Java Chapter 2 Part 1

* Output, Variables, Literals, Data Types, Arithmetic
* CIS 255 • Shelby-Hoover Campus

The Structure of a Java Program

* Every Java program will begin with the same basic structure
* Becoming familiar with this structure through practice over time will make it easier to focus on the elements that vary from one program to the next
* Note that Java is case sensitive
  + Key words must be typed exactly as required by the language
  + Variable names must be typed consistently

Import Statements

* Your code must indicate what optional functionality you plan to include in your program using an import statement
* Import statements generally take this form: import java.package.class;
* Example: the Scanner class for console input: import java.util.Scanner;
* Example: the JOptionPane class for dialogs: import javax.swing.JOptionPane;

The Class

* All Java programs are enclosed in a class (though not all Java programs are object-oriented)
* After the import statements, add the **class header**:  
    
   public class ClassName
* The name you give your class must exactly match the file name (spelling and case) without .java
* The key word publicis an **access specifier** meaning that there are no restrictions as to what other programs can use code in this class
* The class header should be followed by an opening curly bracket / brace, and there should be a corresponding closing bracket at the end

The Main Method

* Java applications (but not applets) require the presence of a **main method**:  
    
   public static void main(String[] args)
* A staticmethod is a procedural method; it does not require an object
* The key word voidindicates that the method will not return any information to a higher-level method
* The contents in the parentheses are what information may be sent to the method (to be explained in chapter 5)
* Another set of curly brackets surrounds the contents of main

The Structure as a Whole

🡨 import statements

public class ClassName

{

public static void main(String[] args)

{

🡨 Your code goes here

}

}

Compiling and Running

* If you don’t have a code editor with direct access to the JDK tools, you can use a command prompt to compile and run your Java applications
* The command to compile a Java source code program is javac:  
    
   javac ClassName.java
* After successfully compiling a program, the byte code version of the program appears in the file ClassName.class
* To run the compiled program, use the command java with the class name (with no filename extension):  
    
   java ClassName

Comments

* To add details for the reader of your code that the compiler should ignore, use **comments**
* **Line Comments** begin with two forward slashes (//) and end at the end of the line of code (Comment1.java, Code Listing 2-24)
* **Block Comments** begin with /\* and end with \*/ on the same line or a different line (Comment2.java, Code Listing 2-25)
* **Javadoc Comments** are similar to block comments but contain special syntax for the javadoc command that generates HTML documentation about your class; these begin with /\*\* and end with \*/ (Comment3.java, Code Listing 2-26)

Punctuation

* Grouping punctuation must be added in pairs
  + Parentheses ( )
  + Curly Brackets { }
  + Quotation Marks " "
  + Square Brackets [ ]
* Every statement (a single command to execute) ends with a semicolon, but comments, headers, and bracket lines generally do not

Console Output

* The System.out object represents output to the console window
* Two methods for output:
  + System.out.print(itemsToDisplay);  
    displays the contents in parentheses, remains on the same line
  + System.out.println(itemsToDisplay);  
    displays the contents in parentheses, starts a new line below
* A value written in parentheses in a method call is known as an **argument**
* Strings of text (also known as **string literals**) to be displayed should be placed in quotation marks within the parentheses:  
  System.out.println("This is my text.");
* Example: TwoLines.java (Code Listing 2-2)
* Example: GreatFun.java (Code Listing 2-3)

Escape Sequences

* Special characters that otherwise can’t be typed are written as **escape sequences** inserted within quotation marks (by themselves or as part of a longer string literal)
* Each escape sequence starts with the backslash (\), not to be confused with a forward slash (/); one or more **control characters** follow the backslash
  + \n 🡨 new line within a string
  + \t 🡨 horizontal tab to the next tab stop
  + \b 🡨 backspace (back to the previous character)
  + \r 🡨 return (to the beginning of the current line)
  + \\ 🡨 printable backslash
  + \' 🡨 printable single quote
  + \" 🡨 printable double quote
* Example: Unruly.java (Code Listing 2-4) uses ordinary print method calls with no separation; Adjusted.java (Code Listing 2-5) uses \n to break the items onto multiple lines; and Tabs.java (Code Listing 2-6) uses \t to indent the three items being displayed

Variables

* A **variable** is a named memory location that holds various values during the execution of a program
* Each statement uses the current value of the variable, not necessarily the original
* Variables must be **declared** before use
  + The declaration statement indicates the type and name to be used
  + The data type indicates how much memory should be set aside for the variable

Declaration Statements

* Declaring a single variable:
  + Pattern: typeName variableName;
  + Example: int numOfStudents;
* Declaring multiple variables of the same type
  + Pattern: typeName var1, var2, var3;
  + Example: int courseNum, capacity;
* Each declaration statement must end with a semicolon
* Don’t separate declarations for different variable types with a comma!

Variable Output

* When displaying the value in a variable, don’t put the variable name in quotation marks:  
    
  System.out.println(variableName);
* A program may display a variable’s value with literal strings of text as a label by separating the different portions of the output statement with a plus sign (don’t forget to include spaces inside the quotation marks before and after the variable):  
    
  System.out.println("I am " + age + " years old.");
* Example: Literals.java (Code Listing 2-9)

Rules for Identifiers

* Programmer-defined names must follow specific guidelines
  + The first character of an identifier must be a letter, an underscore, or (rarely) a dollar sign
  + Other characters can be any of the above or a digit (but an identifier cannot start with a digit)
  + Identifiers cannot include spaces
  + Be consistent with case (year vs. Year vs. YEAR)
  + Class names typically start with an uppercase letter, whereas variable names start with a lowercase letter
  + Use underscores or camel casing (capitalizing first letter of the second and later words) in multiword variable names
  + Key words (reserved words) are off limits as identifiers

Primitive Data Types (1)

* The names of the primitive data types begin with lowercase letters
* Variables of these types store simple values in a fixed memory space (no methods)
* Integer (whole number) Types:
  + byte: consumes 1 byte, ranges from -128 to +127
  + short: consumes 2 bytes, ranges from -32,768 to +32,767
  + int: consumes 4 bytes, ranges from -2,147,483,648 to +2,147,483,647
  + long: consumes 8 bytes, up to 19 digits (+ / -)
  + Integer literals are assumed to be of type int; add L to the end of the literal to force Java to treat it as a long

Primitive Data Types (2)

* Floating-Point (real number / decimal) Types:
  + float: consumes 4 bytes, ranges from ±3.4x10-38 to ±3.4x10-38 with 7 digits of accuracy
  + double: consumes 8 bytes, ranges from ±1.7x10-308 to ±1.7x10308 with 15 digits of accuracy
  + Floating-point literals are assumed to be of type double; add F to the end of the literal to force Java to treat it as a float
  + Normally written with an optional sign, digits, and a decimal point (no commas)
  + Can also be written in **scientific / E notation**:
    - 247.91 = 2.4791E2 (2.4791 x 102)
    - 0.00072 = 7.2E-4 (7.2 x 10-4)

Primitive Data Types (3)

* boolean: stores a value of true or false
  + Don’t put quotation marks around these values
  + Chapter 3 covers how to write Boolean expressions (usually comparisons) whose results can be stored in a boolean variable
* char: stores a single Unicode (two-byte) character in single quotes
  + Unicode is a superset of ASCII (one-byte)
  + Supports international characters

Assignment Statements

* A single equal sign (the assignment operator) assigns the value on the right hand side to the variable on the left hand side
  + This is **not** the operator for comparison!
  + The expression on the right hand side can be a literal value, a variable of a compatible type, or an expression
  + Examples:
    - age = 22;
    - newRate = oldRate + 17;
    - distance = rate \* time;
  + Don’t reverse the left and right sides!
  + Don’t forget the semicolon at the end!
* A program may assign a value to a variable when declaring it
  + This is called **initializing** the variable
  + Example: int age = 22;

Assignment Compatibility

* The declared data type of a variable determines what values can be assigned to it
* Numeric literal values should not be placed in quotes:  
    
  age = 25;
* Character literal values should be placed in single quotes:  
    
  gender = 'M';
* String literal values should be placed in double quotes:  
    
  name = "Priscilla";

Arithmetic Operators

* Unary operators require only one operand (e.g., negation)
* Binary operators require two operands
  + Addition: +
  + Subtraction: -
  + Multiplication: \*
  + Division: /
  + Modulus: %

The Modulus Operator

* When two integers are used in a division operation, the remainder is lost
* The modulus operator can be used to obtain the remainder in integer division
  + leftOver = 9 % 5; /\* leftOver is 4 (5 goes into 9 once with 4 left over) \*/
  + remainder = 12 % 3; /\* remainder is 0 (3 goes into 12 four times with 0 left over) \*/
  + discard = 4 % 10; /\* discard is 4 (10 doesn’t go into 4, so 4 is left over) \*/
* This operator can be used to determine when a number is a multiple of another number
  + if a % 2 is zero, then a is even
  + if a % 2 is one, then a is odd

Arithmetic Guidelines

* Integer division always **truncates** the result
  + The decimal component is lost; no rounding up
  + Use at least one floating-point operand to prevent this
* Operators have varying levels of precedence
  + Unary negation is the highest (evaluated first)
  + Multiplication, Division, Modulus
  + Addition, Subtraction are lowest (evaluated last)
* Operations of the same level are evaluated left to right
* Parentheses may be used to override precedence

Math Methods

* More complex mathematical operations require the use of a method
  + Math.pow(base, exponent) raises the base to the exponent
  + Math.sqrt(operand) obtains the square root of the operand
* The values determined by these methods should be assigned to a variable or displayed:  
    
  double rootOfEight = Math.sqrt(8);  
    
  System.out.println(Math.pow(1.05,10));

Combined Assignment

* Frequently a programmer needs to modify the value of an existing variable:  
    
   count = count + 1; /\* adds 1 to the existing value of count \*/
* Java contains several combined assignment operators to shorten this action
  + Operators: +=, -=, \*=, /=, %=
  + Example: count += 1;
  + Example: price -= discount;

Mixing Types in an Expression

* Operands are considered in pairs
* If an expression involves operands of differing types, the operand of the lowest-ranking type will be converted to the highest-ranking type
  + If both are integer types, the largest capacity type will be chosen
  + If both are floating-point types, double will be chosen
  + If the expression mixes an integer type and a floating-point type, the floating-point type will be used

Type Conversions

* Widening conversions (assigning a value of a low-rank type to a variable of a high-rank type) are automatic
* Narrowing conversions (assigning a high-rank type value to a low-rank type variable) must be done explicitly via **casting**
  + Place the new type in parentheses before the literal or variable to convert
  + Example: if income is a double and adjusted is an int, you could truncate income by writing the following expression:  
      
    adjusted = (int) income;
* Casting is useful to force floating-point division with two integer operands (cast either or both operands to a floating-point type):  
    
   piecesPerPerson = (double) pieces / people;
* Be careful not to cast entire expressions if you are trying to change the type of arithmetic; the cast will be performed after the arithmetic is complete:  
    
   piecesPerPerson = (double) (pieces / people);

Constants

* Normally, the value in a named memory location can change throughout a program’s execution
* You may want to give a name to a value that cannot change
* The keyword final may be added to the beginning of what would normally be a variable declaration to instead make it a named constant
  + Constants must be initialized when declared
  + Usually named with all uppercase letters
  + Example:   
      
    final double CREDIT\_HOURS = 3.0;
* Some constants are built into Java (e.g., Math.PI)

Learning about Java

* The Java web site, java.sun.com, contains extensive documentation on the Java language
* The Java **APIs (application programming interfaces)** detail each class and what operations can be performed with it
* Example: The System class