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

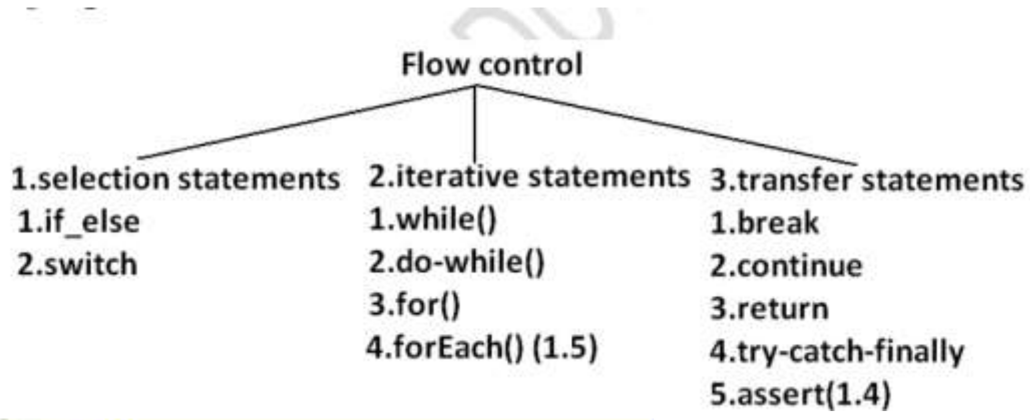
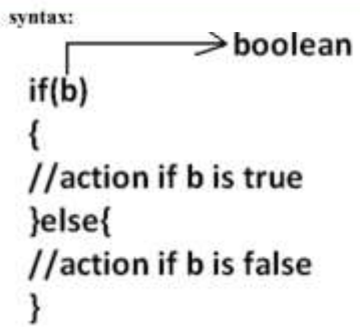


Diagram:

Selection statements:

if-else:


syntax:

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

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

Example 1:

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

OUTPUT:

```
Compile time error:
D:\Java>javac ExampleIf.java
ExampleIf.java:4: incompatible types
found : int
required: boolean
if(x)
```

Example 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");}
}}
```

OUTPUT:
Compile time error

```
D:\Java>javac ExampleIf.java
ExampleIf.java:4: incompatible types
found : int
required: boolean
if(x=20)
```

Example 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");
}}}
```

OUTPUT:

```
Hi
```

Example 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");
}}}
```

OUTPUT:

```
Hello
```

Example 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");
}}}
```

OUTPUT:

```
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.
Example 6:

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

OUTPUT:

```
Hello
```

Example 7:

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

OUTPUT:

```
No output
```

Example 8:

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

OUTPUT:

Compile time error

```
D:\Java>javac ExampleIf.java
ExampleIf.java:4: '.class' expected
int x=10;
ExampleIf.java:4: not a statement
int x=10;
```

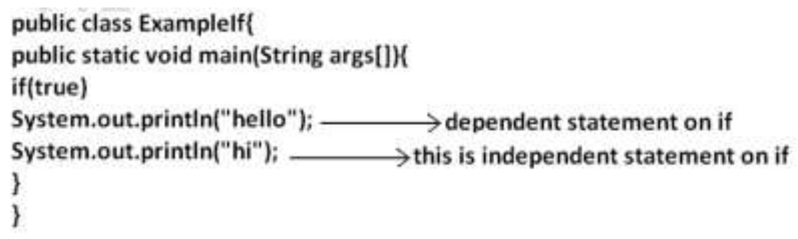
Example 9:

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

OUTPUT:

```
D:\Java>javac ExampleIf.java
D:\Java>java ExampleIf
```

Example 10:



```
public class ExampleIf{
public static void main(String args[]){
if(true)
System.out.println("hello"); ————> dependent statement on if
System.out.println("hi"); ————> this is independent statement on if
}
}
```

OUTPUT:

```
Hello
Hi
```

Semicolon(;) is a valid java statement which is call empty statement and it won't produce any output.

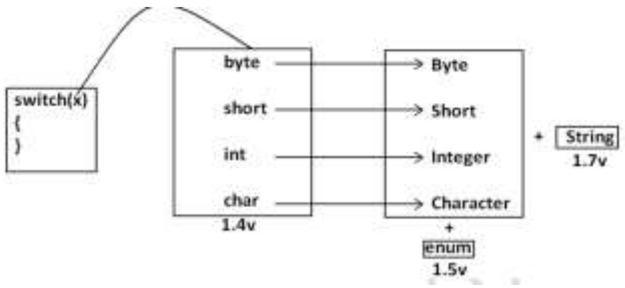
Switch:

If several options 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.



 Curly braces are mandatory.(except switch case in all remaining cases curly braces are optional)
 Both case and default are optional.
 Every statement inside switch must be under some case (or) default. Independent statements are not allowed.
Example 1:
```
public class ExampleSwitch{
public static void main(String args[]){
int x=10;
switch(x)
{
System.out.println("hello");
}}}
```
OUTPUT:
```
Compile time error.
D:\Java>javac ExampleSwitch.java
ExampleSwitch.java:5: case, default, or '}' expected
System.out.println("hello");
```
Every case label should be "compile time constant" otherwise we will get compile time Error.

Example 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
D:\Java>javac ExampleSwitch.java
ExampleSwitch.java:9: constant expression required
case y:
```
If we declare y as final we won't get any compile time error.
Example 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.

Example 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.

Example 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
```
D:\Java>javac ExampleSwitch.java
ExampleSwitch.java:10: possible loss of precision
found : int
required: byte
case 1000:
```
Example :
```
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.
Example 6:

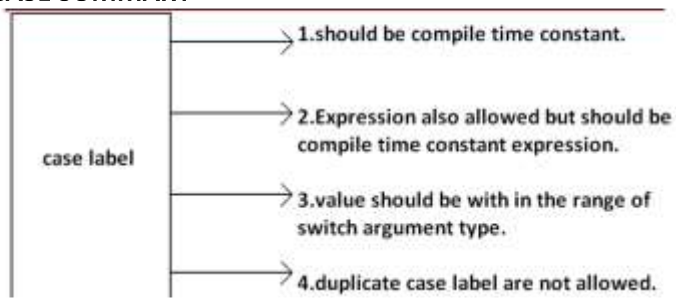
```
public class ExampleSwitch{
public static void main(String args[]){
int x=10;
switch(x)
{
case 97:
System.out.println("97");
case 99:
System.out.println("99");
case 'a':
System.out.println("100");
}}}
```

OUTPUT:
Compile time error.

```
D:\Java>javac ExampleSwitch.java
ExampleSwitch.java:10: duplicate case label

case 'a':
```

CASE SUMMARY



1.should be compile time constant.
2.Expression also allowed but should be compile time constant expression.
3.value should be with in the range of switch argument type.
4.duplicate case label are not allowed.

FALL-THROUGH INSIDE THE SWITCH:
With in 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 .
The main advantage of fall-through inside a switch is we can define common action for multiple cases
Example 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");
}}}
```

OUTPUT:

```
x=0 x=1 x=2 x=3
0 1 2 default
1 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.

Example 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");
}}}
```

OUTPUT:

```
X=0 x=1 x=2 x=3
0 1 2 default
2 0
```