1. FLOW CONTROL:

Flow control describe the order in which the statement will be executed yet runtime.

SELECTION STATEMENT:

-> IF-ELSE

-> SWITCH()

ITERATIVE STATEMENT:

-> WHILE

-> DO-WHILE

-> FOR

-> FOR EACH(1.5V)

TRANSFER STATEMENT:

-> BREAK

-> CONTINUE

-> RETURN

-> TRY-CATCH-FINALLY

-> ASSERT(1.4V)

2. SELECTION STATEMENT:

2.1 IF-ELSE STATEMENT:

Syntax:

If(boolean condition)

{

Action if boolean is true

}

Else

{

Action if boolean is false

}

The argument to the if statement should be boolean type by mistake if we are trying to provide any other type then we will get compile time error.

```
class test
{
public static void main(String args[])
{
int i = 0;
if(i)
{
System.out.println("hello");
}
else
{
System.out.println("hii");
}
}
}
```

in if we use assignment operator = invalid.

```
C:\Users\Atish kumar sahu\desktop>javac test.java
test.java:6: error: incompatible types: int cannot be converted
to boolean
if(i)
   ^
1 error
```

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

here in if we take assignment operator = invalid.

```
C:\Users\Atish kumar sahu\desktop>javac test.java
test.java:6: error: incompatible types: int cannot be converted
to boolean
if(x = 20)
     ^
1 error
```

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

comparison operator gives t/f so we can take this.

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
hii
```

```
class test
{
public static void main(String args[])
{
boolean b = true;
if(b = false)
{
System.out.println("hello");
}
else
{
System.out.println("hii");
}
}     }
```

here variable b is in boolean type which is valid one.

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
hii
```

```
class test
{
public static void main(String args[])
{
boolean b = false;
if(b == false)
{
System.out.println("hello");
}
else
{
System.out.println("hii");
}
}     }
```

“b is boolean type and comparison also give boolean type so it is valid.”

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
hello
```

```
class test
{
public static void main(String args[])
{
if(true)
        System.out.println("hello"); //valid

if(true); //valid

if(true)
        int x =10; //invalid

if(true)
{
int x = 10;  //valid
}

}     }
```

In the left side of program else part and “{}” are optional. Without “{}” only one statement is allowed under if which should not be declarative statement that is -> int x = 10;

“;” is valid java statement which also known as empty statement.

```
C:\Users\Atish kumar sahu\desktop>javac test.java
test.java:9: error: variable declaration not allowed here
        int x =10;
            ^
1 error
```

There is no dangling else problem in java. every else is matched or mapped to the nearest if statement.

```
class test
{
public static void main(String args[])
{
if(true)
        if(true)
                System.out.println("hello");
else
        System.out.println("hii");

}     }
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
hello
```

2.2 SWITCH() STATEMENT:

If several options are available then it is not recommended to use nested if-else. Because it reduces readability. To handle this requirement we should go for switch statement.

Syntax:

```
Switch(x)     {
        Case 1 :
                Action1;
                Break;
        Case 2 :
                Action2;
                Break;
        ……..
        Default :
                Action;
                Break;
}
```

```
class test
{
public static void main(String args[])
{
int x = 5;
switch(x)
{
      case 1 :
            System.out.println("jan");
            break;
      case 2 :
            System.out.println("feb");
            break;
      case 3 :
            System.out.println("mar");
            break;
      case 4 :
            System.out.println("apr");
            break;
      default :
            System.out.println("minimum 4");
}

}     }
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
minimum 4
```

CASE-1:

The allowed argument types for the switch statement are "*byte*" , "*short*", "*char*", "*int*" until 1.4 version. But 1.5 version onwards corresponding "*wrapper classes*" and "*enum*" type also allowed. From 1.7 version onwards "*String*" type also allowed.

Byte, Short, Character, Integer, byte, short, char, int, enum, String.

CASE-2:

"{}" are mandatory except switch everywhere "{}" are optional.

CASE-3:

Both "case" and "default" are optional. That is an empty switch statement is a valid java statement.

```
Switch(x)
{
}                    ==> valid java statement
```

CASE-4:

Inside a switch every statement should be under some case or default that is independent statements are not allowed inside a switch otherwise we will get compile time error.

```
int x = 5;
switch(x)
{
S.O.P("HELLO");
}                              ==> invalid coding and we will get compile time error
```

CASE-5:

```
class test
{
public static void main(String args[])
{
int x = 5;
int y = 10;
switch(x)
{
	case 5 :
		System.out.println(x);
		break;
	case y :
		System.out.println(20);
		break;
	default :
		System.out.println("bye");
}

}	}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java
test.java:12: error: constant expression required
        case y :
             ^
1 error
```

```
class test
{
public static void main(String args[])
{
int x = 5;
final int y = 10;
switch(x)
{
	case 5 :
		System.out.println(x);
		break;
	case y :
		System.out.println(20);
		break;
	default :
		System.out.println("bye");
}

}	}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
5
```

Every case level should be compile time constant that is constant expression. If we declare “y” as final we won’t get any compile time error.

CASE-6:

Both switch argument and case level can be expression but case level should be constant expression.

```
class test
{
public static void main(String args[])
{
int x = 5;
switch(x + 5)
{
      case 5 :
            System.out.println(x);
            break;
      case 10 * 2 + 10 :
            System.out.println(20);
            break;
}

}     }
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test

C:\Users\Atish kumar sahu\desktop>
```

CASE-7:

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

```
C:\Users\Atish kumar sahu\desktop>javac test.java
test.java:14: error: incompatible types: possible lossy conversion
from int to byte
        case 1000 :
             ^
1 error
```

```
class test
{
public static void main(String args[])
{
byte b = 10;
switch(b+1)
{
	case 10 :
		System.out.println(10);
		break;
	case 100 :
		System.out.println(100);
		break;
	case 1000 :
		System.out.println(1000);
		break;
}
}	}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test

C:\Users\Atish kumar sahu\desktop>
```

Every case level should be in the range of switch argument type otherwise we will get compile time error.

CASE-8:

Duplicate case labels are not allowed in switch case otherwise we will get compile time error.

```
class test
{
public static void main(String args[])
{
int x = 10;
switch(x)
{
	case 97 :
		System.out.println(97);
		break;
	case 98 :
		System.out.println(98);
		break;
	case 99 :
		System.out.println(99);
		break;
	case 'a' :
		System.out.println('a');
		break;
}
}	}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java
test.java:17: error: duplicate case label
		case 'a' :
		^
1 error
```

CASE-9:

Case label it should be constant expression. The value should be is in the range of the switch argument type. duplicate case labels are not allowed.

FALL THROUGH INSIDE SWITCH:

With in the switch if any case is matched from that case onwards all statements will be executed until break or end of the switch. This is called fall through inside a switch. The main advantage of fall through inside a switch is we can define common action for multiple cases code reusability.

```
switch(x)
{
Case 1 :
Case 2 :
Case 3 :
        S.O.P("HII");
        Break;
Case 4 :
Case 5 :
Case 6 :
        S.O.P("BYE");
        Break;
}
```

```
Switch(x)
{
Case 0 :
        S.O.P(0);
Case 1 :
        S.O.P(1);
        Break;
Case 2 :
        S.O.P(2);
Default :
        S.O.P("DEF");
}
```

OUTPUT OF PROGRAM:

X = 0 output = 0 1

X = 1 output = 1

X = 2 output = 2 DEF

X = 3 output = DEF

CASE-10:

With in the switch we can take default case at most one time. Default case will be executed if and only if there Is no case matched. With in the switch we can write default case anywhere but it is recommended to write as last case.

```
Switch(x)
{
Default :
        S.O.P("DEF");
Case 0 :
        S.O.P(0);
        Break;
Case 1 :
        S.O.P(1);
Case 2 :
        S.O.P(2);
}
```

OUTPUT OF THE PROGRAM:

X = 0 output = 0

X = 1 output = 1 2

X = 2 output = 2

X = 3 output = DEF 0

3. ITERATIVE STATEMENT:

3.1 WHILE LOOP:

If we don't know number of iteration in advance then we should go for while loop.

Ex: while (rs.next()){} while(itr.hasNext()){}

SYNTAX:

While(x) "x" should be boolean type.

{

Action

}

The argument should be boolean type if we are trying to provide any other type then we will get compile time error.

```
class test
{
public static void main(String args[])
{
      while(1)
      {
            System.out.println("hello");
      }
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java
test.java:5: error: incompatible types: int cannot be converted
 to boolean
        while(1)
              ^
1 error
```

“{}” are optional and without “{}” we can take only one statement under while which should not be declarative statement.

```
class test
{
public static void main(String args[])
{
	while(true)
	{
		int a = 10;		//valid while loop
	}

	while(true)
		System.out.println("hello");		//valid

	while(true);	//valid
}
}
```

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

	while(false)
	{
		System.out.println("hello");
	}
	System.out.println("hii");	//invalid
}
}
```

```
class test
{
public static void main(String args[])
{
	int a = 50;
	int b = 60;
	while(a < b)
	{
		System.out.println("hello"); //output = hello.....
	}
	System.out.println("hii");	//valid

	int a = 50;
	int b = 60;
	while(a > b)
	{
		System.out.println("hello");
	}
	System.out.println("hii");	//valid //output = hii
}
}
```

```
class test
{
public static void main(String args[])
{
        final int a = 50;
        final int b = 60;
        while(a < b)
        {
                System.out.println("hello");
        }
        System.out.println("hii");    //invalid

        final int a = 50;
        final int b = 60;
        while(a > b)
        {
                System.out.println("hello");
        }
        System.out.println("hii");    //invalid
}
}
```

Every final variable will be replaced by the value yet compile time only.

Final int a = 10;

Int b = 20;

S.O.P(a); //S.O.P(10);

S.O.P(b); //S.O.P(b);

If every argument is a final variable (compile time constant) then the operation should be performed yet compile time only.

```
class test
{
public static void main(String args[])
{
      final int a = 50;
      final int b = 60;
      int c = 30;
      System.out.println(a + b); //S.O.P(110);
      System.out.println(a + c); //S.O.P(50 + c);
      System.out.println(a<b); //S.O.P(true);
      System.out.println(a<c); //S.O.P(50 < c);
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
110
80
true
false
```

3.2 DO-WHILE LOOP:

If we want to execute loop body at least ones then we should go for do-while loop.

SYNTAX:

Do

{

Body

}while(b); ";" is mandatory "b" is boolean type

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

```
C:\Users\Atish kumar sahu\desktop>javac test.java
test.java:6: error: variable declaration not allowed here
        int x = 10;
            ^
1 error
```

```
/*
      do
      System.out.println("hello");
      while(true);       // valid

      do;
      while(true);       // valid

      do
      int x = 10;
      while(true);       // invalid

      do
      {
            int z = 50;
      }while(true);      //valid

      do
      while(true);       //invalid
*/
```

"{}" are optional and without "{}" we can take only one statement between do and while which should not be declarative statement.

```
class test
{
public static void main(String args[])
{
        //valid statements
        do while(true)
        System.out.println("hello");
        while(false);
        //output = hello hello.....
}
}
```

```
do
        while
        System.out.println(“hello”);
while(false);
```

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

do
{
System.out.println("hello");
}while(false);
System.out.println("hii"); //valid
}
}
```

```
class test
{
public static void main(String args[])
{
int a = 50 , b = 60;
do
{
System.out.println("hello");
}while(a<b);
System.out.println("hii"); //valid

int a = 50 , b = 60;
do
{
System.out.println("hello");
}while(a>b);
System.out.println("hii"); //valid
}
}
```

```
class test
{
public static void main(String args[])
{
final int a = 50 , b = 60;
do
{
System.out.println("hello");
}while(a<b);
System.out.println("hii"); //invalid

final int a = 50 , b = 60;
do
{
System.out.println("hello");
}while(a>b);
System.out.println("hii"); //valid
}
}
```

If unreachable statement is there then you will get compile time error.

3.3 FOR LOOP:

For loop is the most commonly used loop in java. if we know number of iteration in advance then for loop is the best choice to use.

SYNTAX:

1 2, 5, 8 4, 7

for(initialization; condition; increment or decrement)

{

Loop body 3, 6, 9

}

```
class test
{
public static void main(String args[])
{
	for(int i = 0; i < 5 ; i++)
	{
		System.out.print("a");
	}
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
aaaaa
```

```
class test
{
public static void main(String args[])
{
	for(int i = 0 ; true; i++)
		System.out.println("hello");
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
hello
hello
hello
hello
hello
hello
hello
```

```
class test
{
public static void main(String args[])
{
      for(int i = 0 ; i<10 ; i++);
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test

C:\Users\Atish kumar sahu\desktop>
```

```
class test
{
public static void main(String args[])
{
      for(int i = 0 ; i<10 ; i++)
            int x = 10;
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java
test.java:6: error: variable declaration not allowed here
                int x = 10;
                    ^
1 error
```

"{}" are optional and without "{}" we can take only one statement under for loop which should not be declarative statement.

INITIALIZATION SECTION:

This part will be executed only once in loop life cycle. Here we can declare and initialize local variables of for loop. Here we can declare any number of variables but should be of the same type. by mistake if we are trying to declare different data type variable then we will get compile time error.

Int I = 0, j = 0 ; //valid

Int I = 0 , String s = "aa"; //invalid

Int i = 0 , int j = 0 ; //invalid

```
class test
{
public static void main(String args[])
{
	int i = 0;
	for(System.out.println("hey bro"); i<3 ; i++)
	{
		System.out.println("hii atish");
	}
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
hey bro
hii atish
hii atish
hii atish
```

In the initialization section we can take any valid java statement including "System.out.print".

"check return and break in for loop?"

CONDITIONAL CHECK:

Here we can take any valid java expression but should be of the boolean type. this part is optional and if we are not taking anything then compiler will always place true.

```
class test
{
public static void main(String args[])
{

	for(int i = 0; ; i++)
	{
		System.out.println("hii atish");
	}
}
}
```

INCREMENT AND DECREMENT SECTION:

In the increment and decrement section we can take any valid java statement including "System.out.print".

```
class test
{
public static void main(String args[])
{
	int i = 0;
	for(System.out.println("hii atish");i <3 ; System.out.println("hii"))
	{
		i++;
	}
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
hii atish
hii
hii
hii
```

ALL THREE PARTS OF FOR LOOPS ARE INDEPENDENT OF EACH OTHER AND OPTIONAL.

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

=> both statement output = infinite loops.

```
class test
{
public static void main(String args[])
{
	for(int i = 0; true; i++)
	{
		System.out.println("a");
	}
	System.out.println("b");	//invalid

	for(int i = 0; false; i++)
	{
		System.out.println("a");
	}
	System.out.println("b");	//invalid
}
}
```

```
class test
{
public static void main(String args[])
{
	for(int i = 0; ; i++)
	{
		System.out.println("a");
	}
	System.out.println("b");	//invalid

}
}
```

```
class test
{
public static void main(String args[])
{

        int a = 10 , b = 20;
        for(int i = 0; a<b; i++)
        {
                System.out.println("a");
        }
        System.out.println("b"); //valid

        int a = 10 , b = 20;
        for(int i = 0; a>b; i++)
        {
                System.out.println("a");
        }
        System.out.println("b");      //valid
}
}
```

```
class test
{
public static void main(String args[])
{

        final int a = 10 , b = 20;
        for(int i = 0; a<b; i++)
        {
                System.out.println("a");
        }
        System.out.println("b"); //invalid

        final int a = 10 , b = 20;
        for(int i = 0; a>b; i++)
        {
                System.out.println("a");
        }
        System.out.println("b");      //invalid
}
}
```

3.4 FOR EACH LOOP: VERSION 1.5

It introduced in 1.5 version. It is specially designed loop to retrieve elements of arrays and collections.

```
class test
{
public static void main(String args[])
{
        int [] a = {10,20,30,40,50};

        for(int x1 : a)
        {
                System.out.println(x1);
        }
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
10
20
30
40
50
```

```
class test
{
public static void main(String args[])
{
        int [][] a = {{10,20,30},{40,50}};

        for(int[] x1 : a)
        {
                for(int x2 : x1)
                {
                        System.out.println(x2);
                }
        }
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
10
20
30
40
50
```

```
   for(int[][] x1 : a)
   {
          for(int[] x2 : x1)
          {
                 for(int x3 : x2)
                 {
                        System.out.println(x3);
                 }
          }
   }
```

=> FOR 3-DIMENSION ARRAY.

For each loop is the best choice to retrieve elements of arrays and collections. But its limitation is it is applicable only for array and collections and it is not a general purpose loop.

By using normal for loop we can print array elements either in original order or in reverse order. But by using for each loop we can print array element only in original order but not in reverse order.

ITERABLE INTERFACE(I):

For(eachitem x : target) target can be Iterable object.

The target element in for each loop should be Iterable object. An object is set to be Iterable if and only if corresponding class implements java.lang.Iterable interface. Iterable interface introduced in java 1.5 version and it contains only one method iterator.

“public Iterator iterator()”.

So all array related classes and collection implemented classes already implement Iterable interface. Being a programmer we are not required to do anything just we should aware.

DIFFERENCES BETWEEN ITERATOR AND ITERABLE:

ITERATOR:

It is related to collection.

We can use to retrieve the elements of collection one by one.

Present in java.util package.

It defines three methods hasNext(), next(), remove()

ITERABLE:

It is related to for-each loop.

The target element in for-each loop should be Iterable.

Present in java.lang package.

It contain one method iterator().

4. TRANSFER STATEMENT:

4.1 BREAK STATEMENT:

We can use break statement in inside switch. Inside a loop. Inside a labeled blocks. We can use break statement in the following places. Inside a switch to stop fall through. Inside loops to break loop execution based on some condition. Inside labeled blocks to break block execution based on some condition. These are the only places where we can use break statement. If we are using anywhere else we will get compile time error saying "*break outside switch or loop*".

```
int i = 0;
switch(i)
{
        case 0 :
                System.out.println(0);
        case 1 :
                System.out.println(1);
                break;
        case 2 :
                System.out.println(2);
        default :
                System.out.println("end");
}       //output : 0 1
```

```
for(int j = 0; j <= 10; j++)
{
if(j == 5)
        break;
System.out.println(j);
}       //output: 0 1 2 3 4
```

```
int x = 10;
l:
{
System.out.println("begin");
        if(x == 10)
                break l;
System.out.println("end");
}
System.out.println("hello");
        //output: begin hello
```

4.2 CONTINUE STATEMENT:

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

```
for(int i = 0; i < 10 ; i++)
{
if(i % 2 == 0)
        continue;
System.out.println(i);
}       //output: 1 3 5 7 9
```

Labeled break and continue:

We can use labeled break and continue to break or continue a particular loop in nested loops.

```
l1:
for()
{
	l2:
	for()
	{
		l3:
		for()
		{
			break l1;
			break l2:
			break;
		}
	}
}
```

Case1:

```
l1:
for(int i = 0; i < 3; i++)
{
	for(int j = 0; j < 3 ; j++)
	{
		if(i == j)
			break;
		System.out.println(i+"---"+j);
	}
}
```

```
C:\Users\Atish
1---0
2---0
2---1
```

Case2:

```
l1:
for(int i = 0; i < 3; i++)
{
	for(int j = 0; j < 3 ; j++)
	{
		if(i == j)
			break l1;
		System.out.println(i+"---"+j);
	}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
```

Case3:

```
l1:
for(int i = 0; i < 3; i++)
{
	for(int j = 0; j < 3 ; j++)
	{
		if(i == j)
			continue;
		System.out.println(i+"---"+j);
	}
}
```

```
C:\Users\Atis
0---1
0---2
1---0
1---2
2---0
2---1
```

Case4:

```
l1:
for(int i = 0; i < 3; i++)
{
	for(int j = 0; j < 3 ; j++)
	{
		if(i == j)
			continue l1;
		System.out.println(i+"---"+j);
	}
}
```

```
C:\Users\Atish
1---0
2---0
2---1
```

DO WHILE VS CONTINUE:

```
int x = 0;
do
{
x++;
System.out.println(x);
if(++x < 5)
	continue;
x++;
System.out.println(x);
}while(++x < 10);
```

```
C:\Users\Atish
1
4
6
8
10
```