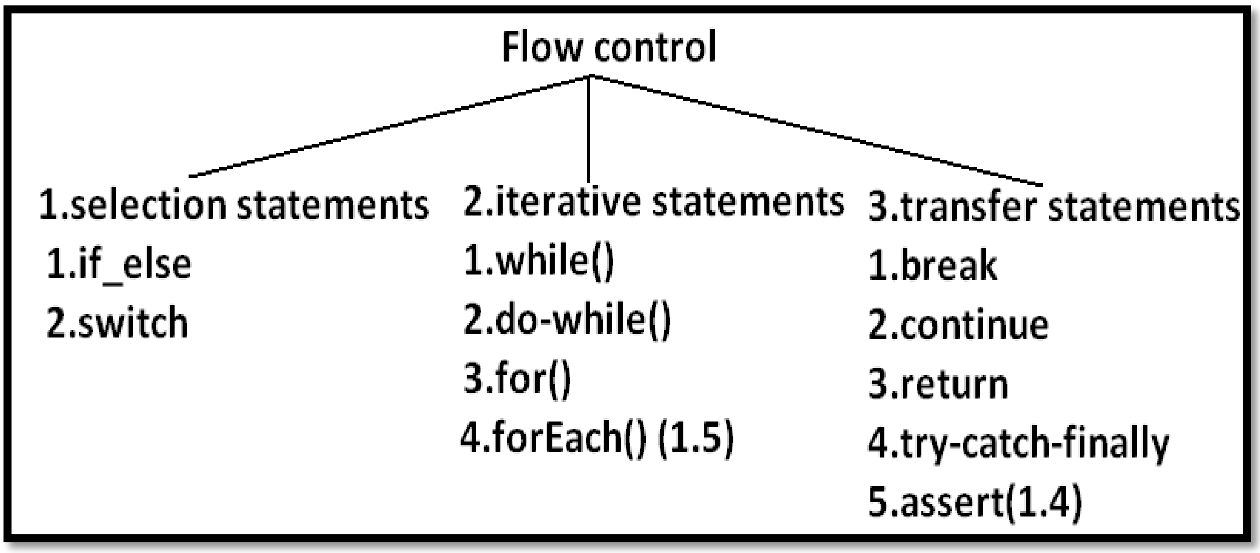
Flow Control

Agenda:

Introduction:

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

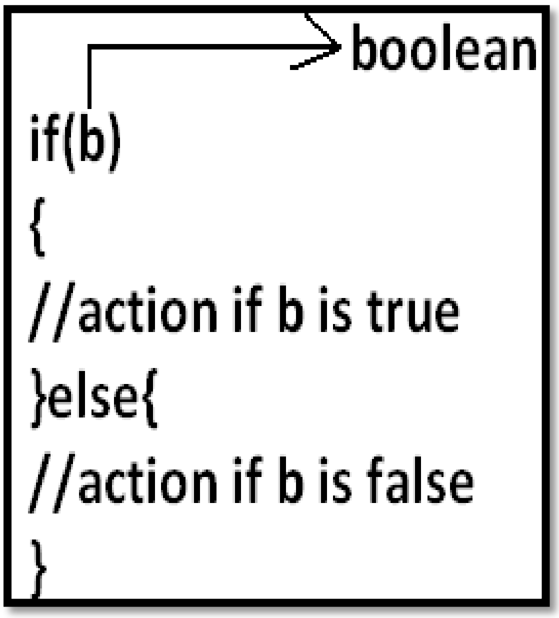
Diagram:



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
```

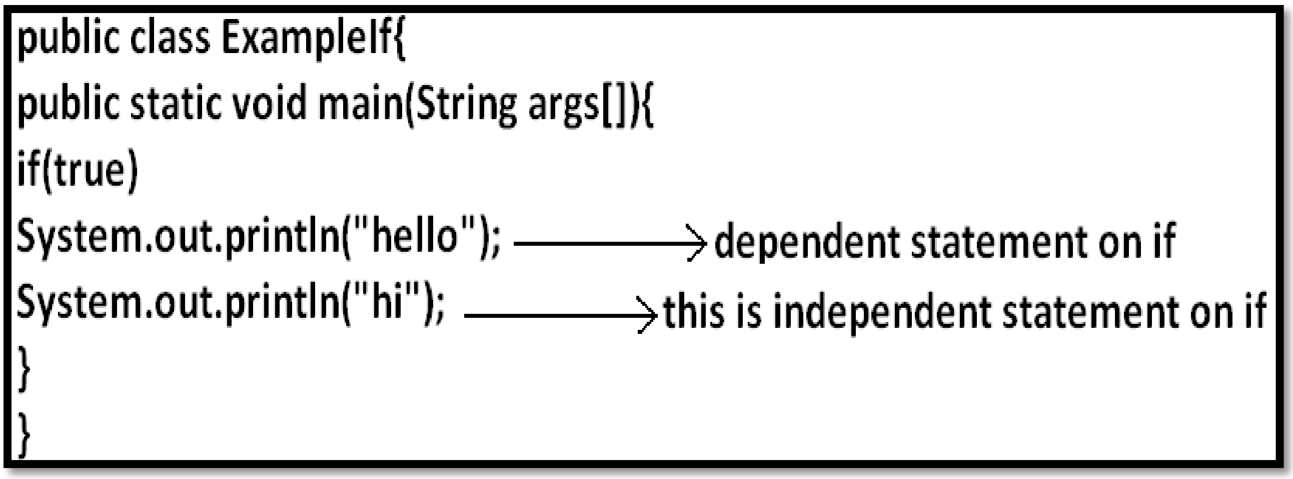
Example 8:

```
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:

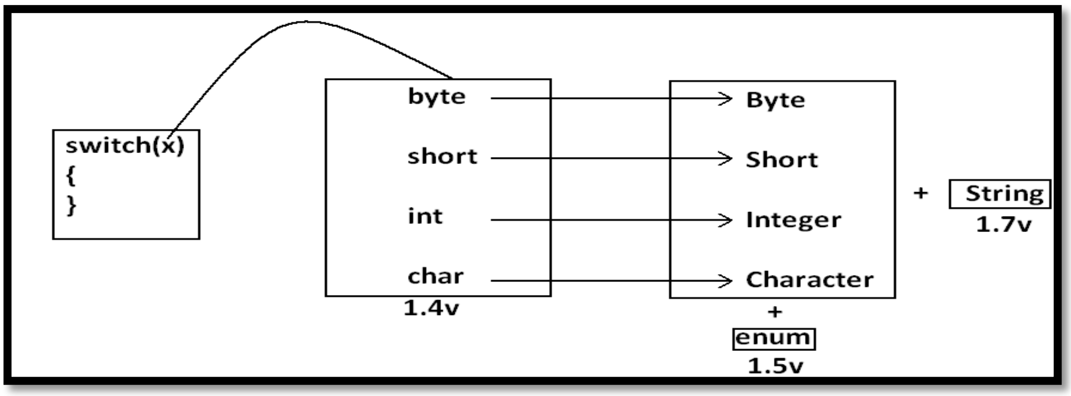
If several options are available then it is not recommended to use if-else we should go for switch statement. Because it improves readability of the code.

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.**

Diagram:



- 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:
```

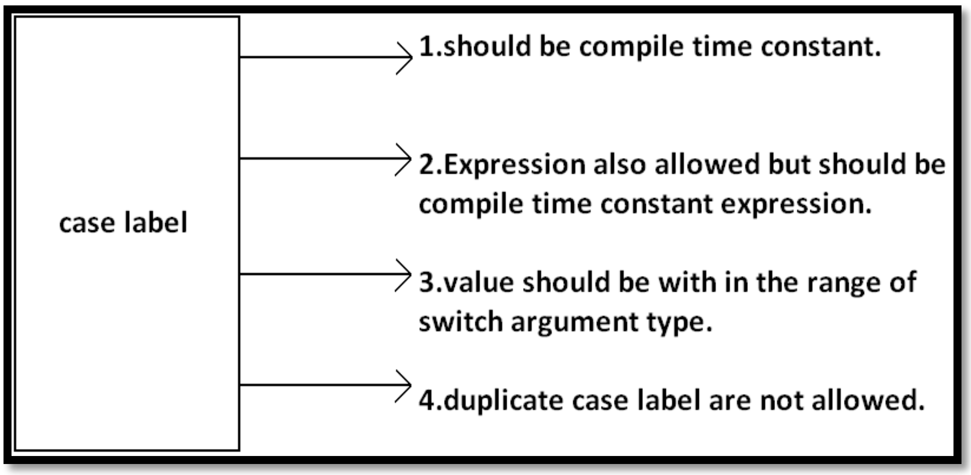
Duplicate case labels are not allowed.

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:

Diagram:


case label
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:

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.

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:

- **Within the switch we can take the default only once**
- **If no other case matched then only default case will be executed**
- **Within the switch we can take the default anywhere, but it is convention 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
```

ITERATIVE STATEMENTS

While loop:

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

Example 1:

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

Example 2:

```
while(e.hasMoreElements())
{
----------
----------
----------
}
```

Example 3:

```
while(itr.hasNext())
{
----------
----------
----------
}
```

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

Example 1:

```
public class ExampleWhile{
public static void main(String args[]){
while(1)
{
System.out.println("hello");
}}}
OUTPUT:
Compile time error.
D:\Java>javac ExampleWhile.java
ExampleWhile.java:3: incompatible types
found   : int
required: boolean
```

```
while(1)
```

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

Example 2:

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

Example 3:

```
public class ExampleWhile{
public static void main(String args[]){
while(true);
}}
OUTPUT:
No output.
```

Example 4:

```
public class ExampleWhile{
public static void main(String args[]){
while(true)
int x=10;
}}
OUTPUT:
Compile time error.
D:\Java>javac ExampleWhile.java
ExampleWhile.java:4: '.class' expected
int x=10;
ExampleWhile.java:4: not a statement
int x=10;
```

Example 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.
D:\Java>javac ExampleWhile.java
ExampleWhile.java:7: unreachable statement
System.out.println("hi");
```

Example 7:

```
public class ExampleWhile{
public static void main(String args[]){
while(false)
{
System.out.println("hello");
}
System.out.println("hi");
}}
OUTPUT:
D:\Java>javac ExampleWhile.java
ExampleWhile.java:4: unreachable statement
{
```

Example 8:

```
public class ExampleWhile{
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).
```

Example 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.
D:\Java>javac ExampleWhile.java
ExampleWhile.java:8: unreachable statement
System.out.println("hi");
```

Example 10:

```
public class ExampleWhile{
public static void main(String args[]){
final int a=10;
while(a<20)
{
System.out.println("hello");
}
System.out.println("hi");
}}
OUTPUT:
D:\Java>javac ExampleWhile.java
ExampleWhile.java:8: unreachable statement
System.out.println("hi");
```

Note:

- **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.**

Example 11:

```
public class ExampleWhile{
public static void main(String args[]){
int a=10;
while(a<20)
{
System.out.println("hello");
}
System.out.println("hi");
}}
OUTPUT:
Hello (infinite times).
```

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.

Example 1:

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

Example 2:

```
public class ExampleDoWhile{
public static void main(String args[]){
do;
while(true);
}}
Output:
Compile successful.
```

Example 3:

```
public class ExampleDoWhile{
public static void main(String args[]){
do
int x=10;
while(true);
}}
Output:
D:\Java>javac ExampleDoWhile.java
ExampleDoWhile.java:4: '.class' expected
int x=10;
ExampleDoWhile.java:4: not a statement
```

```
int x=10;
ExampleDoWhile.java:4: ')' expected
int x=10;
```

Example 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).
```

Rearrange the above Example:

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

Example 6:

```
public class ExampleDoWhile{
public static void main(String args[]){
do
while(true);
}}
Output:
Compile time error.
D:\Java>javac ExampleDoWhile.java
ExampleDoWhile.java:4: while expected
while(true);
ExampleDoWhile.java:5: illegal start of expression
}
```

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.
D:\Java>javac ExampleDoWhile.java
ExampleDoWhile.java:8: unreachable statement
System.out.println("hi");
```

Example 8:

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

Example 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).
```

Example 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
```

Example 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.
D:\Java>javac ExampleDoWhile.java
ExampleDoWhile.java:9: unreachable statement
System.out.println("hi");
```

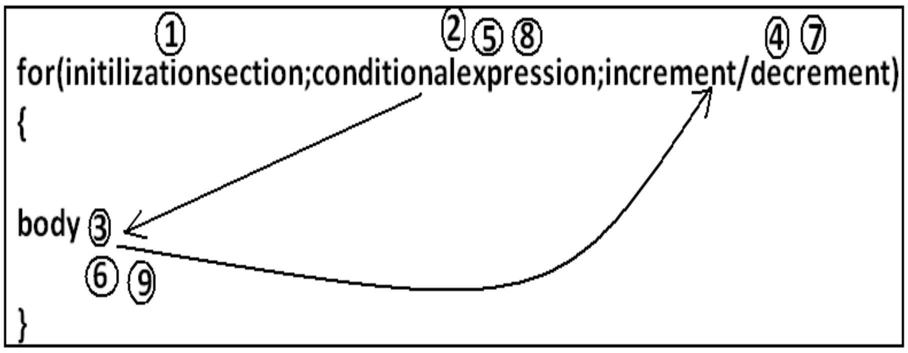
Example 12:

```
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:
D:\Java>javac ExampleDoWhile.java
D:\Java>java ExampleDoWhile
Hello
Hi
```

For Loop:

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

Syntax:



for(initilizationsection;conditionalexpression;increment/decrement)
{
body
}
① ② ⑤ ⑧ ④ ⑦ ③ ⑥ ⑨

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

Initilizationsection:

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 1:

```
public class ExampleFor{
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");
}}}
Output:
D:\Java>javac ExampleFor.java
D:\Java>java ExampleFor
Hello u r sleeping
No boss, u only sleeping
No boss, u only sleeping
No boss, u only sleeping
```

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.

Increment and decrement section:

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

Example:

```
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:
D:\Java>javac ExampleFor.java
D:\Java>java ExampleFor
Hello
Hi
Hi
Hi
```

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

Example:

```
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:

Example 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.
D:\Java>javac ExampleFor.java
ExampleFor.java:6: unreachable statement
System.out.println("hi");
```

Example 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.
D:\Java>javac ExampleFor.java
ExampleFor.java:3: unreachable statement
for(int i=0;false;i++){
```

Example 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.
D:\Java>javac ExampleFor.java
ExampleFor.java:6: unreachable statement
System.out.println("hi");
```

Example 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).
```

Example 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:
D:\Java>javac ExampleFor.java
ExampleFor.java:7: unreachable statement
System.out.println("hi");
```

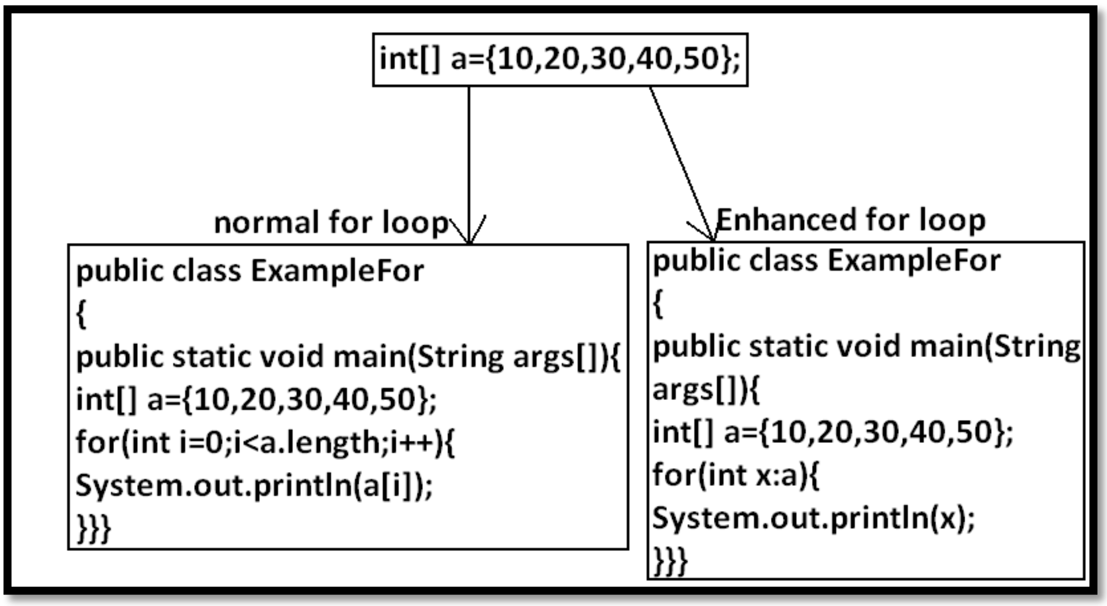
For each:(Enhanced for loop)

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

Example 1:

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

Example:



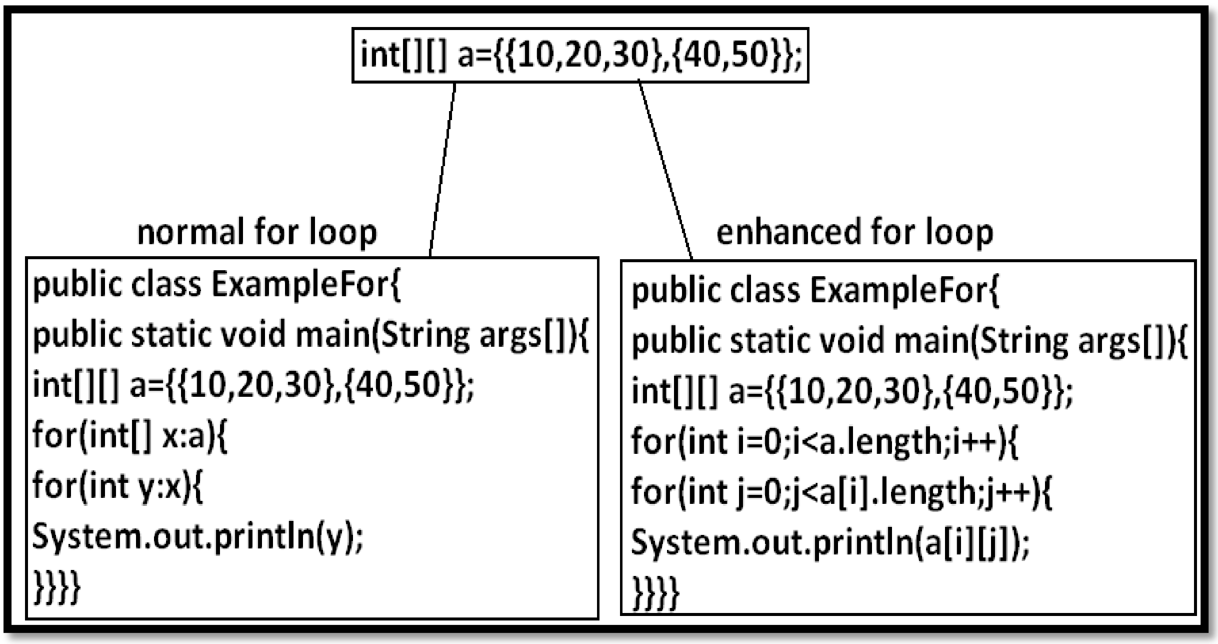
```
public class ExampleFor
{
public static void main(String args[]){
int[] a={10,20,30,40,50};
for(int i=0;i<a.length;i++){
System.out.println(a[i]);
}}}
```

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

```
Output:
D:\Java>javac ExampleFor.java
D:\Java>java ExampleFor
10
20
30
40
50
```

Example 2:

Write code to print the elements of 2 dimensional arrays by using 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[] x:a){
for(int y:x){
System.out.println(y);
}}}}
```

```
public class ExampleFor{
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]);
}}}}
```

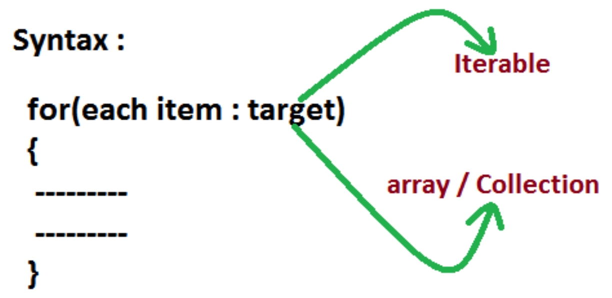
Example 3:

Write equivalent code by For Each loop for the following for loop.

```
public class ExampleFor{
public static void main(String args[]){
for(int i=0;i<10;i++)
{
System.out.println("hello");
}}}
Output:
D:\Java>javac ExampleFor1.java
D:\Java>java ExampleFor1
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

- **We can't write equivalent for each loop.**
- **For each loop is the more convenient loop to retrieve the elements of arrays and collections, but its main limitation is it is not a general purpose loop.**
- **By using normal for loop we can print elements either from left to right or from right to left. But using for-each loop we can always print array elements only from left to right.**

Iterator Vs Iterable(1.5v)



Syntax :

```
for(each item : target)
{
 ---------
 ---------
}
```

- 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 methoditerator().
- <u>Syntax:</u> public Iterator iterator();
- Every array class and Collection interface already implements Iterable interface.

Difference between Iterable and Iterator:

Iterable	Iterator
It is related to forEach loop	It is related to Collection
The target element in forEach loop should be Iterable	We can use Iterator to get objects one by one from the collection
Iterator present in java.lang package	Iterator present in java.util package
contains only one method iterator()	contains 3 methods hasNext(), next(), remove()
Introduced in 1.5 version	Introduced in 1.2 version

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

Example 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:\Java>javac Test.java
D:\Java>java Test
Hello
```

Inside loops :

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

Example 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:
D:\Java>javac Test.java
D:\Java>java Test
0
1
2
3
4
```

Inside Labeled block :

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

Example:

```
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:
D:\Java>javac Test.java
D:\Java>java Test
begin
hello
```

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

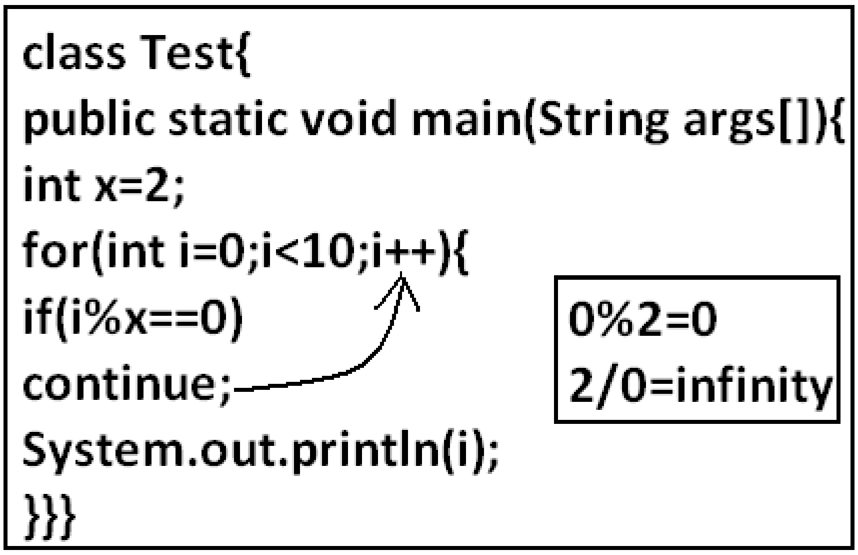
Example:

```
class Test{
public static void main(String args[]){
int x=10;
 if(x==10)
 break;
 System.out.println("hello");
}
}
Output:
Compile time error.
D:\Java>javac Test.java
Test.java:5: break outside switch or loop
break;
```

Continue statement:

We can use continue statement to skip current iteration and continue for the next iteration.

Example:



```
class Test{
public static void main(String args[]){
int x=2;
for(int i=0;i<10;i++){
if(i%x==0)
continue;
System.out.println(i);
}}}
```

```
0%2=0
2/0=infinity
```

```
Output:
D:\Java>javac Test.java
D:\Java>java Test
1
3
5
7
9
```

We can use continue only inside loops if we are using anywhere else we will get compile time error saying "continue outside of loop".

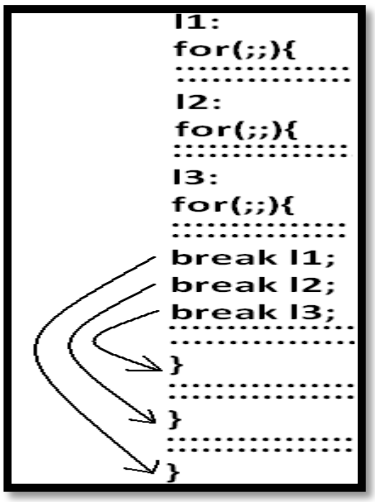
Example:

```
class Test
{
public static void main(String args[]){
int x=10;
if(x==10);
continue;
System.out.println("hello");
}
}
Output:
Compile time error.
D:\Enum>javac Test.java
Test.java:6: continue outside of loop
continue;
```

Labeled break and continue statements:

In the nested loops to break (or) continue a particular loop we should go for labeled break and continue statements.

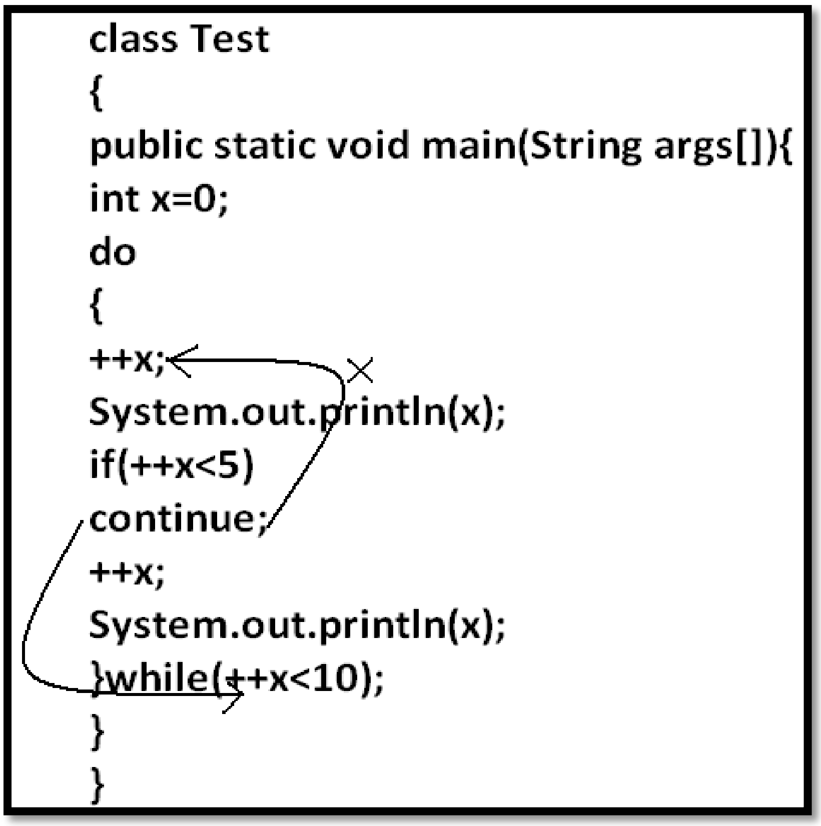
Syntax:


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

Example:

```
class Test
{
public static void main(String args[]){
l1:
 for(int i=0;i<3;i++)
 {
   for(int j=0;j<3;j++)
   {
     if(i==j)
       break;
         System.out.println(i+"........."+j);
    }
   }
 }
}

Break:
1.........0
2.........0
2.........1

Break l1:
No output.

Continue:
0.........1
0.........2
1.........0
1.........2
2.........0
2.........1

Continue l1:
1.........0
2.........0
2.........1
```

Do-while vs continue (The most dangerous combination):



```
class Test
{
public static void main(String args[]){
int x=0;
do
{
++x;
System.out.println(x);
if(++x<5)
continue;
++x;
System.out.println(x);
}while(++x<10);
}
}
```

```
Output:
1
4
6
8
10
```

Compiler won't check unreachability in the case of if-else it will check only in loops.

Example 1:

```
class Test
{
public static void main(String args[]){
while(true)
{
System.out.println("hello");
}
System.out.println("hi");
}
}
Output:
Compile time error.
D:\Enum>javac Test.java
```

```
Test.java:8: unreachable statement
System.out.println("hi");
```

Example 2:

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

Q1. Consider the code

```java
public class Test
{
   public static void main(String[] args)
   {
      String stuff="X";
      String res=null;
      if(stuff.equals("X"))
      {
         res="A";
      }
      else if(stuff.equals("Y"))
      {
         res="B";
      }
      else
      {
         res="C";
      }
   }
}
```

Which of the following code can replace nested if-else?
A) res=stuff.equals("X") ? "A" : stuff.equals("Y") ? "B" : "C";
B) res=stuff.equals("X") ? stuff.equals("Y") ? "A" : "B" : "C";
C) res=stuff.equals("X") ? "A" else stuff.equals("Y") ? "B" : "C";
D) res=stuff.equals("X") ? res="A" : stuff.equals("Y") ? "B" : "C";

Answer: A

Q2. Consider the code

```java
public class Test
{
   public static void main(String[] args)
   {
      //line-1
      switch(x)
      {
         case 10:
            System.out.println("Ten");
            break;
         case 20:
            System.out.println("Twenty");
            break;
      }
   }
}
```

Which 3 code fragments can be independently inserted at line-1 to print Ten

A) byte x =10;
B) short x =10;
C) String x ="10";
D) long x =10;
E) double x =10;
F) Integer x = new Integer(10);

Answer: A,B,F

Q3. Consider the code

```
public class Test
{
   public static void main(String[] args)
   {
      boolean b = true;//line-1
      switch(b)
      {
         case true://line-2
            System.out.print("True");
            break;
         default:
            System.out.print("default");
      }
      System.out.println("Done");
   }
}
```

Which of the following changes are required to print TrueDone?

A) Replace line-1 with String b="true";
Replace line-2 with case "true";

B) Replace line-1 with boolean b=1;
Replace line-2 with case 1;

C) remove break statement

D) remove the default section

Answer: A

Q4. Consider the code

```java
public class Test
{
   public static void main(String[] args)
   {
      String color="Green";
      switch(color)
      {
         case "Red":
            System.out.println("Red");
         case "Blue":
            System.out.println("Blue");
            break;
         case "Green":
            System.out.println("Green");
            break;
         default:
            System.out.println("Default");
      }

   }
}
```

What is the output?

A) Red
Blue

B) Green
Default

C) Default

D) Green

Answer: D

Q5. Which of the following is true about switch statement?

A) It should contain the default section
B) The break statement, at the end of each case block is mandatory
C) Its case lable literals can be changed at runtime
D) Its expression must evaluate a single value

Answer : D

Comparison:

1. If we dont know the number of iterations in advanace then we should go for while loop.

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

3. If we know the number of iterations in advance then we should use for loop.

4. To retrieve the elements of arrays and collections then we should use for-each loop.

Q6. Consider the code

```
public class Test
{
   public static void main(String[] args)
   {
      int i =5;
      while(isAvailable(i))
      {
         System.out.print(i);//Line-1
         //Line-2
      }
   }
   public static boolean isAvailable(int i)
   {
      return i-- > 0 ? true : false;// Line-3
   }
}
```

Which modification enables the code to print 54321

A) Replace Line-1 with System.out.print(--i);
B) At Line-2, insert i--;
C) Replace Line-1 with --i; and , at Line-2 insert System.out.print(i);
D) Replace Line-3 with return (i> 0) ? true : false;

Answer : B

Q7. Consider the code

```
public class Test
{
   public static void main(String[] args)
   {
      int[] x= {1,2,3,4};
      int i =0;
      do
      {
         System.out.print(x[i]+" ");
         i++;
```

```
        }
        while (i<x.length -1);
    }
}
```

What is the output?

A) 1 2 3 4
B) 1 2 3
C) Compilation Fails
D) IndexOutOfBoundsException

Answer: B

Q8. Consider the code

```
public class Test
{
    public static void main(String[] args)
    {
        int x =5;
        do
        {
            System.out.print(x-- +" ");
        }
        while (x==0);
    }
}
```

What is the result?

A) 5 4 3 2 1 0
B) 5 4 3 2 1
C) 4 2 1
D) 5
E) Nothing is printed

Answer: D

Q9. Consider the code

```
public class Test
{
    public static void main(String[] args)
    {
        int[][] x = new int[2][4];
        x[0]= new int[]{2,4,6,8};
        x[1]= new int[]{2,4};
        for(int[] x1: x)
        {
         for(int x2 :x1)
         {
            System.out.print(x2+" ");
         }
         System.out.println();
      }
   }
}
```

What is the output?

A) 2 4 6 8
2 4

B) 2 4
2 4

C) Compilation Fails

D) ArrayIndexOutOfBoundsException

Answer: A

Q10. Consider the code

```
public class Student
{
   String name;
   public Student(String name)
   {
      this.name=name;
   }
}
public class Test
{
   public static void main(String[] args)
   {
      Student[] s = new Student[3];
      s[1]= new Student("Durga");
      s[2]= new Student("Ravi");
      for(Student s1: s)
      {
         System.out.println(s1.name);
      }
   }
}
```

What is the output?

A) Durga
Ravi

B) Durga
 Ravi
 null

C) Compilation Fails

D) ArrayIndexOutOfBoundsException

E) NullPointerException

Answer: E

Q11. Consider the code

```
public class Test
{
   public static void main(String[] args)
   {
      int[]  data={10,20,30,40,50,30};
      int k= 30;
      int count=0;
      for(int x : data)
      {
         if( x!= k)
         {
            continue;
            count++;
         }
      }
      System.out.println(count);
   }
}
```

What is the result?

A) 0
B) 1
C) 2
D) Compilation Fails

Answer: D

Note: In the above code if we take count++; outside of if block then output is 2 (Option C).

Q12. Consider the code

```
public class Test
{
   public static void main(String[] args)
   {
      int wd = 0;
      String[] days={"sun","mon","wed","sat"};
      for(String s : days)
      {
         switch(s)
         {
            case "sat":
                case "sun":
               wd -= 1;
               break;
            case "mon":
               wd++;
            case "wed":
               wd += 2;
         }
      }
      System.out.println(wd);
   }
}
```

What is the output?

A) 3
B) 4
C) -1
D) Compilation Fails

Answer: A

Q13. Consider the code

```
public class Test
{
   public static void main(String[] args)
   {
      int[] x= {1,2,3,4,5}
      for(yyy)
      {
         System.out.print(a[i]);
      }
   }
}
```

Which option can replace yyy to enable the code to print 135?

A) int i=0;i<=4;i++
B) int i=0;i<5;i+=2
C) int i=1;i<=5;i++
D) int i=1;i<5;i+=2

Answer: B

Q14. Consider the code

```
public class Test
{
   public static void main(String[] args)
   {
      String[][] colors=new String[2][2];
      colors[0][0]="red";
      colors[0][1]="blue";
      colors[1][0]="green";
      colors[1][1]="yellow";
   }
}
```

Which code fragment prints red:blue:green:yellow:?

A)

```
for(int i=1;i<2;i++)
{
   for(int j =1; j<2;j++)
   {
      System.out.print(colors[i][j]+":");
   }
}
```

B)

```
for(int i=0;i<2;+++)
{
   for(int j =0; j<i;++j)
   {
      System.out.print(colors[i][j]+":");
   }
}
```

C)

```
for(String c: colors)
{
   for(String s: sizes )
   {
      System.out.print(s+":");
```

```
    }
}
```

D)

```
for(int i=0;i<2;)
{
    for(int j =0; j<2;)
    {
        System.out.print(colors[i][j]+":");
        j++;
    }
    i++;
}
```

Q15. Consider the following code

```
public class Test
{
    public static void main(String[] args)
    {
        int[] s={10,20,30};
        int size=3;
        int i =0;
        /* Line-1 */
        System.out.println("The Top Element:"+ s[i]);
    }
}
```

Which code fragment inserted at line-1 ,prints The Top Element:30 ?

A)

```
do
{
    i++;
}
while ( i>= size );
```

B)

```
while(i<size)
{
    i++;
}
```

C)

```
do
{
    i++;
}
while (i<size-1);
```

D)

```
do
{
   i++;
}
while (i<= size);
```

E)

```
while(i<= size-1)
{
   i++;
}
```

Answer: C

Q16. Consider the following code

```
public class Test
{
   public static void main(String[] args)
   {
      int n[];
      n= new int[2];
      n[0]=100;
      n[1]=200;

      n = new int[4];
      n[2]=300;
      n[3]=400;

      for(int x : n)
      {
         System.out.print(" "+x);
      }
   }
}
```

What is the result?

A) 100 200 300 400
B) 0 0 300 400
C) Compilation Fails
D) An exception thrown at runtime

Answer : B

Q17. Consider the following code

```java
public class Test
{
   public static void main(String[] args)
   {
      int[][] n= {{1,2},{3,4}};
      for(int i =n.length-1;i>=0;i--)
      {
         for(int x:n[i])
         {
            System.out.print(x);
         }
      }
   }
}
```

What is the result?
A) 1234
B) 3412
C) 4321
D) 2143

Answer : B

Q18. Given the code fragment:

int[] x ={10,20,30,40};

And the given requirements:

1. Process all the elements of the array in the order of entry.
2. Process all the elements of the array in the reverse order of entry.
3. Process alternating elements of the array in the order of entry.

Which two statements are true?

A) Requirements 1,2 and 3 can be implemented by using the enhanced for loop
B) Requirements 1,2 and 3 can be implemented by using the standard for loop
C) Requirements 2 and 3 CANNOT be implemented by using the standard for loop
D) Requirement 1 can be implemented by using the enhanced for loop
E) Requirement 3 CANNOT be implemented by using either the enhanced for loop or the standard for loop.

Answer: B,D

Q19. Given the following array:

int[] x = {10,20,30,40,50};

Which two code fragments independently print each element of this array?

A)

```
for(int i : x)
{
  System.out.print(x[i]+" ");
}
```

B)

```
for(int i : x)
{
  System.out.print(i+" ");
}
```

C)

```
for(int i=0 : x)
{
  System.out.print(x[i]+" ");
  i++;
}
```

D)

```
for(int i=0 ; i < x.length; i++)
{
  System.out.print(i+" ");
}
```

E)

```
for(int i=0 ; i < x.length; i++)
{
  System.out.print(x[i]+" ");
}
```

F)

```
for(int i=1 ; i < x.length; i++)
{
  System.out.print(x[i]+" ");
}
```

Answer : B and E

Explanation: Standard for loop is index based where as Enhanced for loop is Element based.

Q20. Consider the code

```
public class Test
{
   public static void main(String[] args)
   {
      String[] s={"A","B","C","D"};
      for(int i =0; i<s.length;i++)
      {
         System.out.print(s[i]+" ");
         if (s[i].equals("C"))
         {
            continue;
         }
         System.out.println("Done");
         break;
      }
   }
}
```

What is the result?

A) A B C D Done
B) A B C Done
C) A Done
D) Compilation fails

Answer : C

Q21. Consider the code

```
public class Test
{
   public static void main(String[] args)
   {
      String[][] s={{"A","B","C"},{"D","E"}};
      for(int i =0; i<s.length;i++)
      {
         for(int j =0; j<s[i].length;j++)
         {
            System.out.print(s[i][j]+" ");
            if(s[i][j].equals("B"))
            {
               break;
            }
         }
```

```
            continue;
        }
    }
}
```

What is the result?

A) A B C D E
B) A B C
C) A B D E
D) Compilation fails

Answer : C

Q22. Consider the following code

```
public class Test
{
    public static void main(String[] args)
    {
        int i = 0;
        int j = 7;
        for( i = 0; i < j-1 ; i= i+2)
        {
            System.out.print(i+" ");
        }

    }
}
```

What is the result?

A) 0 2 4 6
B) 0 2 4
C) 2 4 6
D) Compilation fails

Answer : B

Q23. Consider the following code

```
public class Test
{
    public static void main(String[] args)
    {
        String[][] s= new String[2][];
        s[0] = new String[2];
        s[1] = new String[5];
        int i =97;

        for(int a =0;a < s.length; a++)
        {
          for(int b =0; b< s.length; b++)
          {
            s[a][b]=""+i;
            i++;
          }
        }
        for( String[] s1: s)
        {
          for (String s2 : s1)
          {
            System.out.print(s2+" ");
          }
          System.out.println();
        }
    }
}
```

What is the result ?

A) 97 98
   99 100 null null null

B) 97 98
   99 100 101 102 103

C) Compilation fails

D) NullPointerException is thrown at Runtime

E) ArrayIndexOutOfBoundsException is thrown at Runtime

Q24. Consider the following code

```
public class Test
{
    public static void main(String[] args)
    {
        int[][] x = new int[2][4];
        x[0] = new int[]{1,2,3,4};
        x[1] = new int[]{1,2};

        for(int[] x1 : x)
        {
          for(int x2 : x1)
          {
            System.out.print(x2+" ");
          }
          System.out.println();
        }
    }
}
```

What is the result?

A) 1 2 3 4
1 2

B) 1 2
1 2

C) Compilation fails

D) ArrayIndexOutOfBoundsException is thrown at runtime

Answer : A