**Computer Science Notes**

## Chapter 5: Methods

These notes are meant to accompany Introduction to Java Programming: Brief Version, eighth edition by Y. Daniel Lang.

**Programming Skills in a Nutshell:**

At the end of this chapter you should have the following programming skills:

1. To recognize when a block of code should be placed in a method.
2. To write the method with correct input parameters and return type.
3. To understand how and when to overload methods.
4. Here is a template that uses the key programming skills you should have at this point:

|  |
| --- |
| **import** java.util.Scanner;  **public** **class** Chap05Basics  {  /\*\* Run a four-pick lottery game with input validation.  \* Uses methods to avoid redundancy and enhance modularity.  \* **@param** args is not used.  \*/  **public** **static** **void** main(String[] args)  {  // Tell the user what the program does.  System.*out*.println("This program simulates a four-number lottery game.\n");    Scanner keyboard = **new** Scanner(System.*in*);  **int** continueSentinel = 1;  **do**  {  //Prompt the user for the range of numbers.  String userPrompt = "Please enter a positive lower bound for the " +  "lottery numbers:";  **int** lowerBound = *getIntegerGreaterThan*(0, userPrompt, keyboard);    userPrompt = "Please enter a positive upper bound for the lottery " +  "numbers:";  **int** upperBound = *getIntegerGreaterThan*(lowerBound+2, userPrompt,  keyboard);    //Compute four distinct winning lottery numbers.  **int** lotto1, lotto2, lotto3, lotto4;  lotto1 = *getRandomNonDuplicatingIntegerInRange*(lowerBound, upperBound,  -1, -1, -1);  lotto2 = *getRandomNonDuplicatingIntegerInRange*(lowerBound, upperBound,  lotto1, -1, -1);  lotto3 = *getRandomNonDuplicatingIntegerInRange*(lowerBound, upperBound,  lotto1, lotto2, -1);  lotto4 = *getRandomNonDuplicatingIntegerInRange*(lowerBound, upperBound,  lotto1, lotto2, lotto3);    //Let user decide to enter numbers by hand or let computer pick.  **int** pick1, pick2, pick3, pick4;  System.*out*.println("Enter 1 if you want to make your own picks.");  System.*out*.println("Enter 2 if you want the computer to make your " +  "picks.");  **int** whoMakesThePicksChoice = keyboard.nextInt();  **switch** (whoMakesThePicksChoice)  {  **case** 1: //User does own picks  //Prompt the user for four valid picks  //i.e., in bounds, no duplicates).  //Also, perform input validation on the picks.  userPrompt = "Please enter your first lottery pick:";  pick1 = *getValidNonDuplicatingIntegerInRange*(lowerBound,  upperBound, -1238, -1238, -1238, userPrompt, keyboard);  //Get second pick with input validation.  userPrompt = "Please enter your second lottery pick:";  pick2 = *getValidNonDuplicatingIntegerInRange*(lowerBound,  upperBound, pick1, pick1, pick1, userPrompt, keyboard);  //Get third pick with input validation.  userPrompt = "Please enter your third lottery pick:";  pick3 = *getValidNonDuplicatingIntegerInRange*(lowerBound,  upperBound, pick1, pick2, pick2, userPrompt, keyboard);  //Get fourth pick with input validation.  userPrompt = "Please enter your fourth lottery pick:";  pick4 = *getValidNonDuplicatingIntegerInRange*(lowerBound,  upperBound, pick1, pick2, pick3, userPrompt, keyboard);  **break**;  **case** 2: //Computer does picks.  //Just in case the user enters something besides 2,  //this case falls through to the default.  **default**:  //Compute four distinct winning lottery numbers.  pick1 = *getRandomNonDuplicatingIntegerInRange*(lowerBound,  upperBound, -1, -1, -1);  pick2 = *getRandomNonDuplicatingIntegerInRange*(lowerBound,  upperBound, pick1, -1, -1);  pick3 = *getRandomNonDuplicatingIntegerInRange*(lowerBound,  upperBound, pick1, pick2, -1);  pick4 = *getRandomNonDuplicatingIntegerInRange*(lowerBound,  upperBound, pick1, pick2, pick3);  }    //Show the results:  System.*out*.println("\nThe range of your lottery picks is from " +  lowerBound + " to " + upperBound + ".");  System.*out*.printf("Here are the lotto numbers:%4d %4d %4d %4d\n",  lotto1, lotto2, lotto3, lotto4);  System.*out*.printf("Here are your picks :%4d %4d %4d %4d\n",  pick1, pick2, pick3, pick4);    **if** (pick1 == lotto1 && pick2 == lotto2 && pick3 == lotto3  && pick4 == lotto4)  System.*out*.println("Exact match! You win $10,000!");  **else**  {  **int** numMatches = *getNumberOfMatchesBetween*(pick1, pick2,  pick3, pick4, lotto1, lotto2, lotto3, lotto4);  System.*out*.println("You had " + numMatches + " matches. " +  "You win $" + numMatches + ",000!");  }    System.*out*.println("Enter 1 to play again, any other number to quit.");  continueSentinel = keyboard.nextInt();  }**while** (continueSentinel == 1);//end of do-while loop to repeat program  System.*out*.println("Program Chap05Basics is now terminating.");  }//end of method main(String[])    /\*\*  \* Prompts the user for an integer,  \* and then performs input validation  \* to ensure the integer is greater than some minimum value.  \* **@param** minValue is the minimum value  \* **@param** promptString contains the prompt for the user  \* **@param** keyboard is a Scanner object from which to get the user input  \* **@return** an integer greater than minValue  \*/  **public** **static** **int** getIntegerGreaterThan(**int** minValue, String promptString,  Scanner keyboard)  {  //Print the prompt string.  System.*out*.println(promptString);  **int** userInteger = keyboard.nextInt();  //Perform input validation on userInteger  //to make sure it is greater than the minimum value.  **while** (userInteger <= minValue)  {  System.*out*.println("Invalid entry. Entry must be greater than " +  minValue);  System.*out*.println(promptString);  userInteger = keyboard.nextInt();  }  **return** userInteger;  }//end method getIntegerGreaterThan(int, String, Scanner)  /\*\*  \* Generates a random integer within a range of values  \* that does not duplicate three given numbers.  \* **@param** lowerBound is the lower bound of the range of numbers  \* **@param** upperBound is the upper bound of the range of numbers  \* **@param** n1 is the first number not to duplicate  \* **@param** n2 is the second number not to duplicate  \* **@param** n3 is the third number not to duplicate  \* **@return** a random integer between lowerBound and upperBound, inclusive,  \* that is not equal to n1 or n2 or n3  \*/  **public** **static** **int** getRandomNonDuplicatingIntegerInRange(**int** lowerBound, **int**  upperBound, **int** n1, **int** n2, **int** n3)  {  //Generate a random number between lowerBound and upperBound.  **int** randomNumber = (**int**)(Math.*random*()\*(upperBound - lowerBound + 1)) +  lowerBound;  //Make sure that number does not duplicate n1, n2, or n3.  **while**(randomNumber == n1 || randomNumber == n2 || randomNumber == n3)  {  randomNumber = (**int**)(Math.*random*()\*(upperBound - lowerBound + 1)) +  lowerBound;  };  **return** randomNumber;  }//end method getRandomNonDuplicatingIntegerInRange(int, int, int, int, int)  /\*\*  \* Prompts the user for an integer,  \* and then performs input validation  \* to ensure the integer is within a range of values,  \* and does not duplicate three given numbers.  \* **@param** lowerBound is the lower bound of the range of numbers  \* **@param** upperBound is the upper bound of the range of numbers  \* **@param** n1 is the first number not to duplicate  \* **@param** n2 is the second number not to duplicate  \* **@param** n3 is the third number not to duplicate  \* **@param** promptString contains the prompt for the user  \* **@param** keyboard is a Scanner object from which to get the user input  \* **@return**  \*/  **public** **static** **int** getValidNonDuplicatingIntegerInRange(**int** lowerBound,  **int** upperBound, **int** n1, **int** n2, **int** n3, String promptString,  Scanner keyboard)  {  **int** userInteger;  **do**  {  System.*out*.println(promptString);  userInteger = keyboard.nextInt();  **if** ( (userInteger < lowerBound || userInteger > upperBound))  System.*out*.println("Invalid entry. Entry must be between " +  lowerBound + " and " + upperBound + ".");  **if** ( (userInteger == n1) || (userInteger == n2) || (userInteger == n3))  System.*out*.println("Invalid entry. No duplicate picks allowed.");  }**while**( (userInteger < lowerBound || userInteger > upperBound)  || (userInteger == n1) || (userInteger == n2) ||  (userInteger == n3));  **return** userInteger;  }//end method getValidNonDuplicatingIntegerInRange(int, int,  //int, int, int, String, Scanner)  /\*\*  \* Counts how many matching numbers there are between two sets of  \* four numbers in each set  \* **@param** n1 is a number in the first set  \* **@param** n2 is a number in the first set  \* **@param** n3 is a number in the first set  \* **@param** n4 is a number in the first set  \* **@param** m1 is a number in the second set  \* **@param** m2 is a number in the second set  \* **@param** m3 is a number in the second set  \* **@param** m4 is a number in the second set  \* **@return** how many of the values in n1..n4 match values in m1..m4  \*/  **public** **static** **int** getNumberOfMatchesBetween(**int** n1, **int** n2, **int** n3, **int** n4,  **int** m1, **int** m2, **int** m3, **int** m4)  {  //Thanks to Patrick Kirk for this efficient technique of counting  //the number of matches!  **int** numMatches = 0;  **if** (n1 == m1 || n1 == m2 || n1 == m3 || n1 == m4) numMatches++;  **if** (n2 == m1 || n2 == m2 || n2 == m3 || n2 == m4) numMatches++;  **if** (n3 == m1 || n3 == m2 || n3 == m3 || n3 == m4) numMatches++;  **if** (n4 == m1 || n4 == m2 || n4 == m3 || n4 == m4) numMatches++;    **return** numMatches;  }//end method getNumberOfMatchesBetween(int, int, int, int, int, int, int, int)    }//end of class Chap05Basics |

|  |
| --- |
| This program simulates a four-number lottery game.  Please enter a positive lower bound for the lottery numbers:  1  Please enter a positive upper bound for the lottery numbers:  3  Invalid entry. Entry must be greater than 3  Please enter a positive upper bound for the lottery numbers:  4  Enter 1 if you want to make your own picks.  Enter 2 if you want the computer to make your picks.  2  The range of your lottery picks is from 1 to 4.  Here are the lotto numbers: 4 3 1 2  Here are your picks : 1 4 3 2  You had 4 matches. You win $4,000!  Enter 1 to play again, any other number to quit.  1  Please enter a positive lower bound for the lottery numbers:  1  Please enter a positive upper bound for the lottery numbers:  10  Enter 1 if you want to make your own picks.  Enter 2 if you want the computer to make your picks.  2  The range of your lottery picks is from 1 to 10.  Here are the lotto numbers: 1 2 6 5  Here are your picks : 3 9 6 4  You had 1 matches. You win $1,000!  Enter 1 to play again, any other number to quit.  0  Program Chap05Basics is now terminating. |

**Book’s Statement of Skills:**

1. To define methods. (5.2)
2. To invoke methods with a return value. (5.3)
3. To invoke methods without a return value. (5.4)
4. To pass arguments by value (5.5)
5. To develop reusable code that is modular, easy to read, easy to debug, and easy to maintain. (5.6)
6. To write a method that converts decimals to hexadecimals. (5.7)
7. To use method overloading and understand ambiguous overloading. (5.8)
8. To determine the scope of variables. (5.9)
9. To solve mathematics problems using the methods in the **Math** class. (5.10 – 5.11)
10. To apply the concept of method abstraction in software development. (5.12)
11. To design and implement methods using stepwise refinement. (5.12)

**Section 5.1: Introduction**

A method is a block of code with a name that is designed to perform a specific function. The block of code gets executed by calling the method, which means to type the name of the method as one of your statements. Methods make coding more efficient because you don’t need to repeat blocks of code every time you need them, because the name of the method is usually a good indication of what its functionality is, and because their use makes reading a program easier.

Here are some examples of methods we have used already: **System.out.println(), System.out.print(), Math.pow(), Math.random(), Math.sin(), Math.exp(), Math.abs(), JOptionPane.showMessageDialog(),** etc. …

**Section 5.2: Defining a Method**

The general syntax to define a method is:

*modifier(s) return\_value\_type* method\_name (list\_of\_parameters)

{

*statements*;

}

1. The ***method header*** is what we call the line that specifies the modifiers, return value type, method name, and parameters.
2. The *modifiers* we will use in this chapter will always be **public static**
3. The *return\_value\_type* we will use in this chapter are **int, double, float, String, char**, or **void**.
4. method\_name should always start with a lowercase letter.
5. The *list\_of\_parameters* is a list of variables with their type that will store information we want to send into the method. A single variable in the list is called an *argument*. Methods are not required to have any parameters.
6. The method name together with the parameter list form what we call the ***method signature***.
7. Other programming languages sometimes call methods procedures or functions.
8. We will also use the javadoc style of commenting our methods:

/\*\* Verb phrase telling what the method does.

\* **@param** *parameter\_name* and what it stores.

\* **@param** *parameter\_name* (for each parameter in the list).

\* **@return** what the method returns as a result of its calculation.

\*/

**Practice:**

**Write a program that prompts the user for two integers and then computes and prints out the sum of integers from the lower integer to the upper integer. Use at least one method in your program.**

**Solution:**

|  |
| --- |
| **import** java.util.Scanner;  /\*\* The SumExample class implements an application that sums up all integers  \* between two bounds.  \* **@author** Kevin Mirus  \*  \*/  **public** **class** SumExample {  /\*\*Prompts user for two integers, then add up all integers between them.  \* **@param** args is not used.  \*/  **public** **static** **void** main(String[] args) {  // Prompt the user for two integers.  Scanner keyboard = **new** Scanner(System.*in*);  System.*out*.println("This program sums integers.");  System.*out*.println("Please enter an integer:");  **int** number1 = keyboard.nextInt();    System.*out*.println("Please enter another integer:");  **int** number2 = keyboard.nextInt();  //Figure out which number is smaller and larger.    **int** lowerBound = *min*(number1, number2);  **int** upperBound = *max*(number1, number2);    System.*out*.println("The sum of integers from " + lowerBound +  " to " + upperBound + " = " +  *sumOfIntegersBetween*(lowerBound, upperBound) );      }//end method main(String[])  /\*\*  \* Finds the minimum of two integers.  \* **@param** n1 is the first integer  \* **@param** n2 is the second integer  \* **@return** the smaller of n1 and n2  \*/  **public** **static** **int** min(**int** n1, **int** n2)  {  /\*  //assume that n1 is the smaller number  int minimum = n1;  //change minimum if not true  if (n2 < minimum)  minimum = n2;  return minimum;  \*/    /\*  //use a conditional operator  int minimum = (n1 < n2) ? n1 : n2;  return minimum;  \*/    **return** (n1 < n2) ? n1 : n2;    }//end method min(int, int)    /\*\*  \* Finds the maximum of two integers.  \* **@param** n1 is the first integer  \* **@param** n2 is the second integer  \* **@return** the larger of n1 and n2  \*/  **public** **static** **int** max(**int** n1, **int** n2)  {  **return** (n1 > n2) ? n1 : n2;  }//end method max(int, int)    /\*\*  \* Computes the sum of integers between two bounds.  \* Precondition: low <= high  \* **@param** low is the lower bound of the range of numbers  \* **@param** high is the upper bound of the range of numbers  \* **@return** the sum of all integers from low to high  \*/  **public** **static** **long** sumOfIntegersBetween(**int** low, **int** high)  {  **long** sum = 0;  **for** (**int** counter = low; counter <= high; counter++)  sum += counter;  **return** sum;  }//end method sumOfIntegersBetween(int, int)    }//end class SumExample |

**Section 5.3: Calling a Method**

* When you *implement* a method, you write the code that makes the method do what it is supposed to do.
* When you *call* or *invoke* a method, you write a line of code that uses the method you implemented.
* If the method returns a value, you usually invoke the method with an assignment statement:

**double sqrt\_5 = Math.sqrt(5);**

* If the method does not return a value (i.e., it returns void) , you invoke the method as a statement

*System.out.println*(″Hi!″);

* The **main** method is just like any other method except that it is invoked by the JVM automatically when the program starts running.
* The main method’s header is always the same: **public** **static** **void** main(String[] args), where **String[]** is an array of **String** objects that are used when the program is run from the command line (see chapter 6 on Arrays).
* A **return** statement is required in any method that returns a value.
* Methods allow for code sharing and reuse. USE THEM!
* See <http://www.cs.armstrong.edu/liang/intro8e/html/TestMax.html>

**Section 5.3.1: Call Stacks**

When a method is invoked, the system stores variables and parameters in an area of memory called a *stack*.

Stacks store information in a last-in, first-out manner.

When a method calls another method, its stack space is kept in place, and the information for the new method is added to the stack “on top of” the calling method.

**Section 5.4: void Method Example**

* See <http://www.cs.armstrong.edu/liang/intro8e/html/TestVoidMethod.html>
* A void method does not return any value, so it is invoked as a statement.
* The return statement can be used in a void method to terminate the method and return execution to the method’s caller.
* See <http://www.cs.armstrong.edu/liang/intro8e/html/TestReturnGradeMethod.html>

**Section 5.5: Passing Parameters by Values**

* *Parameter order association*: when calling a method, the ***order*** in which you provide arguments must be in the same order as their respective parameters in the method specification.

Having the same name of variables does not do the trick.

* The values in any variables used to call a method are copied into the parameters
* A method can not change the value of an input call variable
* Example:
* See <http://www.cs.armstrong.edu/liang/intro8e/html/Increment.html>
* See <http://www.cs.armstrong.edu/liang/intro8e/html/TestPassByValue.html>

**Section 5.6: Modularizing Code**

* Methods can be used to reduce redundant code.
* Methods can be used to enable code reuse.
* Methods can be used to improve the quality of a program.
  + The code becomes easier to read because problems are isolated into separate pieces.
  + Debugging becomes easier because you can debug one method at a time instead of the whole program at once.
* See <http://www.cs.armstrong.edu/liang/intro8e/html/GreatestCommonDivisor.html>
* See <http://www.cs.armstrong.edu/liang/intro8e/html/PrimeNumberMethod.html>

**Section 5.7: Problem: Converting Decimals to Hexadecimals**

See <http://www.cs.armstrong.edu/liang/intro8e/html/Decimal2HexConversion.html>

**Section 5.8: Overloading Methods**

* *Method overloading*: Implementing two methods with the same name but different parameter lists within one class.
* The Java compiler decides which method to use based on the method signature.
* See <http://www.cs.armstrong.edu/liang/intro8e/html/TestMethodOverloading.html>
* Overloading methods can make programs clearer and more readable. Methods that perform similar tasks on slightly different data *should* be given the same name.
* Overloaded methods must have different parameter lists. You can’t overload methods based on return type or other modifiers.
* *Ambiguous invocation*: When a method invocation matches two or more methods. Net result: compiler error.
  + See the source code **AmbiguousOverloading.java** in the book…

**public static double min(double a, double b)**

**{**

**Return (a < b) ? a : b;**

**}**

**public static int min(int a, int b)**

**{**

**Return (a < b) ? a : b;**

**}**

**Section 5.9: The Scope of Variables**

* *Scope of a variable*: the portion of a program where a variable can be referenced (i.e., which portion of the call stack is the variable in).
* The scope of a variable is within the curly braces it is defined in, or sub-blocks within those braces.
* *Local variable*: a variable defined in a method. Thus, its scope is only within that method.
* Sometimes you can use the same variable name within a block inside another block, but you really should avoid doing that…

**Section 5.10: The Math Class**

The following table is blatantly plagiarized from <http://download.oracle.com/javase/6/docs/api/java/lang/Math.html>

|  |  |
| --- | --- |
| **Method Summary** | |
| static double | [**abs**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#abs(double))(double a)            Returns the absolute value of a double value. |
| static float | [**abs**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#abs(float))(float a)            Returns the absolute value of a float value. |
| static int | [**abs**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#abs(int))(int a)            Returns the absolute value of an int value. |
| static long | [**abs**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#abs(long))(long a)            Returns the absolute value of a long value. |
| static double | [**acos**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#acos(double))(double a)            Returns the arc cosine of a value; the returned angle is in the range 0.0 through *pi*. |
| static double | [**asin**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#asin(double))(double a)            Returns the arc sine of a value; the returned angle is in the range -*pi*/2 through *pi*/2. |
| static double | [**atan**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#atan(double))(double a)            Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2. |
| static double | [**atan2**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#atan2(double, double))(double y, double x)            Returns the angle *theta* from the conversion of rectangular coordinates (x, y) to polar coordinates (r, *theta*). |
| static double | [**cbrt**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#cbrt(double))(double a)            Returns the cube root of a double value. |
| static double | [**ceil**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#ceil(double))(double a)            Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer. |
| static double | [**copySign**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#copySign(double, double))(double magnitude, double sign)            Returns the first floating-point argument with the sign of the second floating-point argument. |
| static float | [**copySign**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#copySign(float, float))(float magnitude, float sign)            Returns the first floating-point argument with the sign of the second floating-point argument. |
| static double | [**cos**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#cos(double))(double a)            Returns the trigonometric cosine of an angle. |
| static double | [**cosh**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#cosh(double))(double x)            Returns the hyperbolic cosine of a double value. |
| static double | [**exp**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#exp(double))(double a)            Returns Euler's number *e* raised to the power of a double value. |
| static double | [**expm1**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#expm1(double))(double x)            Returns *e*x -1. |
| static double | [**floor**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#floor(double))(double a)            Returns the largest (closest to positive infinity) double value that is less than or equal to the argument and is equal to a mathematical integer. |
| static int | [**getExponent**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#getExponent(double))(double d)            Returns the unbiased exponent used in the representation of a double. |
| static int | [**getExponent**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#getExponent(float))(float f)            Returns the unbiased exponent used in the representation of a float. |
| static double | [**hypot**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#hypot(double, double))(double x, double y)            Returns sqrt(*x*2 +*y*2) without intermediate overflow or underflow. |
| static double | [**IEEEremainder**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#IEEEremainder(double, double))(double f1, double f2)            Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard. |
| static double | [**log**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#log(double))(double a)            Returns the natural logarithm (base *e*) of a double value. |
| static double | [**log10**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#log10(double))(double a)            Returns the base 10 logarithm of a double value. |
| static double | [**log1p**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#log1p(double))(double x)            Returns the natural logarithm of the sum of the argument and 1. |
| static double | [**max**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#max(double, double))(double a, double b)            Returns the greater of two double values. |
| static float | [**max**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#max(float, float))(float a, float b)            Returns the greater of two float values. |
| static int | [**max**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#max(int, int))(int a, int b)            Returns the greater of two int values. |
| static long | [**max**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#max(long, long))(long a, long b)            Returns the greater of two long values. |
| static double | [**min**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#min(double, double))(double a, double b)            Returns the smaller of two double values. |
| static float | [**min**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#min(float, float))(float a, float b)            Returns the smaller of two float values. |
| static int | [**min**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#min(int, int))(int a, int b)            Returns the smaller of two int values. |
| static long | [**min**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#min(long, long))(long a, long b)            Returns the smaller of two long values. |
| static double | [**nextAfter**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#nextAfter(double, double))(double start, double direction)            Returns the floating-point number adjacent to the first argument in the direction of the second argument. |
| static float | [**nextAfter**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#nextAfter(float, double))(float start, double direction)            Returns the floating-point number adjacent to the first argument in the direction of the second argument. |
| static double | [**nextUp**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#nextUp(double))(double d)            Returns the floating-point value adjacent to d in the direction of positive infinity. |
| static float | [**nextUp**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#nextUp(float))(float f)            Returns the floating-point value adjacent to f in the direction of positive infinity. |
| static double | [**pow**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#pow(double, double))(double a, double b)            Returns the value of the first argument raised to the power of the second argument. |
| static double | [**random**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#random())()            Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0. |
| static double | [**rint**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#rint(double))(double a)            Returns the double value that is closest in value to the argument and is equal to a mathematical integer. |
| static long | [**round**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#round(double))(double a)            Returns the closest long to the argument. |
| static int | [**round**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#round(float))(float a)            Returns the closest int to the argument. |
| static double | [**scalb**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#scalb(double, int))(double d, int scaleFactor)            Return d × 2scaleFactor rounded as if performed by a single correctly rounded floating-point multiply to a member of the double value set. |
| static float | [**scalb**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#scalb(float, int))(float f, int scaleFactor)            Return f × 2scaleFactor rounded as if performed by a single correctly rounded floating-point multiply to a member of the float value set. |
| static double | [**signum**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#signum(double))(double d)            Returns the signum function of the argument; zero if the argument is zero, 1.0 if the argument is greater than zero, -1.0 if the argument is less than zero. |
| static float | [**signum**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#signum(float))(float f)            Returns the signum function of the argument; zero if the argument is zero, 1.0f if the argument is greater than zero, -1.0f if the argument is less than zero. |
| static double | [**sin**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#sin(double))(double a)            Returns the trigonometric sine of an angle. |
| static double | [**sinh**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#sinh(double))(double x)            Returns the hyperbolic sine of a double value. |
| static double | [**sqrt**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#sqrt(double))(double a)            Returns the correctly rounded positive square root of a double value. |
| static double | [**tan**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#tan(double))(double a)            Returns the trigonometric tangent of an angle. |
| static double | [**tanh**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#tanh(double))(double x)            Returns the hyperbolic tangent of a double value. |
| static double | [**toDegrees**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#toDegrees(double))(double angrad)            Converts an angle measured in radians to an approximately equivalent angle measured in degrees. |
| static double | [**toRadians**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#toRadians(double))(double angdeg)            Converts an angle measured in degrees to an approximately equivalent angle measured in radians. |
| static double | [**ulp**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#ulp(double))(double d)            Returns the size of an ulp of the argument. |
| static float | [**ulp**](http://java.sun.com/javase/6/docs/api/java/lang/Math.html#ulp(float))(float f)            Returns the size of an ulp of the argument. |

**Section 5.10.1: Trigonometric Methods**

static double acos(double a)

Returns the arc cosine of a value; the returned angle is in the range 0.0 through pi.

static double asin(double a)

Returns the arc sine of a value; the returned angle is in the range -pi/2 through pi/2.

static double atan(double a)

Returns the arc tangent of a value; the returned angle is in the range -pi/2 through pi/2.

static double atan2(double y, double x)

Returns the angle theta from the conversion of rectangular coordinates (x, y) to polar coordinates (r, theta).

static double cos(double a)

Returns the trigonometric cosine of an angle.

static double hypot(double x, double y)

Returns sqrt(x2 +y2) without intermediate overflow or underflow.

static double sin(double a)

Returns the trigonometric sine of an angle.

static double tan(double a)

Returns the trigonometric tangent of an angle.

static double toDegrees(double angrad)

Converts an angle measured in radians to an approximately equivalent angle measured in degrees.

static double toRadians(double angdeg)

Converts an angle measured in degrees to an approximately equivalent angle measured in radians.

**Section 5.10.2: Exponent Methods**

static double cbrt(double a)

Returns the cube root of a double value.

static double cosh(double x)

Returns the hyperbolic cosine ( cosh(x) = (e^x + e^(-x)) / 2 ) of a double value.

static double exp(double a)

Returns Euler's number e raised to the power of a double value.

static double expm1(double x)

Returns e^x -1.

static int getExponent(double d)

Returns the unbiased exponent used in the representation of a double.

static int getExponent(float f)

Returns the unbiased exponent used in the representation of a float.

static double log(double a)

Returns the natural logarithm (base e) of a double value.

static double log10(double a)

Returns the base 10 logarithm of a double value.

static double log1p(double x)

Returns the natural logarithm of the sum of the argument and 1.

static double pow(double a, double b)

Returns the value of the first argument raised to the power of the second argument.

static double sinh(double x)

Returns the hyperbolic sine ( sinh(x) = (e^x – e^(-x)) / 2 ) of a double value.

static double sqrt(double a)

Returns the correctly rounded positive square root of a double value.

static double tanh(double x)

Returns the hyperbolic tangent ( tanh(x) = sinh(x) / cosh(x) ) of a double value.

**Section 5.10.3: The Rounding Methods**

static double ceil(double a)

Returns the smallest (closest to negative infinity) double value that is greater than or equal to the

static double floor(double a)

Returns the largest (closest to positive infinity) double value that is less than or equal to the argument and is equal to a mathematical integer.

static double rint(double a)

Returns the double value that is closest in value to the argument and is equal to a mathematical integer.

static long round(double a)

Returns the closest long to the argument.

static int round(float a)

Returns the closest int to the argument.

**Section 5.10.4: The min, max, and abs Methods**

static double abs(double a)

Returns the absolute value of a double value.

static float abs(float a)

Returns the absolute value of a float value.

static int abs(int a)

Returns the absolute value of an int value.

static long abs(long a)

Returns the absolute value of a long value.

static double max(double a, double b)

Returns the greater of two double values.

static float max(float a, float b)

Returns the greater of two float values.

static int max(int a, int b)

Returns the greater of two int values.

static long max(long a, long b)

Returns the greater of two long values.

static double min(double a, double b)

Returns the smaller of two double values.

static float min(float a, float b)

Returns the smaller of two float values.

static int min(int a, int b)

Returns the smaller of two int values.

static long min(long a, long b)

Returns the smaller of two long values.

**Section 5.10.5: The random Method**

static double random()

Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

**Section 5.11: Case Study: Generating Random Characters**

* See <http://www.cs.armstrong.edu/liang/intro8e/html/RandomCharacter.html>
* See <http://www.cs.armstrong.edu/liang/intro8e/html/TestRandomCharacter.html>

**Section 5.12: Method Abstraction and Stepwise Refinement**

* Method abstraction: separating the use of a method from its implementation (i.e., saying to yourself, :I would like a method that does such-and-such…” without actually writing the coed first).
* Information hiding: when the details of a method are hidden from the client who invokes the method.
* Example: We do not know the code behind the **println()** method; we just know how to use it.
* Divide-and-conquer (stepwise refinement): when you take a big programming problem and break it into smaller pieces.

**Section 5.12.1: Top-Down Design**

* Start with the big picture, then work your way down into the details.

**Section 5.12.2: Top-Down or Bottom-Up Implementation**

*Top down*: create a main program that calls all methods, then implement one method at a time from the big picture, and use stubs for the rest.

*Bottom up*: implement one method, test it, and then add other methods and their test code as well.

* See <http://www.cs.armstrong.edu/liang/intro8e/html/PrintCalendar.html>

**Section 5.12.3: Implementation Details**

* See <http://www.cs.armstrong.edu/liang/intro8e/html/PrintCalendar.html>