COP 2250 – Introduction to Java Programming

**Chapter 1**

The Java content starts on in Section 1.5 on page 10. Start there and continue through to the end of the chapter, but ignore Section 1.11. You will be using Eclipse, not NetBeans.

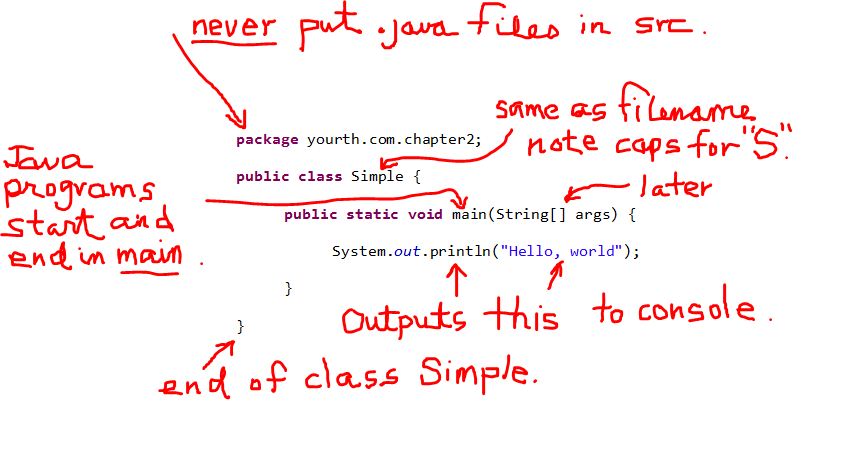
* Java was created by a team led by **James Gosling** at Sun Microsystems (now in Oracle).
* Java programs compile into **byte code**, not machine language. To run a Java program, a computer needs a Java Virtual Machine (**JVM**) that can interpret the byte code.
* A Java program written on a Mac can run on a PC or Linux computer if it has a JVM.

Some Key Points

* **Comments** start with //, on a new line or at the end of a line.
* Java is **case-sensitive**.
* Java statements end with a **semicolon**.
* To start a new Java program, you pick a name for a **class** (1st character in upper case).
* If a class name consists of several words, capitalize the first character of each word.
* This class name must be identical to the name of the **.java** source code file.
* Only one class can be **public** (means access is unrestricted) in a .java file.
* For now, all Java programs will have a **main method**. This makes the program executable. Later, we will create classes to model our data. They won’t have a main.

Section 1.7

* This section covers the parts of a basic Java program with several examples.
* NOTE: the package statement must be at the top of every program file.



Section 1.8

* This section explains how Java source code is compiled into byte code.
* Figure 1.6 on page 16 depicts this well.

Section 1.9

* Programming style and documentation are important.
* Note well the indentation and block styles on page 19**. Us**e the end-of-line style.
* Eclipse (covered below) can quickly help you to create nicely styled, and easily to read, source code files. Use Source -> Format and Source -> Correct Indentation.

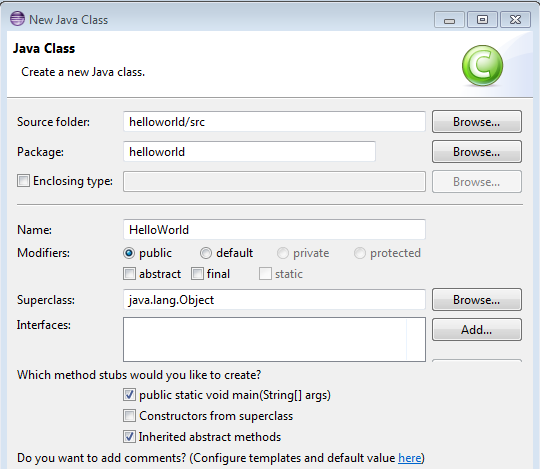
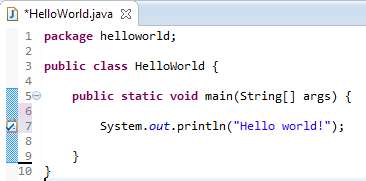
Section 1.10

* Note the three types of programming errors.
* Eclipse will detect and identify your syntax errors and even suggest fixes.

Section 1.12 - Eclipse (most used Java IDE)

* When you start Eclipse, you will be prompted to specify a “**workspace**”. This is just a folder where your Java will be saved. I suggest you create a folder on your desktop with your name on it, like smithjava. Use this workspace throughout the course.
* When the IDE appears, it will be in the **Java perspective,** a set of window panels best for Java. Eclipse can be used for other programming languages.
* Eclipse programs are always part of a **project** that can contain many programs.

Starting a new Eclipse Project and Program

1. Select **File > New > Java Project**
2. This opens a “New Java Project” dialog window.
3. Enter the project name in the box at the top (all lower case). Change nothing else, then click Finish, or Next > Finish if you like mouse-clicking. ☺
4. The project will appear in the **Package Explorer** window on the left. Expand it.
5. Don’t open or mess with the JRE System Library.
6. Right-click on the project name and select **New > Package** from the popup. You can alternatively click the **package icon**  on the toolbar. That will suggest a package name that is identical to the project name, and that is okay.
7. Don’t create your Java programs in the default **src** folder as is shown in the book. Always create a package. A package is a folder in your project.
8. To create a program, right-click on the package icon in the Package Explorer window and then select New > Class. You can also click the class icon  on the toolbar.
9. The “New Java Class” dialog will open. Pick a class name (HelloWorld) and check the public static void main checkbox as shown. Click Finish.
10. By convention, Java class names, and the first character of each word in multiple word class names, begin with an **upper case character**. This is important.
11. Eclipse will then display a skeletal source code file named **HelloWorld.java** in the large editor window.
12. Start Java coding inside the main method as shown here.
13. To run a program in Eclipse, click the arrow icon  on the toolbar.

The print and println Methods

* These are your console output methods. In Eclipse, they generate output in the Output window below the editor window in Eclipse.
* The println() method outputs its argument and causes a new line.
* The print() method does not make a new line, so subsequent output is on the same line.

**Chapter 2**

Keyboard Input from Console

* The **Scanner** class can be used for keyboard input and interactive Java programs.
* Scanner is not in the default **java.lang** package so you have to **import** it from the **java.util** package.
* All import statements follow the package statement and precede the public class header line.
* The keyboard in Java is the **in property** of the System class, **System.in.**
* To use the Scanner class, you create an instance of Scanner and assign its address to System.in.

Scanner keyboard = new Scanner(System.in);

* Eclipse will generate a warning for the line above. To eliminate the warning, enter the following statement when done with inputs:

keyboard.close();

* Once the Scanner object has been instantiated, it can be used to run any one of many methods of the Scanner class.
* Commonly used methods for numeric input are shown in Table 2-2 on page 45. The table shows that each numeric data type has a corresponding Scanner method.

Identifiers

* These are names that you select to refer to your data.
* Always choose identifiers that…um…***identify*** what they are storing in memory.
* For example, weekly**P**ay is a better choice than x for a double type that would store a person’s weekly pay. Note the “camel case” style of weeklyPay is common in Java.
* There are strict rules for choosing your identifiers in Java. See the bullet list and notes on page 40.

Class Names

* By convention, class names start with an upper case character.
* Multiple word class names use upper case for the 1st character in each word.

Some good and not good class names:

AcmeEmployee good

MonthlyRateOfInterest good

Myfirstprogram not good use MyFirstProgram

Variables and Literals

* Java is a strongly typed language. That means that you must declare the type of data that you wish to store in the variable when you are creating the variable in code.
* An integer like 10 or -6 is stored in the **int** type. A number with decimals is stored by default as a **double** type.
* Variables must be declared before they are used. The value in the variable can be optionally assigned when it is declared.

Constants with final

* The final keyword is used to assign a constant.
* The type and value of the constant must be specified when creating a constant.
* By convention, constants in Java are written in all **upper case, with underscores between each word** of a multi-word name.

final double STATE\_SALES\_TAX = 0.06;

final int MAX\_SIZE\_OF\_CLASS = 24;

Numeric (Primitive) Data Types

* In Java, every piece of data you use in a program has a data type.
* Classes are **object types, aka reference types**.
* Numbers, however, are usually stored as non-object **primitive types**.
* See Table 2-1 on page 45 for a list of common Java primitive types for numbers.
* Note also the amount of memory each type needs and the range of possible values that each type can store.
* Since they are the defaults, we will use type **int** for integers and type **double** for floating point numbers, but if memory space is a concern, coders always use the most suitable type.
* Note the Scanner class methods for numeric input in Table 2.2 on page 45.

Arithmetic Operators

* See Table 2-3 on page 46.
* Note that when an integer is divided, any decimal part in the quotient is dropped.
* NOTE: when using the **/** operator with int types, any decimal fraction part is lost.

int num = 25 / 7; // stores 3 in the variable num.

* NOTE: the **%** operator (remainder or **modulus**) yields the remainder after a division.

int num = 25 % 7; // stores 4 in num.

The Math Class

* This class is always available to you. It’s in the default java.lang package.
* The Math class has many useful methods that you can employ.
* Math.pow() is used for exponentiation, and Math.sqrt() for finding square roots.

Floating Point Types

* Note that, as mentioned previously, floating point values are doubles by default.
* If the literal is followed by **F or f**, a float type can be stored.

Operator Precedence

* This is just the same as in math class. “Please Excuse My Dear Aunt Sally” is in effect.

The Combined (Augmented) Assignment Operators

See Table 2-4 on page 54.

Increment and Decrement Operators

* See Table 2-5 and examples on page 55.

Conversion Between Numeric Data Types

* Examine the pages 56-58.
* The primitive data types are ranked on byte size.
* Java will convert a lower ranked type to a higher ranked type without a problem.
* But a higher ranked type cannot be automagically converted to a lower ranked type. In a case like this, a **cast** is required.
* Specify the new type for the cast inside ( )

Casting Example

double num1 = 12.5;

int num2 = num1; ///// error: type mismatch

int num2 = (int)num2; ///// okay after casting  
Primitive Types and Class (Reference) Types

* The value in a variable of primitive type is the actual value assigned to it.
* The value stored in a variable of an object type is a memory address. The variable doesn’t store the object, it stores a reference to the object. Objects are also called “reference types” for this reason.

Common Errors to Avoid

* See pages 65-67.