USN

CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 1 – Sept. 2019

Sub:	Programming in JAVA				Sub Code:	17CS561	Branch:	ECE/TE
Date:	09-09-2019	Duration:	90 min's	Max Marks: 50	Sem / Sec:	5 (ALL SEC)		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1 (a)	Explain three OOP principles.	[2+2+2]	CO1	L2
(b)	Explain steps to execute simple java program and role of JVM in execution.	[2+2]	CO1	L2
2 (a)	Explain scope and lifetime of variables with an example.	[5]	CO1	L2
(b)	What is narrowing and widening (type casting)? Explain with an example.	[5]	CO1	L2
3	Explain all looping control statement with syntax and example.	[10]	CO2	L3
4 (a)	Explain Ternary operator. Write a java program to display largest of three numbers using ternary operator.	[02+04]	CO2	L3
(b)	Evaluate i) (5 >> 2) + 3 ii) 5 ! 4 + 2 >> 1 & 7	[02+02]	CO2	L3
5 (a)	Explain if, if..else, nested if statements with syntax and suitable example.	[2+2+2]	CO2	L3
(b)	Explain use of for..each loop with suitable example. Write program to find sum of first five numbers using for..each loop.	[2+2]	CO2	L3

USN

CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 1 – Sept. 2019

Sub:	Programming in JAVA				Sub Code:	17CS561	Branch:	ECE/TE
Date:	09-09-2019	Duration:	90 min's	Max Marks: 50	Sem / Sec:	5 (ALL SEC)		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1 (a)	Explain three OOP principles.	[2+2+2]	CO1	L2
(b)	Explain steps to execute simple java program and role of JVM in execution.	[2+2]	CO1	L2
2 (a)	Explain scope and lifetime of variables with an example.	[5]	CO1	L2
(b)	What is narrowing and widening (type casting)? Explain with an example.	[5]	CO1	L2
3	Explain all looping control statement with syntax and example.	[10]	CO2	L3
4 (a)	Explain Ternary operator. Write a java program to display largest of three numbers using ternary operator.	[02+04]	CO2	L3
(b)	Evaluate i) (5 >> 2) + 3 ii) 5 ! 4 + 2 >> 1 & 7	[02+02]	CO2	L3
5 (a)	Explain if, if..else, nested if statements with syntax and suitable example.	[2+2+2]	CO2	L3
(b)	Explain use of for..each loop with suitable example. Write program to find sum of first five numbers using for..each loop.	[2+2]	CO2	L3

6 (a)	Write a program to display following pattern	[6]	CO2	L3
(b)	Explain use of Short circuit properties for logical operators (&& and ||).	[4]	CO2	L2
7 (a)	Write program to display Prime numbers between 1 to 100.	[5]	CO2	L3
(b)	Write a program to find factorial of a number 5.	[5]	CO2	L3
8	Explain break, continue and return statement with Syntax and example.	[10]	CO2	L3

6 (a)	Write a program to display following pattern	[6]	CO2	L3
(b)	Explain use of Short circuit properties for logical operators (&& and ||).	[4]	CO2	L2
7 (a)	Write program to display Prime numbers between 1 to 100.	[5]	CO2	L3
(b)	Write a program to find factorial of a number 5.	[5]	CO2	L3
8	Explain break, continue and return statement with Syntax and example.	[10]	CO2	L3

Scheme

Question #	Description	Marks Distribution		Max Marks
1 a	Each OOP principal	2 M 2 M 2M	6M	6M
1 b	steps to execute simple java program role of JVM	2 M 2 M	4M	4M
2a	scope and lifetime of variables with an example	4+1 M	5M	5M
2b	narrowing and widening of expression . Explain with an example.	4+1 M	5M	5M
3	Looping control statements While Do..while for	3 M 3M 4M	10M	10M
4 a	Ternary operator. Write a java program to display largest of three numbers using ternary operator.	2M 4M	6M	6M
4 b	i) (5 >> 2) + 3 ii) 5 ! 4 + 2 >> 1 & 7	2 M 2 M	4M	4M

5 a	if,
	if..else,
	nested if statements
	with syntax and suitable example.
		2M
		2M
		2M
			6M	6M

5 b	For each loop Sum program	2M 2M	4M	4M
6a	program to display following pattern	6M	6M	6M
6 b	short circuit property with example.	4M	4M	4M
7a	Program to display Prime numbers between 1 to 100.	5M	5M	5M
7 b	Program to check given year is leap year or not.	5M	5M	5M
8	Break Continue Return	4M 4M 2M	10M	10M

Solution

Q1 a> Explain three OOP principles.

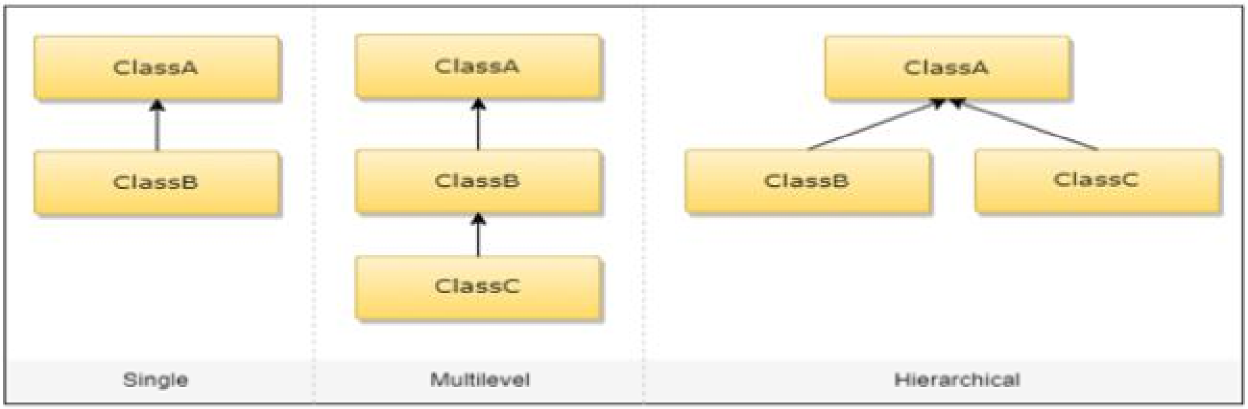
Encapsulation: Encapsulation can be defined as the procedure of casing up of codes and their associated data jointly into one single component.

In simple terms, encapsulation is a way of packaging data and methods together into one unit. Encapsulation gives us the ability to make variables of a class keep hidden from all other classes of that program or namespace.

Hence, this concept provides programmers to achieve data hiding. Programmers can have full control over what data storage and manipulation within the class

Inheritance: Inheritance can be defined as the procedure or mechanism of acquiring all the properties and behavior of one class to another, i.e., acquiring the properties and behavior of child class from the parent class.

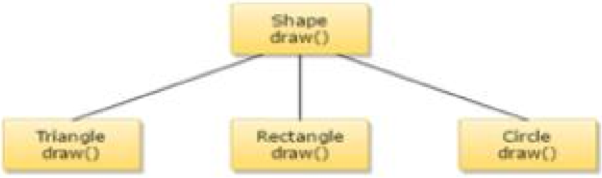
Java supports three types of inheritance. These are:



Java Inheritance

Polymorphism: The word polymorphism means having multiple forms. The term Polymorphism gets derived from the Greek word where poly + morphos where poly means many and morphos means forms.

- Static Polymorphism
- Dynamic Polymorphism.



Polymorphism

b) **Explain steps to execute simple java program and role of JVM in execution** [2+2]

Steps:

a) After typing the program in the terminal we have to type javac progranmane.java and hit enter

b) Then again in the terminal (if no error is there) the we have to type java program name and hit enter key

Role of JVM in execution can be described as follows:

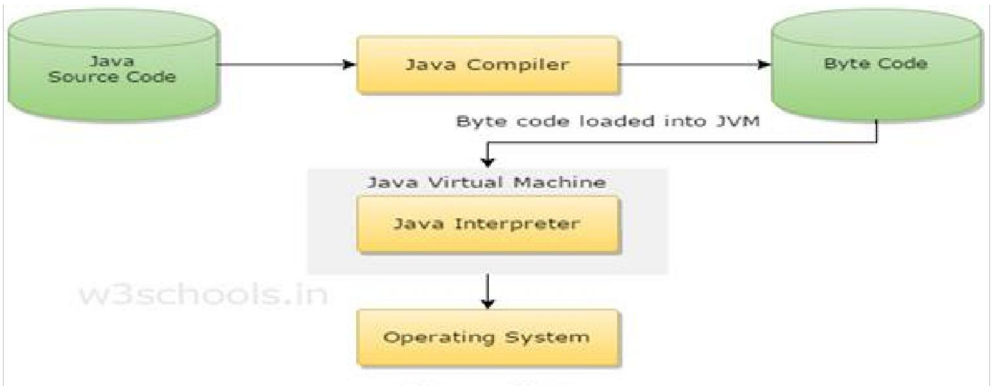


Diagram of JVM

- Reading Bytecode.
- Verifying bytecode.

Linking the code with the library

2) a) Explain scope and lifetime of variables with an example [5]

- Java allows variables to be declared within any block. A block defines a scope. Thus, each time you start a new block, you are creating a new scope.
- A scope determines what objects are visible to other parts of your program. It also determines the lifetime of those objects
- In Java, the two major scopes are:

i) defined by a class and

ii) those defined by a method.

- As a general rule, variables declared inside a scope are not visible (that is, accessible) to code that is defined outside that scope. Thus, when we declare a variable within a scope, we are localizing that variable and protecting it from unauthorized access and/or modification.
- Each time you create a block of code, you are creating a new, nested scope. When this occurs, the outer scope encloses the inner scope. This means that objects declared in the outer scope will be visible to code within the inner scope. However, the reverse is not true. Objects declared within the inner scope will not be visible outside it.

```
// Demonstrate block scope.
class Scope {
public static void main(String args[]) {
int x; // known to all code within main
x = 10;
if(x == 10) { // start new scope
int y = 20; // known only to this block
// x and y both known here.
System.out.println("x and y: " + x + " " + y);
x = y * 2;
}
// y = 100; // Error! y not known here
// x is still known here.
System.out.println("x is " + x);
}
}
```

- Here is another important point to remember: variables are created when their scope is entered, and destroyed when their scope is left.
- If a variable declaration includes an initializer, then that variable will be reinitialized each time the block in which it is declared is entered.

b) What is narrowing and widening. Explain with an example. [5]

Widening : When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:

• The two types are compatible.

• The destination type is larger than the source type.

When these two conditions are met, a widening conversion takes place. For example, the int type is always large enough to hold all valid byte values, so no explicit cast statement is required. For widening conversions, the numeric types, including integer and floating-point types, are compatible with each other.

Narrowing: In case of stornt int value to a byte variable, conversion will not be performed automatically, because a byte is smaller than an int. This kind of conversion is sometimes called a narrowing conversion, To create a conversion between two incompatible types, we must use a cast. A cast is simply an explicit type conversion. It has this general form:

(target-type) value

For example, the following fragment casts an int to a byte. If the integer's value is larger than the range of a byte, it will be reduced modulo (the remainder of an integer division by the) byte's range.

```
int a;
byte b;
// ...
b = (byte) a;
```

3) Explain all loping control statement with syntax and example.

1. **while loop:** A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.
   Syntax :
2. while (boolean condition)
3. {
4.   loop statements...
5. }

- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop**
- Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

```
// Java program to illustrate while loop
class whileLoopDemo
{
   public static void main(String args[])
   {
      int x = 1;

      // Exit when x becomes greater than 4
      while (x <= 4)
      {
         System.out.println("Value of x:" + x);

         // Increment the value of x for
         // next iteration
         x++;
      }
   }
}
```

Output:

```
Value of x:1
Value of x:2
Value of x:3
Value of x:4
```

for loop: for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

Syntax:

```
for (initialization condition; testing condition; increment/decrement)
{
statement(s)
}
```

Initialization condition: Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.

Testing Condition: It is used for testing the exit condition for a loop. It must return a boolean value. It is also an **Entry Control Loop** as the condition is checked prior to the execution of the loop statements.

Statement execution: Once the condition is evaluated to true, the statements in the loop body are executed.

Increment/ Decrement: It is used for updating the variable for next iteration.

Loop termination: When the condition becomes false, the loop terminates marking the end of its life cycle.

```
// Java program to illustrate for loop.
class forLoopDemo
{
    public static void main(String args[])
    {
        // for loop begins when x=2
        // and runs till x <=4
        for (int x = 2; x <= 4; x++)
            System.out.println("Value of x:" + x);
    }
}
```

Output:

```
Value of x:2
Value of x:3
Value of x:4
```

Enhanced For loop

Java also includes another version of for loop introduced in Java 5. Enhanced for loop provides a simpler way to iterate through the elements of a collection or array. It is inflexible and should be used only when there is a need to iterate through the elements in sequential manner without knowing the index of currently processed element.

Also note that the object/variable is immutable when enhanced for loop is used i.e it ensures that the values in the array can not be modified, so it can be said as read only loop where you can't update the values as opposite to other loops where values can be modified.

We recommend using this form of the for statement instead of the general form whenever possible.(as per JAVA doc.)

Syntax:

```
for (T element:Collection obj/array)
{
    statement(s)
}
```

Lets take an example to demonstrate how enhanced for loop can be used to simpify the work. Suppose there is an array of names and we want to print all the names in that array. Let's see the difference with these two examples

Enhanced for loop simplifies the work as follows-

```
// Java program to illustrate enhanced for loop
public class enhancedforloop
{
    public static void main(String args[])
    {
        String array[] = {"Ron", "Harry", "Hermoine"};

        //enhanced for loop
        for (String x:array)
        {
            System.out.println(x);
        }

        /* for loop for same function
        for (int i = 0; i < array.length; i++)
        {
            System.out.println(array[i]);
        }
        */
    }
}
```

Output:

```
Ron
Harry
Hermoine
```

do while: do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of **Exit Control Loop.**

Syntax:

```
do
{
    statements..
}
while (condition);
```

do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time. After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.

When the condition becomes false, the loop terminates which marks the end of its life cycle.

It is important to note that the do-while loop will execute its statements atleast once before any condition is checked, and therefore is an example of exit control loop.

```
// Java program to illustrate do-while loop
class dowhileloopDemo
{
   public static void main(String args[])
   {
      int x = 21;
      do
      {
         // The line will be printed even
         // if the condition is false
         System.out.println("Value of x:" + x);
         x++;
      }
      while (x < 20);
   }
}
```

Output:

```
Value of x: 21
```

4) a) Explain Ternary operator. Write a java program to display largest of three numbers using ternary operator [5]

· ternary (three-way) operator that can replace certain types of if-then-else statements.

· This operator is the ?.

· The ? has this general form: expression1 ? expression2 : expression3

· Here, expression1 can be any expression that evaluates to a boolean value. If expression1 is true, then expression2 is evaluated; otherwise, expression3 is evaluated.

· The result of the ? operation is that of the expression evaluated.

· Both expression2 and expression3 are required to return the same type, which can’t be void.

Largest among three numbers:

```
public class Ternary {

        public static void main(String[] args) {
        int a=40,b=39,c=99,res;
         res=(a>b)?((a>c)?a:c):((b>c)?b:c);
        System.out.println(res);
        }

}
```

4b> i) (5 >> 2) + 3 ii) 5 | 4 + 2 >> 1 & 7

```
i)      4
ii)     5 | 4 + 2 >> 1 & 7
        5|6>>1&7
         5|3&7
          5|3
          7
```

5a> Explain if, if..else, nested if statements with syntax and suitable example.

An if statement consists of a Boolean expression followed by one or more statements.

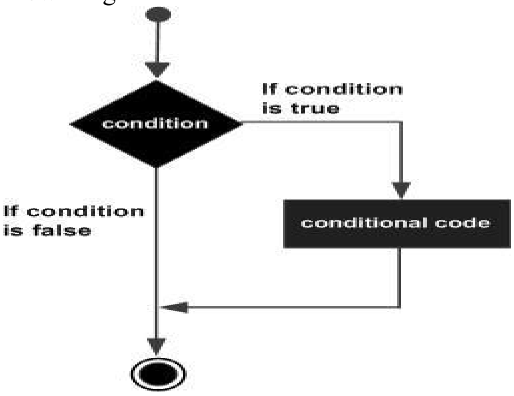
Syntax

Following is the syntax of an if statement −

```
if(Boolean_expression) {
  // Statements will execute if the Boolean expression is true
}
```

If the Boolean expression evaluates to true then the block of code inside the if statement will be executed. If not, the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Flow Diagram



Example

```
public class Test {

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

    if( x < 20 ) {
      System.out.print("This is if statement");
    }
  }
}
```

If..else

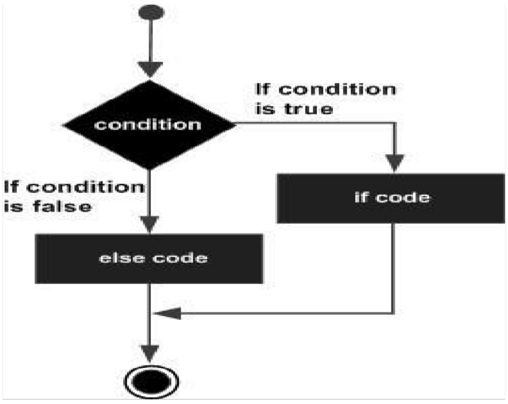
An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

Syntax

Following is the syntax of an if...else statement −

```
if(Boolean_expression) {
  // Executes when the Boolean expression is true
}else {
  // Executes when the Boolean expression is false
}
```

If the boolean expression evaluates to true, then the if block of code will be executed, otherwise else block of code will be executed.



Example

```
public class Test {

   public static void main(String args[]) {
      int x = 30;

      if( x < 20 ) {
         System.out.print("This is if statement");
      }else {
         System.out.print("This is else statement");
      }
   }
}
```

Nested-IF

It is always legal to nest if-else statements which means you can use one if or else if statement inside another if or else if statement.

Syntax

The syntax for a nested if...else is as follows −

```
if(Boolean_expression 1) {
   // Executes when the Boolean expression 1 is true
   if(Boolean_expression 2) {
      // Executes when the Boolean expression 2 is true
   }
}
```

You can nest else if...else in the similar way as we have nested *if* statement.

Example

```
public class Test {

   public static void main(String args[]) {
      int x = 30;
      int y = 10;

      if( x == 30 ) {
         if( y == 10 ) {
            System.out.print("X = 30 and Y = 10");
         }
      }
   }
}
```

5b>Explain use of for..each loop with suitable example. Write same example using simple for loop.

Syntax

Following is the syntax of enhanced for loop −

```
for(declaration : expression) {
   // Statements
}
```

- Declaration − The newly declared block variable, is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.
- Expression − This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

Example

```
public class Test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};
```

```
    for(int x : numbers ) {
      System.out.print( x );
      System.out.print(",");
    }
    System.out.print("\n");
    String [] names = {"James", "Larry", "Tom", "Lacy"};

    for( String name : names ) {
      System.out.print( name );
      System.out.print(",");
    }
  }
}
```

Using simple for loop

```
public class Test {

  public static void main(String args[]) {
    int [] numbers = {10, 20, 30, 40, 50};

    for(int x : numbers ) {
      System.out.print( x );
      System.out.print(",");
    }
    System.out.print("\n");
    String [] names = {"James", "Larry", "Tom", "Lacy"};

    for( int i=0;i<4;i++ ) {
      System.out.print( name[i] );
      System.out.print(",");
    }
  }
}
```

```
6 a> class Nested {
public static void main(String args[]) {
int i, j;
for(i=0; i<5; i++) {
for(j=i; j<5; j++)
System.out.print(".");
System.out.println();
}
}
}
```

6b>Explain use of short circuit property with example.

These are secondary versions of the Boolean AND and OR operators, and are known as short-circuit logical operators. As you can see from the preceding table, the OR operator results in true when A is true, no matter what B is. Similarly, the AND operator results in false when A is false, no matter what B is.

a=10 b=0

if(b&&a)   here since b is 0; result is going to be 0; no need to check a

if(a||b)   here since a is non zero; result is going to be 1; no need to check b

7a>Write program to display Prime numbers between 1 to 100.

```
class PrimeNumbers
{
   public static void main (String[] args)
   {
       int i =0;
       int num =0;
       //Empty String
       String  primeNumbers = "";

       for (i = 1; i <= 100; i++)
       {
          int counter=0;
          for(num =i; num>=1; num--)
        {
             if(i%num==0)
           {
            counter = counter + 1;
           }
        }
        if (counter ==2)
        {
           //Appended the Prime number to the String
           primeNumbers = primeNumbers + i + " ";
        }
       }
       System.out.println("Prime numbers from 1 to 100 are :");
       System.out.println(primeNumbers);
   }
}
```

7b> **Write a program to find factorial of a number 5.**

```
 class FactorialExample{
 public static void main(String args[]){
  int i,fact=1;
  int number=5;//It is the number to calculate factorial
  for(i=1;i<=number;i++){
      fact=fact*i;
  }
  System.out.println("Factorial of "+number+" is: "+fact);
 }
}
```

Output:

```
Factorial of 5 is: 120
```

8 Explain break, continue and return statement with Syntax and example.

Java Break Statement

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

The Java break is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:
jump-statement;
break;

Java Break Statement with Loop

Example:

```
//Java Program to demonstrate the use of break statement
//inside the for loop.
public class BreakExample {
public static void main(String[] args) {
    //using for loop
    for(int i=1;i<=10;i++){
        if(i==5){
            //breaking the loop
            break;
        }
        System.out.println(i);
    }
}
}
```

Output:

```
1
2
3
4
```

Java Break Statement with Inner Loop

It breaks inner loop only if you use break statement inside the inner loop.

Example:

```
//Java Program to illustrate the use of break statement
//inside an inner loop
public class BreakExample2 {
public static void main(String[] args) {
            //outer loop
            for(int i=1;i<=3;i++){
                    //inner loop
                    for(int j=1;j<=3;j++){
                        if(i==2&&j==2){
                            //using break statement inside the inner loop
                            break;
                        }
                        System.out.println(i+" "+j);
                    }
            }
}
}
```

Output:

```
1 1
```

```
1 2
1 3
2 1
3 1
3 2
3 3
```

Java Break Statement with Labeled For Loop

We can use break statement with a label. This feature is introduced since JDK 1.5. So, we can break any loop in Java now whether it is outer loop or inner.

Example:

```
//Java Program to illustrate the use of continue statement
//with label inside an inner loop to break outer loop
public class BreakExample3 {
public static void main(String[] args) {
            aa:
            for(int i=1;i<=3;i++){
                    bb:
                    for(int j=1;j<=3;j++){
                        if(i==2&&j==2){
                            //using break statement with label
                            break aa;
                        }
                        System.out.println(i+" "+j);
                    }
            }
}
}
```

Output:

```
1 1
1 2
1 3
2 1
```

Java Break Statement in while loop

Example:

```
//Java Program to demonstrate the use of break statement
//inside the while loop.
public class BreakWhileExample {
public static void main(String[] args) {
    //while loop
    int i=1;
    while(i<=10){
        if(i==5){
            //using break statement
            i++;
            break;//it will break the loop
        }
        System.out.println(i);
        i++;
    }
}
}
```

Output:

```
1
2
3
4
```

Java Continue Statement

The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.

The Java continue statement is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.

We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:

```
jump-statement;
continue;
```

Java Continue Statement Example

Example:

```
//Java Program to demonstrate the use of continue statement
//inside the for loop.
public class ContinueExample {
public static void main(String[] args) {
   //for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //using continue statement
         continue;//it will skip the rest statement
      }
      System.out.println(i);
   }
}
}
```

Output:

```
1
2
3
4
6
7
8
9
10
```

As you can see in the above output, 5 is not printed on the console. It is because the loop is continued when it reaches to 5.

Java Continue Statement with Inner Loop

It continues inner loop only if you use the continue statement inside the inner loop.

Example:

```
//Java Program to illustrate the use of continue statement
//inside an inner loop
public class ContinueExample2 {
public static void main(String[] args) {
        //outer loop
        for(int i=1;i<=3;i++){
             //inner loop
             for(int j=1;j<=3;j++){
                if(i==2&&j==2){
                    //using continue statement inside inner loop
                    continue;
                }
                System.out.println(i+" "+j);
             }
        }
}
}
```

Output:

```
1 1
1 2
1 3
2 1
2 3
3 1
3 2
3 3
```

Java Continue Statement with Labeled For Loop

We can use continute statement with a label. This feature is introduced since JDK 1.5. So, we can continue any loop in Java now whether it is outer loop or inner.

Example:

```
//Java Program to illustrate the use of continue statement
//with label inside an inner loop to continue outer loop
public class ContinueExample3 {
public static void main(String[] args) {
        aa:
        for(int i=1;i<=3;i++){
             bb:
             for(int j=1;j<=3;j++){
                if(i==2&&j==2){
                    //using continue statement with label
                    continue aa;
                }
                System.out.println(i+" "+j);
             }
```

```
            }
}
}
```

Output:

```
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

Java Continue Statement in while loop

Example:

```
//Java Program to demonstrate the use of continue statement
//inside the while loop.
public class ContinueWhileExample {
public static void main(String[] args) {
    //while loop
    int i=1;
    while(i<=10){
        if(i==5){
            //using continue statement
            i++;
            continue;//it will skip the rest statement
        }
        System.out.println(i);
        i++;
    }
}
}
```

Test it Now

Output:

```
1
2
3
4
6
7
8
9
10
```

Java return

Java return keyword is used to complete the execution of a method. The return followed by the appropriate value that is returned to the caller. This value depends on the method return type like int method always return an integer value.

Points to remember

It is used to exit from the method.

It is not allowed to use return keyword in void method.

The value passed with return keyword must match with return type of the method.

Examples of Java return Keyword

Example 1

Let's see a simple example to return integer value.

```
public class ReturnExample1 {

     int display()
    {
       return 10;
    }
    public static void main(String[] args) {
    ReturnExample1 e =new ReturnExample1();
    System.out.println(e.display());
}

}
```