


CMR INSTITUTE OF TECHNOLOGY		USN <input type="text"/>								
Internal Assessment Test - I										
Sub:	Object Oriented Programming with Java						Code:	20MCA22		
Date:	22/06/2021	Duration:	90 mins	Max Marks:	50	Sem:	I-A&B	Branch:	MCA	
Answer <b>ONE FULL QUESTION</b> from each part										
								Marks	OBE	
									CO	RBT
<b>Part - I</b>										
1 (a)	Briefly explain any five features of Java.						6	CO1	L2	
(b)	What is for-each loop? Write its syntax.						4	CO1	L1	
(OR)										
2	Write a program in Java for String handling which performs the following: i. Checks the capacity of StringBuffer objects. ii. Reverses the contents of a string given on console and converts the resultant string in upper case. iii. Reads a string from console and appends it to the resultant string of ii.						10	CO1	L3	
<b>Part – II</b>										
3 (a)	What is type casting? What is meant by automatic type promotion? Give an example.						5	CO1	L2	
(b)	What is 'this' keyword? Demonstrate 'this' with a suitable program.						5	CO2	L2	
(OR)										
4	What is constructor? Explain different types of constructor with example						10	CO1	L2	

<b>PART - III</b>						
5 (a)	Explain bitwise operator.			5	CO1	L1
(b)	What are various access specifiers in Java? List out the behavior of each of them.			5	CO2	L2
(OR)						
6 (a)	Discuss usage of final keyword in Java. Give suitable examples.			5	CO2	L2
(b)	Write a java program to print the following pattern: 1 123 1234 12345			5	CO1	L5
<b>Part – IV</b>						
7	What is super keyword? Explain use of super with example.			10	CO2	L2
(OR)						
8 (a)	Distinguish between method overriding and method overloading with suitable examples.			5	CO2	L3
(b)	What is Inner class? Write a program to demonstrate inner class			5	CO2	L1
<b>Part – V</b>						
9	What is inheritance? Explain the order of constructor execution in multilevel hierarchy of class.			10	CO2	L2
(OR)						
10	Create an abstract class called Employee. Include the members: Name, EmpID and an abstract method cal_sal(). Create two inherited classes SoftwareEng (with the members basic and DA) and HardwareEng (with members basic and TA). Implement runtime polymorphism (dynamic method dispatch) to display salary of different employees by creating array of references to superclass.			10	CO2	L5

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

**Internal Assessment Test 1 – June. 2021**

<b>Sub:</b>	OS with Unix							<b>Sub Code:</b>	20MCA22
<b>Date:</b>	22-06-2021	<b>Duration:</b>	90 min's	<b>Max Marks:</b>	50	<b>Sem</b>	II	<b>Branch:</b>	MCA

**Note : Answer FIVE FULL Questions, choosing ONE full question from each Module**

PART I		MARKS	OBE	
			CO	RBT
1a	<p><b>Simple :</b> Java is Easy to write and more readable and eye catching. Java has a concise, cohesive set of features that makes it easy to learn and use. Most of the concepts are drew from C++ thus making Java learning simpler.</p> <p><b>Secure :</b> Java program cannot harm other system thus making it secure. Java provides a secure means of creating Internet applications. Java provides secure way to access web applications.</p> <p><b>Portable :</b> Java programs can execute in any environment for which there is a Java run-time system.(JVM) Java programs can be run on any platform (Linux,Window,Mac) Java programs can be transferred over world wide web (e.g applets)</p> <p><b>Object-oriented :</b> Java programming is object-oriented programming language. Like C++ java provides most of the object oriented features. Java is pure OOP. Language. (while C++ is semi object oriented)</p> <p><b>Robust :</b> Java encourages error-free programming by being strictly typed and performing run-time checks.</p>	6	CO1	L2
1b	<p>For-each loop is used to access the elements of an array or list. We can access the elements directly instead of index. The type of the loop variable and array type should be the same. Syntax: for(type var:array) {     Body of the loop; }</p> <p>Example: <b>class</b> ForEachExample1 {     <b>public static void</b> main(String args[]){         <b>int</b> arr[]={12,13,14,44};         <b>for</b>(<b>int</b> i:arr){             System.out.println(i);         }     }</p>	4	CO1	L1
2.	<pre>import java.util.Scanner; public class stringclass { public static void main(String args[]) { StringBuffer sb=new StringBuffer("MCA"); System.out.println("length="+sb.length()); }</pre>	10	CO1	L3

<pre> System.out.println("total capacity="+sb.capacity()); String s = new String("Aslam"); s=s.toUpperCase(); System.out.println(s); String reverse =new StringBuffer(s).reverse().toString(); System.out.println("reversed string is"+reverse); Scanner user = new Scanner(System.in); System.out.println("enter any string"); String s3=user.next(); reverse=reverse.concat(s3); System.out.println(reverse); } } </pre>			
<p>3.a Type casting is needed when we want to store a value of one type into a variable of another type. There are two types of type casting .They are</p> <ol style="list-style-type: none"> <li>1. Implicit Casting</li> <li>2. Explicit Casting</li> </ol> <p>Implicit Casting:</p> <ul style="list-style-type: none"> <li>Automatic Type casting take place when, <ul style="list-style-type: none"> <li>the two types are compatible</li> <li>the target type is larger than the source type</li> </ul> </li> </ul> <p>Example :</p> <pre> class Test { public static void main(String[] args) { int i = 100;  //automatic type conversion long l = i;  //automatic type conversion float f = l; System.out.println("Int value "+i); System.out.println("Long value "+l); System.out.println("Float value "+f); } } </pre> <p>Narrowing or Explicit type conversion</p> <p>When we are assigning a larger type value to a variable of smaller type, then we need to perform explicit type casting.</p> <p><b>Example :</b></p> <pre> public class Test{ Public static void main(String[] k) { Double d=100.87; Long l=(long)d; int i=(int)l; System.out.println("Double value"+d); System.out.println("Long value"+l); System.out.println("Int value"+i); } } </pre>	5	CO1	L2
<p>3.b The 'this' keyword is used for two purposes</p> <ol style="list-style-type: none"> <li>1. It is used to point to the current active object.</li> <li>2. Whenever the formal parameters and data members of a class are similar, to differentiate the data members of a class from formal arguments the data members of a class are preceded with 'this'.</li> </ol> <p>This(): It is used for calling current class default constructor from current class parameterized</p>	5	CO2	L2

constructor.

This(...): It is used for calling current class parameterized constructor from other category constructors of the same class.

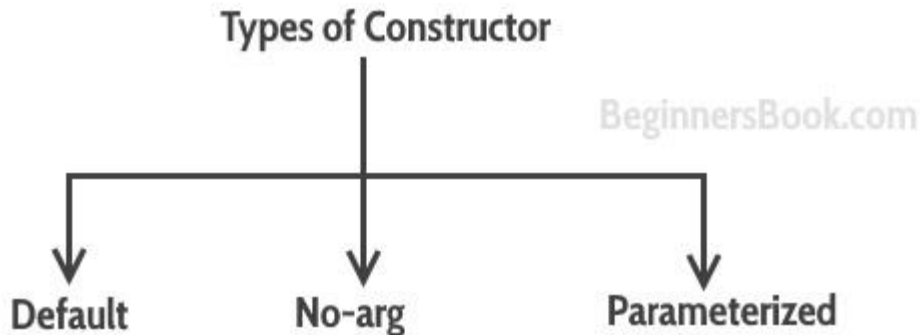
Ex: class Test

```
{
    int a,b;
    Test()
    {
        This(10);
        System.out.println("I am from default constructor");
        a=1;
        b=2;
        System.out.println("Value of a="+a);
        System.out.println("Value of b="+b);
    }
    Test(int x)
    {
        This(100,200);
        System.out.println("I am from parameterized constructor");
        a=b=x;
        System.out.println("Value of a="+a);
        System.out.println("Value of b="+b);
    }
    Test(int a,int b)
    {
        System.out.println("I am from double parameterized constructor")
        this.a=a+5;
        This.b=b+5;
        System.out.println("Value of instance variable a="+this.a);
        System.out.println("Value of instance variable b="+this.b);
        System.out.println("Value of a="+a);
        System.out.println("Value of b="+b);
    }
}
Class TestDemo3
{
    Public static void main(String k[])
    {
        Test t1=new Test();
    }
}
```

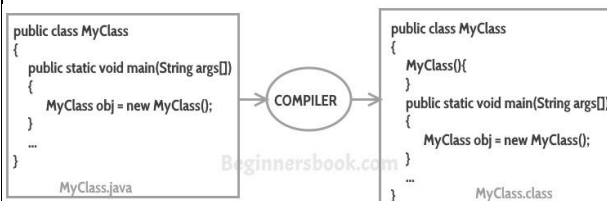
A constructor initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.

Typically, you will use a constructor to give initial values to the instance variables defined by the class, or to perform any other start-up procedures required to create a fully formed object.

All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero. However, once you define your own constructor, the default constructor is no longer used.



**Default constructor:** If you do not implement any constructor in your class, Java compiler inserts a default constructor into your code on your behalf. This constructor is known as default constructor. You would not find it in your source code (the java file) as it would be inserted into the code during compilation and exists in .class file. This process is shown in the diagram below:



#### no-arg constructor:

Constructor with no arguments is known as **no-arg constructor**. The signature is same as default constructor, however body can have any code unlike default constructor where the body of the constructor is empty.

```

class Demo
{
    public Demo()
    {
        System.out.println("This is a no argument constructor");
    }
    public static void main(String args[]) {
        new Demo();
    }
}
  
```

#### Parameterized constructor

Constructor with arguments (or you can say parameters) is known as Parameterized constructor.

```

public class Employee {
    int empId;
    String empName;
  
```

```
//parameterized constructor with two parameters
Employee(int id, String name){
    this.empId = id;
    this.empName = name;
}
void info(){
    System.out.println("Id: "+empId+" Name: "+empName);
}

public static void main(String args[]){
    Employee obj1 = new Employee(10245,"Chaitanya");
    Employee obj2 = new Employee(92232,"Negan");
    obj1.info();
    obj2.info();
}
}
```

### Copy Constructor

A copy constructor is used for copying the values of one object to another object.

```
class JavaExample{
    String web;
    JavaExample(String w){
        web = w;
    }
    JavaExample(JavaExample je){
        web = je.web;
    }
    void disp(){
        System.out.println("Website: "+web);
    }

    public static void main(String args[]){
        JavaExample obj1 = new JavaExample("BeginnersBook");

        /* Passing the object as an argument to the constructor
        * This will invoke the copy constructor
        */
        JavaExample obj2 = new JavaExample(obj1);
        obj1.disp();
        obj2.disp();
    }
}
```

5a Java defines several *bitwise operators that can be applied to the integer types, long, int, short, char, and byte. These operators act upon the individual bits of their operands.*

5

CO1

L1

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

### The Bitwise Logical Operators

— The bitwise logical operators are **&**, **|**, **^**, and **~**.

A	B	A   B	A & B	A ^ B	~A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

The **Bitwise NOT** Also called the *bitwise complement, the unary NOT operator, ~*, inverts all of the bits of its operand.

For example, the number 42, which has the following bit pattern: 00101010 becomes 11010101 after the NOT operator is applied.

The **Bitwise AND** The AND operator, **&**, produces a 1 bit if both operands are also 1. A zero is produced in all other cases. Here is an example:

```
00101010      42
& 00001111
-----
00001010      10
```

The **Bitwise OR** The OR operator, **|**, combines bits such that if either of the bits in the operands is a 1, then the resultant bit is a 1, as shown here:

```
00101010      42
| 00001111      15
-----
00101111      47
```

The **Bitwise XOR** The XOR operator, **^**, combines bits such that if exactly one operand is 1, then the result is 1. Otherwise, the result is zero.

```
00101010      42
^ 00001111      15
-----
00100101      37
```

The **left shift operator**, **<<**, shifts all of the bits in a value to the left a specified number of times.

It has this general form:  $value \ll num$

Here, *num specifies the number of positions to left-shift the value in value. That is, the << moves all of the bits in the specified value to the left by the number of bit positions specified by num.*

For each shift left, the high-order bit is shifted out (and lost), and a zero is brought in on the right.

The **right shift operator**, **>>**, shifts all of the bits in a value to the right a specified number of times.

Its general form is shown here:

$value \gg num$

Here, *num specifies the number of positions to right-shift the value in value. That is, the >> moves all of the bits in the specified value to the right the number of bit positions specified by num.*

The following code fragment shifts the value 32 to the right by two positions, resulting in a **being set to 8**:

```
int a = 32 (10000);
a = a >> 2; // a now contains 8
int a = 35;
a = a >> 2; // a still contains 8
```

5.b The access specifiers also determine which data members (methods or fields) of a class can be accessed by other data members of classes or packages etc.

5

CO2

L2

Access modifiers in Java allow us to set the scope or accessibility or visibility of a data member be it a field, constructor, class, or method.

#### Types Of Access Modifiers In Java

Java provides four types of access specifiers that we can use with classes and other entities.

- **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
- **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
- **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

#### Private Access Modifier

```
class A
```

```
{  
private int data=40;  
private void msg(){System.out.println("Hello java");}  
}
```

```
public class Simple{
```

```
public static void main(String args[]){  
A obj=new A();  
System.out.println(obj.data);//Compile Time Error  
obj.msg();//Compile Time Error  
}  
}
```

#### Default Access Modifier

```
//A.java
```



```
package pack;

class A
{
    void msg()
    {
        System.out.println("Hello");
    }
}
```

//B.java

```
package mypack;

import pack.*;

class B
{
    public static void main(String args[])
    {
        A obj = new A();//Compile Time Error
        obj.msg();//Compile Time Error
    }
}
```

Protected Access Modifier

//save by A.java

```
package pack;

public class A{

protected void msg(){System.out.println("Hello");}

}
```

//save by B.java

```
package mypack;

import pack.*;
```

	<pre> class B extends A{      public static void main(String args[]){          B obj = new B();          obj.msg();      }  }  Public Access Modifier  //save by A.java  package pack;  public class A{  public void msg(){System.out.println("Hello");}  }  //save by B.java  package mypack;  import pack.*;  class B{      public static void main(String args[]){          A obj = new A();          obj.msg();      }  } </pre>			
6a	<p>Final keyword is used to denote constants. It can be used with variables, methods, and classes.</p> <p>Once any entity (variable, method or class) is declared final, it can be assigned only once. That is,</p> <ul style="list-style-type: none"> <li>If you make any variable as final, you cannot change the value of final variable(It will be constant).</li> </ul>	5	CO2	L2

- A final method cannot be overridden. This means even though a sub class can call the final method of parent class without any issues but it cannot override it.
- If you make any class as final, you cannot extend it.

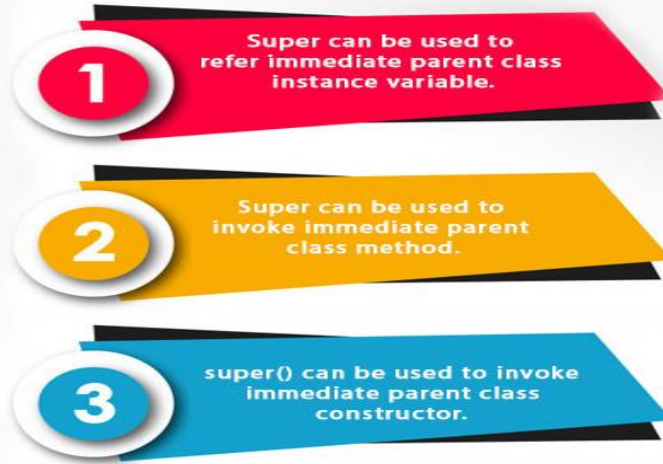
```
class FinalVariable
{
    public static void main(String[] args)
    {
        final int hours=24;
        System.out.println("Hours in 6 days = " + hours * 6);
    }
}
```

**b. Final Method:**

```
class X
{
    final void getMethod()
    {
        System.out.println("X method has been called");
    }
}
class Y extends X
{
    void getMethod() //cannot override
    {
        System.out.println("Y method has been called");
    }
}
class FinalMethod
{
    public static void main(String[] args)
    {
        Y obj = new Y();
        obj.getMethod();
    }
}
```

	<pre> }  <b>c. Final Class</b>  final class X  {      //properties and methods of class X  }  class Y extends X  {      //properties and methods of class Y  }  class FinalClass  {      public static void main(String args[]) {}  } </pre>			
6.b	<pre> import java.util.Scanner; public class Exercise16 {      public static void main(String[] args)      {          int i,j,n;         System.out.print("Input number of rows : ");         Scanner in = new Scanner(System.in);             n = in.nextInt();          for(i=1;i&lt;=n;i++)         {             for(j=1;j&lt;=i;j++)                 System.out.print(j);              System.out.println("");         }     } } </pre>	5	CO1	L5
7	<ul style="list-style-type: none"> <li>• The super keyword refers to superclass (parent) objects.</li> <li>• The most common use of the super keyword is to eliminate the confusion between super classes and subclasses that have methods with the same name.</li> </ul>	10	CO2	L2

## Usage of Super Keyword



1) super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1 {
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
```

```

super.eat();
bark();
}
}

class TestSuper2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}

```

3) super is used to invoke parent class constructor.  
The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

```

class Animal{
Animal(){System.out.println("animal is created");}
}

class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}

class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}

```

8a. There are many differences between method overloading and method overriding in java. A list of differences between method overloading and method overriding are given below:

No.	Method Overloading	Method Overriding
1)	Method overloading is used to <i>increase the readability</i> of the program.	Method overriding is used to <i>provide the specific implementation</i> of the method that is already provided by its super class.
2)	Method overloading is performed <i>within class</i> .	Method overriding occurs <i>in two classes</i> that have IS-A (inheritance) relationship.
3)	In case of method overloading, <i>parameter must be different</i> .	In case of method overriding, <i>parameter must be same</i> .
4)	Method overloading is the example of <i>compile time polymorphism</i> .	Method overriding is the example of <i>run time polymorphism</i> .
5)	In java, method overloading can't be performed by changing return type of the method only. <i>Return type can be same or different</i> in method overloading. But you must have to change the parameter.	<i>Return type must be same or covariant</i> in method overriding.

8b In Java, just like methods, variables of a class too can have another class as its member. Writing a class within another is allowed in Java. The class written within is called the **nested class**, and the class that holds the inner class is called the **outer class**.

5 CO2 L3

5 CO2 L1

	<p><b>Syntax</b>  Following is the syntax to write a nested class. Here, the class <b>Outer_Demo</b> is the outer class and the class <b>Inner_Demo</b> is the nested class.</p> <pre>class Outer_Demo { class Nested_Demo { } }</pre> <p>Nested classes are divided into two types –</p> <ul style="list-style-type: none"> <li>• <b>Non-static nested classes</b> – These are the non-static members of a class.</li> <li>• <b>Static nested classes</b> – These are the static members of a class.</li> </ul> <p>Inner classes are a security mechanism in Java. We know a class cannot be associated with the access modifier <b>private</b>, but if we have the class as a member of other class, then the inner class can be made private. And this is also used to access the private members of a class.</p> <p>Inner classes are of three types depending on how and where you define them. They are –</p> <ul style="list-style-type: none"> <li>• Inner Class</li> <li>• Method-local Inner Class</li> <li>• Anonymous Inner Class</li> </ul> <p><b>Inner Class</b>  Creating an inner class is quite simple. You just need to write a class within a class. Unlike a class, an inner class can be private and once you declare an inner class private, it cannot be accessed from an object outside the class.</p> <p>Following is the program to create an inner class and access it. In the given example, we make the inner class private and access the class through a method.</p> <p><b>Example</b></p> <pre>class Outer_Demo { int num; // inner class private class Inner_Demo { public void print() { System.out.println("This is an inner class"); } } // Accessing the inner class from the method within void display_Inner() { Inner_Demo inner = new Inner_Demo(); inner.print(); } } public class My_class { public static void main(String args[]) { // Instantiating the outer class Outer_Demo outer = new Outer_Demo(); // Accessing the display_Inner() method. outer.display_Inner(); } }</pre>			
9	<p>The process by which one class acquires the properties(data members) and functionalities(methods) of another class is called <b>inheritance</b>. The aim of inheritance is to provide the reusability of code so that a class has to write only the unique features and rest of the common properties and functionalities can be extended from the another class.</p> <p><b>Child Class:</b>  The class that extends the features of another class is known as child class, sub class or derived class.</p> <p><b>Parent Class:</b>  The class whose properties and functionalities are used(inherited) by another class is known as parent class, super class or Base class.</p>	10	CO2	L2

	<p>Order of execution of constructors in inheritance relationship is from base /parent class to derived / child class.</p> <p>We know that when we create an object of a class then the constructors get called automatically.</p> <p>In inheritance relationship, when we create an object of a child class, then first base class constructor and then derived class constructor get called implicitly.</p> <p>NOTE: We cannot call a class constructor explicitly using an object or say manually.</p> <p>If we create object of bottom most derived class i.e. of Testing class in main() program, then constructors of Design class, Coding class and then Testing class will be called.</p> <pre> class Design {     Design(){         System.out.println("Design()...");     } } class Coding extends Design {     Coding(){         System.out.println("coding()...");     } } class Testing extends Coding {     Testing()     {         System.out.println("Testing()...");     } }  public class TestConstructorCallOrder {     public static void main(String[] args) {         //Create object of bottom most class object         System.out.println("Constructor call order...");         new Testing();     } } </pre> <p><b>OUTPUT:</b>  Constructor call order...  Design()...  coding()...  Testing()...</p>			
10	<pre> abstract class Shape {     final double PI= 3.1416;     abstract double area(); } </pre>	10	CO2	L5



```
class Triangle extends Shape
{
int b, h;
Triangle(int x, int y)
{
b=x;
h=y;
}

double area()
{
System.out.print("\nArea of Triangle is:");
return 0.5*b*h;
}
}

class Circle extends Shape
{
int r;
Circle(int rad)
{
r=rad;
}

double area()
{
System.out.print("\nArea of Circle is:");
return PI*r*r;
}
}

class Rectangle extends Shape
{
int a, b;
Rectangle(int x, int y)
{
a=x;
b=y;
}
double area()
{
System.out.print("\nArea of Rectangle is:");
return a*b;
}
}

public class AbstractDemo
{
public static void main(String args[])
{
Shape r[]={ new Triangle(3,9), new Rectangle(9,6),new Circle(4)};

for(int i=0;i<3;i++) System.out.println(r[i].area());
```

	}			
--	---	--	--	--