**Control Statements**

All the programs we have written till now had a sequential flow of control i.e. the statements were executed line by line from the top to bottom in an order. Nowhere were any statements skipped or a statement executed more than once. We will now look into how this can be achieved using control structures. These control structures can be classified into three groups: Decision making statements, repetition statements and branching statements. Decision making statements are used to decide whether or not a particular statement or a group of statements should be executed. Repetition statements are used to execute a statement or a group of statements more than once. Branching statements are used to transfer control to another line in the code. We shall look into the syntaxes of these structures first and then see how we formulate algorithms using these statements.

We have already seen the if statement and the if else statement while writing the set methods for our Student class. The if statement is used to selectively execute a statement or a block of statements. The if statement contains a condition which decides whether or not the statements would be executed. If the condition stated in the parentheses evaluates to true, then the statements are executed, otherwise the statements in the if block are skipped. Conditions, as we have already seen are built using conditional and relational operators. Following is the syntax of the if statement.

if ( condition ) {  
// code  
}

And here is a small code snippet which displays the message "Hi" only if the value of the boolean variable wish is true.

if ( wish == true ) {  
    System.out.println("Hi");  
}

In the above code snippet, the braces could have been omitted. Braces are not required when the block consists of only a single statement. But we recommend that braces always be placed. This ensures that mistakes won't creep into the code when you modify your program at a later time to add another statement to the if block forgetting to insert the necessary braces. Moreover the condition 'wish==true' could have also been written as 'wish'. Both are equivalent and perform the same condition check.

The if statement is known as a single selection structure since we have only a single statement that is either executed or not. No alternative statement would be executed if the condition evaluates to false. For such purpose, we have the if else double selection structure. If the condition is true, the if block will be executed. Otherwise, the else block will be executed. Following is the syntax of if else structure.

if ( condition ) {  
    // code  
} else {  
    // code  
}

And here is an example usage which states whether a number is an even number or an odd number. Even numbers are divisible by zero and hence leave a remainder of zero. The remainder on dividing the number by two can be obtained using the modulo (%) mathematical operator.

if ( num%2==0){   
    System.out.println("Number is even");  
}else{  
    System.out.println("Number is odd");  
}

if else statements can be nested. That is the statements block of an if or else can in turn contain another if else structure. And moreover, an else is always associated with the immediately preceding else. These two concepts are used to form a chained if else structure in which only one of the several alternatives is executed. Look at the following code where if else structures have been nested to print the number stored in the int num in words when the value of num is between one and three. If the number falls outside this range, an appropriate message is displayed.

if (num == 1) {  
    System.out.println("One");  
} else {  
    if (num == 2) {  
        System.out.println("Two");  
    } else {  
        if (num == 3) {  
            System.out.println("Three");  
        } else {  
            System.out.println("Numbers greater than three cannot be processed by this code");  
        }  
    }  
}

This code looks highly unreadable. One thing that we can do is to remove the redundant braces since each of the blocks contain no more than one statement. You may be hesitant to perform this action with the first else statement since it contains more than a single line of code. But all that would still fall under the else statement even if we remove the braces. This is because the statement within the if block is associated with the if header and the succeeding else is associated with the if block. So the entire block will be associated with the else even if no braces are used. So, after we remove the braces, this is what we are left with. We have also written the words if and else on the same line.

if (num == 1) {  
    System.out.println("One");  
} else if (num == 2) {  
    System.out.println("Two");  
} else if (num == 3) {  
    System.out.println("Three");  
} else {  
    System.out.println("Numbers greater than three cannot be processed by this code");  
}

This code looks much better than the earlier once. This is what we refer to as chained if else structure formed using a number of if and else statements. Only one out of the four statements will be executed depending on the value of num. Once a particular statement is executed, the remaining ones would be ignored. This chained control structure is quite frequently used.

We could have used a set of four if statements instead of the above chained sequence but that that code would take a longer time to execute. This is because all the conditions would be checked. But in this chained structure, one a condition evaluates to true, all other statements are skipped which would reduce processing time, hence, making the code more efficient.

As already said, an else is associated with the immediately preceding if. Indenting the code by using tabs has no effect on how the code works. Assuming that the indentation matters may lead to errors in programming. Consider the following

code:

if(x>3)  
 if(y>3)  
  System.out.println("Both x and y are greater than 3");  
else  
 System.out.println("x is not greater than 3");

By looking at the indentation one might be tempted to say that the last else is associated to the first if. But this is erroneous since an else if always associated with the immediately preceding if unless braces are used to associate the else with a different if. This problem of which if the final else is to be associated is known as the dangling else problem. If we wrongfully assume that indentation matters, the result obtained would be incorrect. Include the above code in a program and set x and y to 2. According to what one might think, the statement "x is not greater than 3" is not printed. Given below is a corrected version of the above code.

if(x>3) {   
 if(y>3)  
  System.out.println("Both x and y are greater than 3");  
} else  
 System.out.println("x is not greater than 3");

In order to avoid such trivial mistakes, always use braces unless you are writing a chained if else structure in the case of which using braces makes the code clumsy and hard to decipher.

Apart from the if and else statements, we have the switch structure. The switch control structure however cannot be used in the situations where the condition needs to contain relational operators like greater than, less than and so on. The switch structure can only be used to check for equality. Given below is the syntax of the switch structure in Java.

switch ( expression )  
{  
 case value1 :  
  // code  
  break;  
 case value2:  
  // code  
  break;  
...  
...  
 case valueN:  
  // code  
  break;  
...  
...  
 default:   
  // code  
  break;  
}

The expression should evaluate to one of the following types: byte, short, int, char, String. This value is compared with each of the case values ( value1, value2,... valueN). When a match is found, the statements under that particular case label are executed. Strings are checked for equality by invoking the equals() method and not by using the == operator which is the used for the other data types. If no match is found, the statements under the default label are executed. The break keyword is used to transfer control outside the switch block. If the break keyword is omitted, the statements under the other case labels are also excited whether or not the case value matches. This might be useful in certain situations which we will look into shortly. The last break under the default label is optional as on reaching the line, the control has to anyway move out of the switch block. All the case values should be unique. Duplicates are not allowed. Also note that braces are not required even if we wish to include more than one statement in a particular case label. However, using them doesn't make any difference. Given below is an example which prints the day of the week based on the value stored in the int variable num. 1 stands for Sunday, 2 for Monday and so on:

int day = s.nextInt; // s is a Scanner object connected to System.in  
String str; // stores the day in words  
switch(day)  
{  
 case 1:  
  str="Sunday";  
  break;  
    case 2:  
        str="Monday";  
        break;  
    case 3:  
        str="Tuesday";  
        break;  
    case 4:  
        str="Wednesday";  
        break;  
    case 5:  
        str="Thursday";  
        break;  
    case 6:  
        str="Friday";  
        break;  
    case 7:  
        str="Saturday";  
        break;  
    default:  
        str=" Invalid day";  
}  
System.out.println(str);

The value stored in day is compared with each of the case values 1 to 7. If a match is found, the statements under the corresponding case label are executed. And since, break is encountered, the remainder of the code in the switch structure so skipped. If the value stored in day does not fall in the range 1-7, the default statement is executed.

As already said, at times, there might be a need to omit the break statement under certain situations where we want the same set of statements to be executed as a target of more than a single case label. Look at the following example where the season is printed deepening on the month.

switch(month)  
{  
    case 12:  
    case 1:  
    case 2:  
        System.out.println(" Winter");  
        break:  
    case 3:  
    case 4:  
    case 5:  
        System.out.println(" Spring");  
        break:  
    case 6:  
    case 7:  
    case 8:  
        System.out.println(" Summer");  
        break:  
    case 9:  
    case 10:  
    case 11:  
        System.out.println(" Autumn");  
        break:  
    default:  
        System.out.println(" Invalid month");  
}

Suppose we have the value of month as 4. This value would be compared each of the case values 12 to 4. The value of month equals the case value 4. Therefore statements under this case label would be executed until a break is encountered. However, note that in the statements that follow, the value of month is not compared with any of the case values. In this example, the value of month would not be compared with 5. The next print() statement is executed. After this, the break statement is encountered because of which, control shifts outside the switch structure.

The last of the decision making statements is the ternary operator which provides a concise form for short and simple if lese statements. The ternary operator is represented by ?: Following is the syntax of ternary operator:

expression1 ? expreession2 : expression3

If expression1 evaluates to true, then expression 2 is evaluated, otherwise expression 3 is evaluated. The ternary operators can be used as a standalone statement or in conjugation with other statements like declaration and print statements. Following examples illustrate the use of ternary operator:

Example 1: The value of s (an int ) is either incremented or decrement based on whether it is a positive integer or not.

s>0 ? s++ :  s-- ;

If s is 3, then s is incremented to 4. If s is 0, it is decremented to -1.

Example 2: Division

If the divisor is zero, then the quotient is set to zero, otherwise division is performed and the quotient is stored in the variable res. a and b are the dividend and divisor respectively.

int res = b==0 ? 0 : a/b;

if b is zero, then the expression b==0 evaluates to true, hence the first expression is evaluated. 0 is the result of the ternary operation which is assigned to res.

If b is non zero, expression1 evaluates to true. Hence expression 3 is evaluated i.e. the ternary operation returns the result of dividing a by b. This returned value is assigned to num.

Example 3: Ternary operator in print statements

The following statement prints either "Positive "or "Non positive "depending on the value of num.

System.out.println( num?0:"Positive":"Non positive");

Example 4: Passing arguments to methods

The following statement passes the value 3 or 0 to the method Math.sqrt() depending on the result of the .equals() method call on str.

double root = Math.sqrt ( str.equals("three") ? 3: 0 );

if str equals "three", then the expression evaluates to zero because of which the ternary operator returns 3 which is passed as an argument to Math.sqrt() function. In other cases, 0 is passed to the function.

An important thing to be kept in mind while using control structures is that any variable that is declared within a block is not visible outside that block. If results processed within such blocks are to be accessible outside the blocks, then such values should be stored in variables that have been declared outside the blocks.

Look at the following code:

if ( num % 2 == 0){  
    boolean res =true;  
} else {  
    boolean res = false;  
}  
System.out.println(res);

The above code would produce compilation errors. Given below is a corrected version of the above code.

boolean res;  
if ( num % 2 == 0) {  
    res = true;  
} else {   
    res = false;  
}  
System.out.println(res);