1. Introduction
2. Selection statements
      if,if-else,switch
3. iterative statements
        while,do-while,for,foreach
4. transfer statements
        break,continue,return,[dowhile vs continue]

Flow control statements
=======================
It decides the order in which the control will be trasfered.

flow control
============
a. selective statement
      1. if
      2. if else
      3. switch
b. iterative statement
      1. while
      2. dowhile
      3. for
      4. foreach
c. transfer staements
      a. break
      b. continue
      c. return
      d. System.exit(0)

if block
========
  To select some statements for execution based on condition,we use if block

syntax::
   if(boolean)
   {
        //statement-1
        //statement-2
        //statement-3
   }
   statement-4

workflow
=======
  boolean condition true means statement-1,2,3 will be executed otherwise
statement-4 will be executed.

eg#1.
   int x=0;
   if(x){
      System.out.println(x);
   }
   output:Compile time error

eg#2
   int x=0;
   if(x==0){

```
      System.out.println(x);
   }
   output:: 0

eg#3
   boolean res=false;
   if(res){
      System.out.println(res);
   }
   output:: nothing will be printed

eg#4
   boolean res=false;
   if(res=true){
      System.out.println(res);
   }
   output:: true

eg#5.
if (true)
System.out.println("hello");//dependent on if
System.out.println("hiee");//independent on if

valid cases
 a. if(true);
 b. if(false);
 c. if(true) System.out.println("hello");

invalid cases
 a. if(true) int x =10;
 b. if(true){ int x =10;} System.out.println(x);


Note:
1. If we are writing only one statement inside if block, then cury braces are
optional.
2. When we write if,else block is also optional.
3. only one statement inside if block,curly braces are optional and that one
statement should never be a declarative statement.
4. If we are writing declarative statement inside if block with curly braces that
variable scope is limited to only if block.
5. Nesting of if-else[else-if ladder] is not prefered in coding.
if()
{

}
else if()
{

}
else if()
{

}
else if()
{

}
else
```

```
{

}
```
Solution : switch case.

+++++++
switch
+++++++
 if several options are available then it is not recomended to use if-else ladder we should go for "switch" statement.
 switch statement improves the readiblity of the code.

syntax:
```
 switch(x)
 {
      case 1: action1
      case 2: action2
      case 3: action3
            ;;;;
      default: default action;

 }
```

Unitll JDK1.4 version the allowed types for switch argument are "byte,short,char,int"
From JDK1.5 version onwards Wrapper classes are also possible "Byte,Short,Character,Integer" and enum.
From JDK1.7 version owards we can also give "String".

Conclusions::
 1. Curly braces are mandatory in switch statement.
 2. If we are writing any statement inside switch then those statements should be a part of case or default.
 3. If we are not writing any statement inside switch then case,default is optional.

eg#1.
```
int x= 10;
switch (x){
      System.out.println("hiee");
}
```
Output: CE

eg#2.
```
int x= 10;
int y= 20;
switch (x){//byte,short,int,char
      case 10: System.out.println(10);
      case y: System.out.println(20);//CE: constant required
}
```

eg#3.final -> This keyword makes the variable compile time constant, so compiler will be knowing the value of the variable.
```
int x= 10;
final int y= 20;
switch (x){//byte,short,int,char
      case 10: System.out.println(10);
      case y: System.out.println(20);
```

```
}
Output: Code will be compiled

Note: In java language, the memory for the variable will be given at the runtime,
so we say java language as "Dynamic programming language".


eg#4.switch argument and case label can have expression, but case label should be
constant expression.
int x= 10;
switch (x+1){
      case 10:
      case 10+20:
      case 10+20+30:
}
Output: No ouput

eg#5. case label values should lie in the range of switch argument type, otherwise
it results in "Compiletime Error".
byte b= 10;
switch (b){//byte = -128 + 127
      case 10:System.out.println(10);
      case 100:System.out.println(100);
      case 1000:System.out.println(1000);//CE: not with in the range of byte
}

eg#6.
byte b= 10;
switch (b+1){//byte + int => int
      case 10:System.out.println(10);
      case 100:System.out.println(100);
      case 1000:System.out.println(1000);
}
Output: No Output


eg#7.
int x = 10;
switch (x){ //byte---int,short---int,char('a')---int(97), int
      case 97:System.out.println("97");
      case 99:System.out.println("99");
      case 'a':System.out.println("100");
      //case 97:System.out.println("100");
}
Ouput: CE: duplicate case labels are not allowed.

Conclusion
 a. Case label should be compile time constant.
 b. Case label can have expression, but it should be a compile time constant
expression.
 c. Case label value should be within the range of the switch argument type.
 d. Duplicate case label are not allowed.

++++++++++++++++++++++
Fall Through in Switch
++++++++++++++++++++++
 With in a switch, if any case is matched from that case onwards all the statements
will be executed untill the end of switch or break.
 This is called as "FallThrough" inside the switch.
```

The main advantage of fall through inside switch is we can define common actions for multiple cases.

```
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");
}
```

Ouptut

```
 x= 0
 0
 1

 x=1
  1

 x= 2
  2
 default

 x= 3
 default
```

++++++++++++++

Duplicate case

++++++++++++++

1. within switch we can take default only once.
2. if no cases are matched then only default case will be executed.
3. With in a switch, we can take default anywhere,but it is a convention to take default as last case.

eg::

```
int x = 3;
switch (x){
      default: System.out.println("default");
      case 0: System.out.println("0");
      case 1: System.out.println("1");
                  break;
      case 2: System.out.println("2");
}
```

Output

```
x=3
default
0
1
```

eg::

```
int x = 3;
switch (x){
      default: System.out.println("default");
      default: System.out.println("default");
}
```

Output: CE: duplicate default label.

++++++++++++++++++++

Iterative statements
++++++++++++++++++++
 while loop
  => If we are not aware of how many time to iterate, then we need to go for while loop.

syntax:
 while(boolean){
      ;;;;;;;;;
 }

eg::
while(resultSet.next()){
      ;;;;
      ;;;;
}

while(enumeration.hasMoreElements()){
      ;;;;
      ;;;;
}

while(itr.hasNext()){
      ;;;;
      ;;;;
}

Note: The argument to a while statement should be of "boolean type".if we are using anyother type it results in "CE".

eg#1.
while (1)
{
      System.out.printl("hello");//CE
}

eg#2.
while (true)
{
      System.out.printl("hello");
}
output: hello(infinite times)

eg#3.
while (true)
;

Output: no output(valid)

eg#4.
while (true)
int x= 10; //CE: delcaration not allowed

eg#5.
while (true)
{
      int x =10; //valid
}

```
eg#6.
while (true)
{
      int x =10;
}
System.out.println(x);//CE: can't find symbol.
```

unreachable code
++++++++++++++++

```
eg#1.
while (true)
{
      System.out.println("hello");
}
      System.out.println("hiee");//CE: unreachable code

eg#2
while (false)
{
      System.out.println("hello");//CE: unreachable code
}
      System.out.println("hiee");


eg#3.
int a= 10;
int b= 20;

while (a<b)//JVM :: 10<20 -> true
{
      System.out.println("hello");
}
      System.out.println("hiee");
```

Output: hello(infinite times)

```
eg#4.
final int a= 10;
final int b= 20;

while (a<b)//JVM :: 10<20 -> true
{
      System.out.println("hello");
}
      System.out.println("hiee");//unreachable code
```

Output: CE

```
eg#5.
final int a= 10;
      while (a<20)
      {
            System.out.println("hello");
      }
      System.out.println("hiee");//unreachable code
```

Output: CE

eg#6.

```
int a= 10;
      while (a<20)
      {
            System.out.println("hello");
      }
      System.out.println("hiee");
```

Output: hello(infinite times)

Note::
 => Every final variable will be replaced with corresponding value by the compiler
 => if any operation involves only constants then compiler is responsible to perform operation.
 => if any operation involves at least one variable, then compiler won't perform any operation, jvm will perform that operation.

++++++++++++
do-while loop
++++++++++++
 if we want to execute the body of the loop atleast once then we need to go for do-while loop.

Syntax:
```
 do{
      ;;;;;
      ;;;;;
      ;;;;;
 }while(boolean); // ; is mandatory
```

Conclusion
 a. curly braces are optional
 b. Without curly braces we can take only one statement and that statement should never be declarative statement.

eg::
```
do{
      System.out.println("hello");
}while(true);
```

output:: hello infinite times

eg::
```
do;while(true);
```
output:: compiles succesfull

eg::
```
do
  int x=10;
while(true);
```
output:: compile time error

eg::
```
do{
  int x=10;
}while(true);
```
output:: compilation succesfull

eg::
```
dowhile(true)
```

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

eg::
dowhile(true);
output:: compilation error.

eg::
do{
      System.out.println("sachin");
}while(true);
System.out.println("dhoni");//CE: unreachable code

eg::
do{
      System.out.println("sachin");
}while(false);
System.out.println("dhoni");

Output
sachin
dhoni

eg::
int a=10,b=20;
do{
      System.out.println("sachin");
}while(a<b);
System.out.println("dhoni");

Output: sachin(infinite times)


eg::
int a=10,b=20;
do{
      System.out.println("sachin");
}while(a>b);
System.out.println("dhoni");

output: sachin dhoni


eg::
final int a=10,b=20;
do{
      System.out.println("sachin");
}while(a<b);
System.out.println("dhoni");

output:Ce: unreachable(dhoni)



eg::
final int a=10,b=20;
do{
      System.out.println("sachin");
```

```
}while(a>b);
System.out.println("dhoni");

output: sachin dhoni


eg::
final int a=10,b=20;
do{
      System.out.println("sachin");
}while(a<b);
System.out.println("dhoni"); //unreachable

Output: CE


eg::
final int a=10;
do{
      System.out.println("sachin");
}while(a>20);
System.out.println("dhoni");

Output: sachin dhoni


forloop
======
 It is one of the most commonly used loop and best suitable if we know the no of
iterations in advanced.

Syntax::
  for(initialisation;condition;incr/dec)
  {
      //body of loop
  }

Note: curly braces are optional, we can take only one statement and that statement
should not be declarative.

eg::
int i=0;
for(System.out.println("hello");i<3;System.out.println("hi")){
  i++; //  i = 1,2,3
}

Output:: hello hi hi hi

eg:: //by default condition is kept as true by compiler
for(;;){
      System.out.println("hello");
}
Output:: hello(infinite times)

eg::
for(;;)
  int x=10;
output:: CE
```

```
eg::
for(;;){
      int x=10;
}
output:: no output

eg::
for(int i=0;i<true;i++){
  System.out.println("sachin");
}
System.out.println("dhoni");

output::CE

eg::
for(int i=0;i<false;i++){
  System.out.println("sachin");
}
System.out.println("dhoni");
output::CE

eg::
for(int i=0;;i++){
  System.out.println("sachin");
}
System.out.println("dhoni");//unreachable

Output::CE

eg::
int a=10,b=20;
for(int i=0;a<b;i++){
  System.out.println("sachin");
}
System.out.println("dhoni");
output:: sachin(infinite)


eg::
final int a=10
final int b=20;
for(int i=0;a<b;i++){
  System.out.println("sachin");
}
System.out.println("dhoni");//Unreachable code
output:: CE

+++++++++
Snippets
+++++++++
Q>
Given:
int x = 0;
int y = 10;
do {
      y--; // y = y-1
      ++x; // x = x+1
} while (x < 5);
```

```
System.out.print(x + "," + y);
```

What is the result?
A. 5,6
B. 5,5[Answer]
C. 6,5
D. 6,6

```
y = 9, x = 1 :: while(1<5) true
y = 8, x = 2 :: while(2<5) true
y = 7, x = 3 :: while(3<5) true
y = 6, x = 4 :: while(4<5) true
y = 5, x = 5 :: while(5<5) false
```

Q>

```
public static void main(String[] args) {
      for (int i = 0; i <= 10; i++) {
            if (i > 6) break;
      }
      System.out.println(i);
}
```

What is the result?
A. 6
B. 7
C. 10
D. 11
E. Compilation fails.[Answer: i not accesible outside for loop]
F. An exception is thrown at runtime.

Q>
Consider below code of Test.java file:

```
public class Test {
    public static void main(String[] args) {
        boolean b1 = 0;
        boolean b2 = 1;
        System.out.println(b1 + b2);
    }
}
```

What is the result of compiling and executing Test class?
A. 0
B. 1
C. true
D. false
E. compilation error[Answer: boolean means only true,false]

Q>
Given:

```
float pi = 3.141f;
if (pi > 3) {
      System.out.print("pi is bigger than 3. ");
}else {
      System.out.print("pi is not bigger than 3. ");
}finally {
      System.out.println("Have a nice day.");
}
```

What is the result?
A. Compilation fails.[Answer:finally without try]

B. pi is bigger than 3.
C. An exception occurs at runtime.
D. pi is bigger than 3. Have a nice day.
E. pi is not bigger than 3. Have a nice day.

Q>
Given:

```
public static void main(String[] args) {
int i = 3; //line-12
switch(i) { //line-13
            case 3: System.out.println("three"); break;
            default: System.out.println("other"); break;//line-15
        }
}
```

What is the result?
A. three[Answer]
B. other
C. An exception is thrown at runtime.
D. Compilation fails because of an error on line 12.
E. Compilation fails because of an error on line 13.
F. Compilation fails because of an error on line 15.

Q>
Given:

```
public static void main(String[] args) {
boolean i = true; //line-12
switch(i) { //line-13
            case true: System.out.println("true"); break;
            case false: System.out.println("false"); break;
            default: System.out.println("other"); break;//line-15
        }
}
```

What is the result?
A. true
B. false
C. An exception is thrown at runtime.
D. Compilation fails because of an error on line 12.
E. Compilation fails because of an error on line 13.[Answer: switch arg types can be : byte,short,int,char]
F. Compilation fails because of an error on line 15.