Assignment 8:

Thread and Multi thread

Q. 1 In this challenge, simulate a banking system. Create the Account and Transaction classes.

1. The Account class has a data member int balance, initially assigned to zero. The class should implement the following three methods:

String deposit(int money) to add money to the balance. This method should return a string that describes the deposit transaction, i.e., "Depositing $money".

String withdraw(int money) to subtract money from the balance. This method should return a string that describes the withdraw transaction, i.e., "Withdrawing $money". Note that, if there is insufficient balance to successfully withdraw the desired amount, then the balance should not be adjusted, and

the returned string should be "Withdrawing $money (Insufficient Balance)".int getBalance() to return the account balance.

2. The Transaction class has two data members Account account and List transactions. The class should implement the following three methods:

void deposit(int money) to invoke the deposit method in the Account class. This should add the transaction message to the transactions list.

void withdraw(int money) to invoke the withdraw method in the Account class. This should add the transaction message to the transactions list. List getTransaction() to return the transactions.

```
import java.util.*;

class Account {

int balance=0;

public String deposit(int money){

      balance += money;

      return "Depositing $"+money;

   }

   public String withdraw(int money){

      if(balance<money){

         return "Withdraw $"+money+" (Insufficient Balance)";

      } else{

         balance -= money;
```

```
            return "Withdraw $"+money;
        }
    }
    public int getBalance(){
        return balance;
    }
}
class Transaction {
    Account account = new Account();
    List<String> transactions = new ArrayList<>();
 public Transaction(Account account){
        this.account = account;
    }
    public void deposit(int money){
        transactions.add(account.deposit(money));
    }
       public void withdraw(int money){
        transactions.add(account.withdraw(money));
    }
      public List<String> getTransaction(){
        return transactions;
    }
}
```

Q2. Write a program of producer and consumer

```
public classs threadexample{
```

```
  public static void main(String[] args)  throwsInterruptedException     {
 // Object of a class that has both produce()
// and consume() methods
 final PC pc = new PC();   // Create producer thread  Thread t1 = new Thread(new Runnable() {
    @Override
     public void run()   {
      try {
     pc.produce();
     }
     catch (InterruptedException e) {
     e.printStackTrace(); }   }
       });  // Create consumer thread
    Thread t2 = new Thread(new Runnable() {
       @Override
       public void run() {
       try {
        pc.consume();
      }
     catch (InterruptedException e) {
      e.printStackTrace();
     }   }
});  // Start both threads
 t1.start();
t2.start(); // t1 finishes before t2
t1.join();
 t2.join();
 }   // This class has a list, producer (adds items to list
 // and consumber (removes items).
 public static class PC {
```

```
// Create a list shared by producer and consumer .
// Size of list is 2.
 LinkedList<Integer> list = new LinkedList<>();
int capacity = 2;  // Function called by producer thread    public void produce() throws
InterruptedException  {  int value = 0;
  while (true) {
 synchronized (this) {
 // producer thread waits while list is full
 while (list.size() == capacity)
  wait();
 System.out.println("Producer produced-"+ value);
    // to insert the jobs in the list
    list.add(value++);   // notifies the consumer threadthat  // now it can start consuming
       notify(); // makes the working of program easier
     // to  understand
    Thread.sleep(1000); }   }
    } // Function called by consumer thread
public void consume() throws InterruptedException {   while (true) {
 // consumer thread waits while lis is empty
while (list.size() == 0)
wait();   // to retrive the ifrst job in the list
  int val = list.removeFirst(); System.out.println("Consumer  + val);   // Wake up //producer thread
  notify();  // and sleep
Thread.sleep(1000);   }
  } } }
}
```

Output:

```
Producer produced-0
Producer produced-1
Consumer consumed-0
Consumer consumed-1
Producer produced-2
```

Q3. Write a program of thread to used all the method of thread like (wait, sleep, notify, notifyall,join,getname, currentthreade islive?, priority).

you have to write on sigle program to used this all.

```
class
A
extends
Thread
{
```

```
public void run()
{
    for (int i = 1; i <= 4; i++) {
        try {
            wait();
        }
        catch (Exception e) {
            System.out.println(e);
        }
        System.out.print(i + " ");
    }
}
```

```
}
class B extends Thread {



    public void run()
    {   System.out.println(Thread.currentThread().getName() + " in control");
        for (char i = 'a'; i <= 'd'; i++) {
            try {
                Thread.sleep(100);
            }
            catch (Exception e) {
                System.out.println(e);
            }
            System.out.print(i + " ");



        }
    }
}
class a8q3 extends Thread {


    public static void main(String args[])
    {
        // creating two threads
        A a1 = new A();
        B b1 = new B();


        // starts second thread after when
```

```
            // first thread a1 is died.
            a1.start();
            a1.setPriority(5);
            System.out.println(a1.isAlive());
            b1.start();
            try {
                a1.join();
            }
            catch (Exception e) {
                System.out.println(e);
            }


        }
    }
```

Q. 4 You are required to compute the power of a number by implementing a calculator. Create a class MyCalculator which consists of a single method long power(int, int). This method takes two integers, n and p, as parameters and finds . If either p or n is negative, then the method must throw an exce

ption which says "n and p should not be negative". Also, if both and are zero, then the method must throw an exception which says "n and p should not be zero"

For example, -4 and -5 would result in java.lang.Exception: n or p should not be negative.

Complete the function power in class MyCalculator and return the appropriate result after the power operation or an appropriate exception as detailed above.

```
import java.io.*;
import java.util.*;
```

```
import java.io.*;
//Write your code here
class Calculator{
  public int power(int n,int p) throws Exception  {
 if(n>=0 && p >=0){
          return (int)Math.pow(n,p);
}
       else {
  throw new Exception("n and p should be non-negative");
       } } }

class Solution{
public static void main(String[] args) {
  Scanner in = new Scanner(System.in);
    int t = in.nextInt();
       while (t-- > 0) {
    int n = in.nextInt();
    int p = in.nextInt();
     Calculator myCalculator = new Calculator();
       try {
      int ans = myCalculator.power(n, p);
      System.out.println(ans);
    }
     catch (Exception e) {
     System.out.println(e.getMessage());
          }
       }
       in.close();
   }
   }
```

Output :

5.Java Program Showing Execution of Multiple Tasks with a Single Thread.

```
class Count extends Thread
{
  Count()
  {
    super("my extending thread");
    System.out.println("my thread created" + this);
    start();
  }
  public void run()
  {
    try
    {
      for (int i=0 ;i<10;i++)
      {
        System.out.println("Printing the count " + i);
        Thread.sleep(1000);
      }
    }
    catch(InterruptedException e)
    {
      System.out.println("my thread interrupted");
    }
    System.out.println("My thread run is over" );
  }
```

```
}
class ExtendingExample
{
  public static void main(String args[])
  {
    Count cnt = new Count();
    try
    {
      while(cnt.isAlive())
      {
        System.out.println("Main thread will be alive till the child thread is live");
        Thread.sleep(1500);
      }
    }
    catch(InterruptedException e)
    {
     System.out.println("Main thread interrupted");
    }
    System.out.println("Main thread's run is over”);
```

Output:

```
my thread [my runnable thread,5,main]
Main thread will be alive till the child thread is live
Printing the count 0
Printing the count 1
Main thread will be alive till the child thread is live
Printing the count 2
```

```
Main thread will be alive till the child thread is live
Printing the count 3
Printing the count 4
Main thread will be alive till the child thread is live
Printing the count 5
Main thread will be alive till the child thread is live
Printing the count 6
Printing the count 7
Main thread will be alive till the child thread is live
Printing the count 8
Main thread will be alive till the child thread is live
Printing the count 9
mythread run is over
Main thread run is over
```

Q6.Java Program Showing Two Threads Working Simultaneously Upon Two Objects

```
class RunnableDemo implements Runnable {
private Thread t;
 private String threadName;
 RunnableDemo( String name) {
threadName = name;
 System.out.println("Creating " + threadName );
 }
public void run()
{
```

```
System.out.println("Running " + threadName );
 try { for(int i = 4; i > 0; i--) {
 System.out.println("Thread: " + threadName + ", " + i);
// Let the thread sleep for a while.
Thread.sleep(50); }
 }
catch (InterruptedException e) {
System.out.println("Thread " + threadName + " interrupted.");
}
 System.out.println("Thread " + threadName + " exiting.");
}
public void start ()
{
System.out.println("Starting " + threadName );
if (t == null)
 {
t = new Thread (this, threadName);
t.start ();
 } } }
 public class TestThread
{
public static void main(String args[]) {
 RunnableDemo R1 = new RunnableDemo( "Thread-1");
R1.start();
RunnableDemo R2 = new RunnableDemo( "Thread-2");
R2.start();
 }
 }
```

Output :

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

Q. 7 Java Program Showing Two Threads Acting Upon a Single Object.

```
public class One extends Thread {

PrintNumbers p;

int i = 1;

public One(PrintNumbers p) {

this.p = p;

 }

@Override

 public void run() {

 int prev = 1;

 while (prev < 1111) {

 p.printOne(prev);

 prev = (int)
```

```
(prev + Math.pow(10, i));
 i = i + 1;
  } } }

 public class Two extends Thread  {
            int i = 1;
             PrintNumbers p;
             public Two(PrintNumbers p)
             { this.p = p;
              }
              @Override
              public void run() {
                        int prev = 2;
                        while (prev < 2222) {
                                  p.printTwo(prev);
                                  prev = (int) (prev + 2 * Math.pow(10, i));
                                  i = i + 1;
                                  } } }

public class PrintTest {
            public static void main(String[] args) {
                        PrintNumbers b = new PrintNumbers();
                         One firstThread = new One(b);
                         Two secondThread = new Two(b);
                        firstThread.setName("first: "); firstThread.start();
                         secondThread.start();
                          }
                           }
```

Output:

first: 1
second: 2

first: 11
second: 22

first: 111
second: 222

Q. 8 Java Program with 2 Threads Which Prints Alternatively.

```
public class Test {

static int count = 0;

public static void main(String[] args)

throwsInterruptedException {

final Object lock = new Object();

Thread t1 = new Thread(new Runnable() {

@Override

public void run() {

for (int i = 0; i < 10; i++) {

synchronized (lock) {

count++;

System.out.println("Count incremented to " + count

+ " by " + Thread.currentThread().getName());

try {

lock.wait();

lock.notify();

} catch (InterruptedException e) {

e.printStackTrace();
```

```
}
}}
} });

Thread t2 = new Thread(new Runnable() {
@Override
public void run() {
for (int i = 0; i < 10; i++) {
synchronized (lock) {
lock.notify();
count++;
System.out.println("Count incremented to " + count
+ " by " + Thread.currentThread().getName());
try {
lock.wait();
} catch (InterruptedException e) {
e.printStackTrace();
}
}}
} });

t1.start();
t2.start();
t1.join();
t2.join();
} }
```

Q9.Java Program to Start One Thread More than Once.

No. After starting a thread, it can never be started again. If you does so, an *IllegalThreadStateException* is thrown. In such case, thread will run once but for second time, it will throw exception.

Let's understand it by the example given below:

```
public class TestThreadTwice1 extends Thread {
 public void run(){
  System.out.println("running...");
 }
public static void main(String args[]){
TestThreadTwice1 t1=new TestThreadTwice1();
 t1.start();
t1.start();
 }
}
```

Output :

```
                running
            Exception in thread "main"
 java.lang.IllegalThreadStateException
```

Q10.Java Program to Check CurrentThread in Multi Threading Concept.

```
public class Test extends Thread

{

   public static void main(String[] args)    {       // getting reference to Main thread

    Thread t = Thread.currentThread();      // getting name of Main thread  System.out.println("Current thread: " + t.getName());

        // changing the name of Main thread

      t.setName("Geeks");

     System.out.println("After name change: " + t.getName());

    // getting priority of Main thread

      System.out.println("Main thread priority: "+ t.getPriority());

      // setting priority of Main thread to MAX(10)

     t.setPriority(MAX_PRIORITY);

        System.out.println("Main thread new priority: "+ t.getPriority());

       for (int i = 0; i < 2; i++){

     System.out.println("Main thread");

   }// Main thread creating a child thread

   ChildThread ct = new ChildThread();

      // getting priority of child thread

     // which will be inherited from Main thread

    // as it is created by Main thread

     System.out.println("Child thread priority: "+ ct.getPriority());

  // setting priority of Main thread to MIN(1)

 ct.setPriority(MIN_PRIORITY);

   System.out.println("Child thread new priority: "+ ct.getPriority());

       // starting child thread
```

```
 ct.start();    }
} // Child Thread class
class ChildThread extends Thread {
public void run()     {
  for (int i = 0; i < 2; i++){
   System.out.println("Child thread");   }    }
}
```

Output:

```
Current thread: main
After name change: Geeks
Main thread priority: 5
Main thread new priority: 10
Main thread
Main thread
Child thread priority: 10
Child thread new priority: 1
Child thread
Child thread
```

Q11.Java Program to Create a Server with 2 Threads to Communicate with Serveral Clients

```
import java.io.*;
import java.text.*;
import java.util.*;
import java.net.*;
// Server class
public class Server  {
    public static void main(String[] args)
    throwsIOException     {
  // server is listening on port 5056
```

```
 ServerSocket ss = new ServerSocket(5056);
// running infinite loop for getting  client request
 while (true)        {
     Socket s = null;
   try  {
 // socket object to receive incoming client requests
s = ss.accept();
 System.out.println("A new client is connected : " + s);  // obtaining input and out streams
   DataInputStream dis = new DataInputStream(s.getInputStream()); DataOutputStream dos = new
DataOutputStream(s.getOutputStream());
  System.out.println("Assigning new thread for this client");   // create a new thread object
Thread t = new ClientHandler(s, dis, dos);
  // Invoking the start() method
    t.start();
       }
        catch (Exception e){
       s.close();
      e.printStackTrace();
   } } }
} // ClientHandler class
class ClientHandler extends Thread  {
   DateFormat fordate = new SimpleDateFormat("yyyy/MM/dd");
  DateFormat fortime = new SimpleDateFormat("hh:mm:ss");
   final DataInputStream dis;
 final DataOutputStream dos;
  final Socket s;
  // Constructor
    public ClientHandler(Socket s, DataInputStream dis, DataOutputStream dos)
     {
```

```
    this.s = s;
     this.dis = dis;
       this.dos = dos;    }
 @Overrid
  public void run()     {
   String received;
        String toreturn;
     while (true)      {
 try {
  // Ask user what he wants
   dos.writeUTF("What do you want?[Date | Time]..\n"+    "Type Exit to terminate connection.");
   // receive the answer from client
  received = dis.readUTF();
     if(received.equals("Exit"))   {
    System.out.println("Client " + this.s + " sends exit...");    System.out.println("Closing this connection.");
   this.s.close();
 System.out.println("Connection closed");
break;  }
 // creating Date object
 Date date = new Date();
  // write on output stream based on t
  // answer from the client
switch (received) {
 case "Date" :
 toreturn = fordate.format(date);  dos.writeUTF(toreturn);
 break;
 case "Time" :
 toreturn = fortime.format(date);  dos.writeUTF(toreturn);
break;
```

```
default:
 dos.writeUTF("Invalid input");
 break;
   } } catch (IOException e) {
     e.printStackTrace();
     }   }   try {  // closing resources
   this.dis.close();
    this.dos.close();
       }catch(IOException e){
   e.printStackTrace();   }
 } }
```

Output

```
A new client is connected : Socket[addr=/127.0.0.1,port=60536,localport=5056]
Assigning new thread for this client
Client Socket[addr=/127.0.0.1,port=60536,localport=5056] sends exit...
Closing this connection.
Connection closed
```

Q12.Java Program to Create a Client That Receive Message From the Server

```
 import java.io.*;
import java.net.*;
import java.util.Scanner;
public class Client  {
final static int ServerPort = 1234;
```

```
public static void main(String args[]) throws UnknownHostException, IOException    {
Scanner scn = new Scanner(System.in);
  // getting localhost ip
InetAddress ip = InetAddress.getByName("localhost");
    // establish the connection
    Socket s = new Socket(ip, ServerPort);
    // obtaining input and out streams
    DataInputStream dis = new DataInputStream(s.getInputStream()); DataOutputStream dos = new
DataOutputStream(s.getOutputStream());
    // sendMessage thread
      Thread sendMessage = new Thread(new Runnable()
      {
     @Override
    public void run() {
     while (true) { // read the message to deliver.
     String msg = scn.nextLine();
       try { // write on the output stream
dos.writeUTF(msg);
    } catch (IOException e) {
  e.printStackTrace();
  }   }  }     });


            // readMessage thread
  Thread readMessage = new Thread(new Runnable()
   {
    @Override
public void run() {
 while (true) {
 try {
```

```
 // read the message sent to this client
 String msg = dis.readUTF();
 System.out.println(msg);
 } catch (IOException e) {
  e.printStackTrace();
}    }  }  });
sendMessage.start();
  readMessage.start();
   }}
```

Output :

From client 0:

Hello # client1

Client 1:hii

How are you #client 1

Client 1:fine…how are you?

Q13.Java Program of a Multithreaded Implementation of Any Parallelized Divide-Conquer Algorithm

```
importjava.io.IOException;
                                import java.util.Arrays;
                                import java.util.Random;
                                import java.util.Scanner;


                                public class a8q13{
                                    public static int[] inputArray;
```

```
    public static int[] arr1;
    public static int[] arr2;
    public static int[] arr3;
    public static int t1_status=0;
    public static int t2_status=0;

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

        Scanner in =new Scanner(System.in);

        int arraySize=5;

        inputArray = new int[arraySize];

        Random rand=new Random();

        for(int i=0;i<arraySize;i++)
        {
            inputArray[i]=rand.nextInt(100);
        }

        arr1=Arrays.copyOfRange(inputArray, 0,
inputArray.length/2);

        arr2=Arrays.copyOfRange(inputArray,
(inputArray.length)/2,inputArray.length);

        System.out.print("The original array is array is ");
```

```
for(int h:inputArray)
{
    System.out.println(h);
}

Thread t1=new Thread(new Runnable(){
        public void run()
        {
            mergeSort(arr1);
            System.out.println("t1 started");
        }

    });
Thread t2=new Thread(new Runnable(){
        public void run()
        {
            mergeSort(arr2);
            System.out.println("t2 started");
        }

    });

t1.start();
t2.start();

try {
    t1.join();
```

```
            t2.join();
        }
        catch (InterruptedException e) {

            e.printStackTrace();
        }
        if(t1.isAlive())
        {
            t1_status=1;
 }
        if(t2.isAlive())
        {
            t2_status=1;
        }
        t1.stop();
        t2.stop();

        arr3=new int[inputArray.length];

        merge(arr3,arr1,arr2);

        System.out.println("The sorted array using divide and
conquer algorithm = ");
        for(int m:arr3)
{
            System.out.print(m);
            System.out.print(" ");
        }
```

```
        System.out.println(" ");
    }

    static void mergeSort(int[] A)
    {
        if (A.length > 1)
        {
            int q = A.length/2;
 // divide the array in half
            int[] leftArray = Arrays.copyOfRange(A, 0, q);
            int[] rightArray =
Arrays.copyOfRange(A,q,A.length);
// sort/conquer each half
            mergeSort(leftArray);
            mergeSort(rightArray);
            merge(A,leftArray,rightArray);
 }
    }


    static void merge(int[] a, int[] l, int[] r) {
        int totElem = l.length + r.length;

        int i,li,ri;
        i = li = ri = 0;
        while ( i < totElem) {
            if ((li < l.length) && (ri<r.length)) {
                if (l[li] < r[ri]) {
```

```
                a[i] = l[li];
                i++;
                li++;
            }
            else {
                a[i] = r[ri];
                i++;
                ri++;
            }
        }
        else {
            if (li >= l.length) {
                while (ri < r.length) {
                    a[i] = r[ri];
i++;
                    ri++;
                }
            }
            if (ri >= r.length) {
                while (li < l.length) {
                    a[i] = l[li];
                    li++;
                    i++;
                }
            }
        }
    }
```

```
            if(t1_status==1){arr1=a;}
            else if(t2_status==1){arr2=a;}
            else{arr3=a;}
        }
    }
```