# ICS 102: Lab# 6 Repetition Statements

**Objectives:**

Learn Java repetition statements by covering the following topics:

1. while loops
2. do-while loops
3. for-loops
4. Nested loops
5. String processing using loops

* **while-statement**

A while-statement is used to execute a statement or a compound-statement zero or more times as long as the while-condition is true:

**while(condition)**

**while-body**

The while-body can be a simple statement in which case it must be terminated by a semicolon or it may be a compound-statement in which case it should not be terminated by a semicolon:

|  |  |
| --- | --- |
| Simple statement while-body | Compound-statement while-body |
| while(condition)  statement; | while(condition)  compound-statement |
|  | |

Examples:

|  |  |
| --- | --- |
| while-loop | output |
| int n = 1;  while(n <= 10){  System.out.printf("%d ", n);  n += 2;  } | 1 3 5 7 9 |
| int k = 12;  while(k > 6)  System.out.printf("%d ", k--); | 12 11 10 9 8 7 |
| int num = 2;  System.out.println("number\tsquare root\tsquare\n");  while(num <= 5){  System.out. printf("%d\t%.2f\t%2d\n", num, Math.sqrt(num),  num \* num);  num++;  } | number square root square  2 1.41 4  3 1.73 9  4 2.00 16  5 2.24 25 |

* **Sentinel controlled loops**

In a program, a sentinel is a value that marks the end of a series of data values; but is not a data value itself. Sentinels may be used to control loops:

Example: Write a Java program fragment that prompts for and reads student grades in a quiz. It then calculates and displays the average. Use a negative value or a value greater than 100 as the sentinel.

int count = 0;

double grade, sumOfGrades = 0.0;

System.out.printf("Enter grade#%d (0 < or > 100 to terminate): ", count+1);

grade = scanner.nextDouble();

while(grade >= 0 && grade <= 100){

count++;

sumOfGrades += grade;

System.out.printf("Enter grade#%d (0 < or > 100 to terminate): ", count+1);

grade = scanner.nextDouble();

}

if(count **==** 0)

System.out.println("Error: No valid grade entered");

else

System.out.printf("Average = %.2f%n", sumOfGrades / count);

**Exampe:** Modify the above Java program fragment such that it also displays the maximum and minimum grade.

int count = 0;

double grade, sumOfGrades = 0.0;

System.out.printf("Enter grade#%d (0 < or > 100 to terminate): ", count+1);

grade = scanner.nextDouble();

double max = grade; **// Assume the first grade is the max**

double min = grade; **// Assume the first grade is the min**

while(grade >= 0 && grade <= 100){

count++;

sumOfGrades += grade;

if(grade > max)

max = grade;

else if(grade < min)

min = grade;

System.out.printf("Enter grade#%d (0 < or > 100 to terminate): ", count+1);

grade = scanner.nextDouble();

}

if(count **==** 0)

System.out.println("Error: No valid grade entered");

else{

System.out.printf("Average = %.2f%n", sumOfGrades / count);

System.out.printf("Maximum = %.2f%n", max);

System.out.printf("Minimum = %.2f%n", min);

}

* **do-while statement**

A do-while statement is used to execute a statement or a compound-statement one or more times as long as the do-while condition is true:

**do**

**do-while body**

**while(condition);**

The do-while body can be a simple statement in which case it must be terminated by a semicolon or it may be a compound-statement in which case it MUST NOT be terminated by a semicolon:

|  |  |
| --- | --- |
| Simple statement do-while body | Compound-statement do-while body |
| do  statement;  while(condition); | do  compound\_statement  while(condition); |
|  | |

Examples:

|  |  |
| --- | --- |
| do-while loop | output |
| int n = 1;  do{  System.out.printf("%d ", n);  n += 2;  } while(n <= 10); | 1 3 5 7 9 |
| int x = 25;  do  System.out.printf("%d ", x -= 5);  while(x > 0); | 20 15 10 5 0 |

**Example**: Write a Java program fragment that helps a child learn multiplication. The fragment displays two integers to be multiplied. The child is then given three chances to provide the correct answer.

int num1 = 7, num2 = 6, response, count = 0;

System.out.printf("What is %d \* %d ?\n", num1, num2);

do{ response = scanner.nextInt(); count++;

if((response != num1 \* num2) && (count < 3))

System.out.println("Wrong response. Please try again");

} while((response != num1 \* num2) && (count < 3 ));

if(response != num1 \* num2)

System.out.println("Sorry ! You were not successful");

else

System.out.println("Congratulations ! You got the correct answer");

**Example:** A do-while loop can be used to validate input

int n;

do{

System.out.println("Enter an integer number in the [10,100] interval: ");

n = scanner.nextInt();

if(n<10 || n>100)

System.out.println("Sorry wrong input, try again");

}while (n < 10 || n > 100);

System.out.println("Now your input is correct");

**Example:** Using a do-while loop to recover from Scanner **java.util.InputMismatchException**

**import java.util.\*;**

**public class ErrorRecovery {**

**public static void main(String[ ] args){**

**int num = 0;**

**boolean inputMismatch;**

**Scanner scanner = new Scanner(System.in);**

**do{**

**System.out.println("Enter an integer number in the [10,100] interval: ");**

**try{**

**num = scanner.nextInt( );**

**inputMismatch = false; // statements up to end of try-block Not executed if there is Exception**

**if(num < 10 || num >100) //**

**System.out.println("Sorry input outside the range [10, 100], try again"); //**

**}**

**catch(InputMismatchException e){**

**System.out.println("Error: " + e);**

**inputMismatch = true;**

**scanner.nextLine( ); // clear the input buffer; otherwise the loop becomes infinite**

**}**

**}while (inputMismatch || num < 10 || num > 100);**

**System.out.println("Now your input is correct");**

**}**

**}**

A do-while loop can be used in a menu driven program. The example shown below will continue running as long as the user did not enter the number 5.

int choice;

do{System.out.println("1-Addition");

System.out.println("2-Subtraction");

System.out.println("3-Multiplication");

System.out.println("4-Division");

System.out.println("5-Exit");

System.out.println("Enter your choice: ");

choice = scanner.nextInt();

// Here come the statements to do the different tasks

} while (choice != 5);

* **for-statement**

A for-statement is usually used to repeat a statement or a compound-statement if the number of repetitions is known. The body of a for-loop may be executed zero or more times.

**for(initialization; condition; update)**

**for-body**

The for-body can be a simple statement in which case it must be terminated by a semicolon, or it may be a compound-statement in which case it should not be terminated by a semicolon:

|  |  |
| --- | --- |
| Simple statement for-body | Compound-statement for-body |
| for(initialization; condition; update)  statement; | for(initialization; condition; update)  compound\_statement |
|  | |

Examples:

|  |  |
| --- | --- |
| for-loop | output |
| int k;  for(k = 7; k <= 12; k++)  System.out.printf("%d ", k); | 7 8 9 10 11 12 |
| int x ;  for(x = 8; x > 2; x--)  if( x % 2 == 0)  System.out.printf("%d is even%n", x);  else  System.out.printf("%d is odd%n", x); | 8 is even  7 is odd  6 is even  5 is odd  4 is even  3 is odd |
| int counter = 1;  int factorial = 1;  for(counter = 1; counter <= 5; counter++)  factorial = factorial \* counter;  System.out.printf("factorial of 5 is %d", factorial); | factorial of 5 is 120 |
| // prints 6 characters per line  int count = 1;  for(char ch = 'A'; ch <='Z'; ch++){  System.out.printf("%c\t",ch);  if(count % 6 == 0)  System.out.println();  count++;  } | A B C D E F  G H I J K L  M N O P Q R  S T U V W X  Y Z |

Loops like the ones in the above examples in which a variable is used to control the number of

repetitions of the loop are called **counting loops**.

**Equivalent loops**

A loop can always be converted to an equivalent loop of a different type. Examples:

|  |  |
| --- | --- |
| loop | equivalent loop |
| for(initialization;condition; update)  {  statement1;  statement2;  . . .  statementN;  } | initialization;  while(condition)  {  statement1;  statement2;  . . .  statementN;  update;  } |
| do{  statement1;  statement2;  . . .  statementN;  update;  } while(condition); | statement1;  statement2;  . . .  statementN;  update;  while(condition){  statement1;  statement2;  . . .  statementN;  update;  } |

For example the following do-while loop:

int n = 1;

double x = 0, s;

do{

s = 1.0 / (n \* n);

x = x + s;

System.out.printf("x = %f%n", x);

n++;

} while(s > 0.01);

is equivalent to the while-loop:

int n = 1;

double x = 0, s;

s = 1.0 / (n \* n);

x = x + s;

System.out.printf("x = %f%n", x);

n++;

while(s > 0.01){

s = 1.0 / (n \* n);

x = x + s;

System.out.printf("x = %%n", x);

n++;

}

* **Nested loops**

A loop statement may contain in its body one or more loop statements. There are two types of nested loops: independent and dependent. A nested loop is independent if its number of repetitions does not depend on an outer loop. A nested loop is dependent if its number of repetitions depends on an outer loop.

Examples:

|  |  |
| --- | --- |
| Nested loops | output |
| //Example of independent nested loop  int m, n;  for(m = 5; m >= 1; m--){  System.out.printf("m is now %d%n", m);  for(n = 1; n <= 4; n++)  System.out.printf("n = %d ", n);  System.out.println();  } | m is now 5  n = 1 n = 2 n = 3 n= 4  m is now 4  n = 1 n = 2 n = 3 n= 4  m is now 3  n = 1 n = 2 n = 3 n= 4  m is now 2  n = 1 n = 2 n = 3 n= 4  m is now 1  n = 1 n = 2 n = 3 n= 4 |
| //Example of dependent nested loop  int k,m;  for(k=1;k<=9;k++){  for(m = 1; m <= k; m++){  System.out.printf("%d",k);  }  System.out.println();  } |  |

**Example**: Write a Java program fragment that prompts for and reads three quiz grades for each student in a class of four students. The program then computes and displays the average for each student. Your program must be easily modifiable to handle any number of students and quizzes.

final int NUMSTUDENTS = 4, NUMQUIZES = 3;

double grade, studentTotal, studentAverage;

int m, n;

for(m = 1; m <= NUMSTUDENTS; m++)

{

studentTotal = 0.0;

for(n = 1; n <= NUMQUIZES; n++)

{

System.out.printf("Enter QuizGrade#%d for student#%d%n", n, m);

grade = scanner.nextDouble();

studentTotal += grade;

}

studentAverage = studentTotal / NUMQUIZES;

System.out.printf("The average for student#%d is %.2f%n", studentAverage);

}

**String processing using loops**

**Example1**: Count the number of uppercase characters in a string referenced by **str**:

|  |  |
| --- | --- |
| int count = 0;  for(int k = 0; k < str.length(); k++){  if(Character.isUpperCase(str.charAt(k))  count++;  } | int count = 0; char ch;  for(int k = 0; k < str.length(); k++){  ch = str.charAt(k);  if(ch >= 'A' && ch <= 'Z')  count++;  } |

**Example2**: Assuming a string referenced by **str** has 11 or more characters, create a substring of the string from index 3 to 10:

|  |  |
| --- | --- |
| int count = 0;  String substring;  substring = str.substring(3, 11); | String substring = "";  for(int k = 3; k <= 10 ; k++){  substring += str.charAt(k);  } |

**Example3**: Find the last index of the blank character in a string referenced by **str**:

|  |  |
| --- | --- |
| int index;  index = str.lastIndexOf(' '); | int index = -1;  for(int k = str.length() - 1; k >= 0 ; k--){  if(str.charAt(k) == ' '){  index = k;  break;  }  } |

**Laboratory Tasks**

**Task 1:** Write a Java program that uses a sentinel-controlled loop which does the following as long as the sentinel, **volume <= 0**, has not been read:

* It Prompts for and reads the **volume** of a sphere, it then displays the **surface area** of the sphere with that volume. Assume that each volume is in cubic centimeters.

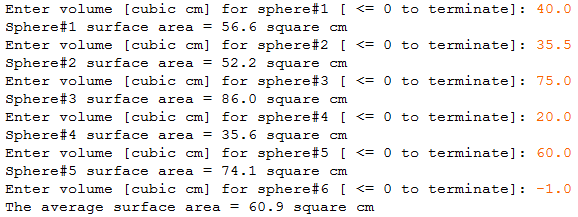
[, ]

When the sentinel is read, the program displays the average of the surface areas.

Note:

* Assume that the value entered for each **volume** is valid.

Sample program run:



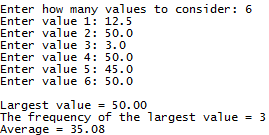
**Task 2:** Write a Java program that given a collection of Nnumbers will find the largest value, the frequency of the largest value (i.e., the number of times the largest value appears in the collection), and the average of the **N** numbers.

1. Get the value of N from the user.
2. If N <= 0 display an appropriate error message and terminate the program; otherwise
3. Read the values as entered from the user. (If N =5, then there are 5 values the user is going to enter).
4. Find the Largest, the frequency of the largest value, and the average of these N values.

**Hint:** To find the largest value, read the first value **before the loop** then assume that this first value entered by the user is the largest value and its frequency is 1. **Within the loop** compare the assumed largest value with other values **read inside the loop** and change the largest value and frequency accordingly.

**Note:** Your solution must not use concepts not covered in ICS 102 so far, like arrays and data files.

Sample program run:



**Task 3:** Write an interactive Java program that prompts for and reads a positive integer, it then displays the positive integer in reverse order. Your program must recover from input error by displaying an error message and looping as long as the input is zero or negative.

**Note:** Your **program** MUST NOT reverse the number; it is only required to **display** the number in reverse.

Sample program runs:

|  |  |
| --- | --- |
|  |  |

Hint: Use quotient and remainder of integer division by 10

**Task 4:** The value for for any x in the interval (0, 2]can be determined by the Taylor series:

Write a Java program to approximate the value of using **n** terms of the above series, where **n** is a positive integer obtained from the user. If n <= 0, display an appropriate error message and terminate the program. Otherwise prompt the user to enter the value of x. If the value of x is not in the interval (0, 2], display an appropriate error message and terminate the program. If the input is valid the program calculates and displays the approximation to. The program must also display the value of ln(x) by calling the standard mathematical function **Math.log(x)**.

**Note:**

Sample program runs:

|  |
| --- |
|  |
|  |
|  |

**Task 5:** Write a Java program that prompts for and reads two strings referenced by **string1** and **string2** each with distinct characters. It then writes the number of lowercase characters of the string referenced by **string1** that are in the string referenced by **string2** followed by those characters, if any.

Sample program runs:

|  |
| --- |
|  |
|  |
|  |
|  |

**Task 6:** Write a Java program that prompts for and reads an integer **n1** in the range [8, 15], it then prompts for and reads an integer **n2** in the range [3, 6]. It finally generates and displays a multiplication table whose rows start from **n1** to **1**, and whose columns start from **n2** to **1** as shown in the sample program runs below. If the input is not valid, your program must display an appropriate error message and terminate.

|  |
| --- |
|  |
|  |
|  |