L.J. Institute of Engineering & Technology *Semester: V (2016)*

Subject Name: Object Oriented Programming using JAVA - QUESTION BANK SOLUTION
Subject Code: 2150704 Unit- 4

Faculties: MS. HEMALI MOJIDRA (SHAH)

UNIT-4 Inheritance and Interfaces:

TOPIC:1 (Basics Of Inheritance)

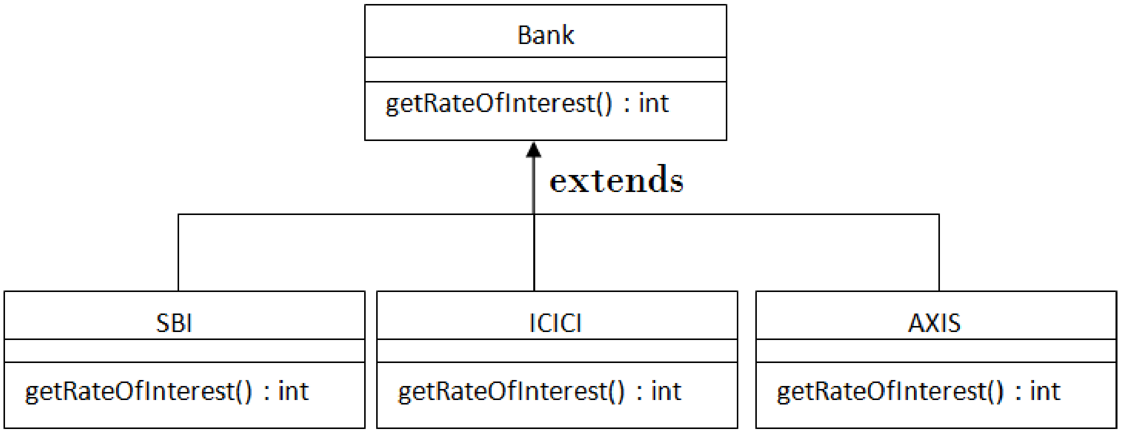
Use of Inheritance, Inheriting Data members and Methods, constructor in inheritance, Multilevel Inheritance – method overriding

1 Explain the followings: . **(Nov-11) [LJIET]** **3/4**

(i) Dynamic Method Dispatch with example. **(June-14) [LJIET]**

Dynamic method dispatch is a mechanism by which a call to an **overridden method** is resolved at **runtime**. This is how java implements **runtime polymorphism**. When an overridden method is called by a reference, java determines which version of that method to execute based **on the type of object it refer to**. In simple words the type of object which it referred determines which version of overridden method will be called

Dynamic method dispatch process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.



Bank
getRateOfInterest() : int
extends
SBI
getRateOfInterest() : int
ICICI
getRateOfInterest() : int
AXIS
getRateOfInterest() : int

Example:

```
class Bank{
int getRateOfInterest(){return 0;}
}

class SBI extends Bank{
int getRateOfInterest(){return 8;}  //Override it
}

class ICICI extends Bank{
int getRateOfInterest(){return 7;}  //Override it
}
class AXIS extends Bank{
```

```
int getRateOfInterest(){return 9;}  //Override it
}

class Test3{
public static void main(String args[]){
// Super Class Variable can refer Sub class instance. Bank is a superclass
Bank b1=new SBI();
Bank b2=new ICICI();
Bank b3=new AXIS();
System.out.println("SBI Rate of Interest: "+b1.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+b2.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+b3.getRateOfInterest());
}
```

Output:

```
Output:
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

OR

Example: [You can take Any Two Sub Classes of Shape only in exam]

```
abstract class Shape
{
        double dim1;
        double dim2;

        Shape(double d1, double d2)
        {
                dim1=d1;
                dim2=d2;
        }
        abstract double area();
}

class Rectangle extends Shape
{
        Rectangle(double d1, double d2)
        {
                super(d1,d2);
        }
        double area()
        {
                return(dim1*dim2);
        }
}

class Triangle extends Shape
{
        Triangle(double d1, double d2)
        {
                super(d1,d2);
        }
        double area()
        {
                return(dim1*dim2)/2;
        }
}

class Circle extends Shape
{
        Circle(double d1)
        {
                super(d1,d1);
        }
        double area()
        {
                return(Math.PI*dim1*dim1);
        }
}

class ShapeMain
{
        public static void main(String[] args)
        {
                Shape s1;
                Rectangle r1=new Rectangle(10.5,20.5);
                Triangle t1=new Triangle(15.5,25.5);
                Circle c1=new Circle(30);

                s1=r1;
                System.out.println("The Area of Rectangle is: "+s1.area());

                s1=t1;
                System.out.println("The Area of Triangle is: "+s1.area());

                s1=c1;
                System.out.println("The Area of Circle is: "+s1.area());
        }
}
```

Output:-

```
        The Area of Rectangle is: 215.25
        The Area of Triangle is: 197.625
        The Area of Circle is: 2826.0
```

1	Differentiate Method Overloading and Method Overriding with example. **(June-12, Dec-13, Dec-14) [LJIET] OR** Explain method overriding and method overloading with the help of examples. . **(June-11) [LJIET]**	**3/4/7**

2 Explain inheritance with its types and example. **(May-15) [LJIET]** 7

OR

What is inheritance in java? Explain different types of inheritance with proper example partial code. **(May-16(old,new)) [LJIET]**

ANS:

Inheritance : can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

Why use inheritance in java

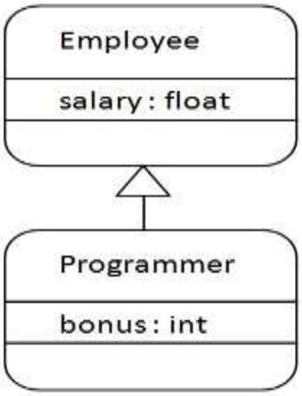
- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.
- increase reusability
- remove redundant code.

extends Keyword :
extends is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.

class Super{ ...
}

class Sub **extends Super**{ ..
}

Understanding the simple Example of inheritance:



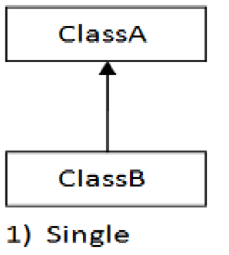
```
class Employee
{
       float salary;
       String eName;
       int eCode;

       void display()
       {
              System.out.println("En-
              "+eName);
       }
}
class Programmer extends Employee
{
       int bonus;
```

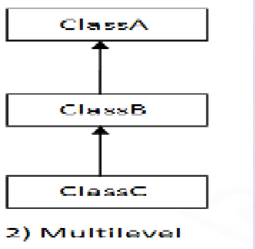
	}

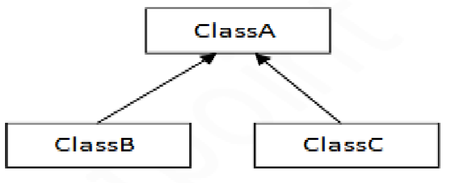
As displayed in the above figure, **Programmer is the subclass and Employee** is the superclass. Relationship between two classes is **Programmer IS-A Employee**. It means that Programmer is a type of Employee.

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.


1) Single
	class A{
	}
	class **B extends A**
	{
	}

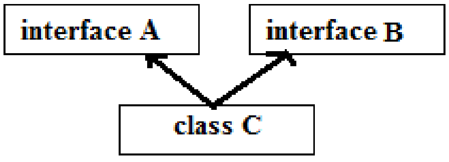

2) Multilevel
	class A
	{
	}
	class **B extends A**
	{
	}
	class **C extends B**
	{
	}

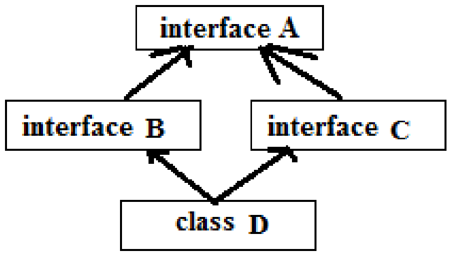

3) Hierarchical
	class A
	{}
	class B **extends A**
	{}
	class C **extends A**
	{}

In java programming, multiple and hybrid inheritance is supported through interface only. When a class

extends multiple classes i.e. known as multiple inheritance. This is not possible in java using classes. Only possible using Interface in java.

For Example:


4) Multiple Inheritance
	interface A
	{}
	
	interface B
	{}
	class C implements A,B
	{}



5) Hybrid Inheritance

```
interface A
{}

interface B extends A
{}

interface C extends A
{}

class D implements B,C
{}
```

3 Define polymorphism with its need. Define and explain static and dynamic binding using program. . **(Dec-10,Jan 13) [LJIET]** **7**

Ans :

Polymorphism: When **one task is performed by different ways** i.e. known as polymorphism. For example: to convense the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

- **Polymorphism** – Single thing is available in many forms (Poly –many, morphism – form)
  - **Types Of Polymorphism**

<table>
<tr><th>- Compile Time Polymorphism</th><th>- Runtime Polymorphism</th></tr>
<tr><td>Over loading</td><td>Over Ridding</td></tr>
<tr><td>Static binding</td><td>Dynamic Binding</td></tr>
<tr><td>Early Binding</td><td>Late Binding</td></tr>
<tr><td>Same thing with differ parameter.</td><td>Same method signature in parent and child class – Only possible in inheritance</td></tr>
<tr><td>- Types Of Over Loading
<table>
<tr><td>1. Method/Function</td><td>2. Constructor</td></tr>
<tr><td>class A
{
void msg(){}
void msg(int i)
{}
}</td><td>class A()
{
A(){}
A(int i){}
}</td></tr>
<tr><td>same method with different parameter</td><td>constructor with different parameter</td></tr>
</table></td><td>- No Types
<table>
<tr><td>class A
{
void msg(){}
void msg(int i){}
}</td></tr>
<tr><td>class B extends A
{
void msg(){}
void msg(int i){}
}</td></tr>
<tr><td>same method with same parameter in s</td></tr>
</table></td></tr>
</table>

Same as Ans-1 – Overloading And Overriding

No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the

		method that is already provided by its super class.
2)	Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5)	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

6)	class OverloadingExample{
	int add(**int a,int b**)
	{return a+b;}
	int add(**int a,int b,int c**)
	{return a+b+c;}
	}
		class Animal{
		void say()
		{System.out.println("don't know ");
		}
		}
		class Dog **extends Animal**{
		void say()
		{
		System.out.println("Bhow");
		}
		}

TOPIC:2 (super And final)

Handle multilevel constructors – super keyword, Stop Inheritance – Final keywords,

1 (i) Explain keywords - super and throws . **(Dec-15) [LJIET]** 3/4

Ans:

i) super:

The **super** keyword in java is a reference variable that is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

Usage of java super Keyword

to refer immediate parent class instance variable.	**to invoke parent class method**	**invoke parent class constructor.**
Syntax: super.variable	**Syntax:** super.method(parameter);	**Syntax:** super (parameter list); **- super(parlist) must be in first line of constructor**

Example:

```
class A {
    int i;
    A() {
        System.out.println("A()");
    }

    A(int i) {
        this.i = i;
    }
    void display() {
        System.out.println("A-Display()");
    }
}

class B extends A {
    int i;
    B() {
        //Default super() is first line of all constructor
        System.out.println("B()");
    }

    B(int i) {
        super(i);
        this.i = i + 5;
    }

    void display() {
        System.out.println("B-Display()");
        //Call Super class display
        super.display();
        System.out.println("This-i - " + this.i);
        System.out.println("Super-i - " + super.i);
    }
}

public class UseOfSuper {

    public static void main(String args[]) {
        System.out.println("b1- With No arg constructor");
        B b1 = new B();
        b1.display();
        System.out.println("b2 -With argument-int constructor");
        B b2 = new B(5);
        b2.display();
    }
}
Output:
b1- With No arg constructor
A()
B()
B-Display()
A-Display()
This-i - 0
Super-i - 0
b2 -With argument-int constructor
B-Display()
A-Display()
This-i - 10
Super-i - 5
```

ii)trows in Unit-6 – Exception Handling

2 Explain following with example:
iii) super iv) final . **(May-16) [LJIET]** **3/4**

Ans: i)super : same as Ans-1 i)super

ii)final:

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

final variable	**final method**	**final class**

If you make any variable as final, you **cannot change** the value of final variable(It will be constant).
- > Use to stop value change
 	
Example:
class A
{
    final int i =10;
    void change()
    {
      i=15//Error
    }
}
	If you make any method as final, you **cannot override** it.
	-> Use To Stop Overriding
	
	Example:
	class A
	{
	    final void m1()
	    {}
	}
	class B **extends A**
	{ //try to override it
	    void m1() // Error
	    {}
	
	}
		If you make any class as final, **you cannot extend** it.
		-> Use to stop Inheritance.
		
		Example:
		final class A
		{
		}
		class B extends A // Error
		{
		}

1 Explain the followings: (i) this, super, final. **(Nov-11,Jan-13, Dec-13,June-14) [LJIET]** **7**

OR Explain following key words: this, super, instance of **(May-15) [LJIET]**

Ans:

1)this:

this is the keyword in java. It is use to refer current object of class

Use of this keyword:

to refer current instance variable.	**to invoke current class method (default)**	**invoke current class constructor.**

Syntax:
this.variable
 	
-Use : to differentiate class instance variable and local variable when both having same name
Example:
class A
{
int a;
A(**int a**)
{
this.a = a;
//class instance=local variable
}
}
	Syntax:
	this.method(parameter);
		Syntax:
		this (parameter list);
		
		- this(parlist) must be in first line of constructor

Example:

```
class A {
    int i;
    A() {
        this(-1);//To call currenrt constructor
        System.out.println("A()");
    }
    A(int i) {
        this.i = i;
        System.out.println("A(int)");
    }
    void display() {
        //this to call current class method
        this.useMethod();//Same as  useMethod();//
        System.out.println("A-Display()- i=" + i);
    }
    void useMethod()
    {
        System.out.println("useMethod()");
    }
}
class UseOfThis {
    public static void main(String args[]) {
        System.out.println("a1- With No arg constructor");
        A a1 = new A();
        a1.display();
        System.out.println("a2 -With argument-int constructor");
        A a2 = new A(5);
        a2.display();
    }
}
```

Output:

```
a1- With No arg constructor
A(int)
A()
useMethod()
A-Display()- i=-1
a2 -With argument-int constructor
A(int)
useMethod()
A-Display()- i=5
```

2)super: Same as ans-1 i)super
3)insatnceof :

instanceof Operator:

USE: to test whether the object is an instance of the specified type (class or subclass or interface). To check type of object

The instanceof in java is also known as type ***comparison operator*** because it compares the instance with type.

It returns either **true or false**. I

if we apply the instanceof operator with any variable that has **null** value, it returns **false**.

Syntax : object instanceof **ClassName**

Example:

```
class A
{}
class B extends A
{}
class D extends B
{}
class C extends A
{}
```

```
public class InstanceOfDemo {
    public static void main(String[] s)
    {
        A a = new A();
        B b = new B();
        C c = new C();
        D d = new D();
        System.out.println("a - A = "+(a instanceof A));
        System.out.println("a - B = "+(a instanceof B));
        System.out.println("a - C = "+(a instanceof C));
        System.out.println("a - D = "+(a instanceof D));
        System.out.println("b - A = "+(b instanceof A));
        System.out.println("b - B = "+(b instanceof B));
        System.out.println("b - D = "+(b instanceof D));
        System.out.println("c - C = "+(c instanceof C));
        System.out.println("d - A = "+(d instanceof A));
        System.out.println("d - B = "+(d instanceof B));
        System.out.println("d - D = "+(d instanceof D));
    }
}
```

Output:

```
a - D = false
b - A = true
b - B = true
b - D = false
c - C = true
d - A = true
d - B = true
d - D = true
```

2 What are final class, final function and final variable in java? Explain with example. **(May-15) [LJIET]** 7

Ans: Same As topic-2 Answer-2

3 Explain Cosmic superclass and its methods. . **(Nov-11) [LJIET]** 7

The **Object** class is the ultimate ancestor—every class in Java extends Object. However, you never have to write “class Employee extends Object”.

In Java, every class that is defined without an explicit extends clause automatically extends the class Object. That is, the class Object is the direct or indirect superclass of every class in Java.

The **Object class** is the parent class of all the classes in java bydefault. In other words, it is the topmost class of java.

The Object class is **beneficial if you want to refer any object whose type you don't know**. Notice that **parent/super class reference variable can refer the child class object**, know as upcasting.

Method	Description
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.

public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified miliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

Use Of toString() Example:

```
class A
{
      int a;
      A(int a)
      {
            this.a = a;
      }
      public String toString()
      {
            return “Class-A Object – a = ” + a;
      }

}
class ToStringDemo
{
      public static void main(String args[]) {
            A a1 = new A(10);
            System.out.println(a1); // Automatic call toString()

      }
}
```

Output:

```
Class-A Object – a =10 // Object String Value – no need display
```

4 Explain final and super by giving examples. . **(June-11, Dec-13) [LJIET]** **7**

Ans:

1)final: same as topic-2 Ans-2 ii)final
2)super : same as topic-2 Ans-1 i)super

5 Explain use of final, static and super keyword by giving examples. **(May-16) [LJIET]** 7

Ans:
1)final: same as topic-2 Ans-2 ii)final
2)static:

The **static keyword** in java is used for memory management mainly. We can apply java static keyword **with variables, methods, blocks and nested class**. The static keyword belongs to the class than instance of the class.

The static can be:

static variable	**static method**	**static block**	**static Inner/Nested class**

Syntax:
class A
{
static int a;
}
 	
Use : To declare **global variable** in java.
 	
- Java static property is shared to all objects.
	Syntax:
	class A
	{
	static void ma()
	{}
	}
		Syntax:
		class A
		{
		static {}
		}
		
		- Is used to **initialize the static** data member.
		- It is executed at **the time of class loading.**
			class A
			{
			static class B{}
			}
			
			- static inner class only can access static properties of outer class

Rules Of Static :

- A static method belongs to the class rather than object of a class.
-A static method can be invoked **without the need for creating an instance of a class**.
- static method can access **static data** member and can change the value of it.
- The static method, block,inner class **can not use non static** data member or **call non-static method directly[Without creating local object]**.
-**this and super** cannot be used in static context-method/block/class.
- static data member and static method can call using classname without creating object of class.

Syntax :
ClassName.variable
ClassName.staticMethodCall(parameter);

Example:

```
class ObjectCount{
    static int count;
    int i;
    ObjectCount(int i)
    {
        count++;
        this.i = i;
    }
    static{
        System.out.println("I am static block - read count from file
-it is 5");
        count = 5;
    }
    void display() {
        System.out.println("count - "+ count + " i - " + i);
    }
    static void showCount()
    {
        System.out.println("showCount() - count - "+ count);
    }
    static class Inner
    {
        void innerDisplay()
        {   //can not use i -b/c i is non static data meber of outer
class
            System.out.println("innerDisplay() - count - "+ count);
        }
    }
}
class UseOfStatic {

    public static void main(String args[]) {
        System.out.println("O1");
        ObjectCount o1 = new ObjectCount(1);
        o1.display();
        System.out.println("O=2");
        ObjectCount o2 = new ObjectCount(2);
        o2.display();
        System.out.println("O3");
        ObjectCount o3 = new ObjectCount(3);
        o3.display();
        System.out.println("Using Class Name" + ObjectCount.count);
        ObjectCount.showCount();
        System.out.println("Using Inner Class");
        ObjectCount.Inner io = new ObjectCount.Inner();
        io.innerDisplay();
    }
}
```

Output:

```
O1
I am static block - read count from file -it is 5
count - 6 i - 1
O=2
count - 7 i - 2
O3
count - 8 i - 3
Using Class Name8
showCount() - count - 8
Using Inner Class
innerDisplay() - count - 8
```

3)super : same as topic-2 Ans-1 i)super

TOPIC:3 (Interface)

Creation and Implementation of an interface, Interface reference, instanceof operator, Interface inheritance, Dynamic method dispatch ,Understanding of Java Object Class, Comparison between Abstract Class and interface, Understanding of System.out.println – statement.

1 (i) Explain instanceof operator. **(Dec-15) [LJIET]** 3/4

Ans: Same as Ans 1) – iii)instanceof

1 Differentiate between abstract class and interface specifying matrices of differences. Write a program to define abstract class, with two methods addition() and subtraction(). addition() is abstract method. Implement the abstract method and call that method using a program(s). **(Dec-10) [LJIET]** 7

OR

Differentiate between abstract class and Interface. **(Dec-14) [LJIET]**

OR

Explain : Abstract Class and Interface with example. Compare Both. **(Dec-15,May-16) [LJIET]**

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can have static methods, main method and constructor**.	Interface **can't have static methods, main method or constructor**.
5) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
6) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

7) **Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Example:

```
abstract class Operation
{
        int x,y;
        Operation(int i,int j)
        {
                x=i;
                y=j;
        }
        abstract void addition();
        void subtraction()
        {
                int sub=x-y;
                System.out.println("The Subtraction is: "+sub);
        }
}

deliver class OperationSub extends Operation
{
	OperationSub(int i,int j)
	{
		super(i,j);
	}

	void addition()
	{
		int add=x+y;
		System.out.println("The Addition is: "+add);
	}
}

class OperationMain
{
	public static void main(String[] args)
	{
		OperationSub obj=new OperationSub(50,30);
		obj.addition();
		obj.subtraction();
	}
}
```

Output:-

```
	The Addition is: 80
	The Subtraction is: 20
```

2 How interface are useful in java? Explain with example. **(May-15) [LJIET] OR** 7

(i) Explain with the help of example(s), use of interface. **(Dec-15,May-16)(3Marks) [LJIET]**

Intrface: An **interface in java** is a blueprint of a class. It is fully abstract class. It has static constants(final) and abstract methods only.

Interface **fields are public, static and final** by default, and methods are **public and abstract**.

The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.

Java Interface also **represents IS-A relationship**.

It cannot be instantiated just like abstract class.(OR It is not possible to create object of interface)

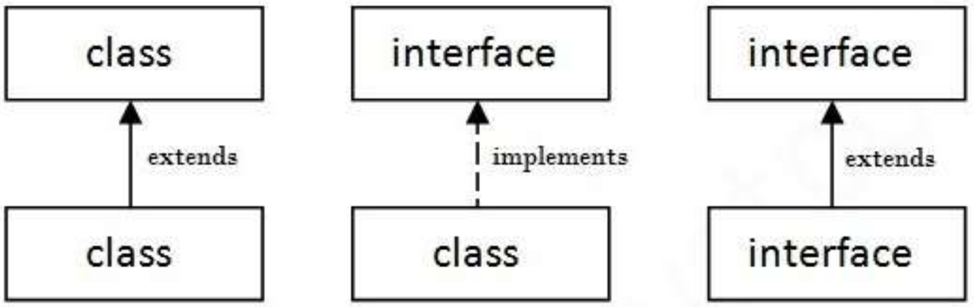
Use Of Java interface:

- It is used to achieve fully abstraction.
- It is used to give basic structure to the child classes.
- By interface, we can support the functionality of **multiple inheritance**.

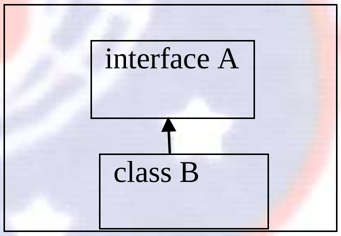
- It can be used to achieve loose coupling. (Overriding and dynamic method dispatch and runtime polymorphism)

Note : The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

Understanding relationship between classes and interfaces



Example:

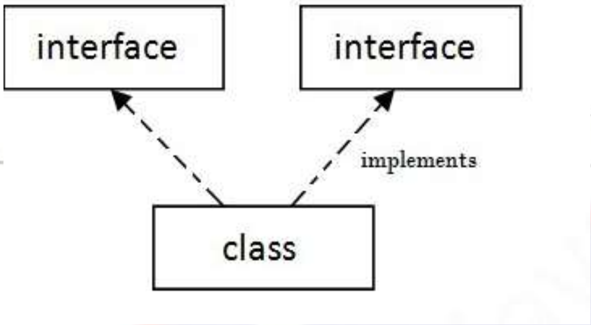


```
interface A {
        int i=10;
        void m1();
        void m1(int a);
        void m2();
}
class B implements A
{
        public void m1() {
           System.out.println("m1 called");
        }
        public void m1(int a) {
           System.out.println("m1(int) called");
        }
        public void m2() {
            System.out.println("m2 called");
        }
}
class UseInerfaceDemo
{
        public static void main(String[] args)
        {
           A a = new B();
           a.m1();
           a.m1(6);
           a.m2();
        }
}
```

Output:

```
m1 called
m1(int) called
m2 called
```

Multiple inheritance in Java by interface



```
interface A1 {
            int i=10;
            void m1();
            void m1(int a);
            void m2();
}
interface A2 {
            void m3();
}
class B implements A1,A2
{
            public void m1() {
               System.out.println("m1 called");
            }
            public void m1(int a) {
               System.out.println("m1(int) called");
            }
            public void m2() {
                System.out.println("m2 called");
            }

            public void m3() {
               System.out.println("m3 called");
            }
}
class UseInerfaceDemo
{
            public static void main(String[] args)
            {
               B b = new B();
               b.m1();
               b.m1(6);
               b.m2();
               b.m3();
            }

}
```

3	Explain interface in JAVA. How do interfaces support polymorphism? **(June-12) [LJIET]** **OR**	7

(ii) Explain interface and its usage. . . **(May-13) [LJIET]**

Ans: interface:

4 Explain super, instanceof, and volatile. . **(May-13) [LJIET]** 7

Ans:

1) super : same as Ans-1 i)super

2) instanceof :Same as Ans 1) – iii)instanceof

3) volatile:[not in syllabus)

TOPIC:4 (Programs)

1 Declare a class called book having author_name as private data member. Extend book class to have two sub classes called book_publication&paper_publication. Each of these classes have private member called title. Write a complete program to show usage of dynamic method dispatch (dynamic polymorphism) to display book or paper publications of given author. Use command line arguments for inputting data. . **(May-13,Dec-15) [LJIET]** 7

Note : you can write it in Overriding OR dynamic method dispatch OR dynamic polymorphism

```
class Book {
    private String authorName;
    public void setName(String authorName) {
        this.authorName = authorName;
    }
    public String getName() {
        return this.authorName;
    }
    public void setTitle(String title) {
        //Will override by Sub classes
    }
    public void display() {
        //Will override by Sub classes
    }
}

class BookPublication extends Book {

    private String title;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public void display() {
        System.out.println("The book's author name is : " + getName() + " and the book title is : " +
this.title);
    }
}

class PaperPublication extends Book {
    private String title;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public void display() {
        System.out.println("The paper's author name is : " + getName() + " and the paper title is : " +
this.title);
    }
}

class BookPaperPublicationDemo {

    public static void main(String a []) {
        Book b = null;
        BookPublication bp = new BookPublication();
        PaperPublication pp = new PaperPublication();
        b = bp;
        b.setName(a[0]);
        b.setTitle(a[1]);
        b.display();
        b = pp;
        b.setName(a[2]);
        b.setTitle(a[3]);
        b.display();
    }
}
```

Run::

```
Command Line Run : >java  BookPaperPublicationDemo  HemaliShah Java HemaliMojidra AdJava
```

Output:

```
The book's author name is : HemaliShah and the book title is : Java
The paper's author name is : HemaliMojidra and the paper title is : AdJava
```

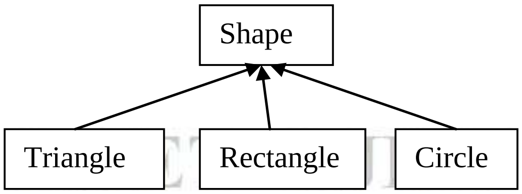
2 The abstract Vegetable class has three subclasses named Potato, Brinjal and Tomato. Write an application that demonstrates how to establish this class hierarchy. Declare one instance variable of type String that indicates the color of a vegetable. Create and display instances of these objects. Override the toString() method of Object to return a string with the name of the vegetable and its color. . **(Nov-11, May-16) [LJIET]** 7

```
abstract class Vegetable
{
        String vegColor;
}

class Potato extends Vegetable
{
        public String toString() // Method Of Object – inbuilt class
        {
                vegColor="Yellow";
                return vegColor;
        }
}

class Brinjal extends Vegetable
{
        public String toString()
        {
                vegColor="Violet";
                return vegColor;
        }
}

class Tomato extends Vegetable
{
        public String toString()
        {
                vegColor="Red";
                return vegColor;
        }
}
class VegetableMain
{

        public static void main(String[] args)
        {

                Vegetable p=new Potato();
                Vegetable b=new Brinjal();
                Vegetable t=new Tomato();

                System.out.println("The Color of the Potato is: "+p);
                 // Object directly call toString() when we use in println()
                System.out.println("The Color of the Brinjal is: "+b);
                System.out.println("The Color of the Tomato is: "+t);

        }

}
```

Output:-

```
        The Color of the Potato is: Yellow
        The Color of the Brinjal is: Violet
        The Color of the Tomato is: Red
```

3	Describe abstract class called Shape which has three sub classes say Triangle, Rectangle, Circle. Define one method area() in the abstract class and override this area() in these three subclasses to calculate for specific object i.e. area() of Triangle subclass should calculate area of triangle etc. Same for Rectangle and Circle. **(June-12,May-16) [LJIET]**	7



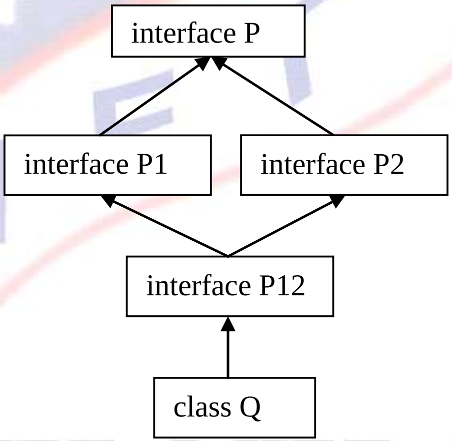
```
abstract class Shape
{
        double dim1;
        double dim2;

        Shape(double d1, double d2)
        {
                dim1=d1;
                dim2=d2;
        }
        abstract double area();
}

class Rectangle extends Shape
{
        Rectangle(double d1, double d2)
        {
                super(d1,d2);
        }
        double area()
        {
                return(dim1*dim2);
        }
}

class Triangle extends Shape
{
        Triangle(double d1, double d2)
        {
                super(d1,d2);
        }
        double area()
        {
                return(dim1*dim2)/2;
        }
}

class Circle extends Shape
{
        Circle(double d1)
        {
                super(d1,d1);
        }
        double area()
        {
                return(Math.PI*dim1*dim1);
        }
}

class ShapeMain
{
        public static void main(String[] args)
        {
                Shape s1;
                Rectangle r1=new Rectangle(10.5,20.5);
                Triangle t1=new Triangle(15.5,25.5);
                Circle c1=new Circle(30);

                s1=r1;
                System.out.println("The Area of Rectangle is: "+s1.area());

                s1=t1;
                System.out.println("The Area of Triangle is: "+s1.area());

                s1=c1;
                System.out.println("The Area of Circle is: "+s1.area());
        }
}
```

Output:-

```
        The Area of Rectangle is: 215.25
        The Area of Triangle is: 197.625
        The Area of Circle is: 2826.0
```

4 Write a program that illustrates interface inheritance. Interface P is extended by P1 And P2. Interface P12 inherits from both P1 and P2.Each interface declares one constant and one method. Class Q implements P12.Instantiate Q and invokes each of its methods. Each method displays one of the constants. **(June-12) [LJIET]** 7

```
interface P
{
        int p=10; // Default is - public static final- no need to declare it- public static final
        void method_p();
}

interface P1 extends P
{
        int p1=20;
        void method_p1();
}

interface P2 extends P
{
        int p2=30;
        void method_p2();
}

interface P12 extends P1,P2
{
        int p12=40;
        void method_p12();
}

class Q implements P12
{
```



```
    public void method_p()
    {
        System.out.println("The Value of p from Interface P: "+p);
    }
    public void method_p1()
    {
        System.out.println("The Value of p1 from Interface P1: "+p1);
    }
    public void method_p2()
    {
        System.out.println("The Value of p2 from Interface P2: "+p2);
    }
    public void method_p12()
    {
        System.out.println("The Value of p12 from Interface P12: "+p12);
    }
}

class InterfaceMain
{

    public static void main(String[] args)
    {
        Q obj_q=new Q();
        obj_q.method_p();
        obj_q.method_p1();
        obj_q.method_p2();
        obj_q.method_p12();

    }

}
```

Output:-

```
The Value of p from Interface P: 10
The Value of p1 from Interface P1: 20
The Value of p2 from Interface P2: 30
The Value of p12 from Interface P12: 40
```

5 Write a program that illustrates interface inheritance. Interface A is extended by A1 and A2. Interface A12 inherits from both A1 and A2.Each interface declares one constant and one method. Class B implements A12.Instantiate B and invoke each of its methods. Each method displays one of the constants. **(Dec-14) [LJIET]** 7

Ans : Same as Above One-(4)

Replace – P by A in interface name and Q by B in class name

6 The Transport interface declares a deliver() method. The abstract class Animal is the superclass of the Tiger, Camel, Deer and Donkey classes. The Transport interface is implemented by the Camel and Donkey classes. Write a test program that initialize an array of four Animal objects. If the object implements the Transport interface, the deliver() method is invoked. . **(Nov-11,Jan-13) [LJIET]** 7

```
interface Transport
```

```
{
        void deliver();
}

abstract class Animal
{
        abstract void jungle();
}

class Tiger extends Animal
{
        public void jungle()
        {
                System.out.println("This is the Tiger Method!!!");
        }
}

class Camel extends Animal implements Transport
{
        public void jungle()
        {
                System.out.println("This is the Camel Method!!!");
        }
        public void deliver()
        {
                System.out.println("Hey Camel! We need to Deliver you to your Owner!!!");
        }
}

class Deer extends Animal
{
        public void jungle()
        {
                System.out.println("This is the Deer Method!!!");
        }
}

class Donkey extends Animal implements Transport
{
        public void jungle()
        {
                System.out.println("This is the Donkey Method!!!");
        }
        public void deliver()
        {
                System.out.println("Hey Donkey! We also need to Deliver you to your Owner!!!");
        }
}
class AnimalMain
{
                public static void main(String[] args)
        {
```

```
                                Transport[] ta = new Transport[4];
                                ta[0]=new Camel ();
                                ta[1] = new Donkey();
                                ta[2]=new Camel ();
                                ta[3] = new Donkey();

                                for(int i =0;i < ta.length;i++)
                                {
                                        ta[i]. deliver();
                                }

                                Tiger t1=new Tiger();
                                t1.jungle();
                                Deer d2=new Deer();
                                d2.jungle();

                }

}
```

Output:-

```
        Hey Camel! We need to Deliver you to your Owner!!!
        Hey Donkey! We also need to Deliver you to your Owner!!!
        Hey Camel! We need to Deliver you to your Owner!!!
        Hey Donkey! We also need to Deliver you to your Owner!!!
        This is the Tiger Method!!!
        This is the Deer Method!!!
```

7 Explain single level and multiple inheritances in java. Write a program to demonstrate combination of both types of inheritance as shown in figure 1. i.e.hybrid inheritance . **(Dec-10) [LJIET]** 7

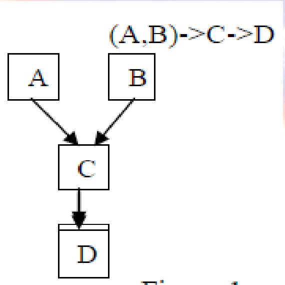


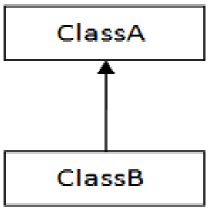
Figure 1

1) Single Inheritance

Single inheritance is damn easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be a **child class** of A.

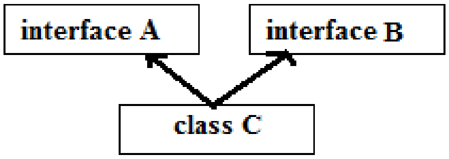
2) Multiple Inheritance

"**Multiple Inheritance**" refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes.



1) Single

```
class A{
}
class B extends A
{
}
```

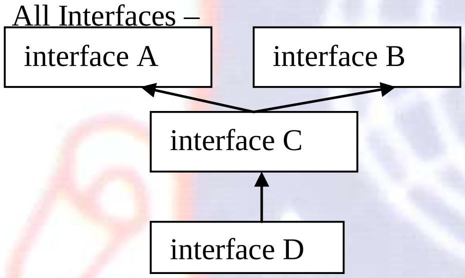


4) Multiple Inheritance

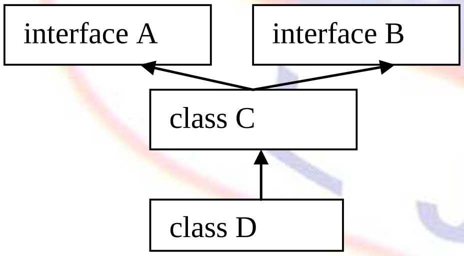
```
interface A
{}

interface B
{}
class C implements A,B
{}
```

Possible Ans Of Figure:

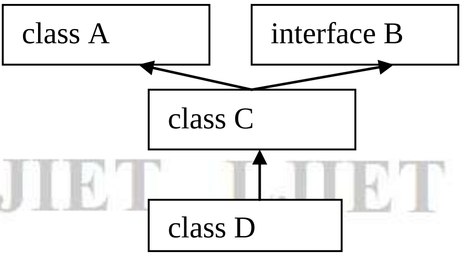
All Interfaces –



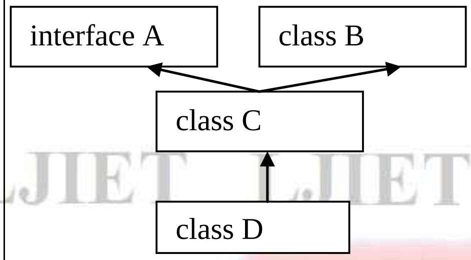
```
interface A
{}
interface B
{}
interface C extends A,B
{}
interface D extends C
{}
```



```
interface A
{}
interface B
{}
class C implements A,B
{}
class D extends C
{}
```



```
interface B
{}
class A
{}
class C extends A implements B
{}
class D extends C
{}
```



```
interface A
{ }
class B
{ }
class C extends B implements A
{ }
class D extends C
{}
```

8 Write a program to demonstrate the multipath inheritance for the classes having relations as shown in figure 2 A-> (B, C) ->D. **(Dec-10, Dec-14) [LJIET]** **7**

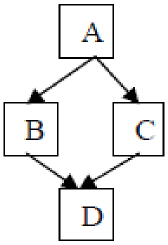


Figure 2

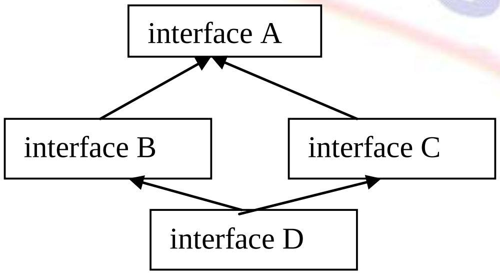
Multipath Inheritance :

When a class is derived from two or more classes which are derived from the same base class then such type of inheritance is called multipath inheritance. [Draw given figure as in example]

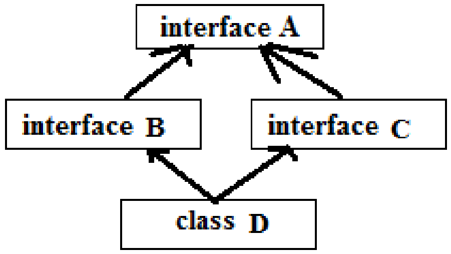
In given example - **class 'D' derived from classes 'B' and 'C', which are derived from same base class 'A'.**

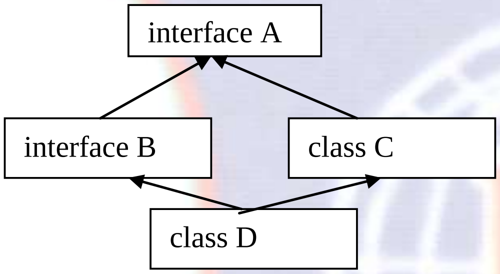
Possible Answers Are :

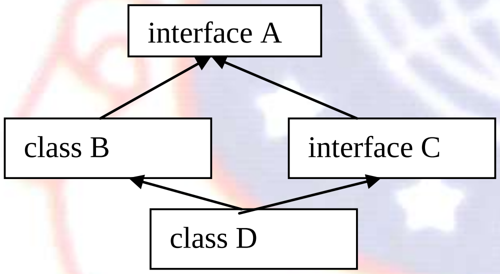
All Interfaces –



```
interface A
{ }
interface B extends A
{ }
interface C extends A
{ }
interface D extends B,C
{}
```


 	
	`interface A`
	`{ }`
	
	`interface B extends A`
	`{ }`
	
	`interface C extends A`
	`{ }`
	
	`class D implements B,C`
	`{ }`

All Interfaces –

 	
	`interface A`
	`{ }`
	`interface B extends A`
	`{ }`
	`class C implements A`
	`{ }`
	`class D extends C implements B`
	`{ }`

All Interfaces –

 	
	`interface A`
	`{ }`
	`interface C extends A`
	`{ }`
	`class B implements A`
	`{ }`
	`class D extends B implements C`
	`{ }`