

## **Control Statements**

Control statements are the statements which alter the flow of execution and provide better control to the programmer on the flow of execution. In Java control statements are categorized into selection control statements, iteration control statements and jump control statements

A programming language uses *control* statements to cause the flow of execution to advance and branch based on changes to the state of a program. Java's program control statements can be put into the following categories: selection, iteration, and jump.

*Selection* statements allow your program to choose different paths of execution based upon the outcome of an expression or the state of a variable. *Iteration* statements enable program execution to repeat one or more statements (that is, iteration statements form loops). *Jump* statements allow your program to execute in a nonlinear fashion. All of Java's control statements are examined here.

➤ **Java's Selection Statements:** Java supports two selection statements: *if* and *switch*. These statements allow us to control the flow of program execution based on condition.

**if Statement:** *if* statement performs a task depending on whether a condition is true or false

**Syntax:** *if* (*condition*)

*statement1*;

*else*

*statement2*;

Here, each statement may be a single statement or a compound statement enclosed in curly braces (that is, a block). The condition is any expression that returns a boolean value. The *else* clause is optional

**Program 1:** Write a program to find biggest of three numbers.

```
//Biggest of three numbers
class BiggestNo
{
    public static void main(String args[])
    {
        int a=5,b=7,c=6;
        if ( a > b && a>c)
            System.out.println ("a is big");
        else if ( b > c)
            System.out.println ("b is big");
        else
            System.out.println ("c is big");
    }
}
```

**Output:**

b is big

**Switch Statement:** When there are several options and we have to choose only one option from the available ones, we can use switch statement

**Syntax:**

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

Here, depending on the value of the expression, a particular corresponding case will be executed.

**Program 2:** Write a program for using the switch statement to execute a particular task depending on color value.

```
//To display a color name depending on color value
class ColorDemo
{
    public static void main(String args[])
    {
        char color = 'r';
        switch (color)
        {
            case 'r': System.out.println ("red");           break;
            case 'g': System.out.println ("green");        break;
            case 'b': System.out.println ("blue");         break;
            case 'y': System.out.println ("yellow");       break;
            case 'w': System.out.println ("white");        break;
            default: System.out.println ("No Color Selected");
        }
    }
}
```

**Output:**

red

➤ **Java's Iteration Statements:** Java's iteration statements are for, while and do-while. These statements are used to repeat same set of instructions specified number of times called loops. A loop repeatedly executes the same set of instructions until a termination condition is met.

o **while Loop:** while loop repeats a group of statements as long as condition is true. Once the condition is false, the loop is terminated. In while loop, the condition is tested first; if it is true, then only the statements are executed. while loop is called as entry control loop.

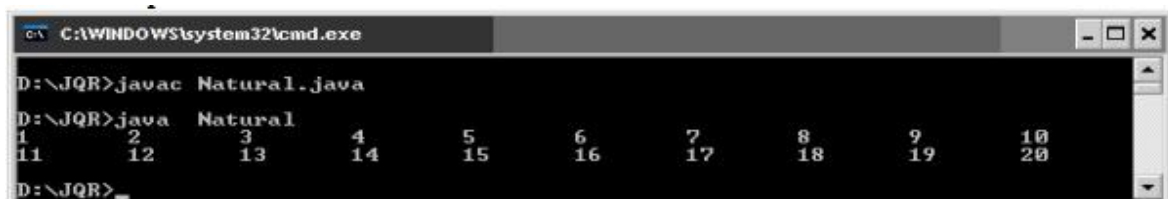
**Syntax:**

```
while (condition)
{
    statements;
}
```

**Program 3:** Write a program to generate numbers from 1 to 20.

```
//Program to generate numbers from 1 to 20.
class Natural
{
    public static void main(String args[])
    {
        int i=1;
        while (i <= 20)
        {
            System.out.print (i + "\t");
            i++;
        }
    }
}
```

**Output:**



o **do...while Loop:** do...while loop repeats a group of statements as long as condition is true. In do...while loop, the statements are executed first and then the condition is tested. **do...while** loop is also called as exit control loop

**Syntax:**

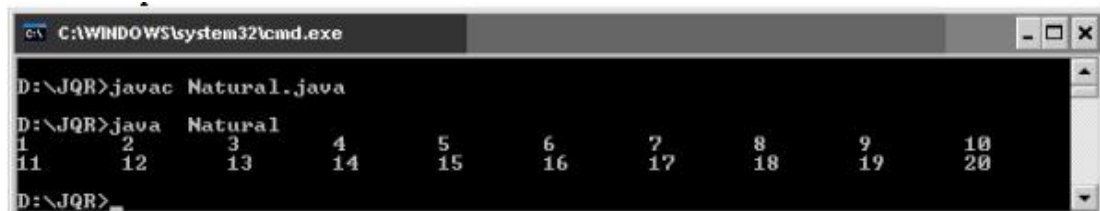
```
do
{
    statements;
} while (condition);
```

**Program 4:** Write a program to generate numbers from 1 to 20.

//Program to generate numbers from 1 to 20.

```
class Natural
{
    public static void main(String args[])
    {
        int i=1;
        do
        {
            System.out.print (i + "\t");
            i++;
        } while (i <= 20);
    }
}
```

**Output:**



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Natural.java
D:\JQR>java Natural
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
D:\JQR>
```

o **for Loop:** The for loop is also same as do...while or while loop, but it is more compact syntactically. The for loop executes a group of statements as long as a condition is true.

**Syntax:**

```
for (expression1; expression2; expression3)
{
    statements;
}
```

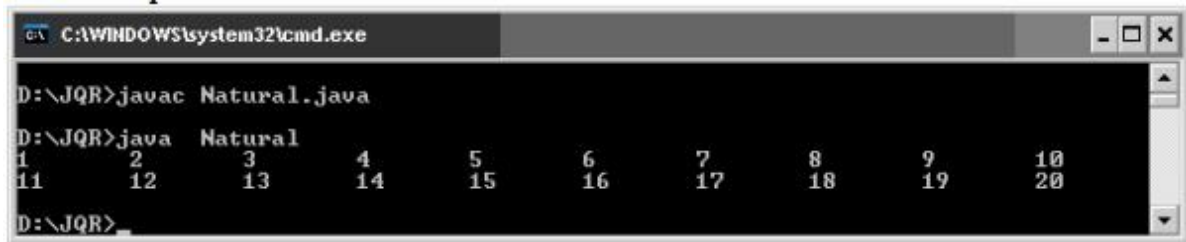
Here, expression1 is used to initialize the variables, expression2 is used for condition checking and expression3 is used for increment or decrement variable value

**Program 5:** Write a program to generate numbers from 1 to 20.

//Program to generate numbers from 1 to 20.

```
class Natural
{
    public static void main(String args[])
    {
        int i;
        for (i=1; i<=20; i++)
            System.out.print (i + "\t");
    }
}
```

**Output:**



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Natural.java
D:\JQR>java Natural
1      2      3      4      5      6      7      8      9      10
11     12     13     14     15     16     17     18     19     20
D:\JQR>
```

**Java’s Jump Statements:** Java supports three jump statements: break, continue and return. These statements transfer control to another part of the program.

**o break:**

- break can be used inside a loop to come out of it.
- break can be used inside the switch block to come out of the switch block.
- break can be used in nested blocks to go to the end of a block. Nested blocks represent a block written within another block.

**Syntax:** break; (or) break label;//here label represents the name of the block.

**Program 6:** Write a program to use break as a civilized form of goto.

//using break as a civilized form of goto

```
class BreakDemo
{
    public static void main (String args[])
    {
        boolean t = true;
        first:
        {
            second:
            {
                third:
                {
                    System.out.println ("Before the break");
                    if (t) break second; // break out of second block
                    System.out.println ("This won't execute");
                }
                System.out.println ("This won't execute");
            }
            System.out.println ("This is after second block");
        }
    }
}
```

**Output:**



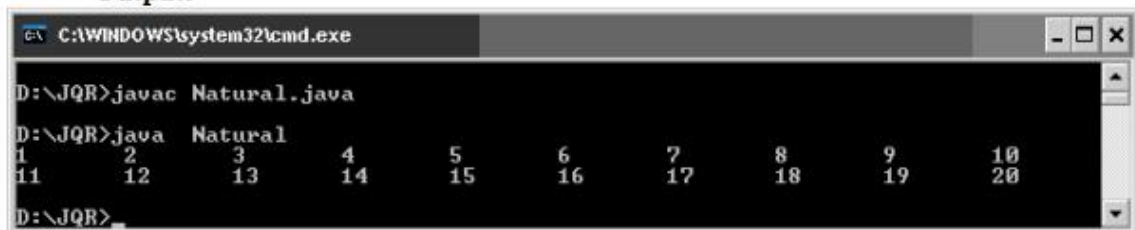
```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac BreakDemo.java
D:\JQR>java BreakDemo
Before the break
This is after second block
D:\JQR>
```

**continue:** This statement is useful to continue the next repetition of a loop/ iteration. When continue is executed, subsequent statements inside the loop are not executed.      **Syntax:** continue;

**Program 7:** Write a program to generate numbers from 1 to 20.

```
//Program to generate numbers from 1 to 20.
class Natural
{
    public static void main (String args[])
    {
        int i=1;
        while (true)
        {
            System.out.print (i + "\t");
            i++;
            if (i <= 20 )
                continue;
            else
                break;
        }
    }
}
```

**Output:**



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Natural.java
D:\JQR>java Natural
1      2      3      4      5      6      7      8      9      10
11     12     13     14     15     16     17     18     19     20
D:\JQR>
```

**o return statement:**

- return statement is useful to terminate a method and come back to the calling method.
- return statement in main method terminates the application.
- return statement can be used to return some value from a method to a calling method.

**Syntax:**                   return;  
                                 (or)  
                                 return value; // value may be of any type

**Program 8:** Write a program to demonstrate return statement.

```
//Demonstrate return
class ReturnDemo
{
    public static void main(String args[])
    {
        boolean t = true;
        System.out.println ("Before the return");
        if (t)
            return;
        System.out.println ("This won't execute");
    }
}
```

**Output:**



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac ReturnDemo.java
D:\JQR>java ReturnDemo
Before the return
D:\JQR>
```

**Note:** goto statement is not available in java, because it leads to confusion and forms infinite loops.

### Accepting Input from Keyboard

A stream represents flow of data from one place to other place. Streams are of two types in java. Input streams which are used to accept or receive data. Output streams are used to display or write data. Streams are represented as classes in java.io package.

- **System.in:** This represents InputStream object, which by default represents standard input device that is keyboard.
- **System.out:** This represents PrintStream object, which by default represents standard output device that is monitor.
- **System.err:** This field also represents PrintStream object, which by default represents monitor. System.out is used to display normal messages and results whereas System.err is used to display error messages.

### To accept data from the keyboard:

- Connect the keyboard to an input stream object. Here, we can use

InputStreamReader that can read data from the keyboard.

```
InputStreamReader obj = new InputStreamReader (System.in);
```

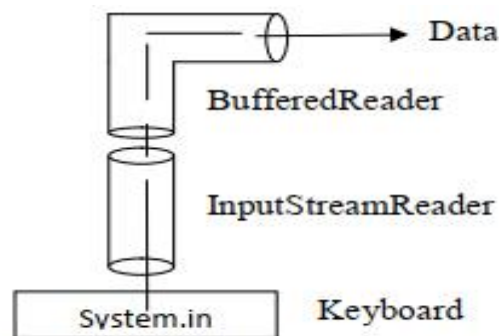
· Connect InputStreamReader to BufferedReader, which is another input type of stream. We are using BufferedReader as it has got methods to read data properly, coming from the stream.

```
BufferedReader br = new BufferedReader (obj);
```

The above two steps can be combined and rewritten in a single statement as:

```
BufferedReader br = new BufferedReader (new InputStreamReader  
(System.in));
```

· Now, we can read the data coming from the keyboard using read () and readLine () methods available in BufferedReader class.



**Figure: Reading data from keyboard**

### **Accepting a Single Character from the Keyboard:**

· Create a BufferedReader class object (br).

· Then read a single character from the keyboard using read() method as:

```
char ch = (char) br.read();
```

· The read method reads a single character from the keyboard but it returns its ASCII number, which is an integer. Since, this integer number cannot be stored into character type variable ch, we should convert it into char type by writing (char) before the method. int data type is converted into char data type, converting one data type into another data type is called type casting.

### **Accepting a String from Keyboard:**

· Create a BufferedReader class object (br).

· Then read a string from the keyboard using readLine() method as:

```
String str = br.readLine ();
```



· `readLine ()` method accepts a string from keyboard and returns the string into `str`. In this case, casting is not needed since `readLine ()` is taking a string and returning the same data type.

### **Accepting an Integer value from Keyboard:**

· First, we should accept the integer number from the keyboard as a string, using `readLine ()`

```
as: String str = br.readLine ();
```

· Now, the number is in `str`, i.e. in form of a string. This should be converted into an `int` by using `parseInt ()` method, method of `Integer` class as:

```
int n = Integer.parseInt (str);
```

If needed, the above two statements can be combined and written as:

```
int n = Integer.parseInt (br.readLine() );
```

· `parseInt ()` is a static method in `Integer` class, so it can be called using class name as `Integer.parseInt ()`.

· We are not using casting to convert `String` type into `int` type. The reason is `String` is a class and `int` is a fundamental data type. Converting a class type into a fundamental data type is not possible by using casting. It is possible by using the method `Integer.parseInt()`.

### **Accepting a Float value from Keyboard:**

· We can accept a float value from the keyboard with the help of the following statement: `float n = Float.parseFloat (br.readLine() );`

· We are accepting a float value in the form of a string using `br.readLine ()` and then passing the string to `Float.parseFloat ()` to convert it into float. `parseFloat ()` is a static method in `Float` class.

### **Accepting a Double value from Keyboard:**

· We can accept a double value from the keyboard with the help of the following statement:

```
double n = Double.parseDouble (br.readLine() );
```

· We are accepting a double value in the form of a string using `br.readLine ()` and then passing the string to `Double.parseDouble ()` to convert it into double.

parseDouble () is a static method in Double class.

**Program 1:** Write a program to accept and display student details.

```
// Accepting and displaying student details.
import java.io.*;
class StudentDemo
{
    public static void main(String args[]) throws IOException
    {
        // Create BufferedReader object to accept data
        BufferedReader br =new BufferedReader (new InputStreamReader (System.in));
        //Accept student details
        System.out.print ("Enter roll number: ");
        int rno = Integer.parseInt (br.readLine());
        System.out.print ("Enter Gender (M/F): ");
        char gender = (char)br.read();
        br.skip (2);
        System.out.print ("Enter Student name: ");
        String name = br.readLine ()
        System.out.println ("Roll No.: " + rno);
        System.out.println ("Gender: " + gender);
        System.out.println ("Name: " + name);
    }
}
```