Java Chapter 2 Part 2

* Strings, Style, Console Input, Dialogs
* CIS 255 • Shelby-Hoover Campus

String

* String is a **class type**, not a primitive type
  + Class type names are capitalized
  + Objects of class types support methods
  + Variables hold the addresses of objects rather than the objects themselves (varying sizes)
* Declare, initialize, and assign values to String variables like primitives
  + Declaration: String firstName;
  + Initialization: String lastName = "Smith";
  + Assignment: firstName = "Priscilla";

Class Methods

* To invoke a method on an object, you use the dot notation / dot operator
* The general form is  
    
  variableName.methodName(arguments)
* Example: String length (no arguments)  
    
  String name = "Priscilla Smith";  
  System.out.println("name has " + name.length() + " characters.");
* Output would be   
    
  name has 15 characters.  
    
  (the space is included)

Other String Methods

* If a method **returns** a value, the value must be assigned to a variable or displayed
* charAt(location) returns a char with the character at that location (first location is 0)
* toLowerCase() returns a String with all letters converted to lowercase
* toUpperCase() returns a String with all letters converted to uppercase
* None of these methods modify the existing String – they simply return a new value
* Example: StringMethods.java (Code Listing 2-22)

Identifier Scope

* When you declare a variable or constant within a particular block (a set of statements in curly braces; e.g., the main method) , that identifier has **scope** to the end of that block (it exists and has the specified meaning in that block)
* A variable declared within a block is said to be a **local variable**
* The declaration statement must appear before the first statement that uses the variable, or the program will not compile (Scope.java, Code Listing 2-23)
* Only one declaration for each identifier is allowed per scope (don’t use the same variable name in another declaration in the same scope)

Programming Style

* The compiler is not concerned with the appearance of your code
* Good programming style helps you and other programmers to understand your code
  + Indentation helps the reader identify code blocks and multiline statements
  + Comments should be added to explain non-obvious details (units, formulas, etc.)
  + An identifier should explain the purpose of the item it describes
* Compare Compact.java (Code Listing 2-27) to Readable.java (Code Listing 2-28)

Console Input

* Just as the System.out object provides output access to the console, System.in provides a means to gather input from the keyboard
* The methods to manipulate System.in to gather input are not straightforward (gathering bytes rather than values of the different types)
* Many programmers use other means to process the input from System.in into the various types

The Scanner Class

* Another class, Scanner, provides the methods needed to gather input of the various types from the keyboard (this chapter) or from a file (chapter 4)
* This class requires an import statement:  
    
  import java.util.Scanner;
* A Scanner object must be declared and associated with the System.in object:  
    
  Scanner keyboard = new Scanner(System.in);

Using a Scanner Object

* The variable for a Scanner object may have any name (the textbook usually uses keyboard)
* Use the dot notation to obtain values of various types
  + Numbers:   
      
    nextByte() nextInt() nextFloat()  
    nextShort() nextLong() nextDouble()
  + Strings: nextLine()
  + To obtain a single character, read the input as a String and use charAt(0) to extract the first character (there is no nextChar())
* Example:   
    
  value = keyboard.nextInt();

Mixing Numeric and Line Input

* When a user enters a value via the console, the program does not receive the value until the user presses the enter key
* The numeric input methods capture the number but leave the new line (the result of pressing the enter key) in the console
* If the next method call after this is a call to nextLine(), the method will capture the end of the current line rather than waiting for the user to enter an additional line
* Example: InputProblem.java (Code Listing 2-30)
  + The user is prompted to enter his / her age, annual income, and name
  + Because the input statement for the name (nextLine()) follows the input statement for the income (nextDouble()), the program captures the pressing of the enter key as the “next line”

Adding a “Dummy” Input

* To transition from reading a numeric value to reading a line of text, use an extra call to nextLine() that isn’t assigned to a variable:  
    
  keyboard.nextLine();
* This extra “dummy” call consumes the pressing of the enter key and prepares the Scanner object to read the next line of text
* The call to nextLine() that follows will then properly obtain the text that the user enters
* Example: CorrectedInputProblem.java (Code Listing 2-31)

Dialog Boxes

* GUI programming tends to be complex (creating the GUI elements, placing them in the proper location in the window)
* Dialog boxes are a simple way to communicate with the user in a graphical manner without complex programming
* The dialog box methods are part of the JOptionPane class and require an import statement:  
    
  import javax.swing.JOptionPane;
* A program with dialog boxes should end with the following statement to force it to terminate properly after the last dialog box is dismissed:  
    
  System.exit(0);

Message Dialogs

* A message dialog is a simple dialog that displays text and an “OK” button
* Method call structure:  
    
  JOptionPane.showMessageDialog(null, "Stuff to be displayed");
* What you display in a dialog may be a mixture of literal text and variables (written with + between values as in the System.out methods)
* Don’t miss the null argument before the values to display: this instructs Java to center the dialog on the screen rather than over another GUI element

Input Dialogs

* The method to capture input via a dialog displays a prompt (instructional message) and returns the value the user enters in a text box as a String:  
    
  String txt;  
  txt = JOptionPane.showInputDialog("Please enter some text.");
* The showInputDialog method does not require the extra argument null
* Example: NamesDialog.java (Code Listing 2-32)

Numeric Data in Input Dialogs

* The showInputDialog method always returns a String, even if the user enters an integer or a real number
* In order to convert the String value to a numeric type, use a special method from a wrapper class called a parse method
  + A wrapper class is a class type that corresponds to one of the primitive types
  + Wrapper classes provide special processing methods for data of the primitive types

The Parse Methods (Table 2-18)

* byte: Byte.parseByte
* double: Double.parseDouble
* float: Float.parseFloat
* int: Integer.parseInt
* long: Long.parseLong
* short: Short.parseShort
* char: Use the charAt(0) method call to obtain the first character from the String
* Parsing example: PayrollDialog.java (Code Listing 2-33)  
    
  inputString = JOptionPane.showInputDialog(  
   "What is your hourly pay rate?");  
  payRate = Double.parseDouble(inputString);

Reminders for Chapter 2

* Make sure that every opening bracket has a corresponding closing bracket in the right place
* Use the exact spelling and case of Java keywords
* Don’t use a keyword as an identifier
* Spell identifiers consistently (including case)
* Don’t put a space in an identifier (use an underscore or camel casing)
* End each statement with a semicolon
* Add an L to an integer literal to treat it as a long; add F to a floating-point literal to treat it as a float
* Don’t use any extra punctuation (dollar signs, commas) in numeric literals

More Reminders

* Integer division always truncates the result
* Arithmetic operations have precedence rules; use parentheses to override these
* Combined assignment operators and escape sequences should not contain spaces
* Variables can only be assigned values compatible with their types
* Block and Javadoc comments must end with \*/
* Scanner and JOptionPane require import statements
* A program with dialog boxes requires a call to System.exit(0);
* Message dialogs require an extra argument of null
* Numeric values from an input dialog must be parsed