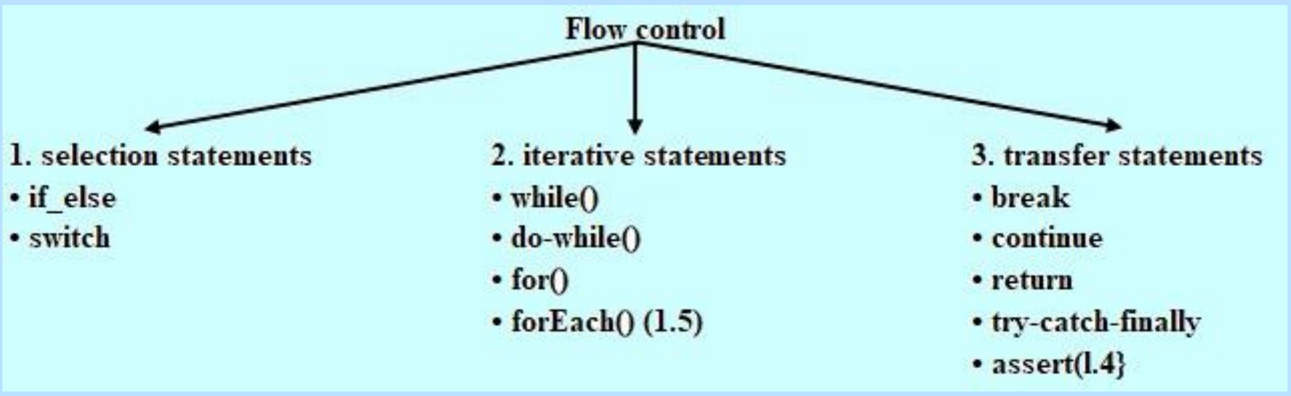
Flow Control

Flow control describes the order in which all the statements will be executed at run time.

Selection statements



if-else

```
if(b)
{
        //action if b is true
}
else
{
        // action if b is false
}
```

The argument to the if statement should be Boolean if we are providing any other type we will get "compile time error".

Example

```
public class Exif
{
        public static void main(String args[])
        {
                int x=O;
                if(x)
                {
                        System.out.println("hello");
                }
                else
                {
                        System.out.println("hi");
                }
        }
}
```

Output

```
Compile time error
incompatible types
found: int
required: boolean if(x)
```

Example

```
public class Exif
{
	public static void main(String args[])
	{
		int x=10;
		if(x=20)
		{
			System.out.println("hello");·
		}
		else
		{
			System.out.println("hi");
		}
	}
}
```

Output

```
incompatible types
found: int
required: boolean if (x=20)
```

Example

```
public class Exif
{
	public static void main(String args[])
	{
		int x=10;
		if(x==20)
		{
			System.out.println("hello");
		}
		else
		{
			System.out.println("hi");
		}
	}
}
```

Output

```
Hi
```

Switch

If several options are available then it is not recommended to use if-else we should go for switch statement.

Syntax:

```
switch(x)
{
	case 1:
		actionl
	case 2:
		action2
	default:
		default action
}
```

- Curly braces are mandatory.
- Both case and default are optional.

- Every statement inside switch must be under some case (or) default. Independent statements are not allowed.
- Until 1.4 version the allow types for the switch argument are byte, short, char, int but from 1.5 v.ersion on wards the corresponding wrapper classes (Byte, Short, Character, Integer) and "enum" types are allowed.
- Every case label should be "compile time constant" otherwise we will get compile time error.

```
public class swich
{
	public static void main(String[] args)
	{
		int x=10;
		int y=20;
		switch(x)
		{
		case 10:
			System.out.println("lO");
		case y:
			System.out.println("20");
		}
	}
}
```

Output

```
Compile time error: constant expression required case y:
```

- If we declare y as final we won't get any compile time error.

```
public class swich
{
	public static void main(String[] args)
	{
		int x=10;
		final int y=20;
		switch(x)
		{
		case 10:
			System.out.println("lO");
		case y:
			System.out.println("20");
		}
	}
}
```

Output

```
10
20
```

- Switch argument and case label can be expressions also, but case should be constant expression.

```
public class swich
{
	public static void main(String[] args)
	{
		int x=10;
		final int y=20;
		switch(x+1)
```

```
                {
                case 10:
                        System.out.println("10");
                case y:
                        System.out.println("20");
                }
        }
}
```

No output

- Every case label should be within the range of switch argument type.

Example

```
public class swich
{
        public static void main(String args[])
        {
                byte b=lO;
                switch(b)
                {
                        case 10:
                                System.out.println("lO");
                        case 100:
                                System.out.println("lO0");
                        case 1000:
                                System.out.println("lO00");
                }
        }
}
```

```
CE: possible loss of precision
found : int
required: byte
```

- Duplicate case labels are not allowed.

```
class swich
{
        public static void main(String args[])
        {
                int x=lO;
                switch{x)
                {
                        case 97:
                                System.out.println("97”);
                        case 99:
                                System.out.println("99");
                        case 'a':
                                System.out.println("100");
                }
        }
}
```

Output

```
Compile time error.
duplicate case label
```

case 'a':

FALL-THROUGH inside the switch

Within the switch statement if any case is matched from that case onwards all statements will be executed until end of the switch (or) break. This is call "fall-through" inside the switch .

```
class switch
{
        public static void main(String args[])
        {
                int x=0;
                switch(x)
                {
                        case 0:
                                System.out.println("0”);
                        case 1:
                                System.out.println("1");
                                break;
                        case 2:
                                System.out.println(“2”);
                        default:
                                System.out.println("default");
                }
        }
}
```

Within the switch we can take the default anywhere, but at most once it is convention to take default as last case.

```
class Switch
{
        public static void main(String args[])
        {
                int x=0;
                switch(x)
                {
                case 0:
                        System.out.println("0");
                        break;
                case 1:
                        System.out.println("1");
                        break;
                case 2:
                        System.out.println("2");
                        break;
                default:
                        System.out.println("default");
                        break;
                }
        }
}
```

Iterative statements

While loop

if we don't know the no of iterations in advance then best loop is while loop

Example

```
while(rs.next())
{

}
```

The argument to the while statement should be Boolean type. If we are using any other type we will get compile time error.

```
Class While
{
        public static void main(String args[])
        {
                while(l)
                {
                System.out.println("hello");
                }
        }
}
```

- Curly braces are optional and without curly braces we can take only one statement which should not be declarative statement.

Example

```
class While
{
        public static void main(String args[])
        {
                while(true)
                System.out.println("hello");
        }
}
```

Output

Hello (infinite times).

```
Class While
{
        public static void main(String args[])
        {
                while(true)
                int x=10;
        }
}
```

Output

No output.

Unreachable statement in while:

```
public While
{
        public static void main(String args[])
        {
                while(true)
                {
```

```
                System.out.println("hello");
                }
                System.out.println("hi");
        }
}
```

Output

Compile time error
unreachable statement
System.out.println("hi");

```
public While
{
        public static void main(String args[])
        {
                while(false)
                {
                System.out.println("hello");
                }
                System.out.println("hi");
        }
}
```

Output

Compile time error
unreachable statement

```
public While
{
        public static void main(String args[])
        {
                int a=10, b=20;
                while(a<b)
                {
                System.out.println("hello");
                }
                System.out.println("hi");
        }
}
```

Output

Hello (infinite times)

```
public While
{
        public static void main(String args[])
        {
                final int a=10, b=20;
                while(a<b)
                {
                System.out.println("hello");
                }
                System.out.println("hi");
        }
}
```

Output

Compile time error
unreachable statement

```
public While
{
	public static void main(String args[])
	{
		final int a=10;
		while(a<b)
		{
			System.out.println("hello");
		}
		System.out.println("hi");
	}
}
```

Output

Compile time error
unreachable statement

- Every final variable will be replaced with the corresponding value by compiler.
- If any operation involves only constants then compiler is responsible to perform that operation.
- If any operation involves at least one variable compiler won't perform that operation. At runtime jvm is responsible to perform that operation.

Do-while

- If we want to execute loop body at least once then we should go for do-while.
- Curly braces are optional.
- Without curly braces we can take only one statement between do and while and it should not be declarative statement.

Example

```
public class doWhile
{
	public static void main(String args[])
	{
		do
		System.out.println("hello");
		while(true);
	}
}
```

Example

```
class doWhile
{
	public static void main(String args[])
	{
		do;
		while(true);
	}
}
```

Output

Compile successful.

```
class doWhile
{
	public static void main(String args[])
	{
		do
		int x=10;
		while(true);
	}
}
```

Compile time error

Example

```
class doWhile
	{
		public static void main(String args[])
		{
			do
			{
			int x=10;
			}while(true);
		}
	}
```

Output

Compile successful.

For Loop

This is the most commonly used loop and best suitable if we know the no of iterations in advance.

for(lnitilizationsection;Conditional check;Increment and decrement section)

1. **Lnitilizationsection**
   - This section will be executed only once. • Here usually we can declare loop variables and we will perform initialization.
   - We can declare multiple variables but should be of the same type and we can't declare different type of variables.

   Example

   ```
   int i=0,j=0; valid
   int i=0,Boolean b=true; invalid
   int i=0,int j=0; invalid
   ```

   - In initialization section we can take any valid java statement including "s.o.p" also.

   Example

   ```
   class For
   {
   	public static void main(String args[])
   	{
   		int i=0;
   		for(System.out.println("hello u r sleeping");i<3;i++)
   		{
   			System.out.println("no boss, u only sleeping");
   		}
   	}
   ```

```
        }
```

2. **Conditional check**
    - We can take any java expression but should be of the type Boolean.
    - Conditional expression is optional and if we are not taking any expression compiler will place true.
3. **Increment and decrement section**
    - Here we can take any java statement including s.o.p also.

Example

```
class For
{
        public static void main(String argsI])
        {
                int i=0;
                for(System.out.println("hello");i<3;System.out.println("hi"))
                {
                        i++;
                }
        }
}
```

- All three parts of for loop are independent of each other and all optional.

Example

```
class For
{
        public static void main{String args[])
        {
                for(;;)
                {
                        System.out.println("hello");
                }
        }
}
```

Output

```
Hello (infinite times).
```

- Curly braces are optional and without curly braces we can take exactly one statement and it should not be declarative statement.

For each

- For each Introduced in 1.5 version.
- Best suitable to retrieve the elements of arrays and collections.

Example

Write code to print the elements of single dimensional array by normal for loop and enhanced for loop.

```
public class ExampleFor
{
        public static void main (String args[])
        {
                int[] a={10,20,30,40,50);
                for(int i=0;i<a.length;i++)
                {
                        System.out.println(a[i]);
                }
        }
}

public class ExampleFor
{
        public static void main(String args[])
        {
                int[] a={10,20,30,40,50);
                for(int x:a)
                {
                        System.out.println(x);
                }
        }
}

class For
{
        public static void main(String args[])
        {
                int[][] a={{10,20,30},{40,50}};
                for(int i=0;i<a.length;i++)
                {
                        for(int j=0;j<a[i].length;j++)
                        {
                                System.out.println(a[i][j]);
                        }
                }
        }
}
```

Transfer statements
Break statement
We can use break statement in the following cases.
- Inside switch to stop fall-through.
- Inside loops to break the loop based on some condition.
- Inside label blocks to break block execution based on some condition.

Continue statement
- We can use continue statement to skip current iteration and continue for the next iteration.
- We can use continue only inside loops if we are using anywhere else we will get compile time error saying "continue outside of loop".