Chapter 5

Monday, August 28, 2017

5.1 Introduction to Methods

Method - bundle of statements that performs a particular task

2 types of methods:
- Void methods
  - Perform a task then terminate
    - `System.out.println("Hi");`
- Value returning methods
  - Perform a task then send a value back to the rest of the program
    - `int x = Integer.parseInt("100");`
    - `double y = Math.pow(2, 3);`

Why Write Methods?
- Functional Decomposition - (Divide and Conquer), breaks a problem down into small, manageable pieces.
- Reuse - Methods can be used from several locations in a program

Defining a Method
- Method definitions consist of a header and a body.

```
public static void displayMessage()
{
        System.out.println("Hello");
}
```

- Method header
  - Contains important information about the method
  - Method modifiers
    - Access Modifiers

	Class	Package	Subclass	World
public	y	y	y	y
protected	y	y	y	n
no access modifier	y	y	n	n
private	y	n	n	n

    - static - methods belongs to a class, not a specific object
      - Math.pow(2,3);
      - Math.sqrt(9);
  - Return type - void or the data type from a value-returning method
  - Method name - name that is descriptive of what the method does
  - Parentheses - contain nothing or a list of one or more variable declarations if the method is capable of receiving arguments

- Calling a Method
  - A method executes when it is called.
  - The main method is automatically called when a program starts, but other methods are executed by method call statements.

- Three types of method call statements:
Method name by itself	To call a method in the same class	max(num1, num2);
Reference to an object . method name	To call a method of the referenced object	yankees.getWins();
Class name (.) method name	To call a static method of a class	Math.sqrt(16.0);

Documenting a Method

- A method should be documented by writing comments that appear just before the method's definition
- The comments should provide a brief explanation of the purpose of the method
- Documentation comments begin with /** and end with */

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package basketballcalculator;

import java.util.Scanner;

/**
 *
 * @author student
 */
public class BasketballCalculator
{

   /**
    * @param args the command line arguments
    */
   public static void main(String[] args)
   {
         Scanner keyboard = new Scanner(System.in);
         System.out.println("Enter the number of free throws attempted: ");
         int x1 = keyboard.nextInt();
         System.out.println("Enter the number of free throws made: ");
         int x2 = keyboard.nextInt();
         double result = freeThrowPerc(x1, x2);
         System.out.printf("Free Throw %% is: %.2f\n", result);

   }

   /** This method calculates free throw percentage.
    *  @param freeThrowsTaken number of attempted free throws
    *  @param freeThrowsMade number of free throws made
    *  @return percentage of free throws that were made
    */

   public static double freeThrowPerc(int freeThrowsTaken, int freeThrowsMade)
{
   //declare the double that needs to be returned
   double rate;

   rate = 100.0 * freeThrowsMade / freeThrowsTaken;

   return rate;

   //   return 100.0 * freeThrowsMade / freeThrowsTaken;

}

}
```

5.2 Passing Arguments to a Method

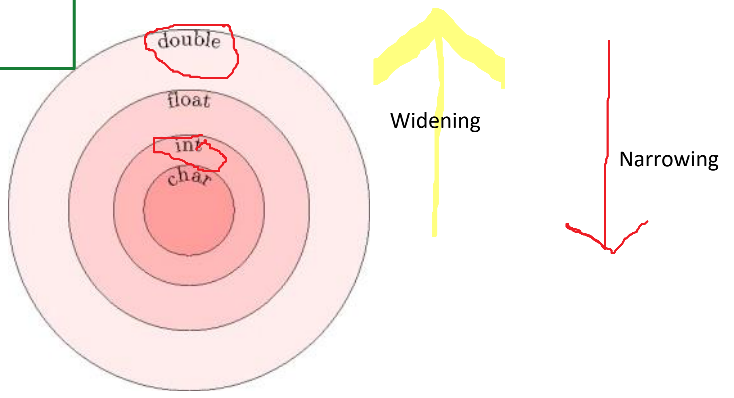
- Argument - Value that is passed into a method when it is called
    - showSum(18, 82);
    - fullName("Abraham", "Lincoln");
- Parameter - Variable declaration in the parentheses of the method definition
    - public static int sum(int num1, int num2)
    - public static String fullName(String firstName, String lastName)

- The data type of an argument must correspond to the parameter in the method definition

```
public static double showSum(double num1, double num2){
        return num1 + num2;
}
```

- Java will automatically perform widening conversions

```
showSum(18, 82);
```



Argument Promotion and Casting

Promotion (i.e. Widening)- converting an argument's value if possible to the type the method declared in its parameter. Java automatically performs promotion/ widening conversions.

Promotion/Widening Rules

Data type	Is automatically promoted/widened to:
double	None
float	double
long	float or double
int	long, float, or double
char	int, long, float, or double
short	int, long, float, or double

byte	short, int, long, float, or double
boolean	None (booleans are not considered numbers in Java)

Java does not automatically perform narrowing conversions.

Without casting, Compilation Error results
    e.g.passing a double into a method that expects an int // BAD

Why does Java automatically perform widening conversions but not narrowing conversions?
- Narrowing would result in a lossy conversion. The decimal part of a double is lost if we try to convert it to an int.

- In cases where information may be lost due to conversion, cast operator can force the conversion to occur (see Chapter 2, Conversion between Primitive Data Types).

5.3 More about Local Variables

- A local variable - variable declared inside a method
- Local variables only have scope from line in which they are declared to the } in the block of code in which they are declared
- Different methods can have local variables with the same name.
- Parameters are local variables.
- When the method ends, the local variables (including the parameter variables) are destroyed and any values are lost.
- Local variables are not automatically initialized with a default value and must be given a value before they can be used.

5.4 Returning a Value from a Method

```
/**
*    This method adds two numbers
*    @param num1 our first number to add
*    @param num2 our second number to add
*    @return returns the sum of the two integer values
*/

public static int sum(int num1, int num2){
     return num1 + num2;
}
```

- The return value has a data type of int
- Methods either return a value or are void.
- Value-returning methods return a value that corresponds to the data type in the
- method header, right before the method name
- @return tag
- Once the return statement runs, if there is one, the method ends and program control Is passed back to the place where the method was called

5.5 Problem Solving with Methods

Crate.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package crate;

/**
 *
 * @author student
 */
public class Crate
{
   //global constants
   final double COST =  .33;
   final double PRICE = .50;

   //Cost (to company):     .33 per cubic foot
   //Price (for customers): .50 per cubic foot

   //instance variables
   double length, width, height;

   //setters and getters
   //(mutators and accessor)
   public void setLength(double l){
      length = l;
   }

   public double getLength(){
      return length;
   }

   public void setWidth(double w){
      width = w;
   }

   public double getWidth(){
      return width;
   }

   public void setHeight(double h){
      height = h;
   }

   public double getHeight(){
     return height;
   }

   public double calculateVolume(){
     return getLength() * getWidth() * getHeight();
   }

   public double cost(){
      return COST * calculateVolume();
   }

   public double price(){
      return PRICE * calculateVolume();
   }

   public double profit(){
      return price() - cost();
   }

}
```

```
}
```

Driver class - class with a main method that creates objects of another class and calls their methods.

CrateTest.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package crate;

/**
 *
 * @author student
 */
public class CrateTest
{
   //6 x 10 x 12
   //14 x 3 x 5

   public static void main(String[] args)
   {
      //first crate:    6 x 10 x 12
      Crate ourCrate = new Crate();
      ourCrate.setLength(6);
      ourCrate.setWidth(10);
      ourCrate.setHeight(12);

      //second crate:    14 x 3 x 5
      Crate ourCrate2 = new Crate();
      ourCrate2.setLength(14);
      ourCrate2.setWidth(3);
      ourCrate2.setHeight(5);

      //Display volume, cost, price, and profit for first crate:
      System.out.println("First Crate:");
      System.out.printf("Dimensions are: %.2f  x %.2f  x %.2f \n",
               ourCrate.getLength(), ourCrate.getWidth(), ourCrate.getHeight());
      System.out.printf("Volume (sq. ft): %.2f%n", ourCrate.calculateVolume());
      System.out.printf("Cost: $%.2f%n", ourCrate.cost());
      System.out.printf("Price: $%.2f%n", ourCrate.price());
      System.out.printf("Profit: $%.2f%n", ourCrate.profit());

      //Display volume, cost, price, and profit for second crate:
      System.out.println("Second Crate:");
      System.out.printf("Dimensions are: %.2f  x %.2f  x %.2f \n",
              ourCrate2.getLength(), ourCrate2.getWidth(), ourCrate2.getHeight());
      System.out.printf("Volume (sq. ft): %.2f%n", ourCrate2.calculateVolume());
      System.out.printf("Cost: $%.2f%n", ourCrate2.cost());
      System.out.printf("Price: $%.2f%n", ourCrate2.price());
      System.out.printf("Profit: $%.2f%n", ourCrate2.profit());


   }
}
```

```
    }
}
```

Output:

```
First Crate:
Dimensions are: 6.00  x 10.00  x 12.00
Volume (sq. ft): 720.00
Cost: $237.60
Price: $360.00
Profit: $122.40
Second Crate:
Dimensions are: 14.00  x 3.00  x 5.00
Volume (sq. ft): 210.00
Cost: $69.30
Price: $105.00
Profit: $35.70
```

If a method calls another method that has a throws clause in its header,
then the calling method should have the same throws clause *(or as we'll see later, can handle the exception)*

Below, the main method calls a constructor for the PrintWriter and File. These both can throw IOExceptions, so include throws IOException in the header:

```
package throwsclause;

import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class FileInputOutput
{

   public static void main(String[] args) throws FileNotFoundException
   {
      PrintWriter output = new PrintWriter("Concatenated.txt");

      //Set up a Scanner object and associate it with FirstNames.txt
      Scanner infile = new Scanner(new File("FirstNames.txt"));
      Scanner infile2 = new Scanner(new File("LastInitials.txt"));

      while (infile.hasNextLine())
      {
         String line1 = infile.nextLine();
         String line2 = infile2.nextLine();

         output.println(line1 + " " + line2);
     }

     infile.close();
     infile2.close();
    output.close();
   }

}
```

Method-Call Stack

Pushing - placing an item at the top of a stack
Popping - removing an item from the top of a stack

Stack - LIFO Data structure

Last in, First Out

"The last item pushed onto the stack is the first item popped from the stack"

Every method call has a method call stack

		Math.sqrt		
	fourthRoot	fourthRoot	fourthRoot	
main	main	main	main	main