CONTROL FLOW STATEMENTS

Introduction

When you write a program, you type statement into a file. Without control flow statements (CFS), the interpreter executes these statements in the order they appear in the file from left to write, top to bottom.

Basically, there are 3 types of control structures in a programming language.

Types of CFS

Type of control	Meaning	Control structure
Sequential control	The program runs from line one to the last line without branching or testing for any condition	All the programs we have been writing before now are examples of this structure.
Selection/Branching	One or more conditions are tested. Statements are executed upon fulfillment of the conditions	Simple if, if else,else if, goto, nested if, arithmetic if, Switch, break and Continue structures
Looping/ Repetition Iteration	Statements are repeatedly executed either on fulfilling a condition or for some number of times	For, while and do-while loops.

Using CFS

U can use control flow statements in your programs:

- To conditionally execute statements
- To repeatedly execute a block of statements
- To otherwise change the normal sequential flow .

For example:

The if statement conditionally executes the System.out.println statement within the braces based on the return value of isUpperCase()

```
Character.isUpperCase(aChar):
char c;
.....
    if Character.isUpperCase(aChar)){
      System.out.println(“the character “ +aChar
   +” is upper case.”);
}
```

Java PL. CFS

The Java PL provides several control flow statements, which are listed in the following table.

Statement Type	Keyword
Looping/Repetition	while, do-while, for
Decision making/selection	if-else, switch-case
Exception handling	try-catch-finally, throw
Branching	break,continue,label:,return

if statements

The if statements enables your program to selectively executes other statements, based on some criteria.

Generally, the syntax of simple if statement goes thus:

```
if(logical expression){
    statement(s)
}
```

The statement(s) will only be executed if the logical expression evaluates to true. If the expression is not true, the control simply passes to the next statement following the statements block,{….} and continues downwards from there.

*if/*statements(2)

Ex:

```
If (x<50){
   System.out.println(“the score is below cut
   off”);
 }
```

The *else* statement

To perform a different set of statements if the expression is false, we use the *else* statement for that.

Example:

The *else* block is executed if the '*if*' part is false.

```
System.out.println("Enter the grade:");
String X = input.readLine();
int grade = Integer.parseInt(X);
if(grade>=50){
System.out.println("The grade is above cut off point");
}
else{
System.out.println("Grade below cut off point");
}
```

The *else if* statement

Another form of the else statement , else if, executes a statement based on another expression. An *if* statements can have a number of companion *else if* statements but only one *else*.

Example:

```
public class IfElse{
   public static void main(Strings args){
        int testscore = 76;
        char grade;
        if(testscore >= 70){
          grade = 'A';
        }
        else if(testscore >= 60){
          grade = 'B';
        }
        elseif(testscore >= 50){
          grade = 'C';
        }
        elseif(testscore >= 45){
          grade = 'D';
        }
        elseif(testscore >= 40){
          grade = 'E';
        }
        else{
          grade = 'F';
        }
        System.out.println("Grade="+grade);
     }
   }
```

- Note: You may have noticed that the value of the testscore satisfy more than one expression in the compound if statement.

76>=70 and 76>= 50.

However as the run time processes a compund statement such as this one, once a condition is satisfied, the appropriate statements are executed (grade = ‘A;), and control passes out of the if statement without evaluating the remaining conditions.

Exercise

Write a program to compute the roots of quadratic equation

ans

```
Importjava.util.Scanner;
class quadratic{
public static void main(Strings args){
    Scanner input = new Scanner(System.in);
    //getting the values of the coefficients a, b and c


        System.out.println("Enter the value of a");
        float a = input.nextFloat();
        System.out.println("Enter the value of b");
        float b = input.nextFloat();
        System.out.println("Enter the value of c");
        float c = input.nextFloat();
        //computing the discriminant d
        double d = b*b-4.0*a*c;



}
```

Ans(2)

```
//testing for solution path
          if(d<0){
                    System.out.println("Complex roots");
          }
          else if(d==0){
                    System.out.println("Only one real root exists with value….");
                    double x = (-b)/2*a);
                    System.out.println(x);
          }
          else{
                    double x1 = ((-b) + Math.sqrt(d) )/2*a;
                    double x2 = ((-b) - Math.sqrt(d) )/2*a;
                    System.out.println("two real root exists with value….");
                    System.out.println(x1+ "and" +x2};
          }
    System.exit(0);
}
}
```

The *for* statement

The for statement provides a compact way to iterate over a range of values. It is sued when we know ahead the no of times a section of code is to be repeated.

The general form of the for statement can be expressed like this:

```
for(initialization; termination/continuation criterion;increment){
statements;
}
```

The *for* statement(2)

The *initialization*: is an expression that initializes the loop. It is executed once at the beginning of the loop.

The *termination/continuation* criterion: determines when to terminate the loop. It gives the condition to continuing the loop until when the loop will terminate. When the expression evaluates to false, the loop terminates.

The increment is an expression that gets invoked after each iteration through the loop.

Example

```
public class AddOddNos{
    public static void main (String[] args) {
        int sum = 0;
        for(int I = 1;i<=10;i++){
          sum += i;
        }
        System.out.println(“Sum = ” +sum);
    }
}
```

Output = ?

Exercise

Write a program to sum all odd numbers together from 1 to n.

Solution

```
importjava.util.Scanner;
public class Sum_OddNos{
   public static void main(Strings args){
       Scanner input = new Scanner(System.in);
       System.out.println("Enter the maximum no to
   compute ");
       int n = input.nextInt();
       int sum = 0;
       for(int i = 1;i<=n;i+= 2){
          sum += i;
       }//next i
       System.out.println("Sum of odd from 1 to " +n
   +"= "+sum);
   }
}
```