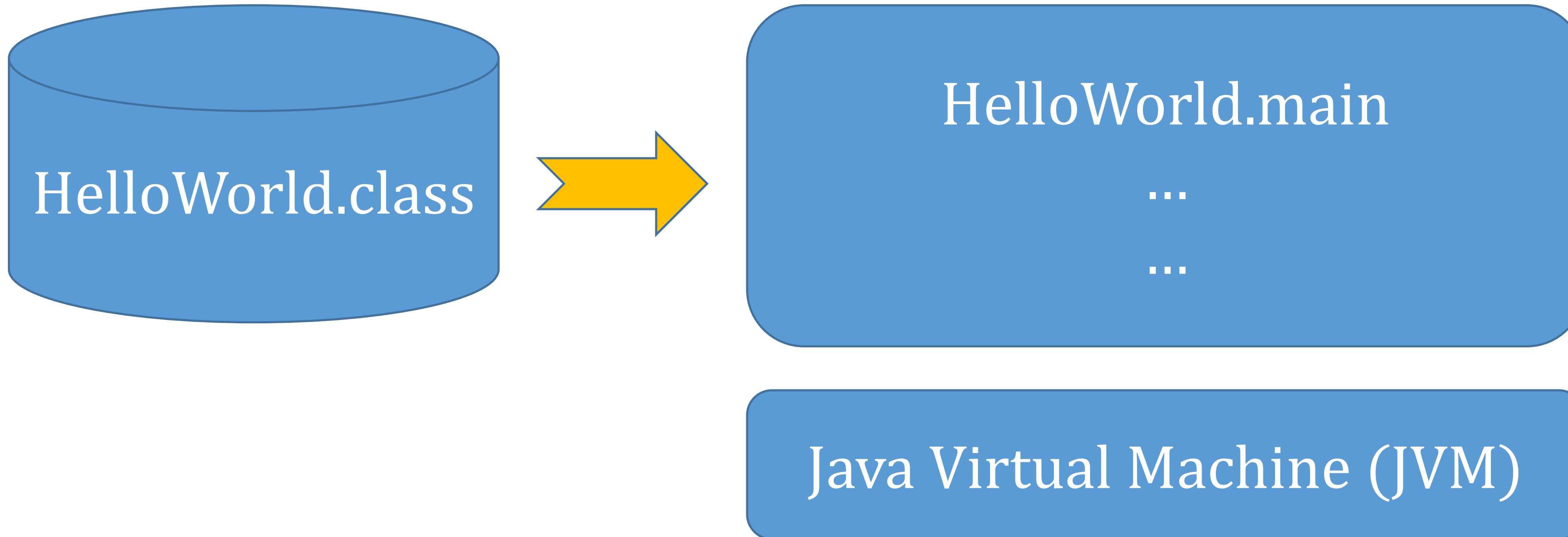


javac HelloWorld.java

Machine Code (aka Object Code)

- Instructions for the computer
- Binary code – strings of ones and zeroes
  - Tells the computer what to do (add, subtract, multiply, etc.)
  - Tells the computer where to get the input data (operands)
  - Tells the computer where to put the output data (results)
- Each computer supports its own machine code
  - Some agreements on “architecture” – e.g. x86, RISC, etc.
- Java has a special feature… “Java Virtual Machine”
  - Reads “java” bytecode from .class files
  - Converts it to machine code instructions, and executes those instructions

Running Java Code



java HelloWorld

Benefits of the Virtual Machine

- Run the same class files on any computer!
- JVM on every machine supports the same machine code
- Needs a different JVM on each platform
  - In contrast, traditionally C and C++ programs must be compiled separately to run on Solaris, Linux, Windows, Mac OS, etc.
- Microsoft's .NET platform is a similar idea to the JVM but more powerful and multi-language.

Java Run-Time

- The java command
  - Loads the java run-time environment
  - Loads the class file associated with the argument specified
  - Interprets the bytecode for the "main" method in that class

- Often you will see:

  Error: Could not find or load main class HelloWorld

  - Either Java couldn't find the HelloWorld.class file
  - Or it couldn't find the "main" method in HelloWorld.class

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 the Cars class, drivers may be in the Drivers class
  - Multiple source files: Cars.java and Drivers.java
  - Multiple bytecode files: Cars.class and Drivers.class
- Packages in Java enable packaging classes to make a single program or integrated set of programs

Typical Packages

- Source (.java) files in a package are kept in a single directory
- Bytecode (.class) files in a package are kept in a single directory

- Package explicitly identified in each source file via a "package" statement at the beginning of the .java file
  - Syntax: `package` *name*;
  - For instance: `package pck;`
  - By convention, package names start with a lower case letter

CLASSPATH

- Java utilities look for bytecode .class files using the "class path"
  - Compiler finds definitions for other classes using the class path
  - Run time 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 compiler (javac) package rules

- When you compile your code, by default the .class bytecode file is written to the current directory
  - whether your code is in a package or not
- If you use the -d flag with javac, the compiler will write the .class file to:
  - The directory specified by –d if the code is not in a package
  - A package sub-directory of the directory specified by –d if the code is in a package

javac examples

No Package keyword

- javac HelloWorld.java
  Writes: ./HelloWorld.class
- javac –d ~/cs140 HelloWorld.java
  writes ~/cs140/HelloWorld.class
- javac –d . HelloWorld.java
  writes ./HelloWorld.class

"package pck;"

- javac HelloWorld.java
  Writes: ./HelloWorld.class
- javac –d ~/cs140 HelloWorld.java
  Writes: ~/cs140/pck/HelloWorld.class
- javac –d . HelloWorld.java
  Writes: ./pck/HelloWorld.class

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 pck.HelloWorld”
- When running the “java” command, use the “-cp” flag to specify where to look for the package sub-directory
  - e.g. “java –cp ~/cs140 pck.HelloWorld”
- If you make a mistake, you will get:
  Error: Could not find or load main class HelloWorld

In CS-140

- Use a package for each lab and assignment: lab01
- Make a sub-directory with the package name:
  mkdir ~/cs140/lab01
- Make the package subdirectory the current directory:
  cd ~/cs140/lab01
- Edit .java files in the current directory: gedit HelloWorld.java&
- Add package keyword to each .java file: “package lab01;”
- Compile using –d . : javac –d . HelloWorld.java
- Run specifying package: java –cp . lab01.HelloWorld

Resulting Directory Structure