Java Chapter 5 Part 2

* Methods: Return Values, Design, and Overloading
* CIS 255 • Shelby-Hoover Campus

Review of Lecture 8

* Write method definitions inside the class brackets but outside any other method definition
  + Header: access level (public or private), the key word static (for non-OO methods), return type (void or a data type), method name, parentheses (may include parameter variables), **no semicolon**
  + Body: enclosed in curly brackets
* A call to a method may be placed in the body of any other method
  + When calling a void method, just list the method name, parentheses, and a semicolon
  + The argument list in the parentheses must match the parameter list in the method header (types, how many)
  + A method may be called multiple times within a program

Value-Returning Methods

* The body of a value-returning method must contain a **return statement**
  + return *some value, variable, or expression that matches the return type*;
  + Most value-returning methods have a single return statement as the last statement in the method body
  + A void method may also have a plain return statement (with no value), but this is optional

Value-Returning Method Examples

* Returning a literal value of type double:  
    
  public static double iNeedPi()  
  {  
   return 3.14159;  
  }
* Returning a calculated value of type int:  
    
  public static int sumOfThreeItems(int item1, int item2, int item3)  
  {  
   return (item1 + item2 + item3);  
  }

Calling a Value-Returning Method

* If a method returns a value, that value must be handled in the method call  
  + Assignment: mySum = sumOfThreeItems(3, 7, 12);
  + Arithmetic: area = iNeedPi() \* radius \* radius;
  + Output: System.out.println("pi equals " + iNeedPi());
* When you place a method call inside another statement, the outer statement uses the value returned by the method
* Don’t place a void method call inside another statement!

Value-Returning Method Call Example

* In this program, the method rollTwoDice returns a value of type int back to main, where the value is assigned to the variable rSum and displayed:  
    
  import java.util.Random;  
    
  public class DiceRoll  
  {  
   public static void main(String[] args)  
   {  
   int rSum;  
    
   rSum = rollTwoDice();  
   System.out.println("You rolled " + rSum);  
   }  
    
   public static int rollTwoDice()  
   {  
   int random1, random2, sum;  
   Random generator = new Random();  
    
   random1 = generator.nextInt(5) + 1;  
   random2 = generator.nextInt(5) + 1;  
   sum = random1 + random2;  
    
   return sum;  
   }  
  }
* Another example: using methods for obtaining input from the user, converting the input to a different unit measurement, and displaying the results (CupConverter.java, Code Listing 5-10)

Javadoc Notation for Return

* As mentioned in Lecture 8, a method is often preceded by a javadoc comment with special tags letting the javadoc tool know what kind of documentation to generate about the method
  + The first segment of the comment is a general description of the method’s behavior
  + For each parameter, include a line with the tag @param, the parameter name, and a general description of its purpose
* If the method returns a value, include a line with @return and a description of the return value

Javadoc Comment Example

* The cupsToOunces method in CupConverter.java (Code Listing 5-10):  
    
  /\*\*  
   The cupsToOunces method converts a number of  
   cups to fluid ounces, using the formula  
   1 cup = 8 fluid ounces.  
   @param numCups The number of cups to convert.  
   @return The number of ounces.  
  \*/  
    
  public static double cupsToOunces(double numCups)  
  {  
   return numCups \* 8.0;  
  }

Designing Methods

* When planning the logic of a program, you may start with a single listing of all of the code required before separating the code into methods
  + Remember that each method only has direct access to its own variables
  + When two methods need the same information, use arguments to parameters and return values
* Don’t use the same name for a variable and a method (this confuses the compiler)
* If multiple methods are using an input or output file, the PrintWriter and / or Scanner objects must be passed from one method to the next
  + Open the file in the first method to use it; close it in the last
  + Any method that uses these objects, and any method that calls one of these methods, needs the throws clause in its header
  + Example: SalesReport.java (Code Listing 5-12)

Overloading Method Names

* In any given scope, there can only be one definition for an identifier (variable, constant)
* Each method definition must have a unique **signature** (the combination of the method name and parameter types / number / order)
* If a method name applies equally to more than one method definition, a program can have multiple method definitions with the same method name as long as the parameter lists differ
* Using the same method name for multiple method definitions in a single program is known as **overloading** (discussed in chapter 6)

Overloading Example

* These methods could coexist in the same program:  
    
  public static void displayBill(double bal)  
  {  
   System.out.println("Thank you for your order.");  
   System.out.printf("Your balance is $%,.2f\n", bal);  
  }  
    
  public static void displayBill(double bal, double discRate)  
  {  
   double discount, newBal;  
    
   discount = bal \* discRate;  
   newBal = bal – discount;  
    
   System.out.println("Thank you for your order.");  
   System.out.printf("Your balance is $%,.2f\n", newBal);  
  }  
    
  public static void displayBill(double bal, String message)  
  {  
   System.out.println("Thank you for your order.");  
   System.out.println(message);  
   System.out.printf("Your balance is $%,.2f\n", bal);  
  }

Calling an Overloaded Method

* The compiler determines which definition to use based on matching the argument list in the method call to the parameter list in the method definition
* Examples:
  + Calling the definition with a single parameter:  
      
    displayBill(myBalance);
  + Calling the definition with two double parameters:  
      
    displayBill(myBalance, 0.10);
  + Calling the definition with a double parameter and a String parameter:  
      
    displayBill(myBalance, "We appreciate you.");

Overloading Guidelines

* Overloaded methods may have the same or different return types; the return type is not a factor in overloading
* Changing the parameter names alone is not sufficient: you must change the number of parameters, their types, and / or the order in which they are listed
* These method definitions could **not** coexist in the same program:  
    
  public static void displayBill(double bal, double discRate)  
  {  
   double discount, newBal;  
    
   discount = bal \* discRate;  
   newBal = bal – discount;  
    
   System.out.println("Thank you for your order.");  
   System.out.printf("Your balance is $%,.2f\n", newBal);   
  }  
    
  public static void displayBill(double bal, double discAmt)  
  {  
   double newBal = bal – discAmt;  
    
   System.out.println("Thank you for your order.");  
   System.out.printf("Your balance is $%,.2f\n", newBal);   
  }

Reminders for Chapter 5

* Writing a method definition:
  + Each method definition should be inside the class brackets but not inside the brackets of another method
  + The header should include the access level, the keyword static, the return type, the name of the method, and a set of parentheses
  + If a method needs to send a value back to the calling method, the return type should match that value’s data type; if not, use void
  + If a method needs to receive values from the calling method, include a set of parameter variables (each with a name and type) inside the parentheses after the method name
  + Do not place a semicolon after the closing parenthesis of the method header (the same as the header of an if, else if, while, or for statement)
  + The body of one method (inside curly brackets) has access only to its own variables (parameters and other local variables)
  + If a method’s return type is anything other than void, it must have a return statement with a value that matches that type
  + void methods typically contain output statements to display results

More Reminders for Chapter 5

* Calling a method:
  + Include the method name and argument names / values, but do not include other keywords or types
  + A statement containing a method call should end with a semicolon
  + When calling a void method, no other syntax is required
  + When calling a value-returning method, place the call inside a statement that will use the return value
  + A method may be called inside any other method, and it may be called in a block within a method (an if statement, a loop, etc.)
* Parameters:
  + Primitive arguments are passed by value (parameter gets a copy)
  + Object arguments are passed by reference (parameter gets the original)
  + File-related objects should be passed to every method that needs them
  + Any method that uses a file-related object, or calls a method that uses a file-related object, needs the throws clause
* Overloading (writing multiple method definitions with the same method name) is allowed as long as the parameter lists are different