**Computer Science Notes**

## Chapter 4: Loops

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. Use a **while** loop to repeat a block of code.
2. Use a **do-while** loop to repeat a block of code.
3. Use a **for** loop to repeat a block of code.
4. Be able to translate from one type of loop to the other.
5. Know the advantages of one type of loop over the other
6. Here is a template that uses the key programming skills you should have at this point:

|  |
| --- |
| **import** java.util.Scanner;**public** **class** Chap04Basics{ /\*\* Run a three-pick lottery game with input validation. \* **@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 three-number lottery game.\n");  Scanner keyboard = **new** Scanner(System.*in*);  **int** continueCondition = 1; **while** (continueCondition == 1) { //Prompt the user for the range of numbers. //Perform input validation on the lower bound to ensure it's positive. **int** lowerBound; **do** { System.*out*.println("\nPlease enter a positive lower bound " + "for the lottery numbers:"); lowerBound = keyboard.nextInt(); **if** (lowerBound <= 0) System.*out*.println("Invalid lower bound. Lower bound" + " must be positive."); } **while** (lowerBound < 1);  //Perform input validation on the upper bound to ensure it is  //two more or more than the first. **int** upperBound; **do** { System.*out*.println("Please enter a positive upper bound for the " + "lottery numbers:"); upperBound = keyboard.nextInt(); **if** (upperBound < (lowerBound + 2)) System.*out*.println("Invalid upper bound. Upper bound " + "must be two or more than the lower bound."); } **while** (upperBound < (lowerBound + 2));  //Pick three distinct numbers. **int** lotto1, lotto2, lotto3; lotto1 = (**int**) (Math.*random*() \* (upperBound - lowerBound + 1)) + lowerBound; **do** { lotto2 = (**int**) (Math.*random*() \* (upperBound - lowerBound + 1)) + lowerBound; } **while** (lotto2 == lotto1);  **do** { lotto3 = (**int**) (Math.*random*() \* (upperBound - lowerBound + 1)) + lowerBound; } **while** (lotto3 == lotto1 || lotto3 == lotto2);  //Prompt the user for three picks. //Perform input validation to ensure the picks are in the range //and to ensure there are no duplicate picks. **int** pick1 = -1, pick2 = pick1, pick3 = pick1; **for** (**int** i = 1; i <= 3; i++) { **int** tempNum; **boolean** tempNumIsOutsideRange = **false**; **boolean** tempNumIsDuplicate = **false**; **do** { System.*out*.println("Please enter a number between " + lowerBound + " and " + upperBound + " for lottery pick #" + i + ":"); tempNum = keyboard.nextInt(); tempNumIsOutsideRange = (tempNum < lowerBound) || (tempNum > upperBound); **if** (tempNumIsOutsideRange) System.*out*.println("Error: Pick is outside " + "of range."); tempNumIsDuplicate = (tempNum == pick1) ||  (tempNum == pick2); **if** (tempNumIsDuplicate) System.*out*.println("Error: Pick duplicates " + "another pick."); } **while** (tempNumIsOutsideRange || tempNumIsDuplicate); **if** (i == 1) pick1 = tempNum; **else** **if** (i == 2) pick2 = tempNum; **else** pick3 = tempNum; }  //Show the results: System.*out*.println("\nThe range of your lottery picks are from " + lowerBound + " to " + upperBound + "."); System.*out*.printf("Here are the lotto numbers:%4d %4d %4d\n", lotto1, lotto2, lotto3); System.*out*.printf("Here are your picks :%4d %4d %4d\n", pick1, pick2, pick3); **if** (pick1 == lotto1 && pick2 == lotto2 && pick3 == lotto3) System.*out*.println("Exact match! You win $10,000!"); **else** { //Thanks to Patrick Kirk for this efficient technique  //of counting the number of matches! **int** numMatches = 0; **if** (pick1 == lotto1 || pick1 == lotto2 || pick1 == lotto3) numMatches++; **if** (pick2 == lotto1 || pick2 == lotto2 || pick2 == lotto3) numMatches++; **if** (pick3 == lotto1 || pick3 == lotto2 || pick3 == lotto3) numMatches++; **if** (numMatches > 0) System.*out*.println("You had " + numMatches + " matching picks." + " You win $" +  numMatches + ",000!"); **else** System.*out*.println("No matches anywhere. You win " + "nothing!"); } System.*out*.println("\nEnter 1 to continue or any other number " + "to quit"); continueCondition = keyboard.nextInt(); }//end while () loop System.*out*.println("Program Chap04Basics is now terminating."); }//end method main(String[])}//end class Chap04Basics |

|  |
| --- |
| This program simulates a three-number lottery game.Please enter a positive lower bound for the lottery numbers:-9Invalid lower bound. Lower bound must be positive.Please enter a positive lower bound for the lottery numbers:0Invalid lower bound. Lower bound must be positive.Please enter a positive lower bound for the lottery numbers:2Please enter a positive upper bound for the lottery numbers:-5Invalid upper bound. Upper bound must be two or more than the lower bound.Please enter a positive upper bound for the lottery numbers:3Invalid upper bound. Upper bound must be two or more than the lower bound.Please enter a positive upper bound for the lottery numbers:4Please enter a number between 2 and 4 for lottery pick #1:1Error: Pick is outside of range.Please enter a number between 2 and 4 for lottery pick #1:1Error: Pick is outside of range.Please enter a number between 2 and 4 for lottery pick #1:2Please enter a number between 2 and 4 for lottery pick #2:2Error: Pick duplicates another pick.Please enter a number between 2 and 4 for lottery pick #2:3Please enter a number between 2 and 4 for lottery pick #3:2Error: Pick duplicates another pick.Please enter a number between 2 and 4 for lottery pick #3:3Error: Pick duplicates another pick.Please enter a number between 2 and 4 for lottery pick #3:4The range of your lottery picks are from 2 to 4.Here are the lotto numbers: 2 4 3Here are your picks : 2 3 4You had 3 matching picks. You win $3,000!Enter 1 to continue or any other number to quit1Please enter a positive lower bound for the lottery numbers:1Please enter a positive upper bound for the lottery numbers:10Please enter a number between 1 and 10 for lottery pick #1:2Please enter a number between 1 and 10 for lottery pick #2:5Please enter a number between 1 and 10 for lottery pick #3:7The range of your lottery picks are from 1 to 10.Here are the lotto numbers: 4 5 3Here are your picks : 2 5 7You had 1 matching picks. You win $1,000!Enter 1 to continue or any other number to quit0Program Chap04Basics is now terminating. |

**Book’s Statement of Skills:**

1. To write programs for executing statements repeatedly using a **while** loop. (4.2)
2. To develop a program for **GuessNumber**. (4.2.1)
3. To follow the loop design strategy to develop loops. (4.2.2)
4. To develop a program for **SubtractionQuizLoop**. (4.2.3)
5. To control a loop with a sentinel value (4.2.4)
6. To obtain large input from a file using input redirection rather than typing from the keyboard (4.2.4)
7. To write loops using **do-while** statements. (4.3)
8. To write loops using **for** statements. (4.4)
9. To discover the similarities and differences between the three types of loops. (4.5)
10. To write nested loops. (4.6)
11. To learn techniques for minimizing numerical error (4.7)
12. To learn loops from a variety of examples (**GCD**, **FutureTuition**, **MonteCarloSimulation**) (4.8)
13. To implement program control with **break** and **continue**. (4.9)
14. (GUI) To control a loop with a confirmation dialog. (4.10)

**Section 4.1: Introduction**

A loop is a structure that allows you to repeat a block of statements as many times as is desired or needed (or more, if your program has an infinite loop).

**Example:** Print Welcome to Java! 100 times

|  |  |
| --- | --- |
| **Without loops** | **Using loops** |
| // Print ″Welcome to Java! 100 timesSystem.*out*.println("Welcome to Java!″);System.*out*.println("Welcome to Java!″);System.*out*.println("Welcome to Java!″);System.*out*.println("Welcome to Java!″);System.*out*.println("Welcome to Java!″);//… 94 more println statements…System.*out*.println("Welcome to Java!″); | // Print ″Welcome to Java!″ 100 timesfor (int count = 1; count <= 100; count++) System.*out*.println("Welcome to Java!″); |

**Section 4.2: The while Loop**

A **while** loop checks to see if a condition is true, executes a block of code if the condition is true, and then keeps checking the condition and executing the block of code as long as the condition is true.

**Syntax of a while statement:**

while (*condition*)

{

*conditionTrueStatement(s)*;

}

*nextStatement;*

* When the condition is true, the program executes *conditionTrueStatement(s)*.
* When *conditionTrueStatement(s)* are done executing, the program goes back to the condition, and if the condition is still true, the program executes *conditionTrueStatement(s)* again.
* This process of looping back to the condition keeps repeating until the condition is false, at which point the program jumps down and executes *nextStatement*.

**Example:**

|  |
| --- |
| **public** **class** PrintLoop{ **public** **static** **void** main(String[] args) { // Print "Welcome to Java!" 100 times **int** counter = 1; **int** maxPrintLines = 100; **while** (counter <= maxPrintLines) { System.*out*.println("Welcome to Java!"); counter++; } }} |

**Example:**

|  |
| --- |
| **import** java.util.Scanner;**public** **class** Guessing7Game{ **public** **static** **void** main(String[] args) { // a loop that plays a guessing game Scanner keyboard = **new** Scanner(System.*in*); System.*out*.println("Enter a guess:"); **int** guess = keyboard.nextInt(); **while** (guess != 7) { System.*out*.println("Bad guess."); System.*out*.println("Enter a guess:"); guess = keyboard.nextInt(); } System.*out*.println("You guessed correctly!"); }} |

**Example: INPUT VALIDATION: Notice that this loop keeps on repeating until a valid piece of data is entered.**

|  |
| --- |
| **import** java.util.Scanner;**public** **class** CircleCircumference{ **public** **static** **void** main(String[] args) { System.*out*.println("This program will compute the circumference of " + "a circle given its radius.");  Scanner keyboard = **new** Scanner(System.*in*);  // Get the user’s input for a circle radius System.*out*.println("Enter the radius of a circle:"); **double** radius = keyboard.nextDouble(); // a loop that validates an input for a radius **while** (radius < 0.0) { System.*out*.println("Bad entry; you entered r = " + radius +  ", and the radius of a circle should NOT be negative."); System.*out*.println("Enter a positive radius for the circle:"); radius = keyboard.nextDouble(); }  System.*out*.println("For a circle of radius " + radius +  ", the circumference is: " + (2\*Math.*PI*\*radius) + "."); }} |

**Example: PROGRAM REPETITION: This is useful if you want to repeat the entire body of a program.**

|  |
| --- |
| **import** java.util.Scanner;**public** **class** CompareNumbers{ /\*\* Prompts the user for two numbers to compare.  \* **@param** args is not used \*/ **public** **static** **void** main (String[ ] args) { System.*out*.println("This program compare two numbers as many times as you like."); //Declare variables for the data to be read in: two numbers for comparison. **double** num1, num2; //Create a Scanner object for reading in the user's input Scanner keyboard = **new** Scanner(System.*in*); **int** compareAgain = 1; **while** (compareAgain == 1) { //get the user's data System.*out*.println("\nPlease enter a number:"); num1 = keyboard.nextDouble(); System.*out*.println("Please enter a second number:"); num2 = keyboard.nextDouble();  //Compare the numbers System.*out*.println("Here is the comparison:"); **if** (num1 < num2) System.*out*.println(num1 + " < " + num2); **else** **if** (num1 > num2) System.*out*.println(num1 + " > " + num2); **else** System.*out*.println(num1 + " = " + num2); System.*out*.println("Enter 1 to continue or 0 to quit:"); compareAgain = keyboard.nextInt(); } System.*out*.println("\nProgram Terminated."); }} |

**Example: Notice that a variable can be defined in a loop, but that it only exists in memory while the loop is executing.**

|  |
| --- |
| **public** **class** InvestmentMaturityDate{ **public** **static** **void** main(String[] args) { // a loop that computes how many years  // until a $10,000 investment reaches a target balance of $20,000 // at an interest rate of 8% **double** initialBalance = 10000.00; **double** targetBalance = 20000.00; **double** rate = 0.08; **double** currentBalance = initialBalance; **int** years = 0; **while** (currentBalance < targetBalance) { years++; **double** interest = currentBalance \* rate; currentBalance = currentBalance + interest; } //NOTE: the interest variable no longer exists outside the loop //SO, it can not be used for output or further calculations. System.*out*.println("An investment of $" + initialBalance +  " earning " + (rate\*100) + "% interest per year" + " grows to at least $" + targetBalance +  " after " + years + " years."); System.*out*.printf("In fact, the investment is worth $%,9.2f " + " at the end of the %2d years\n", currentBalance, years); }} |

**Example: A loop that executes a pre-determined number of times.**

|  |
| --- |
| **public** **class** FutureValueOfAnInvestment{ **public** **static** **void** main(String[] args) { // a loop that computes the value of a $10,000 investment  // after 5 years when earning at an interest rate of 8% **double** initialBalance = 10000.00; **double** rate = 0.08; **int** maxYears = 5; **double** currentBalance = initialBalance; **int** years = 0; **while** (years < maxYears) { years++; **double** interest = currentBalance \* rate; currentBalance = currentBalance + interest; } System.*out*.println("An investment of $" +  String.*format*("%,10.2f",initialBalance) + " invested at " + (rate\*100) + "% interest per year " + "grows to $" + String.*format*("%,10.2f",currentBalance) +  " after " + maxYears + " years."); }} |

It is VERY common that loops are controlled by an integer variable that gets incremented every time the loop executes. Such controlling integer variables are called **control variables or counters**.

Common errors with counters:

* Infinite loops: due to incrementing the counter incorrectly or not at all, or having a bad condition.

|  |
| --- |
| The following loop is “infinite” because the counter years is never incremented, so it will always be less than maxYears. |
| **while** (years < maxYears){ **double** interest = currentBalance \* rate; currentBalance = currentBalance + interest;} |

* Off-by-one errors: due to using a bad starting value for the counter or an incorrect condition.

|  |
| --- |
| The following loop will compute one year too many of interest payments because when starting a counter at zero, you need to terminate the loop at one less than the number of iterations you want. |
| **int** years = 0;**while** (years <= maxYears){ years++; **double** interest = currentBalance \* rate; currentBalance = currentBalance + interest;} |

**Section 4.2.1: Problem: Guessing Numbers**

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

**Section 4.2.2: Loop Design Strategies**

Step1: Identify the statements that need to be repeated.

Step 2: Wrap those statements in a basic loop just to get the process started:

**while (true) {**

 **statements;**

**}**

Step 3: Replace the **true** with an appropriate loop continuation condition, and add a statement or statements into the body of the loop for controlling the loop.

**Section 4.2.3: Problem: An Advanced Math Learning Tool**

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

**Section 4.2.4: Controlling a Loop with a Sentinel Value**

* If you want the user of a program to decide when to end a loop, then another trick you can use is to give the user a special input value to enter that does not make sense in terms of the other data being entered. When this special input value is encountered, the loop ends. This special input value is called a ***sentinel value***.
* See <http://www.cs.armstrong.edu/liang/intro8e/html/SentinelValue.html>

**Example: Computing the average of a list of user-entered numbers.**

|  |
| --- |
| **import** java.util.Scanner;**public** **class** Average{ **public** **static** **void** main (String[ ] args) { System.*out*.println("This program computes the average of non-negative " + "numbers."); //Declare variables for the data entries, number of entries, and average. **double** dataEntry = 0.0, average = 0.0; **int** numEntries = 0; //Create a Scanner object for reading in the user's input Scanner keyboard = **new** Scanner(System.*in*); **while** (dataEntry >= 0.0) { //get the user's data System.*out*.println("Please enter a number (or a negative value " + "to quit):"); dataEntry = keyboard.nextDouble(); **if** (dataEntry >= 0.0) { average += dataEntry; numEntries++; } } **if** (numEntries > 0) { average = average / numEntries; System.*out*.println("\nThe average of your " + numEntries +  " values is: " + average); } **else** System.*out*.println("\nNo data enetered."); System.*out*.println("\nProgram Terminated."); }//end main()} |

|  |
| --- |
| This program computes the average of non-negative numbers.Please enter a number (or a negative value to quit):2Please enter a number (or a negative value to quit):3Please enter a number (or a negative value to quit):4Please enter a number (or a negative value to quit):-1The average of your 3 values is: 3.0Program Terminated. |

**Section 4.2.5: Input and Output Redirections**

* To read data from a text file rather than from the keyboard, you can use ***input redirection*** to redirect the program to use data from a file instead of from the keyboard
	+ Example: from the command line, type the following (**SentinelValue.java** and **input.txt** have to be in the same folder):
	**java SentinelValue < input.txt**
* To write data to a text file rather than to the console, you can use ***output redirection*** to redirect the program to write the data to a file instead of the console
	+ Example: from the command line, type the following:
	**java ClassName > output.txt**
* You can do both input and output redirection at the same time…
	+ Example: from the command line, type the following (**SentinelValue.java** and **input.txt** have to be in the same folder):
	**java SentinelValue < input.txt > output.txt**

**Section 4.3: The do-while Loop**

A **do-while** loop executes the body of the loop ***before*** checking the condition. This can result in slightly more efficient code when you know you want the loop to execute at least once.
Notice the semicolon after the **while** clause.

**Syntax of a do-while statement:**

do

{

*Statement(s)*;

} while (*condition*);

*nextStatement;*

* When the condition is true, the program goes back to the **do** and executes the *Statement(s)*.
* When the condition is false, the program executes *nextStatement*.

**Example:** Notice that the program below does not have to have the prompt and input lines of code repeated like in the while loop, but that you do need an **if** statement to handle the error message to the user when an incorrect guess is entered.

|  |
| --- |
| **import** java.util.Scanner;**public** **class** Guessing7Game{ **public** **static** **void** main(String[] args) { // a loop that plays a guessing game Scanner keyboard = **new** Scanner(System.*in*); **int** guess; **do** { System.*out*.println("Enter a guess:"); guess = keyboard.nextInt(); **if** (guess != 7) System.*out*.println("Bad guess."); } **while** (guess != 7); System.*out*.println("You guessed correctly!"); }} |

* See <http://www.cs.armstrong.edu/liang/intro8e/html/TestDoWhile.html> for another example.

**Section 4.4: for Loops**

Loops that depend on the value of an incremented counter for the condition are so common that a looping structure called the **for** loop was created that makes this type loop more efficient:

**Syntax of a simple for statement:**

for (*initial\_action*; *loop\_continuation\_condition*; *action\_after\_each\_iteration*)

*Statement*;

*nextStatement;*

**Syntax of a block for statement:**

for (*initial\_action*; *loop\_continuation\_condition*; *action\_after\_each\_iteration*)

{

*Statement(s)*;

}

*nextStatement;*

* The *initial\_action* is statement that is usually used to declare and initialize a control variable for the loop.
* As long as the *loop\_continuation\_condition* is true, the program executes the *Statement(s)*.
* When the *loop\_continuation\_condition* is false, the program executes *nextStatement*.
* The *action\_after\_each\_iteration* is a statement that executes after the last statement in the body of the loop; it is usually used to adjust the control variable (usually by incrementing it or decrementing it).
* Control variables declared inside the header of a loop can not be accessed by statements outside the loop.

**Example:** A loop that executes a pre-determined number of times. It is possible to both define and initialize a variable in the header, but that variable only exists in memory while the loop is executing…

|  |
| --- |
| **public** **class** FutureValueOfAnInvestment{ **public** **static** **void** main(String[] args) { // a loop that computes the value of a $10,000 investment  // after 5 years when earning at an interest rate of 8% **double** initialBalance = 10000.00; **double** rate = 0.08; **int** maxYears = 5; **double** currentBalance = initialBalance; **int** years = 0; **for** (years = 1; years <= maxYears; years ++) { **double** interest = currentBalance \* rate; currentBalance = currentBalance + interest; } System.*out*.println("An investment of $" +  String.*format*("%,10.2f",initialBalance) + " invested at " + (rate\*100) + "% interest per year " + "grows to $" + String.*format*("%,10.2f",currentBalance) +  " after " + maxYears + " years."); }} |

**Also notice that the loop could be made more efficient as follows:**

**for** (years = 1; years <= maxYears; years ++)

 currentBalance = currentBalance + currentBalance \* rate;

**Or as follows:**

**for** (years = 1; years <= maxYears; years ++)

 currentBalance += currentBalance \* rate;

**Example:**

|  |
| --- |
| **public** **class** IntegerSum{ **public** **static** **void** main(String[] args) { // a loop that sums the numbers from 1 to 10 **int** sum = 0; **int** lastInteger = 1000; **for** (**int** i= 1; i <= lastInteger; i++) sum += i; System.*out*.println("The sum of integers from 1 to " + lastInteger +  " = " + sum); }} |

**For loop hints:**

1. use **for** loops for their intended purpose; do not change the counter, change the starting value, change the ending value, or increment the counter in the body of the loop; only do this in the **for** loop header.
2. don’t forget a semicolon after a for loop in which all the work of the loop is done in the header.
**(Bad) Example: This works, but violates hint #1; … notice the semicolon immediately after the header.**

|  |
| --- |
| **public** **class** InvestmentMaturityDate{ **public** **static** **void** main(String[] args) { // a loop that computes how many years  // until a $10,000 investment reaches a target balance of $20,000 // at an interest rate of 8% **double** initialBalance = 10000.00; **double** targetBalance = 20000.00; **double** rate = 0.08; **double** currentBalance = initialBalance; **int** years = 0;  **for** (years = 1;  ( currentBalance = currentBalance \* (1 + rate) ) < targetBalance;  years ++) ;  System.*out*.println("An investment of $" + initialBalance +  " earning " + (rate\*100) + "% interest per year" + " grows to at least $" + targetBalance +  " after " + years + " years."); System.*out*.printf("In fact, the investment is worth $%,9.2f " + " at the end of the %2d years\n", currentBalance, years); }} |

1. don’t use a semicolon after the header when you want the loop to iterate the statement after the header.
2. don’t use != to test the end of a range; use <= instead.
3. when the counter is defined in the header of a for loop, it only has scope while the loop is iterating; it is no longer defined after the loop
4. you can declare multiple variables and have multiple updates (all separated by semicolons) in the header of a for loop, but this is sometimes considered bad form

**Section 4.5: Which Loop to Use?**

* **while** and **for** loops are pre-test loops.
* **do-while** loops are post-test loops.
* Use a **while** loop when you want to test the condition first.
* Use a **do-while** loop when you want the body to execute at least once.
* Use a **for** loop when you want to test the condition first, ***and*** the loop will execute a pre-determined number of times.

|  |
| --- |
| **public** **class** IntegerSum{ **public** **static** **void** main(String[] args) { //Sum the numbers from 1 to 1000. **int** lastInteger = 1000; **int** sum = 0; //Perform the calculation with a for loop. **for** (**int** i= 1; i <= lastInteger; i++) sum += i; System.*out*.println("The sum of integers from 1 to " + lastInteger +  " = " + sum);  //Perform the calculation with a while loop. sum = 0; **int** i = 0; **while** (i <= lastInteger) { sum += i; i++; } System.*out*.println("The sum of integers from 1 to " + lastInteger +  " = " + sum); //Perform the calculation with a do loop. sum = 0; i = 0; **do** { sum += i; i++; }**while** (i < lastInteger+1); System.*out*.println("The sum of integers from 1 to " + lastInteger +  " = " + sum); }} |

|  |
| --- |
| The sum of integers from 1 to 1000 = 500500The sum of integers from 1 to 1000 = 500500The sum of integers from 1 to 1000 = 500500 |

**Section 4.6: Nested Loops**

A nested loop is a loop within another loop. Each time the outer loop iterates, the entire inner loop gets executed. Nested loops are used when you have to process data that depends on two “dimensions”. One common application is processing graphics, and another is processing any “two-dimensional” type of information.

**Example:** Print a triangular shape as follows:

X

XX

XXX

XXXX

XXXXX

|  |
| --- |
| **public** **class** PrintRightTriangle { **public** **static** **void** main(String[] args) { // Print a right triangle of x's. **int** maxRows = 5; **for** (**int** row = 1; row <= maxRows; row++) { **for** (**int** col = 1; col <= row; col++) { System.*out*.print("X"); } System.*out*.println(); } }} |

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

**Section 4.7: Minimizing Numerical Errors**

Tips:

* Use **double** instead of **float**.
* Use integers to count instead of floating-point numbers.
* Process small numbers first.
* Add or subtract numbers of similar magnitude.
* See <http://www.cs.armstrong.edu/liang/intro8e/html/TestSum.html>

**Section 4.8: Case Studies**

**Section 4.8.1: Problem: Finding the Greatest Common Divisor**

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

**Section 4.8.2: Problem: Predicting the Future Tuition**

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

**Section 4.8.3: Problem: Monte Carlo Simulation**

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

**Section 4.9: Keywords break and continue**

* **break** immediately ends the innermost loop that contains it. **break** breaks out of a loop. It is generally used with an if statement.
* **continue** only ends the current iteration of the loop that contains it. Program control goes to the end of the loop body. **continue** breaks out of an iteration. It is generally used with an **if** statement.
* See <http://www.cs.armstrong.edu/liang/intro8e/html/TestBreak.html>
* See <http://www.cs.armstrong.edu/liang/intro8e/html/TestContinue.html>
* See <http://www.cs.armstrong.edu/liang/intro8e/html/GuessNumberUsingBreak.html>

**Section 4.9.1: Problem: Displaying Prime Numbers**

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

 **Random Numbers using the Random Class and their use in Simulations**

A lot of times, you need a random number, like to simulate the roll of some dice, a card picked at random from a deck, or other things. The Random class generates random numbers for you.

* You can import the class **java.util.Random** instead of using Math.random()…
* The **nextInt(n)** method returns a random integer between **0** (inclusive) and **n** (exclusive).
* The **nextDouble()** method returns a random floating-point number between **0** (inclusive) and **1** (exclusive).

**Example: simulate a pair of dice.**

|  |
| --- |
| **import** java.util.Random;**import** java.util.Scanner;/\*\* Simulates rolling dice. \*/**public** **class** Dice{ /\*\* Rolls the dice.  \* **@param** args is not used \*/ **public** **static** **void** main (String[] args) { Random generator = **new** Random(); Scanner keyboard = **new** Scanner(System.*in*); **int** repeatSentinel;  System.*out*.println("This program will simulate the roll " + "of a pair of dice."); **do** { **int** die1 = generator.nextInt(6) + 1; **int** die2 = generator.nextInt(6) + 1;  System.*out*.println("The roll is: " + die1 + " and " + die2);  System.*out*.println("Enter 1 to continue or 0 to quit."); repeatSentinel = keyboard.nextInt(); }**while** (repeatSentinel > 0); }} |

**Example: Compute the area of a circle by “throwing darts” at a quarter circle of radius 1 in the square of
0 < x < 1 and 0 < y < 1. Techniques like this are called Monte Carlo simulations.**

|  |
| --- |
| **import** java.util.Random;/\*\* Computes area of a circle using random numbers.\*/**public** **class** MonteCarloCircle{ /\*\* Computes area of a circle using random numbers. \* **@param** args is not used \*/ **public** **static** **void** main (String[ ] args) { Random generator = **new** Random(); **int** totalPoints = 10000; **int** interiorHits = 0;  System.*out*.println("This program will compute the approximate " + "area of a circle of radius 1 using " + totalPoints +  " random numbers.\n");  **for** (**int** i = 1; i <= totalPoints; i++) { **double** x = generator.nextDouble(); **double** y = generator.nextDouble(); **if** ( (x\*x + y\*y) < 1 ) interiorHits++; }  System.*out*.println("The approximate area of a radius=1 circle is "  + 4. \* interiorHits / totalPoints); }} |

##

**Section 4.10: Controlling a Loop with a Confirmation Dialog**

If you want the user of a program to decide when to end a loop, you can prompt them if they want to continue using a confirmation dialog box (as opposed to prompting for input from the console in some of the examples above).

**Example: PROGRAM REPETITION using a confirmation dialog.**

|  |
| --- |
| **import** javax.swing.JOptionPane;**public** **class** Chap04BasicsGUI{ /\*\* Prompts the user for two numbers to compare.  \* **@param** args is not used \*/ **public** **static** **void** main (String[ ] args) { //Tell the user what the program does JOptionPane.*showMessageDialog*(**null**, "This program compare two " + "numbers as many times as you like." , "Chap04BasicsGUI Intro", JOptionPane.*INFORMATION\_MESSAGE*); //Declare variables for the data to be read in: two numbers for comparison. **double** num1, num2; String numberString; **int** compareAgain = JOptionPane.*YES\_OPTION*; **while** (compareAgain == JOptionPane.*YES\_OPTION*) { //get the user's data numberString = JOptionPane.*showInputDialog*(**null**, "Please enter the " + "first number:", "First Number", JOptionPane.*QUESTION\_MESSAGE*); num1 = Double.*parseDouble*(numberString); numberString = JOptionPane.*showInputDialog*(**null**, "Please enter the " + "second number:", "Second Number", JOptionPane.*QUESTION\_MESSAGE*); num2 = Double.*parseDouble*(numberString);  //Compare the numbers String resultString = "Here is the comparison:\n"; **if** (num1 < num2) resultString += (num1 + " < " + num2); **else** **if** (num1 > num2) resultString += (num1 + " > " + num2); **else** resultString += (num1 + " = " + num2);  JOptionPane.*showMessageDialog*(**null**, resultString, "Results", JOptionPane.*INFORMATION\_MESSAGE*); compareAgain = JOptionPane.*showConfirmDialog*(**null**, "Do you want to continue?"); } JOptionPane.*showMessageDialog*(**null**, "Program Terminated.", "All Done", JOptionPane.*INFORMATION\_MESSAGE*); }} |