MODULE 2

1. Write a Java program to facilitate the multilevel inheritance. (Also demonstrate the use of constructor in multilevel inheritance)

```
class travel
{
	travel()
	{
		System.out.println("Travel constructor called");
	}
}
class Indonesia extends travel
{
	Indonesia()
	{
		System.out.println("Indonesia constructor called");
	}
}
class Bali extends Indonesia
{
	Bali()
	{
		System.out.println("Bali constructor called");
	}
}
public class MultipleInheritance
{
	public static void main()
	{
		System.out.println("Order of execution is ");
		Bali b=new Bali();
	}
}
```

Output:

```
Order of execution is
Travel constructor called
Indonesia constructor called
Bali constructor called
```

2. Demonstrate the use of super keyword in inheritance and try to implement the program which can able to perform following operations…

a. Use super to call base class variable.

b. Use super to call base class method.

c. Use super to call base class constructor.

Super a:

```
class fruits {String taste="sweet";}
class grape extends fruits
{
String taste="sour";
void printTaste()
{
System.out.println(taste);
System.out.println(super.taste);
}
}
class supera101
{
public static void main(String args[])
{
grape g=new grape();
g.printTaste();
}
}
```

Output:

```
sour
sweet
```

Super b:

```
class fruits {void color() {System.out.println("fruits have various colors");}}
class grapes extends fruits{
void color(){System.out.println("green");}
void water(){System.out.println("81% water");}
void print(){
super.color();
water();
}
}
class superb101{
public static void main(String args[]){
grapes g=new grapes();
g.print();
}}
```

Output:

```
fruits have various colors
81% water
```

Super c:

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class superc101{
public static void main(String args[]){
Dog d=new Dog();
}}
```

Output:

```
animal is created
dog is created
```

3. Demonstrate the use of final keyword with respect to

a. Variable

b. Method

c. Class

Final c:

```
final class Bike{}

class TVS extends Bike{
  void run(){System.out.println("running safely with 60kmph");}

  public static void main(String args[]){
  TVS tvs= new TVS();
  tvs.run();
  }
}
```

Output:

```
error: cannot inherit from final Bike
```

Final b:

```
class Bike{
  final void run(){System.out.println("running");}
}

class TVS extends Bike{
   void run(){System.out.println("running safely with 60kmph");}

   public static void main(String args[]){
   TVS tvs= new TVS();
   tvs.run();
   }
}
```

Output:

```
error: run() in TVS cannot override run() in Bike
```

Final a:

```
class Bike{
 final int speedlimit=90;
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike obj=new  Bike();
 obj.run();
 }
}
```

Output:

```
error: cannot assign a value to final variable speedlimit
```

4. Write a Java Program to demonstrate the concept of hierarchical inheritance. (implement this program with super keyword, final keyword, constructor and method overriding concept)

```
class Employee{
    final float salary = 40000;
    Employee(){System.out.println("Employee constructor called");}
    void yearsOfExp()
    {
        int Exp=25;
        System.out.println("Years of experience for an Employee is:"+Exp);
    }
    }
    class PermanentEmp extends Employee{
    double hike = 0.5;
    PermanentEmp(){System.out.println("PermanentEmp constructor called");}
    void yearsOfExp()
    {
        int Exp=35;
        System.out.println("Years of experience for Permanent Employee is:"+Exp);
    }
    }
    class TemporaryEmp extends Employee{
    double hike = 0.35;
    TemporaryEmp(){System.out.println("TemporaryEmp constructor called");}
    void yearsOfExp()
    {
        int Exp=15;
        System.out.println("Years of experience for Temporary Employee is:"+Exp);
    }
    }
    public class hier101
    {
    public static void main(String args[]){
    PermanentEmp p = new PermanentEmp();
    TemporaryEmp t = new TemporaryEmp();
    System.out.println("Salary for all Employees is :" +p.salary);
    p.yearsOfExp();
    System.out.println("Hike for Permanent Employee is:" +p.hike);
    t.yearsOfExp();
    System.out.println("Hike for Temporary Employee is :" +t.hike);
    }
    }
```

Output:

```
Employee constructor called
PermanentEmp constructor called
Employee constructor called
TemporaryEmp constructor called
Salary for all Employees is 40000
Years of experience of Permanent Employees is 35
Hike for permanent employee is 0.5
Years of experience of Temporary Employee is 15
Hike for temporary employee is 0.35
```

5. Write a java program to demonstrate dynamic method dispatch and abstract keyword with class and methods.

```
abstract class Bank{
abstract float getRateOfInterest(){}
}
class SBI extends Bank{
float getRateOfInterest(){return 8.4f;}
}
class AXIS extends Bank{
```

```
float getRateOfInterest(){return 9.7f;}
}
class dynabs101{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());
b=new ICICI();
System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());
b=new AXIS();
System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());
}
}
```

Output:

```
SBI Rate of Interest 8.4
ICICI Rate of Interest 9.7
```

6. Try to implement the concept of multiple inheritance in Java with the use of interface.

```
interface IEat {
  void eat();
}
interface ITravel {
  void travel();
}
class Me implements IEat, ITravel {
  public void eat() {
    System.out.println("I am eating");
  }
  public void travel() {
    System.out.println("I am traveling");
  }
}
public class multipleinherit101 {
  public static void main(String args[]) {
    Me m = new Me();
    m.eat();
    m.travel();
```

```
   }
}
```

Output:

```
I am eating
I am traveling
```

7. write a Java program to demonstrate the hybrid inheritance with example

```
class GrandFather
{
 public void printGrandFather()
 {System.out.println("GrandFather's class");}
}

class Father extends GrandFather
{
 public void printFather()
 {System.out.println("Father class has inherited GrandFather class");}
}

class Son extends Father
{
 public Son()
 {System.out.println("Inside the Son Class");}
 public void printSon()
 {System.out.println("Son class has inherited Father class");}
}

class Daughter extends Father
{
 public Daughter()
 {
   System.out.println("Inside the Daughter Class");
 }

 public void printDaughter()
 {
   System.out.println("Son class has inherited Father class");
 }
}

public class hybrid101
{
 public static void main(String[]args)
 {
   Son obj = new Son();
   obj.printSon();
   obj.printFather();
   obj.printGrandFather();

   Daughter obj2 = new Daughter();
   obj2.printDaughter();
   obj2.printFather();
   obj2.printGrandFather();
 }
}
```

Output:

```
Inside Son’s class
Son class has inherited Father class
Father class has inherited Grandfather class
Grandfather’s class
Inside Daughter class
Daughter class has inherited Father class
Father class has inherited Grandfather class
Grandfather’s class
```

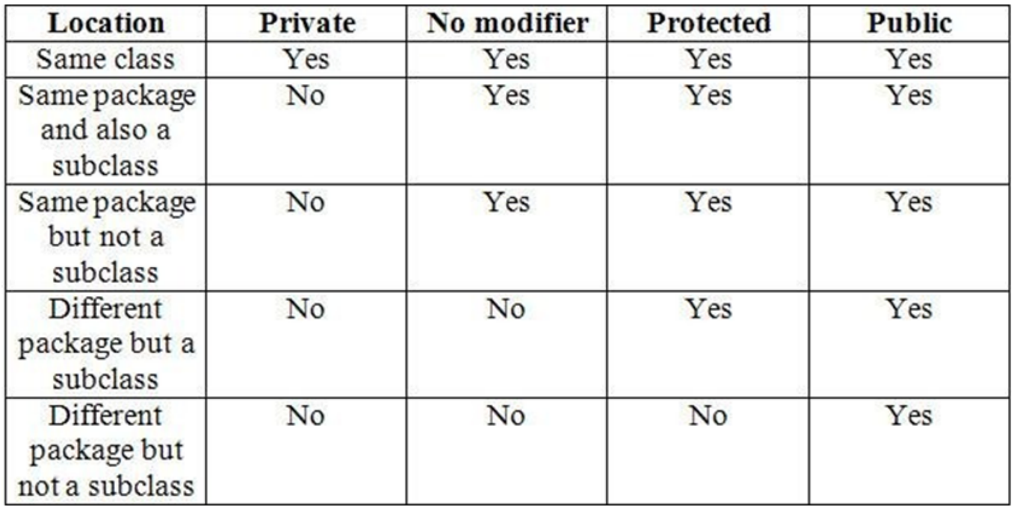
8. Write a program that demonstrates use of packages & import statements. (Simple addition program).

```
import java.util.*;
package mypack;
class packageDemo {
    public static void main(String[] args)
    {
        Scanner myObj = new Scanner(System.in);
        String userName;
        System.out.println("Enter You Name");
        userName = myObj.nextLine();
        System.out.println("Your Name IS : " + userName);
    }
}
```

Output:

```
Enter your name:
Shalini
Your name is Shalini
```

9. Write java program to check all the available access specifier with package and try to prove that the below given table is true.



Location	Private	No modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package and also a subclass	No	Yes	Yes	Yes
Same package but not a subclass	No	Yes	Yes	Yes
Different package but a subclass	No	No	Yes	Yes
Different package but not a subclass	No	No	No	Yes

File 1:

```
class File 1
{
        private int a=1;
        int b=2;
        protected int c=3;
        public int d=4;
        public static void main(String args[])
        {
                a=10; b=20; c=30; d=40;
        }
}
```

File 2:

```
package p1;
class File2
{
	private int a=1;
	int b=2;
	protected int c=3;
	public int d=4;
}
```

File 3:

```
package p1;
class File3 extends File2
{
	public static void main(String[] args)
	{
		a=10; b=20; c=30; d=40;
	}
}
```

File 4:

```
package p1;
class File4
{
	public static void main(String[] args)
	{
		a=10; b=20; c=30; d=40;
	}
}
```

File 5:

```
package p2;
import p1.*;
class File5 extends File2
{
	public static void main()
	{
		a=10; b=20; c=30; d=40;
	}
}
```

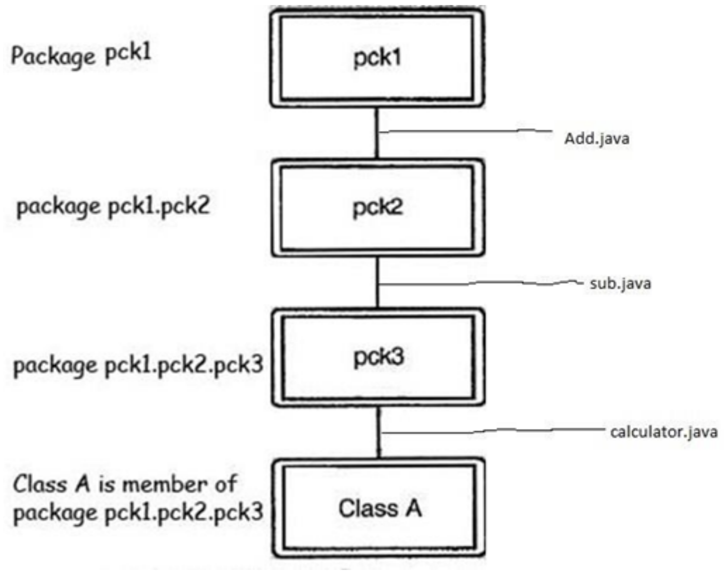
File 6:

```
package p3;
import p1.*;
class File6
{
        public static void main()
        {
                a=10; b=20; c=30; d=40;
        }
}
```

Outputs:

```
Output(File 3):
Error: 'a' cannot be accessed
Output(File 4):
Error: 'a' cannot be accessed
Output(File 5):
Error: 'a' cannot be accessed
Error: 'b' cannot be accessed
Output(File 6):
Error: 'a' cannot be accessed
Error: 'b' cannot be accessed
Error: 'c' cannot be accessed
```

10. Write a java program to create the hierarchy like below figure to perform addition and subtraction operation.



addition.java:

```
package addition1;
public class addition
{
	public int add(int a, int b)
	{
		return (a+b);
	}
}
```

subtraction.java:

```
package subt;
import addition1.*;
public class subtraction
{
	public int sub(int a, int b)
	{
		return (a-b);
	}
}
```

simplecalc.jav:

```
package calci;
import subt.*;
import java.util.*;
public class simplecalc
{
	Scanner ob=new Scanner(System.in);
	addition1.addition ob1=new addition1 addition();
	subtraction ob2=new subtraction();
	public void calculate()
	{
		System.out.println("Enter two numbers");
		int a=ob.nextInt();
```

```
                int b=ob.nextInt();
                System.out.println("1. Addition\n 2. Subtraction\n Enter your choice: ");
                int ch=ob.nextInt();
                switch(ch)
                {
                        case 1: System.out.println("The sum is "+ob1.add(a,b));
                        break;
                        case 2: System.out.println("The diff is "+ob2.sub(a,b));
                        break;
                        default: System.out.println("Enter correct choice");
                        break;
                }
        }
}
```

Main.java:

```
import calci.Simplecalc;
import java.util.*;
class Main
{
        public static void main(String[] args)
        {
                Simplecalc ob=new Simplecalc();
                ob.calculate();
        }
}
```

Output:

```
Enter 2 numbers:
30
35
1. Addition
2. Subtraction
Enter your choice
1
The sum is: 65
```

11. Write a different java program for generating following types of exception

a. NullPointerException
b. ArrayIndexOutOfBoundException
c. ArithmeticException
d. NumberFormatException
e. StringIndexOutOfBoundException

```
            import java.util.*;
class ExceptionTypes {
        public static void main(String[] args) {
                try {
                        int a=3/0;
                }
                catch(ArithmeticException e) {
                        System.out.println(e);
                }
                System.out.println("Arithmetic Exception");
                try {
                        String str=null;
                        System.out.println(str.length());
                }
                catch(NullPointerException e) {
                        System.out.println(e);
                }
                System.out.println("Null Pointer Exception");
                try {
                        int a[]=new int[3];
                        a[4]=15;
                }
                catch(ArrayIndexOutOfBoundsException e) {
                        System.out.println(e);
                }
                System.out.println("Array Index Out Of Bounds Exception");
                try {
                        String s="Shalini";
                        char ch=s.charAt(25);
                        System.out.println(s);
                        System.out.println(ch);
                }
                catch(StringIndexOutOfBoundsException e) {
                        System.out.println(e);
                }
                System.out.println("String Index Out Of Bounds Exception");
                try {
                        String w="Shalini";
                        int n=Integer.parseInt(w);
                }
                catch(NumberFormatException e) {
                        System.out.println(e);
                }
                System.out.println("Number Format Exception");
        }
}
```

Output:

```
java.lang.ArithmeticException: / by zero
Arithmetic Exception
java.lang.NullPointerException
Null Pointer Exception
java.lang.ArrayIndexOutOfBoundsException: Index 4 out of bounds for length 3
Array Index Out Of Bounds Exception
java.lang.StringIndexOutOfBoundsException: String index out of range: 25
String Index Out Of Bounds Exception
java.lang.NumberFormatException: For input string: "Shalini"
Number Format Exception
```

12. Write a program to demonstrate user defined exception.

```
import java.util.*;

class UserDefinedException extends Exception{
        public static void main(String[] args) {
            String s="Shalini";
            try {
                if(s.length()<20)
                    throw new Exception("String length should be more than 20.");
                }
                catch(Exception e) {
                        System.out.println("Exception detected");
                        System.out.println(e.getMessage());
                }
        }
}
```

Output:

```
Exception detected
String length should be more than 20.
```

13. Write a java program to demonstrate the use of nested try block.

```
import java.util.*;
class NestedTryBlock {
         public static void main(String args[]){
          try{
            try{
             System.out.println("Dividing by 0");
             int b =10/0;
             }
            catch(ArithmeticException e)  {
              System.out.println(e);
            }
            try{
              int a[]=new int[5];
              a[8]=4;
            }
            catch(ArrayIndexOutOfBoundsException e)  {
               System.out.println(e);
            }
            System.out.println("some statements");
          }
          catch(Exception e)  {
            System.out.println("OUTER CATCH");
          }
          System.out.println("Rest of the statements");
         }
```

```
    }
```

Output:

```
java.lang.ArithmeticException: / by zero
java.lang.ArrayIndexOutOfBoundsException: Index 8 out of bounds for length 5
some statements
Rest of the statements
```

14. Write a java program to demonstrate exception with overloaded methods

```
import java.util.*;

public class ExceptionOverloading {
  double method(int i) throws Exception {
    return i/0;
  }
  boolean method(boolean b) {
    return !b;
  }
  static double method(int x, double y) throws Exception {
    return x + y ;
  }
  static double method(double x, double y) {
    return x + y - 3;
  }
  public static void main(String[] args) {
        ExceptionOverloading mn = new ExceptionOverloading();
    try {
      System.out.println(method(10, 20.0));
      System.out.println(method(30.0, 10));
      System.out.println(method(20.0, 40.0));
      System.out.println(mn.method(10));
    } catch (Exception ex) {
      System.out.println("Exception: "+ ex);
    }
  }
}
```

Output:

```
30.0
```

```
37.0
57.0
Exception: java.lang.ArithmeticException: / by zero
```

15. Write program in java to demonstrate the use of throw and throws in exception.

```
import java.io.IOException;
class ThrowThrows{
	void fun()throws IOException{
		throw new IOException("Device Error");
	}
}
public class ThrowAndThrows {
	public static void main(String[] args) {
		try {
			ThrowThrows obj=new ThrowThrows();
			obj.fun();
		}
		catch(Exception e) {
			System.out.println("Exception handled");
		}
		System.out.println("Other statements");
	}
}
```

Output:

```
Exception handled
Other statements
```

16. Write a program for creating three threads randomly using following methods:

a. By extending Thread class
b. By implementing Runnable interface

a.

```
import java.util.*;

class ThreeThreads extends Thread{
	ThreeThreads(String threadName) {
		super(threadName);
	}
	public void run() {
		System.out.println("Thread "+Thread.currentThread().getName()+" is
runnning");
	}
	public static void main(String[] args) {
		ThreeThreads t1=new ThreeThreads("t1");
		ThreeThreads t2=new ThreeThreads("t2");
		ThreeThreads t3=new ThreeThreads("t3");
		t1.start();
		t2.start();
		t3.start();
	}
}
```

Output:

```
Thread t1 is running
Thread t2 is running
Thread t3 is running
```

b.

```
import java.io.*;
class threeb implements Runnable
{
	public void run()
	{
		System.out.println("Thread is running inside");
	}
	public static void main(String args[])
	{
		threeb th1=new threeb();
		Thread t1=new Thread(th1);
		t1.start();
		Thread t2=new Thread(th1);
		t2.start();
		Thread t3=new Thread(th1);
		t3.start();
	}
```

}

Output:

```
Thread is running inside
Thread is running inside
Thread is running inside
```

17. Write a thread program to demonstrate stop(), sleep(), isAlive () and join ().

```
import java.util.*;
public class ThreadOperations extends Thread{
	public void run(){
		for(int i=0;i<=3;i++){
			try{
				Thread.sleep(2000);
			}catch(Exception e){
				System.out.println(e);
			}
			System.out.println(i);
		}
	}
	public static void main(String[] args){
		ThreadOperations t1=new ThreadOperations();
		ThreadOperations t2=new ThreadOperations();
		t1.start();
		System.out.println("Thread t1 is "+t1.isAlive());
		try{
			t1.join();
		}
		catch(Exception  e){
			System.out.println(e);
		}
		System.out.println("Thread t1 is "+t1.isAlive());
		t2.start();
	}
}
```

Output:

```
Thread t1 is true
```

```
0
1
2
3
Thread t1 is false
0
1
2
3
```

18. Write a program to create a new thread by extending a thread class.

a. Get the current thread name
b. Set the highest priority to the newly created thread
c. Pause a thread for 2.5 seconds.
d. Check whether the thread is in running state or not.
e. Verify your newly created thread must be completed first before your main thread is completed.

```
import java.util.*;

class NewThread extends Thread{
	public void run() {
		System.out.println("Thread "+Thread.currentThread().getName()+" is
running");
	}
	public static void main(String[] args) {
		NewThread t1=new NewThread();
		t1.start();
		System.out.println("Currect thread name is
"+Thread.currentThread().getName());
		t1.setPriority(MIN_PRIORITY);
		System.out.println("Thread priority is "+t1.getPriority());
		try {
			Thread.sleep(2500);
		}catch(InterruptedException e) {
			System.out.println(e);
		}
		System.out.println("New thread state is "+t1.getState());
```

```
            System.out.println("Main thread state is "+Thread.currentThread().getState());
        }
}
```

Output:

```
Thread priority is 1
Thread Thread-0 is running
New thread state is TERMINATED
Main thread state is RUNNABLE
```

19. Write a Java program to create deadlock type situation using multiple threads. Also provide solution to come out from the deadlock.

```
import java.util.*;

class Deadlock {
  public static Object lock1= new Object();
  public static Object lock2= new Object();
  public static void main(String args[]) {
    Thread1 t1 = new Thread1();
    Thread2 t2 = new Thread2();
    t1.start();
    t2.start();
  }
  private static class Thread1 extends Thread {
    public void run() {
      synchronized (lock1) {
        System.out.println("Thread 1: Holding lock 1...");
        try { Thread.sleep(10); }
        catch (InterruptedException e) {}
        System.out.println("Thread 1: Waiting for lock 2...");
        synchronized (lock2) {
          System.out.println("Thread 1: Holding lock 1 & 2...");
        }
      }
    }
  }
  private static class Thread2 extends Thread {
    public void run() {
      synchronized (lock2) {
        System.out.println("Thread 2: Holding lock 2...");
        try { Thread.sleep(10); }
        catch (InterruptedException e) {}
        System.out.println("Thread 2: Waiting for lock 1...");
        synchronized (lock1) {
          System.out.println("Thread 2: Holding lock 1 & 2...");
        }
      }
    }
  }
}
```

Output:

```
Thread 1: Holding lock 1...
Thread 2: Holding lock 2...
Thread 1: Waiting for lock 2...
Thread 2: Waiting for lock 1...
^C
```

Solution:

```
public class AvoidDeadlockExample
{
        public static void main(String[] args) throws InterruptedException
        {
                Object object1 = new Object();
                Object object2 = new Object();
                Object object3 = new Object();
                Thread thread1 = new Thread(new SynchroniseThread(object1, object2),
"thread1");
                Thread thread2 = new Thread(new SynchroniseThread(object2, object3),
"thread2");
                thread1.start();
                Thread.sleep(2000);
                thread2.start();
                Thread.sleep(2000);
        }
}
```

```
class SynchroniseThread implements Runnable
{
	private Object object1;
	private Object object2;
	public SynchroniseThread(Object o1, Object o2)
	{
		this.object1=o1;
		this.object2=o2;
	}
	public void run()
	{
		String name = Thread.currentThread().getName();
		System.out.println(name + " acquire lock on " + object1);
		synchronized (object1)
		{
			System.out.println(name + " acquired lock on " + object1);
			work();
		}
	System.out.println(name + " released lock of " + object1);
	System.out.println(name + " acquire lock on " + object2);
	synchronized (object2)
	{
		System.out.println(name + " acquire lock on " + object2);
		work();
	}
	System.out.println(name + " released lock of " + object2);
	System.out.println(name + " execution is completed.");
	}
	private void work()
	{
		try
		{
			Thread.sleep(5000);
		}
		catch (InterruptedException e)
		{
			e.printStackTrace();
		}
	}
}
```

Output:

```
thread1 acquire lock on java.lang.Object@5ebab9e8
thread1 acquired lock on java.lang.Object@5ebab9e8
thread2 acquire lock on java.lang.Object@181b35c2
thread2 acquired lock on java.lang.Object2181b35c2
thread1 released lock of java.lang.Object@5ebab9e8
thread1 acquire lock on java.lang.Object181b35c2
thread2 released lock of java.lang.Object2181b35c2
thread2 acquire lock on java.lang.Object@16dcc8b1
thread1 acquire lock on java.lang.Object@181b35c2
thread2 acquire lock on java.lang.Object@16dcc8b1
thread1 released lock of java.lang.Object2181b35c2
thread2 released lock of java.lang.Object@16dcc8b1
thread1 execution is completed.
thread2 execution is completed.
```

20. Write a Java Program to implement the concept of inter thread communication with very famous producer consumer problem.

```
import java.util.*;

class ThreadCommunication {
	public static void main(String[] args) throws InterruptedException	{
		final PC pc = new PC();
		Thread t1 = new Thread(new Runnable() {
			public void run(){
				try {
					pc.produce();
				}
				catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		});
		Thread t2 = new Thread(new Runnable() {
			public void run(){
				try {
```

```
                    pc.consume();
                }
                catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        t1.start();
        t2.start();
        t1.join();
        t2.join();
    }
public static class PC {
        LinkedList<Integer> list = new LinkedList<Integer>();
        int capacity = 1;
        public void produce() throws InterruptedException{
            int value = 0, i=0;
            while (i<10) {
                synchronized (this){
                    while (list.size() == capacity)
                        wait();
                    System.out.println("Producer produced-"+ value);
                    list.add(value++);
                    notify();
                    Thread.sleep(1000);
                }
              i++;
            }
        }
        public void consume() throws InterruptedException        {
          int i=0;
            while (i<10) {
                synchronized (this){
                    while (list.size() == 0)
                        wait();
                    int val = list.removeFirst();
                    System.out.println("Consumer consumed-"+ val);
                    notify();
                    Thread.sleep(1000);
                }
```

```
                    i++;
                }
            }
        }
}
```

Output:

```
Producer produced-0
Consumer consumed-0
Producer produced-1
Consumer consumed-1
Producer produced-2
Consumer consumed-2
Producer produced-3
Consumer consumed-3
Producer produced-4
Consumer consumed-4
Producer produced-5
Consumer consumed-5
Producer produced-6
Consumer consumed-6
Producer produced-7
Consumer consumed-7
Producer produced-8
Consumer consumed-8
Producer produced-9
Consumer consumed-9
```

21. Assume that only one copy of the book is available in Amazon and four customers are trying to place the order for book at the same time. Write a java program using threads which prints book confirmed for one person and "out of stock" for others.

```
public class Amazon implements Runnable{
        static int noOfBooks=1;
        synchronized public void run() {
                if(noOfBooks>0) {
                        noOfBooks--;
                        System.out.println("Confirmed");
                }
                else {
                        System.out.println("Out of stock");
                }
        }
        public static void main(String[] args) {
                Thread t1=new Thread(new Amazon(),"Thread1");
                Thread t2=new Thread(new Amazon(),"Thread2");
                Thread t3=new Thread(new Amazon(),"Thread3");
                Thread t4=new Thread(new Amazon(),"Thread4");
                t1.start();
                t2.start();
                t3.start();
                t4.start();

        }
}
```

Output:

```
Confirmed
Out of stock
Out of stock
Out of stock
```

22. Write a java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.

```
import java.util.Random;

class RandomNumberThread extends Thread{
    public void run() {
        Random random=new Random();
        for(int i=0;i<5;i++) {
            int randomInteger=random.nextInt(1000);
            System.out.println(randomInteger);
            if(randomInteger%2==0) {
                SquareThread sThread=new SquareThread(randomInteger);
                sThread.start();
            }
            else{
                CubeThread cThread=new CubeThread(randomInteger);
                cThread.start();
            }
            try {
                Thread.sleep(2000);
            }
            catch(Exception e) {
                System.out.println(e);
            }
        }
    }
}
class SquareThread extends Thread{
    int number;
    SquareThread(int n){
        number=n;
    }
    public void run() {
        System.out.println("Square of number is "+number*number);
    }
}
class CubeThread extends Thread{
    int number;
    CubeThread(int n){
        number=n;
    }
    public void run() {
        System.out.println("Cube of number is "+number*number*number);
    }
}
public class RandomGenerator {
    public static void main(String[] args) {
        RandomNumberThread r=new RandomNumberThread();
        r.start();
    }
}
```

Output:

```
223
Cube of number is 11089567
273
Cube of number is 20346417
595
Cube of number is 210644875
164
Square of number is 26896
710
Square of number is 504100
```