++++++++++++++++++++
Flow Control statement
++++++++++++++++++++
 1. Simple control statements
      a. if
      b. if-else
      c. switch(tricky in java)
 2. Looping control statements
      a. while[not aware of how many times iterations executes]
      b. dowhile[Atleast once if we want to execute]
      c. for[iterations if it is known at the begining]
      d. foreach(jdk1.5v)[Arrays and oops]
 3. Transfer control statements
      a. break
      b. continue
      c. return[in methods]
      d. System.exit(0)[in methods]

++++++++++++++++++++++++++
Transfer control statements
++++++++++++++++++++++++++
Break statement
  a. Inside switch to avoid fallthrough
  b. Inside loops to break the loop based on some condition
  c. Inside label block to break label block execution based on some condition.

1. break is used to avoid "fallthrough" in switch.
```
int x= 0;
switch(x){
      case 0: System.out.println("hello");
            break;
      case 1: System.out.println("hiee");
}
```

2. we can use break inside loop to break the loop based on some condition.
```
for (int i=0;i<=10 ;i++ )
{
      if (i==5)
         break;
      System.out.println(i);//0 1 2 3 4
}
```

3. break can be used along with blocks also.
```
int x= 10;
l1:{
      System.out.println("begin");
      if (x==10)
         break l1;
      System.out.println("end");
}
System.out.println("hello");
```

Output
```
begin
hello
```

Note: break can be used inside "switch or loop", any other places if we try to use it results in "CompileTime-Error."

```
int x= 10;
System.out.println("hello");
if (x==10)
  break;
System.out.println("hiee");
```

++++++++
continue
++++++++
=> we can use continue statement to skip the current iteration and continue for next iteration

eg#1.

```
int x= 2;
for (int i=0;i<10 ;i++ )
{
      if (i%x==0)
         continue;
      System.out.println(i);
}
```

Output
 1 3 5 7 9

eg#2. we can use continue only inside loops, if we try to use outside it would result in "CompileTime Error".

```
int x= 10;
if (x==10)
{
   continue;
}
System.out.println("hello");
```

do-while with continue
++++++++++++++++++++++

```
int x= 0;
do
{
      x=x+1;
      System.out.println(x);

      if((x=x+1) < 5)
            continue;
      x=x+1;
      System.out.println(x);
}while (++x<10);
```

output
1
4
6
8
10

Note: Compiler won't check for unreachablity in case of "if-else" statement, whereas it checks for unreachability in case of loops.

eg#1.

```
while(true)
  System.out.println("hello");
System.out.println("hiee");//CE: unreachable statement

eg#2.
if (true)
    System.out.println("hello");//hello
else
    System.out.println("hiee");


++++++++++++++++++++++++++++++Operators++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
1. Increment and decrement operator
2. Arithmetic operator
3. Relational operator
4. Equality operator
5. instanceof operator
6. bitwise operator
7. shortcircuit operator
8. typecast operator
9. assignment operator
10.conditional operator
11. [] operator
12. precedence of operator
13. evaluation of java operands

++++++++++++++++++++++++++++++++
Increment and decrement operator
++++++++++++++++++++++++++++++++
increment operator
  a. pre-increment[first increment and then use it]
            eg: int x = 10;
                int y = ++x; // x  = 11, y = 11

  b. post-increment[first use it and then increment it]
            eg: int x = 10;
                int y = x++; // x= 11 , y = 10

Note:
 1. increment or decrement can be applied only on variables, but not on values
directly.
            eg: int y = 10++; //CE

 2. we can't perform nesting of increment or decrement  operator, it would result
in compile time error.
        int x= 10;
      int y= ++(++x);//CE
      System.out.println("x = " + x);
      System.out.println("y = " + y);

 3. For a final variable, increment or decrement operation can't be done.
      final int x=  4;
      x++;//CE
      System.out.println(x);

 4. we can't apply increment or decrement operator on boolean type, where as it can
be applied on other primitive types.
            int x= 10;
```

```
            x++;
            System.out.println(x);//11

            char ch='a';
            ch++;
            System.out.println(ch);//b

            double d = 10.5;
            d++;
            System.out.println(d);//11.5


            boolean b= true;
            b++;//CE
            System.out.println(b);
```

5. What is the difference b/w b = b+1 and b++?

```
      byte b= 10;
        b = b+1; //CE (byte+int :: int)
      System.out.println(b);


      byte b = 10;
      b++;  // b = (byte)(b+1);
      System.out.println(b);//11
```

++++++++++++++++++++
Arithmetic operator
++++++++++++++++++++
operators : +,-,*,/,%

Note: if we apply arithmetic operators b/w 2 variables then the result type is always max(int,typeof a, typeof b)

```
      eg: byte + byte   = int
          byte + short  = int
          int + double  = double
          char + char   = int
            char + double = double
```

```
eg#1.
System.out.println('a' + 'b');//195
System.out.println('a' + 1);// 98
System.out.println('a' + 1.2);//98.2
```

eg#2. In intergral arithmetic(byte,short,int,long) there is no way to represent "Infinity", so result will be
"ArithmeticException".
In case of double,float types, there is a possiblity of storing "Infinity" so the result will be "Infinity".

```
System.out.println(10/0.0); // int/double = +double
System.out.println(-10/0.0); // -int/double = -double
System.out.println(0/0); // int/int = int
```

```
output
Infinity
-Infinity
ArithmeticException :/by zero
```

++++++++
Snippets
++++++++
What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        boolean b = true;
        switch(b) {//switch arg are : byte,short,int,char,String,enum
            case true:
                System.out.println("ONE");
            case false:
                System.out.println("TWO");
            default:
                System.out.println("THREE");
        }
    }
}
```

A. ONE
   TWO
   THREE

B. TWO
   THREE

C. THREE

D. None of the above options

Answer: D

Q>

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

What is the result?
A. 6
B. 7
C. 10
D. 11
E. Compilation fails.
F. An exception is thrown at runtime.

Answer: E

Q>
Given code of Test.java file:

```
public class Test {
      public static void main(String[] args) {
          byte b1 = 10; //Line n1
        int i1 = b1; //Line n2
          byte b2 = i1; //Line n3[implicit type casting is not possible]
          System.out.println(b1 + i1 + b2);
    }
}
```

What is the result of compiling and executing Test class?

A. Line n1 causes compilation error
B. Line n2 causes compilation error.
C. Line n3 causes compilation error.
D. 30 printed on to console.

Answer: C

Q>
What will be the result of compiling and executing the Test class?

```
public class Test {
    public static void main(String[] args) {
        int grade = 75;
        if(grade > 60)
            System.out.println("Congratulations");
        System.out.println("You passed");
        else
            System.out.println("You failed");
    }
}
```

A. Congratulations
B. Congratulations
   You passed
C. You failed
D. copmilation error[misplaced else]

Q>
What will be the output of compiling and executing the Test class?

```
public class Test {
    public static void main(String[] args) {
        int x = 2;
        switch (x) {
            default:
                System.out.println("Still no idea what x is");
            case 1:
                System.out.println("x is equal to 1");
                break;
            case 2:
                System.out.println("x is equal to 2");
                break;
            case 3:
                System.out.println("x is equal to 3");
                break;
        }
    }
}
```

A. x is equal to 2[Answer]
B. Compilation error
C. Still no idea what x is
D. Produces nooutput