```
// Encapuslation

package com.prakash;
class Student
{
	private int rollno;
	private String name;//these private variables can assigned or accessed only
using methods
	public int getRollno() {          // Encapsulation is binding data through
methods
		return rollno;
	}
	public void setRollno(int rollno) {
		System.out.println("The user is changing the value");
		this.rollno = rollno;             // The need of encapsulation is to be
ensure that the data is safe
		                                   // when data is public any one can
access
	}                                          // so by keeping it as private it
changed by method & we also let
		                                   //to know that the user is changing the
value
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}

}
public class EncapsulationImp {

	public static void main(String[] args) {



		Student s1 = new Student();
		s1.setRollno(112);
		s1.setName("Halls");

		System.out.println(s1.getRollno()+" "+s1.getName());
	}

}
```

Inheritance

```java
package com.prakash;
class Calc
{
	public int add(int i,int j)
	{
		return i+j;
	}
}
class CalcAdv extends Calc
{
	public int sub(int i,int j)
	{
		return i-j;
	}
}
class CalcVeryAdv extends CalcAdv
{
	public int mul(int i,int j)
	{
		return i*j;
	}
}

public class MultiLevelInheritance {

	public static void main(String[] args) {

		CalcVeryAdv obj = new CalcVeryAdv();
		System.out.println(obj.add(1, 2));
		System.out.println(obj.sub(2, 1));
		System.out.println(obj.mul(3, 4));


	}

}
```

```
//POLYMORPHISM
package com.prakash;
class A
{
	void show()
	{
		System.out.println("in A");
	}
}
class B extends A
{
	void show()
	{
		System.out.println("in B");
	}
	void config()
	{
		System.out.println("in config");
	}
}
class C extends B
{
	void show()
	{
		System.out.println("in C");
	}
}
public class OverRidingExample {

	public static void main(String[] args) {

		A obj = new B();// creating obj with reference to A class to B class
		obj.show();

		obj = new C();
		obj.show();
	//	obj.config() This gives an error because config doesnt present in A
class
		   //

	}

}



// Abstraction
package com.prakash;

 class Sanjay
{
	public void print(Number i)// Number is an abstract class // Integer also
extends number //Float also extends Number

	{
		System.out.println(i); // So only 1 method is enough for passing all 
type of arguments
	}
}
abstract class Writer
{
	abstract void show();

}
class Pen extends Writer
{
	void show()		// Integer or Float both extends Number so we can pass 
Number .This example is same as it is
	{
		System.out.println("Im pen");
	}
}
class Pencil extends Writer
{
	void show()
	{
		System.out.println("Im pencil");
	}
}
class Kit
{
	void doSomething(Writer w)
	{
		w.show();
	}


}

public class AbstractionAnotherDemo {

	public static void main(String[] args) {

		//Sanjay s1 = new Sanjay();
		//s1.print(5);
	//	s1.print(5.5);

		Kit k = new Kit();
		Writer p= new Pen();
		Writer pc = new Pencil();

		k.doSomething(p);
		k.doSomething(pc);

	}

}
package com.prakash;
abstract class Human
{
	public abstract void show();


	void walk()
	{

	}

}
class Men extends Human
{
	public void show()// When we extend an abstract class we should define the 
abstract method
				// Other wise this sub class will also changed as abstract 
class
	{
		System.out.println("In subclass");
	}
}
public class AbstractionDemo {

	public static void main(String[] args) {

	//	Human obj1 = new Human();// we cannot create object for abstract class
		Human obj = new Men();
		obj.show();

	}

}
```

// Interface

```
package com.prakash;
interface Demo
{
	default void show() // default key word is used to define a method in 
interface
	{
		System.out.println("In Demo Show");
	}
}
interface MyDemo
{
	default void show()
	{
		System.out.println("In MyDemo Show");
	}
}
class DemoImp implements Demo,MyDemo
{
	public void show()
	{
		Demo.super.show();
		MyDemo.super.show();
	}
}
public class MultipleInheritanceIssue {

	public static void main(String[] args) {

		Demo obj = new DemoImp();
		obj.show();

	}

}

package com.prakash;

//Types of Interface
// 1.Normal Interface-Which has more than 2 method
//2.single abstract interface-Which has only 1 abstract method
//      |->functional interface - Lambda expressions
//3.Marker Interface -Which does not have any methods



interface Abc
{
	void show();
	// it assume it has an abstract method// we can only declare the class// we 
cannot define the class
}
class Implementor implements Abc
{
	public void show()
	{
		System.out.println("Implemented");
	}

}
public class InterfaceDemo {

	public static void main(String[] args) {

		Abc obj = new Implementor(); // we can create object for interface by 
creating an another class
		obj.show();


		//The another way to create a object for interface is by using anonymous 
class

		Abc obj1 = new Abc()
				{
			public void show()
```

```
                              {

                                   System.out.println("Implemented using anonymous");
                               }
                                 };


                                 obj1.show();

        }

}
```

EXCEPTION HANDLING

```
package com.prakash;

public class ExceptionHandlingDemo {

        public static void main(String[] args) {

                try
                {
                        int[] arr= new int[6];
                        arr[6]=13;
                        int i=13;
                        int j=0;
                        int k=i/j;


                }
                catch(ArithmeticException e)
                {
                        System.out.println("Zero division Exception");
                }
                catch(ArrayIndexOutOfBoundsException e)
                {
                        System.out.println("Limit Exceeded..");
                }
                catch(Exception e)// When we give it before the first catch block it 
gives error
                                              // because (Exception e) extends or handle all 
Exception
                {
                        System.out.println("Any other Exception");
                }
                finally
                {
                        System.out.println("Finally Block");
                }
        }

}

// MultiThreading
package com.halls;
import java.util.*;
class Hi extends Thread // One method of achieving Multithreading by extending thread
{
        public void run()
        {
                for(int i=1;i<=5;i++)
                {
                        System.out.println("Hi");
                        try{Thread.sleep(1000);}catch(Exception e){}
                }
        }

}
class Hello extends Thread
{
        public void run()
        {
                for(int i=1;i<=5;i++)
                {
                        System.out.println("Hello");
                        try{Thread.sleep(1000);}catch(Exception e){}
                }
        }

}
public class MultiThreadingExample {

        public static void main(String[] args) {// main is the default thread

                Hi obj1 = new Hi();
                Hello obj2 = new Hello();
                // MultiThreading is the process of doing multiple task simultaneously

                obj1.start();// This line will call run method
                try{Thread.sleep(10);}catch(Exception e){}
                obj2.start();

        }

}
package com.halls;
class Hii implements Runnable
{

        @Override
        public void run() {
                for(int i=1;i<=5;i++)
                {
                        System.out.println("Hi");
                        try{Thread.sleep(1000);}catch(Exception e){}
                }


        }

}
class Helloo implements Runnable
{

        @Override
        public void run() {
                for(int i=1;i<=5;i++)
                {
                        System.out.println("Hello");
                        try{Thread.sleep(1000);}catch(Exception e){}
                }
        }

}
public class MultiThreadingInterfaceExample {

        public static void main(String[] args) throws Exception{

                Runnable obj1= new Hii();
                Runnable obj2= new Helloo();
        //Name
                Thread t1= new Thread(obj1,"My Thread1");
                Thread t2= new Thread(obj2,"My Thread2");
                System.out.println(t1.getName());
                System.out.println(t2.getName());

        //Name
                t1.setName("Hi Thread");
                t2.setName("Hello Thread");

                System.out.println(t1.getName());
                System.out.println(t2.getName());


        // Priority
                t1.setPriority(Thread.MIN_PRIORITY);
                t2.setPriority(Thread.MAX_PRIORITY);

                System.out.println(t1.getPriority());
                System.out.println(t2.getPriority());


                t1.start();
                try{Thread.sleep(10);}catch(Exception e) {};
                t2.start();

                t1.join();// This will show exception so we should handle exception in
main by throws exception
                t2.join();
```

```
		System.out.println(t1.isAlive());// this shows whether the tread is 
alive or not
		System.out.println(t2.isAlive());
		System.out.println("Bye");// When we doesn't use join main thread is 
also simultaneously
		//running so it will before new threads t1 and t2 are completed

	}

}
```