	UNIT 5
1	Explain throw and finally in Exception Handling of java language.
2	Write a java program to write a text file in java.
3	Write a Java Program to develop user defined exception.
4	Explain Synchronization in Thread.
5	Compare checked and Unchecked Exception.
6	Define Thread in Java.
7	Explain 'try' and 'catch' statements in exceptional handling of Java language
8	Explain 'Extending Thread class' in Java.
9	Explain life cycle of thread in Java.
10	List types of Errors in exceptional handling and explain any one of them.
11	Write a Java program that executes three threads. One thread displays "Thread – I" every 2500 millisecond, second thread displays "Thread – II" every 5000 millisecond and third thread displays "Thread – III" every 7500 millisecond.
12	List types of error in Java
13	List any four inbuilt exceptions.
14	Write a program in Java to demonstrate use of synchronization of threads when multiple threads are trying to update common variable.
15	Write a program in Java for Banking Application in which user deposits the amount Rs 2000/- and then start withdrawing of Rs 1000/-, Rs 500/- and it throws exception "Not Sufficient Fund" when user withdraws Rs. 750/- thereafter.
16	Explain Thread life cycle.
17	Explain different methods used to create thread with example
18	Write a program in Java to demonstrate multiple try block and multiple catch exception.
19	List four different in-built exception class in Java.
20	List different thread priorities.
21	Explain the throw and throws Keyword.
22	Explain the use of synchronized keyword in Java.
23	Explain the types of errors in Java.
24	Differentiate between Single Threaded Program and Multi Threaded Program
25	List any four inbuilt exceptions.
26	List different thread priorities
27	List and explain different types of errors.
28	Explain basic concept of Exception Handling.
29	Write a Java program that executes two threads. One thread will print the numbers divisible by 3 and another thread print numbers divisible by 5 between 1 to 50.
30	Write a Java program to handle user defined exception for 'Divide by Zero' error.
31	What is Thread? List different methods used to create Thread. Explain Thread life cycle.
32	Write a program in Java to demonstrate use of synchronization of threads when multiple threads are trying to update common variable.

Java Unit 5

Q1) Explain basic concept of Exception Handling.

An exception (or exceptional event) is a problem that arises during the execution of a program. When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled. An exception can occur for many different reasons

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Exception handeling is done by following keyword

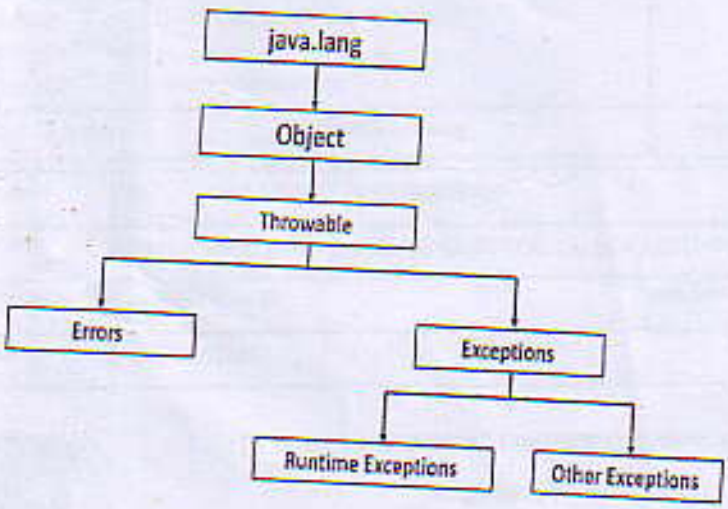
- Try
- Catch
- Throw
- Throws
- Finally

Q2) Explain types of java exceptions

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception(Runtime Exception)
3. Error

Hierarchy of Java Exception classes



Q3) List and explain different types of errors.

There are two types of errors: exception class and error class

Examples of error in java

- Stack overflow
- Virtual machine error
- Out of memory error

List any four inbuilt exceptions.

There are two types of exceptions

Checked and unchecked

inBuilt unchecked exception	inBuilt checked exception
1) **ArithmeticException** Arithmetic error, such as divide-by-zero 2) **ArrayIndexOutOfBoundsException** Array index is out-of-bounds. 3) **ArrayStoreException** Assignment to an array element of an incompatible type. 4) **IndexOutOfBoundsException** Some type of index is out-of-bounds.	1) **ClassNotFoundException** Class not found. 2) **CloneNotSupportedException** Attempt to clone an object that does not implement the Cloneable interface. 3) **IllegalAccessException** Access to a class is denied. 4) **InstantiationException** Attempt to create an object of an abstract class or interface. 5) **InterruptedException** One thread has been interrupted by another thread.

Q4) List types of Errors in exceptional handling and explain any one of them.

Or

List and explain different types of errors.

Examples of error in java

- Stack overflow
- Virtual machine error
- Out of memory error

StackOverflowError

```
class Test {
public static void main(String[] args)
    {
        m1();
    }
public static void m1()
    {
        m2();
    }
public static void m2()
    {
        m1();
    }
}
```

Output:

```
Exception in thread "main" java.lang.StackOverflowError
```

Q5) Explain 'try' and 'catch' statements in exceptional handling of Java language

Java **try** block is used to enclose the code that might throw an exception. It must be used within the method.

If an exception occurs at the particular statement of try block, the rest of the block code will not execute. So, it is recommended not to keeping the code in try block that will not throw an exception.

Java try block must be followed by either catch or finally block.

Syntax of Java try-catch

```
try{
//code that may throw an exception
}catch(Exception_class_Name ref){}
```

Java catch block

Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception (i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.

The catch block must be used after the try block only. You can use multiple catch block with a single try block.

Example of try and catch block

```
public class TryCatchExample2 {
    public static void main(String[] args) {
     try
     {
     int data=50/0; //may throw exception
     }
       //handling the exception
     catch(ArithmeticException e)
     {
       System.out.println(e);
             }
             System.out.println("rest of the code");
          } }
```

Output:

```
java.lang.ArithmeticException:
/ by zero

rest of the code
```

Q6) Write a program in Java to demonstrate multiple try block and multiple catch exception.

```
public class MultipleCatchBlock1 {

    public static void main(String[] args) {

        try{
            int a[]=new int[5];
            a[5]=30/0;
           }
           catch(ArithmeticException e)
            {
             System.out.println("Arithmetic Exception occurs");
            }
          catch(ArrayIndexOutOfBoundsException e)
            {
             System.out.println("ArrayIndexOutOfBounds Exception occurs");
            }
          catch(Exception e)
            {
             System.out.println("Parent Exception occurs");
            }
          System.out.println("rest of the code");
    }
}
```

Output

```
Arithmetic Exception occurs
rest of the code
```

Q7) Explain nested try{} block

```
public class Nested {
    public static void main(String[] args) {
        try {
            System.out.println("Outer try block starts");
            try {
                System.out.println("Inner try block starts");
                int res = 5 / 0;
            } catch (InputMismatchException e) {
                System.out.println("InputMismatchException caught");
            } finally {
                System.out.println("Inner final");
            }
        } catch (ArithmeticException e) {
            System.out.println("ArithmeticException caught");
        } finally {
            System.out.println("Outer finally");
        }
    }
}
```

The output is

```
Outer try block starts
Inner try block starts
Inner final
ArithmeticException caught
Outer finally Outer finally
```

Q8) Explain finally keyword

- A **finally** keyword is used to create a block of code that **follows a try or catch** block.
- A **finally block** of code always executes **whether or not exception has occurred.**
- A finally block appears at the **end of catch** block.

Syntax:

```
try
    {
        //Protected code
    }
    catch(ExceptionType1 e1)
    {
        //Catch block 1

    }
    catch(ExceptionType2 e2)
    {

    }
    finally
    {

    }
```

Example

```
class demoFinally
{
    public static void main(String args[])
    {
        int a[] = new
        int[2]; try
        {
            System.out.println("Access element three :" + a[3]);
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
System.out.println("Exception thrown :" + e);
        }
        finally
        {
a[0] = 10;
System.out.println("First element value: " +a[0]); System.out.println("The finally
block is always executed");}
System.out.println("out of try catch");}}
```

OUTPUT

```
Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 10
        The finally block is always executed
        out of try catch
```

Q9) Compare checked and Unchecked Exception.

Or

Explain checked and unchecked exception

Ans :

Checked Exception	Unchecked Exception
The Compiler checks the checked exception.	The Compiler does not checks the Unchecked exception.
Except "RuntimeException" class all the child classes of the class "Exception", and the "Error" class and its child classes are Checked Exception.	"RuntimeException"class and its child classes, are unchecked Exceptions.
If we do not handle the checked exception, then the compiler objects.	Even if we do not handle the unchecked exception, the compiler doesn't object.
The Program doesn't Compile if there is an unhandled chaecked exception in the program code.	The program compiles successfully even if there is and unhandled unchecked exception in the program code.
Examples of checked exceptions are IOException, ClassNotFound, Data Access Exception etc.	Examples of unchecked exception is Runtime Exceptions or errors.
This Exception is checked by Compiler.	This Exception are not checked by Compiler.

Q10) Explain the throw and throws Keyword.

Throw keyword

The Java throw keyword is used to explicitly throw an exception. We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

Syntax: `throw exception;`

Example:

```
public class TestThrow1{
  static void validate(int age){
   if(age<18)
    throw new ArithmeticException("not valid");
   else
    System.out.println("welcome to vote");
  }
  public static void main(String args[]){
    validate(13);
    System.out.println("rest of the code...");
  }
}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:not valid
```

Q11) Write difference between throw and throws

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

Q12) Explain throw and finally in Exception Handling of java language.

Throw

The Java throw keyword is used to explicitly throw an exception. We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

Example:

```
public class TestThrow1{
  static void validate(int age){
   if(age<18)
    throw new ArithmeticExceptio
n("not valid");
   else
    System.out.println("welcome t
o vote");
  }
  public static void main(String a
rgs[]){
   validate(13);
   System.out.println("rest of the
code...");
  }
}
```

Output:

```
Exception in thread main
java.lang.ArithmeticException:not
valid
```

Finally

Finally is used to place important code, it will be executed whether exception is handled or not.

Example

```
class FinallyExample{
public static void main(String[] args){

try{
int x=300;
}catch(Exception e){System.out.println
(e);}
finally{System.out.println("finally block
is executed");}
}}
```

Throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Syntax:

```
return_type method_name() throws exception_class_name{
//method code
}
```

Example:

```
import java.io.IOException;
class Testthrows1{
 void m()throws IOException{
  throw new IOException("device error");//checked exception
 }
 void n()throws IOException{
  m();
 }
 void p(){
 try{
  n();
 }catch(Exception e){System.out.println("exception handled");}
 }
 public static void main(String args[]){
 Testthrows1 obj=new Testthrows1();
 obj.p();
 System.out.println("normal flow...");
 }
}
```

Output: Runtime Exception

Q13) Explain Difference between final, finally and finalize

No.	final	finally	finalize
1	Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.	Finallze is used to perform clean up processing just before object is garbage collected.
2	Final is a keyword.	Finally is a block.	Finalize is a method.

3	**class** FinalExample{
	public static void main(String[] args){
	final int x=100;
	x=200;//Compile Time Error
	}}
		class FinallyExample{
		public static void main(String[] args){
		try{
		int x=300;
		}**catch**(Exception e){System.out.println(e);}
		finally{System.out.println("finally block is executed");}
		}}
			class FinalizeExample{
			public void finalize(){System.out.println("finalize called");}
			public static void main(String[] args){
			FinalizeExample f1=**new** FinalizeExample();
			FinalizeExample f2=**new** FinalizeExample();
			f1=**null**;
			f2=**null**;
			System.gc();
			}}

Q14) Write a Java Program to develop user defined exception.

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

Example 1

```
class InvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);
 }
}
```

Example 2

```
class demoUserException extends Exception
{
        private int ex;
        demoUserExcep
        tion(int a)
        {
                ex=a;
        }
        public String toString()
        {
                return "MyException[" + ex +"] is less than zero";
        }}
class demoException
{
        static void sum(int a,int b) throws demoUserException
        {
                if(a<0)
                {
                }
                else
                {
                }}
        public static void main(String[] args)
        {
                try
```

```
                    {
                    }
                    catch(demoUserException e)
        {System.out.println(e);}}} Output: MyException[-10] is less than zero
```

Q15) Write a Java program to handle user defined exception for 'Divide by Zero' error.

Or

Arithmetic exception

```
class ArithmeticException_Demo {
public static void main(String args[])
    {
        try {
            int a = 30, b = 0;
            int c = a / b; // cannot divide by zero
            System.out.println("Result = " + c);
        }
        catch (ArithmeticException e) {
            System.out.println("Can't divide a number by 0");
        }
    }
}
```

Output: Can't divide a number by 0

Q16) Write a program to explain ArrayIndexOutOfBounds Exception

```
class ArrayIndexOutOfBound_Demo {
public static void main(String args[])
    {
        try {
            int a[] = new int[5];
            a[6] = 9; // accessing 7th element in an array of
            // size 5
        }
        catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array Index is Out Of Bounds");
        }
    }
}
```

Output: Array Index is Out Of Bounds

Q17) Example of ClassNotFoundException :

This Exception is raised when we try to access a class whose definition is not found.

```
class Bishal {

} class Geeks {

} class MyClass {
public static void main(String[] args)
    {
        Object o = class.forName(args[0]).newInstance();
        System.out.println("Class created for" + o.getClass().getName());
    }
}
```

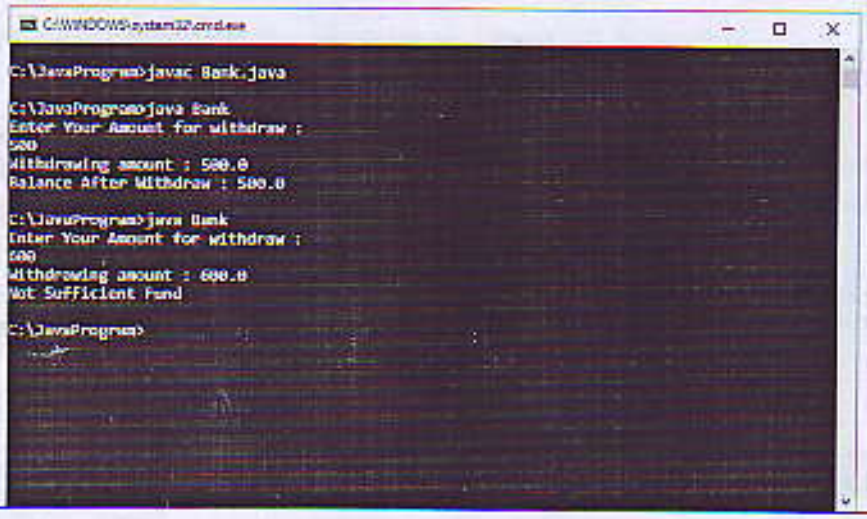
Output: ClassNotFoundException

Q18) Write an small application in Java to develop Banking Application in which user deposits the amount Rs 1000.00 and then start withdrawing of Rs 400.00, Rs 300.00 and it throws exception "Not Sufficient Fund" when user withdraws Rs. 500 thereafter.

```
import java.util.*;
class Bank
{
  float fund;
  void deposit(float amount)
  {
    fund=amount;
  }
  void withdraw(float money) throws Exception
  {
    float newFund=fund-money;
    if(newFund<500)
    {
      throw new Exception("Not Sufficient Fund");
    }
    else
    {
      fund=newFund;
      System.out.println("Balance After Withdraw : "+fund);
    }}
    public static void main(String arg[])
    {
      Bank b=new Bank();
      b.deposit(1000.00f);
      try
      {
        float money;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Your Amount for withdraw : ");
        money=sc.nextInt();
        System.out.println("Withdrawing amount : "+money);
        b.withdraw(money);
        /* here test with static data so don't worry
        money=300;
```

```
            System.out.println("Withdrawing amount : "+money);
            b.withdraw(money); */
        }
        catch(Exception e)
        {
            System.out.println(e.getMessage());}}}}
```

Output:



Q19) Define Thread in Java.

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

As shown in the figure, a thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS, and one process can have multiple threads.

Advantages of Java Multithreading

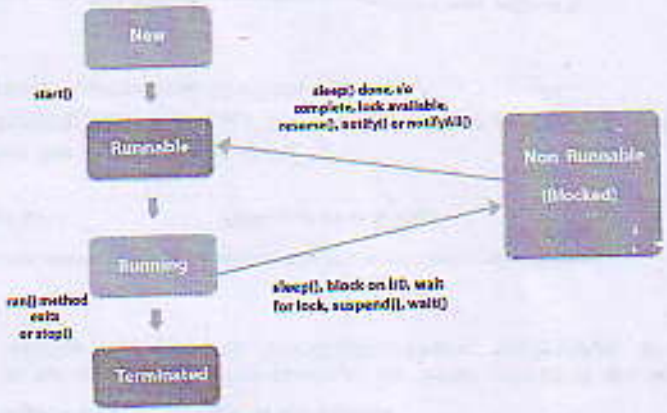
1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time.**

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

Q20) Explain life cycle of thread in Java.

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated



1) New

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

2) Runnable

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3) Running

The thread is in running state if the thread scheduler has selected it.

4) Non-Runnable (Blocked)

This is the state when the thread is still alive, but is currently not eligible to run.

5) Terminated

A thread is in terminated or dead state when its run() method exits.

Q21) How to create Thread?

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Explain 'Extending Thread class' in Java.

- Second way to create a thread is to **create a new class** that extends **Thread** class and then create an instance of that class.
- The extending class must override the **run( )** method, which is the entry point for the new thread.
- Once **Thread object** is created, you can start it by calling **start( )** method, which executes a call to **run( )** method.

Example:

```
class demoThread2 extends Thread
{
        public void run()
        {
                System.out.println("Thread is running...");
        }
        public static void main(String args[])
        {
                demoThread2  t1 = new demoThread2();
                t1.start();
        }
}
```

Output:

```
Thread is running...
```

How to create thread using runnable interface?

- The easiest way to create a thread is to create a class that implements the runnable interface.
- After implementing runnable interface, the class needs to implement the run() method, which has following form:

  public void run( )
- This method provides entry point for the thread and you will put you complete business logic inside this method.
- After that, you will instantiate a Thread object using the following constructor:

  Thread (Runnable threadObj, String threadName);
- Where, threadObj is an instance of a class that implements the Runnable interface and threadName is the name given to the new thread.
- Once Thread object is created, you can start it by calling start( ) method, which executes a call to run( ) method.

  void start ( );

Example:

```
class demoThread3 implements Runnable
{
        public void run()
        {
                System.out.println("Thread is running...");
        }
        public static void main(String args[])
        {
                demoThread3 d1=new demoThread3 ();
                Thread t1 =new Thread(d1);
                t1.start();
        }
}
```

Output:

```
Thread is running...
```

Q22) Explain Synchronization in Thread.

Thread Synchronization

- When two or more threads need access to a **shared** resource, they need some way to ensure that the resource will be used by **only one thread at a time.**
- The process by which this synchronization achieved is called **thread synchronization.**
- The **synchronized** keyword in Java creates a block of code referred to as **a critical section.**
- Every Java object with a critical section of code gets a **lock associated with the object.**
- To enter a **critical section**, a thread needs to obtain the corresponding object's lock.

Syntax:

```
synchronized(object)
{
        // statements to be synchronized
}
```

  - Here, **object** is a reference to the object being **synchronized.**
  - A synchronized block ensures that a call to a method that is a member of **object** occurs only after the current thread has successfully entered object's critical section.

Example

```
class Table{
 synchronized void printTable(int n){//synchronized method
  for(int i=1;i<=5;i++){
   System.out.println(n*i);
   try{
    Thread.sleep(400);
   }catch(Exception e){System.out.println(e);}
  }

 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
```

```
    }
    public void run(){
    t.printTable(5);
    }

    }
    class MyThread2 extends Thread{
    Table t;
    MyThread2(Table t){
    this.t=t;
    }
    public void run(){
    t.printTable(100);
    }
    }

    public class TestSynchronization2{
    public static void main(String args[]){
    Table obj = new Table();//only one object
    MyThread1 t1=new MyThread1(obj);
    MyThread2 t2=new MyThread2(obj);
    t1.start();
    t2.start();
    }
    }
```

```
Output: 5
        10
        15
        20
        25
        100
        200
        300
        400
        500
```

Q22) What is Inter-thread communication?

- **Inter-thread communication or Co-operation** is all about allowing **synchronized threads** to communicate with each other.
- Inter-thread communication is a mechanism in which a **thread is paused running in its critical section** and another thread is allowed to enter (or lock) in the same critical section to be executed.
- To avoid **polling**(It is usually implemented by loop), **Inter-thread communication** is implemented by following methods of **Object class**:
  - **wait( ):** This method tells the calling thread to give up the **critical section** and go to **sleep** until some other thread enters the same **critical section** and calls notify( ).
  - **notify( ):** This method **wakes up** the first thread that called **wait( )** on the **same object.**
  - **notifyAll( ):** This method **wakes up** all the threads that called **wait( )** on the **same object.** The highest priority thread will run first.
- Above all methods are implemented as **final** in Object class.
- All three methods can be called only from **within a synchronized context.**

Q23) Write a program in Java to demonstrate use of synchronization of threads when multiple threads are trying to update common variable.

```
class Table{
 synchronized void printTable(int n){//synchronized method
  for(int i=1;i<=5;i++){
   System.out.println(n*i);
   try{
    Thread.sleep(400);
   }catch(Exception e){System.out.println(e);}
  }

 }
}
 class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
  public class TestSynchronization2{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
```

OUTPUT

```
5
10
15
20
25
100
200
300
400
```

```
        MyThread2 t2=new MyThread2(obj);
        t1.start();
        t2.start();  }}
```

Q24) Explain thread priorities

Thread Priorities

- Every Java thread has a **priority** that helps the operating system determine the **order in which threads are scheduled.**
- Java priorities are in the range between **MIN_PRIORITY** (a constant of 1) and **MAX_PRIORITY** (a constant of 10).
- By default, every thread is given priority **NORM_PRIORITY** (a constant of 5).
- Threads with **higher** priority are more important to a program and should be allocated **processor time** before **lower**-priority threads.
- The thread scheduler mainly uses **preemptive** or **time slicing** scheduling to schedule the threads.

Q25) List methods of thread class

Method	Description
public void **run**()	Entry point for a thread
public void **start**()	Start a thread by calling run() method.
public String **getName**()	Return thread's name.
public void **setName**(String name)	To give thread a name.
public int getPriority()	Return thread's priority.
public int setPriority(int priority)	Sets the priority of this Thread object. The possible values are between 1 and 10.
public final boolean isAlive()	Checks whether thread is still running or not.
public static void sleep(long millisec)	Suspend thread for a specified time.
public final void join(long millisec)	Wait for a thread to end.

Q26) Write a program that executes two threads. One thread displays "Thread1" every 2,500 milliseconds, and the other displays "Thread2" every 4,000 milliseconds. Create the threads by extending the Thread class.

```
class ThreadExample extends Thread
{
    ThreadExample(String s)
    {
        super(s);
        start();
    }
    public void run()
    {
        for(int i=0;i<5;i++)
        {
            System.out.println(Thread.currentThread().getName());
            try
            {
                if(Thread.currentThread().getName()=="Thread1")
                {
                    Thread.sleep(2000);
                }
                else
                {
                    Thread.sleep(4000);
                }
else
{
Thread.sleep(7500);
}

            }
            catch(Exception e){}
        }
    }
}
class TwoThread
{
    public static void main(String arg[])
    {
        System.out.println("Thread name : "+Thread.currentThread().getName());
        ThreadExample e1=new ThreadExample("Thread1");
        ThreadExample e2=new ThreadExample("Thread2");
```

```
    }
}
```

OUTPUT

```
C:\JavaProgram>javac TwoThread.java

C:\JavaProgram>java TwoThread
Thread name : main
Thread2
Thread1
Thread1
Thread1
Thread2
Thread1
Thread1
Thread2
Thread2
Thread2
```

Q27) Write a Java program that executes two threads. One thread will print the numbers divisible by 3 and another thread print numbers divisible by 5 between 1 to 50.

```
public class Exercise50 {
	public static void main(String args[]) {
		System.out.println("\nDivided by 3: ");
		for (int i=1; i<150; i++) {
			if (i%3==0)
			System.out.print(i +", ");
		}

		System.out.println("\n\nDivided by 5: ");
		for (int i=1; i<50; i++) {
			if (i%5==0) System.out.print(i +", ");
		}

		System.out.println("\n\nDivided by 3 & 5: ");
		for (int i=1; i<50; i++) {
			if (i%3==0 && i%5==0) System.out.print(i +", ");

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

class Thread1 extends Thread

{

int i;

{

Public void run()

{ system.out.Println("divisable

for (int i=1; i<50; i++)

{ if (i%3==0){

System.out.Println(i);

}

}

}

```
class Thread2 extends Thread
{
  int i;
  Public void run()
  {
System.out.Println("divisible by 5");
  for(i=1;i<50;i++)
  {
  if(i%5==0);
  {
System.out.Println(i);
  }
  }
}
class main
{
Public static void main(String args[])
{
Thread1 t1 = new Thread1();
t1.run();
Thread2 t2 = new Thread2();
t2.run();
}
}
```