CODER UNIVERSE TECHNOLOGY

https://coderuniversetechnology.info/

https://www.facebook.com/profile.php?id=61569717476118

https://www.instagram.com/coder_universe_technology/

https://www.youtube.com/@CoderUniverseTechnology/videos

Contact us on: 7709701108

Introduction :

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

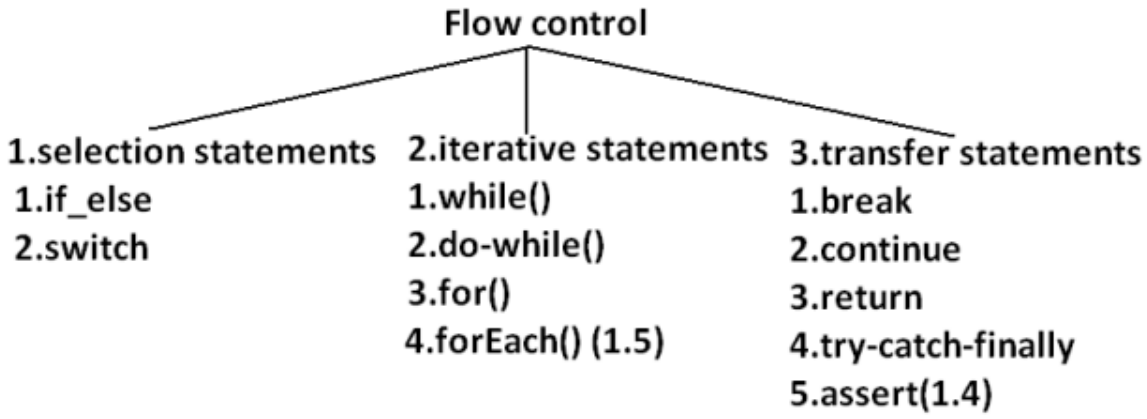
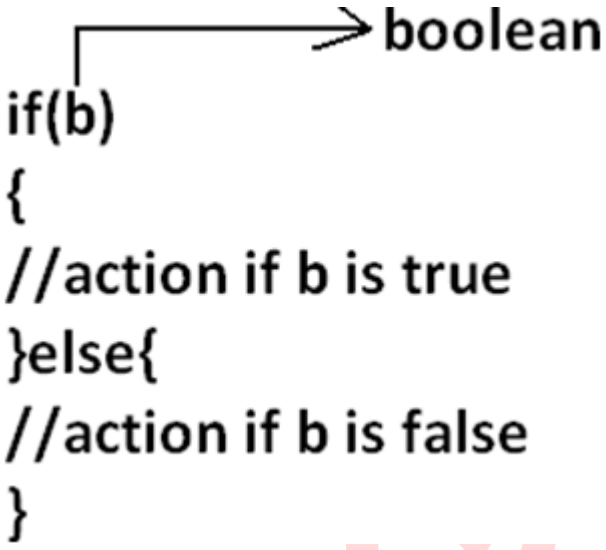


Diagram:

If else syntax:



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

Program 1:

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

Program 2:

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

Program 3:

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

Program 4:

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

Program 5:

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

Both else part and curly braces are optional.

Without curly braces we can take only one statement under if, but it should not be declarative statement.

Program 6:

```
public class ExampleIf{
public static void main(String args[]){
 if(true)
System.out.println("hello");
}} OUTPUT:
Hello
```

Program 7:

```
public class ExampleIf{
public static void main(String args[]){ if(true);
}}
```

OUTPUT:

No output

Program 8:

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

OUTPUT: Compile time error

Program 9:

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

Switch:

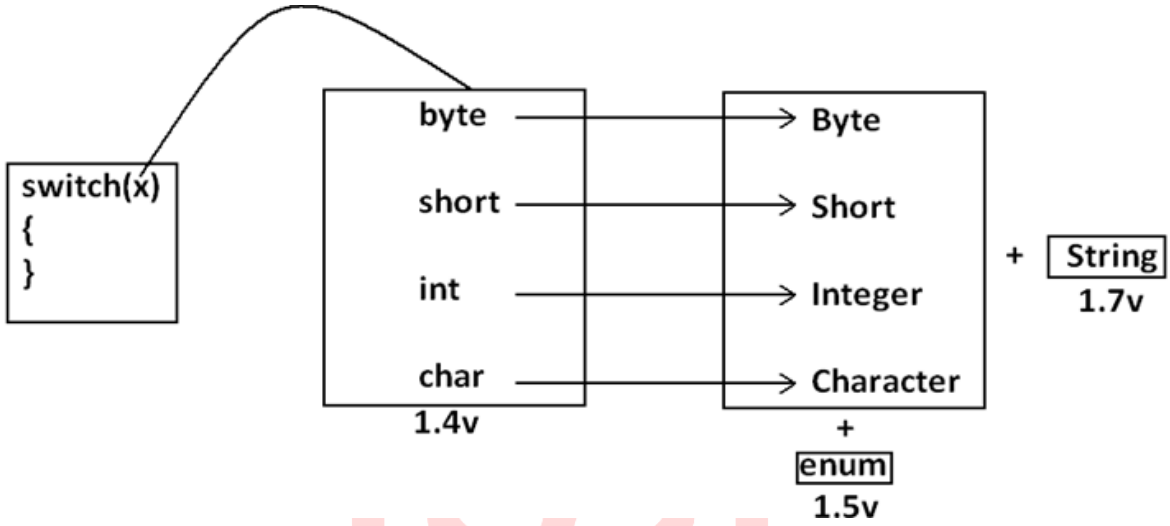
If multiple conditions are there are available then it is not recommended to use if-else we should go for switch statement. Because it improves readability of the code.

Syntax:

```
switch(x)
{
case 1:
action1 case 2:
action2
default:
default action
}
```


Until 1.4 version the allow types for the switch argument are byte, short, char, int but from 1.5 version on wards the corresponding wrapper classes (Byte, Short, Character, Integer) and "enum" types also allowed.

Diagram:



Program 1:

```
public class ExampleSwitch{
public static void main(String args[]){
 int x=10;
switch(x)
{
System.out.println("hello");
}}}
```

OUTPUT:

Compile time error.

Every case label should be "compile time constant" otherwise we will get compile time error.

Program 2:

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

}}}
```

OUTPUT:

Compile time error

Program 3:

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

But switch argument and case label can be expressions , but case label should be constant expression.

Program 4:

```
public class ExampleSwitch{
public static void main(String args[]){ int x=10;
switch(x+1)
{
case 10:
case 10+20:
case 10+20+30:
}}} OUTPUT: No output.
```

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

Program 5:

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

OUTPUT:

Compile time error

Program 6:

```
public class ExampleSwitch{
public static void main(String args[]){
 byte b=10;
switch(b+1)
{
case 10:
```

```
System.out.println("10");
 case 100:
 System.out.println("100");
 case 1000:
System.out.println("1000");
}}}
```

OUTPUT:

==========

Duplicate case labels are not allowed in below program.

Program 6:

```
public class ExampleSwitch{
public static void main(String args[]){
int x=10;
switch(x)
{
case 97:
 System.out.println("97");
 case 99: System.out.println("99");
 case 'a': System.out.println("100");
}}}
```

OUTPUT:

Compile time error.

FALL-THROUGH INSIDE THE SWITCH:

The main advantage of fall-through inside a switch is we can define common action for multiple cases

Program 7:

```
public class ExampleSwitch{
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");
}}}
```

Default Case

- With in the switch we can take the default only once
- If no other case matched then only default case will be executed
- With in the switch we can take the default any where, but it is convension to take default as last case.

Program 8:

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

While Loop

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

Syntax

```
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.

Program 1:

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

OUTPUT:

Compile time error.

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

Program 2:

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

OUTPUT:

Hello (infinite times).

Program 3:

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

while(true);
}}
```

OUTPUT:

No output.

Program 4:

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

```
while(true)
int x=10;
}}
```
OUTPUT:

Compile time error

Program 5:

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

No output.

Unreachable statement in while:

Example 6:

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

}} OUTPUT:

Compile time error.

Program 9:

```
public class ExampleWhile{
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

Note:

Do-while:

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

Syntax:

```
do
{
----------
----------
----------
}while(b); ——————>semicolon is the mandatory.
```

Curly braces are optional.

Without curly braces we can take only one statement between do and while and it should not be declarative statement.

Program 1:

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

Output:

Hello (infinite times).

Program 2:

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

Output:

Compile successful.

Program 3:

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

Program 4:

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


Output:

Compile successful.

Example 5:

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

```
System.out.println("hello"); while(true);
}}
```

Output:

Hello (infinite times).

Program 6:

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

Output:

Compile time error

Unreachable statement in do while:

Example 7:

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

Output: Compile time error.

Program 8:

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

Output:

Hello Hi

Program 9:

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

Output:

Hello (infinite times).

Program 10:

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

Output:

Hello Hi

Program 11:

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

}}

Output:

Compile time error.

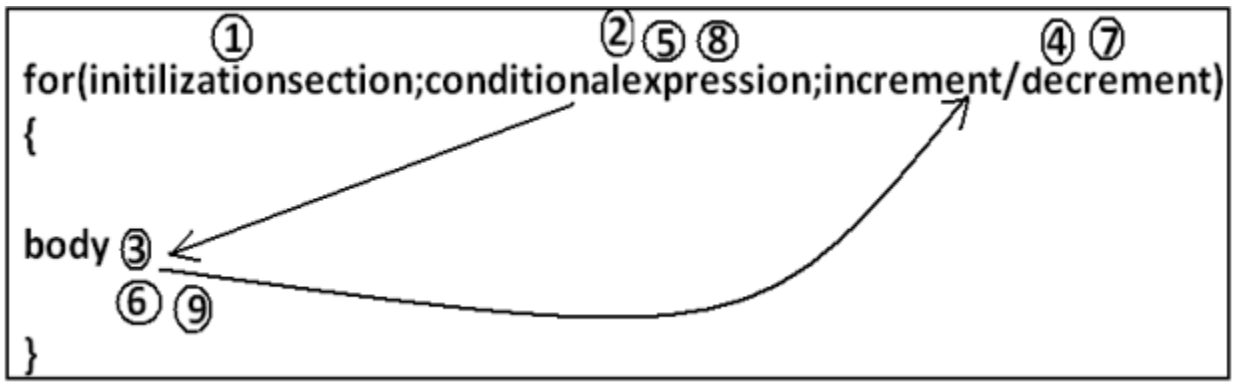
Example 12:

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

Output:

Hi

For Loop



In initialization section

- **we can take any valid java statement including "s.o.p" also.**
- This section will be executed only once. Here usually we can declare loop variables and we will perform initialization.

Programe 1:

```
public class ExampleFor{

public static void main(String args[]){ int i=0;

for(System.out.println("hello-intial");i<3;i++){ System.out.println("Inc-dec");

}}}
```

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.

Here we can take any java statement including s.o.p also.

Program 2:

```
public class ExampleFor{

public static void main(String args[]){ int i=0;

for(System.out.println("hello");i<3;System.out.println("hi")){ i++;

}}}
```

Output:

All 3 parts of for loop are independent of each other and all optional.

Program 3:

```
public class ExampleFor{
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.

Unreachable statement in for loop:

Program 1:

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

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

Output:

Compile time error

Program 2:

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

Output:

Compile time error.

Program 3:

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

```
}}
```

Output:

Compile time error

Program 4:

```
public class ExampleFor{
public static void main(String args[]){
int a=10,b=20;
for(int i=0;a<b;i++){
System.out.println("hello");
}
System.out.println("hi");
}}
```

Output:

Hello (infinite times).

Program 5:

```
public class ExampleFor{
public static void main(String args[]){
 final int a=10,b=20;
for(int i=0;a<b;i++){
```

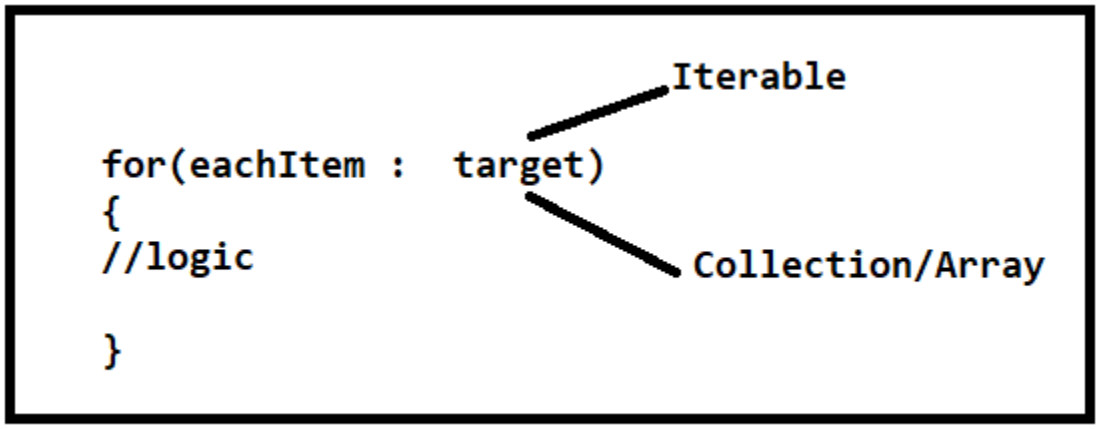
```
 System.out.println("hello");
}
System.out.println("hi");
}}
```

Output:

For each:(Enhanced for loop)

- **Best suitable to retrieve the elements of arrays and collections.**

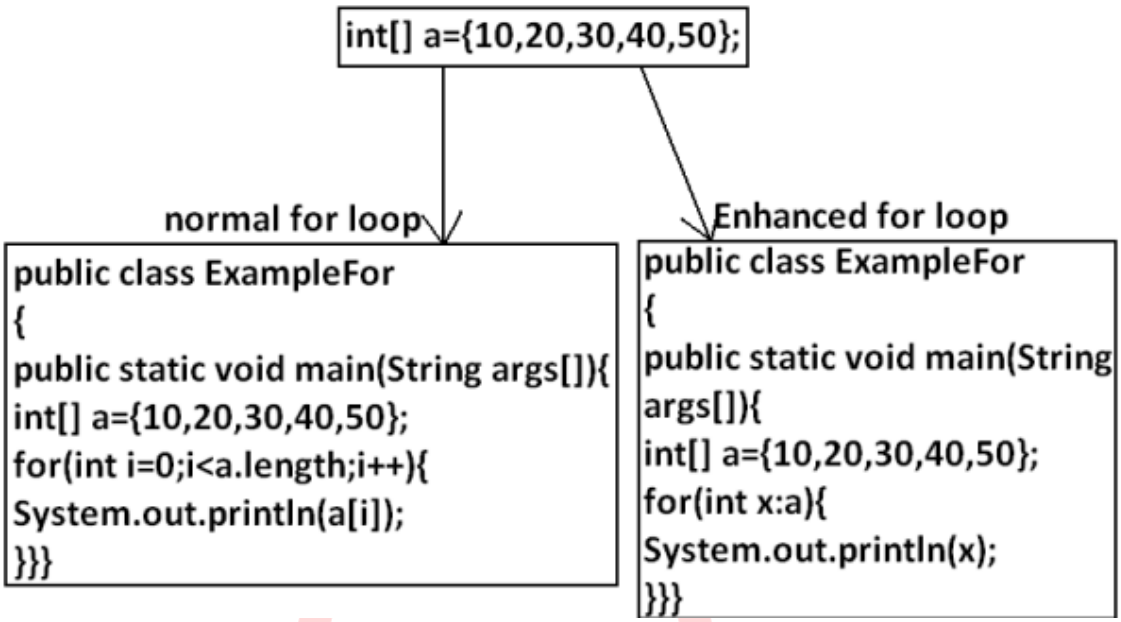
Syntax :



Example 1:

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

Program:



Iterator Vs Iterable(1.5v)

- **The target element in for-each loop should be Iterable object.**
- **An object is set to be iterable iff corresponding class implements java.lang.Iterable interface.**
- **Iterable interface introduced in 1.5 version and it's contains only one method iterator().**

Syntax : public Iterator iterator();

Every array class and Collection interface already implements Iterable interface.

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.

Inside switch :We can use break statement inside switch to stop fall-through

Program 1:

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

Output:

D Hello

Inside loops :

We can use break statement inside loops to break the loop based on some condition.

Program 2:

```
class Test{

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

if(i==5) break;

System.out.println(i);

}

}}
```

Output:

```
0

1

2

3

4
```


Inside Labeled block :

We can use break statement inside label blocks to break block execution based on some condition.

Program:

```
class Test{

public static void main(String args[]){ int x=10;
```

```
l1 : {
System.out.println("begin");
if(x==10)
break l1;
System.out.println("end");
}
System.out.println("hello");
}
}
```

Output:

```
hello
```

These are the only places where we can use break statement. If we are using anywhere else we will get compile time error.

Program:

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

Output:

Compile time error.