05. Exception Handling, Thread, Networking

Introduction:

While doing programming in Java, there is possibility to happen errors. These errors may be encountered at two times-

1) Compile time error
2) Runtime error

Let's see these types of errors in details-

1) Compile time error:

- These type of errors detected by compiler at compile time of program.
- Such type of errors are due to wrong syntax of programming languages.
- Or when programmer violate the rules of writing Java statements.
- These compiler error indicates something that must be fixed before the code can be compiled.
- All these errors are detected by compiler and thus are known as "compile-time errors"
- Most frequent compile time errors happen if we make following mistake:
  - Missing Parenthesis (})
  - Use of variable without declaring it.
  - Missing semicolon at end of statement.
- Note that such type of errors are handled by compiler and it generate corresponding error message such that we can correct it.
- These type of errors are not harmful for computer because it is easily detected by compiler at compile time.

2) Run time error (Exception):

- Exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time and that disturbs the normal flow of the program's instructions.
- An exception is a problem or error that arises during the execution of a program (program running).
- Exceptions are run-time abnormal conditions that a program encounters during its execution.
- An exception is a response to an exceptional situation that arises while a program is running, such as an attempt to divide a number by zero.
- When an Exception occurs the normal flow of the program is disturbed and the program/Application/software terminates abnormally that cause's application failure which is harmful, which is not recommended, therefore, these exceptions should to be handled.
- An exception can occur for many different reasons. Following are some scenarios where an exception occurs.
  - A user has entered an invalid data.
  - Attempt to open a file that cannot be found.
  - A network connection has been lost in the middle of communications.

What happen if we do not handle exceptions?

- When an exception occurred, if we don't handle it, the program terminates abnormally and remaining program code will not get executed that cause's application failure which is harmful.

Example-

- We know that, an array is of fixed size and each element is accessed using the indices. Consider, we have created an array with size 7. Then the valid expressions to access the elements of this array will be a[0], a[1], .. to a[6] (i.e. length-1).
- Whenever, we used a negative index value or, the index value greater than the size of the array, then there will be exception occurs.

Types of Exceptions:

Java has two types of exceptions.

1. Checked exception
2. Unchecked exception

Let's see these exception in details-

1) Checked Exceptions:

- The exception which occur at compile time of program is called "Checked Exception".
- Java compiler checks program contains the checked exception or not at the time of compilation.
- All these exceptions are subclass of Exception class.
- Developer has overall control on checked exception because these occur at compile time.
- For example: SQLException, IOException, ClassNotFoundException, MalformedURLException etc.

Example: Sample program for checked exception.

```
import   java.io.*;
public   class   CheckExpDemo
{
   public  static  void   main(String args[])
   {
                FileInputStream    in = new  FileInputStream("read.txt");
                int    c;
                while((c=in.read())!=-1)
                {
                      System.out.println((char)c);
                }
                in.close( );
   }
}
```

Above program generates compile time exception like-

error: unreported exception IOException

Because IOException must be caught or declared to be thrown.

Here, IOException occur at compile time therefore it is checked exception.

2. Unchecked Exceptions:

- The exception which occur at run time of program is called "Unchecked Exception".
- JVM detects for unchecked exception at runtime of program.
- Such exception occur due to logical errors or improper use of API.
- Developer has no control on unchecked exception because these occur at run time of program.

For example: ArrayIndexOutOfBoundException, NullPointerException, ArithmeticException.

Example : Sample program for unchecked exception

```
class     UnCheckExpDemo
{
    public  static  void main(String args[])
    {
        int  a = 10;
        int  b = 0;
        int  c = a/b;
        System.out.println(c);
    }
}
```

Above program raise run time exception like-

Exception in thread "main" java.lang.ArithmeticException: / by zero

Difference between error and exception:

Error	Exception
It cannot be caught.	It can be handled by try and catch block.
Errors are by default unchecked type.	Exception can be either checked or unchecked type.
It is defined in *java.lang.Error* package.	It is defined in *java.lang.Exception* package.
Errors are generally caused by environment in which application is running.	Exceptions are generally caused by application itself.
Example: StackOverflowError, OutOfMemoryError etc.	**Example:** SQLException, IOException, ClassCastException, ArrayOutOfBoundException

Built-in exceptions:

- Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

1. ArithmeticException

It is thrown when an exceptional condition has occurred in an arithmetic operation.

2. ArrayIndexOutOfBoundsException

It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

3. ClassNotFoundException
This Exception is raised when we try to access a class whose definition is not found
4. FileNotFoundException
This Exception is raised when a file is not accessible or does not open.
5. IOException
It is thrown when an input-output operation failed or interrupted
6. InterruptedException
It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.
7. NoSuchFieldException
It is thrown when a class does not contain the field (or variable) specified.
8. NoSuchMethodException
It is thrown when accessing a method which is not found.
9. NullPointerException
This exception is raised when referring to the members of a null object. Null represents nothing
10. NumberFormatException
This exception is raised when a method could not convert a string into a numeric format.
11. RuntimeException
This represents any exception which occurs during runtime.
12. StringIndexOutOfBoundsException
It is thrown by String class methods to indicate that an index is either negative than the size of the string
Examples of Built-in Exception:

```
// Java program to demonstrate ArithmeticException
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

```
//Java program to demonstrate NullPointerException
class     NullPointer_Demo
{
public static void main(String args[])
{
  try
  {
       String a = null; //null value
       System.out.println(a.charAt(0));
  }
 catch(NullPointerException    e)
 {
      System.out.println("NullPointerException !!");
  }
}
}
```

Output:
NullPointerException !!

```
//Java program to demonstrate StringIndexOutOfBoundsException
class        StringIndexOutOfBoundDemo
{
  public static void main(String args[])
  {
     try
    {
         String a = "Box is Heavy ";   // length is 12
         char c = a.charAt(20);      // 20th index element
         System.out.println(c);
     }
     catch(StringIndexOutOfBoundsException    e)
    {
System.out.println("StringIndexOutOfBoundsException");
    }
  }
}
```

Output: StringIndexOutOfBoundsException

```
//Java program to demonstrate NumberFormatException
class      NumberFormat_Demo
{
public static void main(String args[])
{
try
{
     int   n= Integer.parseInt ("GOOD") ;
     System.out.println(n);
  }
catch(NumberFormatException     e)
 {
   System.out.println("Number format exception");
}
}
}
```

Output:
Number format exception

```
// Java program to demonstrate ArrayIndexOutOfBoundException
class ArrayIndexOutOfBound_Demo
{
public static void main(String args[])
{
  try
{
     int a[] = new  int[5];
     a[6] = 9;   // assigning 9 element at 6th index but array size is 5
 }
catch(ArrayIndexOutOfBoundsException     e)
{
   System.out.println ("Array Index is Out Of Bounds");
}
}
}
```

Output:
Array Index is Out Of Bounds

```
// Java program to demonstrate FileNotFoundException
import     java.io.*;
class      FileNotFoundException _Demo
{
public static void main(String args[])
{
  try
{
    FileReader   in=new FileReader("Read.txt");
}
catch(FileNotFoundException      e)
{
   System.out.println("File Not found..");
}
}
}
```

Output:
File Not found..

Exception Handling in Java:

- Exception handling is the mechanism to handle the abnormal termination of the program without failure.
- By handling exception we can protect application from failure.
- In Java, there are different kinds of exceptions which are handled by **Exception** class which was derived from **Throwable** class.
- The Exception class Hierarchy in Java is shown as-

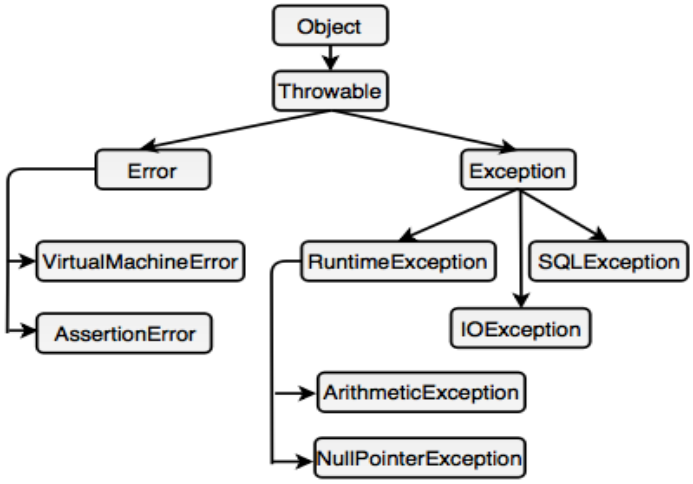

Fig: Exception Hierarchy

- To protect application it is necessary to handle the exception.
- In java, exceptions are handled using five keywords which are as follow-
  1) **try**
  2) **catch**
  3) **throw**
  4) **throws**
  5) **finally**

Let's see these keywords in details-

1) The try block:

- The try block contains the code or statements that might be possibility to occur an exception.
- The try block must have least one catch block or finally block.

2) The catch block:

- A catch block must be declared after try block. It contains the error handling code.
- A catch block executes if an exception occurs in corresponding try block.

- Syntax of try and catch:

```
try
{
    //code that cause exception;
}
catch(Exception_type      e)
{
    //exception handling code
}
```

Example: Following example shows use of try and catch block in java.

Program 1)

```
class     TryCatchDemo
{
      public  static  void  main(String args[])
      {
          int    a  = 30, b = 0;
          int    res;
          try
          {
              res = a / b;
          }
          catch(ArithmeticException        ae)
          {
              System.out.println("Divided by zero: "+ae);
          }
      }
}
```

Output :

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

Program 2)

```
class     ExceptionDemo
{
  public static void main(String args[])
  {
      try
    {
    int    arr[] = new int[5];
    arr[2] = 5;
    System.out.println("Access element two: " + arr[2]);
    arr[7] = 10;    //invalid
   System.out.println("Access element seven: "+ arr[7]);
    }
  catch(ArrayIndexOutOfBoundsException     e)
  {
        System.out.println("Exception thrown:" + e);
   }
  }
}
```

Output :

Access element two: 5
Exception thrown:
java.lang.ArrayIndexOutOfBoundsException: 7

Multiple catch blocks:

- In a single program, there is a possibility to occur multiple exceptions of different types. Then to handle such multiple exceptions, we requires multiple catch blocks.
- A single try block has multiple catch blocks to handle different types of exceptions.
- Syntax:

```
try
{
    // code which generate exception
}
catch(Exception_type1      e1)
{
    //exception handling code
}
catch(Exception_type2       e2)
{
    //exception handling code
}
```

Example: Following program illustrating multiple catch blocks that handles different exception.

```
import   java.util.*;
public  class     MultiCat
{
    public static void main(String args[])
    {
                    Scanner   sc=new Scanner(System.in);
                    int     a,b,c;
          try
          {
            System.out.println("Enter Two NO=");
            a = sc.nextInt();
            b = sc.nextInt();
            c = a / b;
           System.out.println("Result = "+c);
          }
          catch(ArithmeticException    e)
          {
               System.out.println("Number cannot divide by zero.");
          }
          catch(InputMismatchException    e)
          {
                System.out.println("Enter only numeric value.");
          }
    }
}
```

Output:

Enter Two NO= 6 2 Result = 3	Enter Two NO= 7 0 Number cannot divide by zero.	Enter Two NO= 5 k Enter only numeric value.

3) The finally block:

- The code present in finally block will always be executed even if try block generates some exception.
- Finally block must be followed by try or catch block.
- It is used to execute some important code like closing file, closing database connection, taking back up etc.
- Finally block executes after try and catch block.
- Syntax:

```
try
{
    // code
}
catch(Exception_type1   ex)
{
    // catch block1
}
catch(Exception_type2    ex)
{
    //catch block 2
}
finally
{
    //finally block
    //always execute
}
```

Example: A program to implement finally block

```
class    FinallyTest
{
    public   static  void   main(String args[])
    {
        int arr[] = new   int[5];
```

```
        try
        {
            arr[7]=10;
        }
        catch(ArrayIndexOutOfBoundsException        ai)
        {
            System.out.println("Exception thrown : " + ai);
        }
        finally
        {
            System.out.println("The finally statement is executed");
        }
    }
}
```

Output:

Exception thrown : java.lang.ArrayIndexOutOfBoundsException: 7
The finally statement is executed

4) 'throw' keyword in Java:

- The **throw** keyword in Java is used to explicitly or forcefully throw the exception.
- That is with the help of **throw** any kind of exception like ArithmeticException, FileNotFoundException etc. can be explicitly thrown.
- It can be used for both checked and unchecked exception.

Syntax:

throw new Exception_subclass;

Here,

throw is keyword that throws exception.
new is operator
Exception_subclass is any exception class.

Example: A program to illustrate throw keyword in java

```
public  class  throwDemo
{
   public static void main(String args[])
   {
        try
        {
            throw  new  ArithmeticException("Not valid ");
        }
        catch(ArithmeticException  e)
        {
            System.out.println("ArithmeticException occur "+e);
        }
   }
}
```

OUTPUT:

ArithmeticException occur java.lang.ArithmeticException: Not valid

5) 'throws' keyword in Java

- The throws keyword is generally used for handling checked exception.
- If we do not want to handle exception by try and catch block, then it can be handled by throws.
- Without handling checked exceptions program can never be compiled.
- The **throws** keyword is always added after the method signature.
- Syntax of throws:

```
Return_type     method_name() throws    Exception_Class_Name
{
    Body of method
}
```

Example: A program to illustrate uses of throws keyword

```
import    java.io.*;
class   MyClass
{
 void   method()  throws IOException
 {
  System.out.println("My method…");
 }
}
class     throwsDemo
{
   public static void main(String args[])throws IOException
  {
     MyClass   m=new   MyClass();
     m.method();
     System.out.println("normal flow...");
  }
}
```

Difference between throw and throws keyword in Java:

Throw	Throws
It is used to throw own exception.	It is used when program does not handle exception via try block.
It handles explicitly thrown exceptions.	It handles all exceptions generated by program.
Cannot throw multiple exceptions.	Declare multiple exception E.g. public void method() throws, IOException, SQLException.
It is used within the method.	It is used within method signature.

Creating Custom Exceptions: (User defined exception)

- The exceptions that are created by programmer are known as “Custom exception.”
- These exceptions are created to generate a solution for anticipated errors as per programmer’s need.
- To create user defined exception, “Custom exception class” must be extended from “Exception” class.
- Syntax to create custom exception:

```
class    Custom_Excep_ClassName    extends    Exception
{
     Body of exception class
}
```

Example : Program to create custom exception in Java

```
import java.util.*;
class   custom   extends   Exception
{
   public custom(String   msg)
   {
                super(msg);
   }
}
public   class   MyExcep
{
   public static void main(String  arg[])
   {
                Scanner   sc=new Scanner(System.in);
                int    age;
                try
                {
                  System.out.println("Enter your Age");
                  age=sc.nextInt();
                  if(age<18)
                  {
                          throw   new  custom("Age should be greater than 18");
                  }
```

```
                else
                {
                        System.out.println("You are Eligible for voting..");
                }
        }
        catch(custom   e)
        {
                System.out.println(e);
        }
}
}
```

OutPut:

Enter your Age 52 You are Eligible for voting..	Enter your Age 15 custom: Age should be greater than 18

Thread:

- We know that, to get result or output from method its statements must be executed. And Thread is such a concept which is responsible to execute the statements of method.
- In short, we can say that because of thread methods of program executes and we will get result.
- C, C++ language supports for single threading concept i.e. to execute entire C, C++ program there is only one thread that execute entire statements of program.
- Java language supports for multithreading concept i.e. to execute Java program there will be allocate more threads that can execute multiple methods of program concurrently. (All methods will executes parallel with the help of threads)

Multithreading:

- Multithreading means executing multiple parts (methods) of the same program concurrently.
- Multithreading is a part of multitasking.
- A program may contain more than one thread and each thread complete its task simultaneously.
- Following diagram shows multithreading:

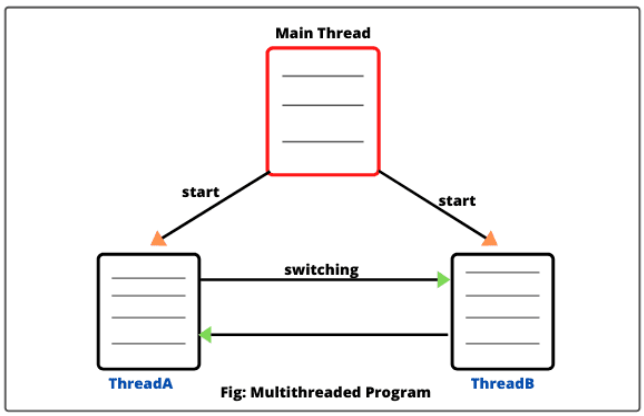

Fig: Multithreaded Program

Life Cycle of Thread:

- Life Cycle of Thread in Java represents all states of a thread from birth to its death.
- Every thread in Java has life cycle which is shown in following diagram:

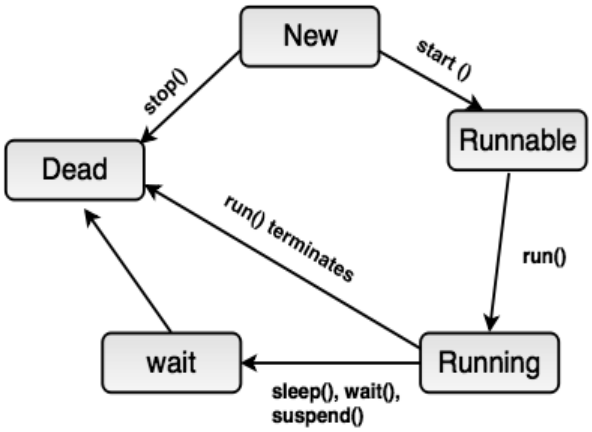

Fig: Life Cycle of Thread

Thread life cycle is explained bellow-

1) **New State:** When we create a thread, then it is said to be in the new state.
2) **Runnable State:** The life of a thread starts after invoking the start () method. The start() method invokes the run () method.
3) **Running State:** When the thread is currently running its task, then it is in running state.
4) **Wait State:** When other threads are holding the resources, the current thread is said to be in wait state
5) **Dead State:** When the thread has completed its task from run () method, then it is said to be in dead state.

Thread Methods in Java:

Following are the some important methods available in thread class.

Methods	Description
public void start( )	This method invokes the **run()** method, which causes the thread to begin its execution.
public void run( )	Contains the actual functionality of a thread and is usually invoked by **start()** method.
public static void sleep (long millisec)	Blocks currently running thread for at least certain millisecond.
public static void yield ( )	Causes the current running thread to halt temporarily and allow other threads to execute.
public static Thread currentThread ( )	It returns the reference to the current running thread object.
public final void setPriority (int priority)	Changes the priority of the current thread. The minimum priority number is 1 and the maximum priority number is 10.
public final void join (long millisec)	Allows one thread to wait for completion of another thread.
public final boolean isAlive ( )	Used to know whether a thread is live or not. It returns true if the thread is alive. A thread is said to be alive if it has been started but has not yet died.
public void interrupt ( )	Interrupts the thread, causing it to continue execution if it was blocked for some reason.

Creating Thread:

There are two ways to create a thread in Java.

1. By extending a class from **Thread** class.
2. By implementing a class from **Runnable** interface.

Let's see these way in details-

1. Extending a class from Thread class:

We can create a new thread by **extending from Thread class**.

- Thread class has constructor and method to create and perform the operation on thread.
- Following program shows creating Thread by extending Thread class

```
class  MyThread  extends  Thread
{
    public  void   run()
    {
          for(int i=1;i<=5;i++)
          {
              System.out.print(" "+i);
          }
          for(int j=100;j>=95;j--)
          {
              System.out.print(" "+j);
          }
    }
}
```

```
public  class  ThreadDemo
{
    public static void main(String[] args)
    {
        MyThread  t1 = new  MyThread();
        MyThread  t2 = new  MyThread();
        t1.start();
        t2.start();
    }
}
```

OUTPUT:

Whenever we run this program we will get different-different outputs. Since, Thread t1 and t2 parallely execute same block of code.

2. Implementing a class from Runnable Interface:

- We can also create a new thread by **implementing Runnable interface.**
- Runnable interface is available in *java.lang* package.
- The purpose of Runnable interface is to provide a set of rules common for objects to execute the functionality of thread while they are active.
- Example : Sample program to create thread class by implementing Runnable interface

```
class MyThread implements Runnable
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.print(" "+i);
        }
        for(int j=100;j>=95;j--)
        {
                    System.out.print(" "+j);
        }
    }
}
```

```
public class MyRunThread
{
    public static void main(String[] args)
    {
        MyThread dt1=new MyThread();
        MyThread dt2=new MyThread();
        Thread t1 = new Thread(dt1);
        Thread t2 = new Thread(dt2);
        t1.start();
        t2.start();
    }
}
```

OUTPUT:

Whenever we run this program we will get different-different outputs. Since, Thread t1 and t2 parallely execute same block of code.

Thread Priorities:

- Thread priorities is the integer number from 1 to 10 assigned to thread which helps to determine the order in which threads are to be scheduled for running.
- Each thread has a priority. Priorities are represented by a number between 1 and 10. In most cases, the thread scheduler schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.
- It decides when to switch from one running thread to another thread.
- Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.
- Thread scheduler selects the thread for execution on the first-come, first-serve basis. That is, the threads having equal priorities share the processor time on the first-come, first-serve basis.
- When multiple threads are ready for execution, the highest priority thread is selected and executed by JVM. In case when a high priority thread stops, yields, or enters into the blocked state, a low priority thread starts executing.
- If any high priority thread enters into the runnable state, it will preempt the currently running thread forcing it to move to the runnable state. Note that the highest priority thread always preempts any lower priority thread.

Setter & Getter Method of Thread Priority:

- The Thread class has getter and setter methods related with Thread priority which are discussed bellow-

1) int getPriority():

- This method of Thread class returns the priority of the given thread.

2) void setPriority(int newPriority):

- This method updates or assign the priority of the thread to newPriority.
- The method throws IllegalArgumentException if the value newPriority goes out of the range, which is 1 (minimum) to 10 (maximum).
- Following program shows use of MAX_PRIORITY, MIN_PRIORITY and NORM_PRIORITY

```
public class ThreadPriorityDemo extends Thread
{
   public void run()
   {
      System.out.println("Priority of thread is: "+Thread.currentThread().getPriority());
   }
   public static void main(String args[])
   {
    ThreadPriorityDemo t1=new ThreadPriorityDemo ();
    ThreadPriorityDemo t2=new ThreadPriorityDemo ();
    ThreadPriorityDemo t3=new ThreadPriorityDemo ();
```

```
    t1.setPriority(Thread.MAX_PRIORITY);
    t2.setPriority(Thread.MIN_PRIORITY);
    t3.setPriority(Thread.NORM_PRIORITY);
    t1.start();
    t2.start();
    t3.start();
   }
}
```

OUTPUT:

```
Priority of thread is: 10
Priority of thread is: 1
Priority of thread is: 5
```

Example : Following program shows use of getPriority() and setPriority() method.

```
class  newThread  extends  Thread
{
                public void run()
                {
                  System.out.println(Thread.currentThread()+" has Priority "+Thread.currentThread().getPriority());
                }
}

public class AllThread
{
  public static void main(String []ar)
  {
                    newThread t1=new newThread();
                    newThread t2=new newThread();
                    newThread t3=new newThread();
                    newThread t4=new newThread();
     t1.setPriority(Thread.MAX_PRIORITY);
     t2.setPriority(Thread.NORM_PRIORITY);
     t3.setPriority(Thread.MIN_PRIORITY);
     t4.setPriority(4);
     t1.start();
     t2.start();
     t3.start();
     t4.start();
  }
}
```

OUTPUT:

```
Thread[Thread-0,10,main] has Priority 10
Thread[Thread-1,5,main] has Priority 5
Thread[Thread-3,4,main] has Priority 4
Thread[Thread-2,1,main] has Priority 1
```

Example : Following program shows execution of thread depending on their priority.

```
public  class  ThreadExec  extends  Thread
{
public  void  run()
{
 System.out.println(Thread.currentThread());
}
public static void main(String[] args)
{
 ThreadExec   t1 = new  ThreadExec();
 ThreadExec   t2 = new  ThreadExec();
 ThreadExec   t3 = new  ThreadExec();

 t1.setPriority(4);
 t2.setPriority(2);
 t3.setPriority(8);
```

```
t1.start();
t2.start();
t3.start();
 }
}
```

Note that: By executing above program again and again, we will get different-different results.

OUTPUT:

Thread[Thread-2,8,main] Thread[Thread-0,4,main] Thread[Thread-1,2,main]	Thread[Thread-0,4,main] Thread[Thread-2,8,main] Thread[Thread-1,2,main]	Thread[Thread-2,8,main] Thread[Thread-1,2,main] Thread[Thread-0,4,main]

Thread Synchronization:

- We know that in multi-threaded programs where multiple threads try to access the same resources (like variable, memory, printer etc.) at same time and finally produce incorrect and unpredicted results.
- Therefore it is necessary to allow only one thread can access to the resource at one time where as other threads may remain in waiting state. And this task in Java is done with the help of synchronized block and synchronized method.
- Definition: Thread synchronization is such a mechanism that allows only one thread to act on object or shared resources while other threads are in waiting state. The next thread only allow to enter in shared resources, if and only if current working thread finishes its task.
- In short, because of Thread synchronization multiple threads cannot enters in shared resources at one time.
- Java provides a way of creating threads and synchronizing their tasks using **synchronized blocks**.
- Synchronized blocks in Java are marked with the **synchronized** keyword.
- All synchronized blocks synchronize on the same object can only have one thread executing inside them at a time. All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits that block.
- Thread synchronization achieved by two ways-
  1) synchronized block 2) synchronized method.
- Let's see these in details-

1) Thread synchronization by synchronized block-

Following is the general form of a synchronized block:

```
synchronized(sync_object)
{
  // Access shared variables and other shared resources
}
```

From above syntax-

- Only one thread can execute synchronized block at a time.
- sync_object is a reference to an object whose lock associates with the monitor.
- The code is said to be synchronized on the monitor object

- This synchronization is implemented in Java with a concept called monitors. Only one thread can own a monitor at a given time.
- When a thread acquires a lock, it is said to have entered the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.
- Following is an example of **multi-threading with synchronized block**:

```
class   ThreadJob
{
                public  void  myjob()
                {
                  for(int  i=11;i<=15;i++)
                  {
                            System.out.println(i);
                  }
                }
}
class  newThread  extends  Thread
{
               ThreadJob     TJ;
               newThread(ThreadJob  tj)
```

```
                {
                    TJ=tj;
                }
                public  void  run()
                {
                  synchronized(TJ)
                  {
                    System.out.println(Thread.currentThread());
                    TJ.myjob();
                  }
                }
}
public  class  ThreadSyncDemo
{
   public  static  void  main(String []arg)
   {
                ThreadJob    obj=new ThreadJob();
                newThread   t1=new  newThread(obj);
                newThread   t2=new  newThread(obj);
                newThread   t3=new  newThread(obj);
                newThread   t4=new  newThread(obj);
                t1.start();
                t2.start();
                t3.start();
                t4.start();
   }
}
```

OUTPUT:

```
Thread[Thread-0,5,main]
11
12
13
14
15
Thread[Thread-2,5,main]
11
12
13
14
15
Thread[Thread-1,5,main]
11
12
13
14
15
Thread[Thread-3,5,main]
11
12
13
14
15
```

2) Thread synchronization by Synchronization Method:

- This is another way of doing multithreading with synchronized method.
- If we declare any method using **synchronized** keyword then it is known as "synchronized method"
- The working of synchronized block and Synchronized method is same which is used to lock an object for any shared resource if any thread enters it.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```
class   ThreadDemo2  extends Thread
{
                synchronized  void  threadWork()
                {
                  System.out.println("Active Thread="+Thread.currentThread());
                  for(int   i=11;i<=15;i++)
                  {
                          System.out.println(i);
                  }
                }
                public  void  run()
                {
                    threadWork();
                }
}

public class MultiThread
 {
   public  static  void  main(String arg[] )
   {
                ThreadDemo2    t1=new ThreadDemo2();
                ThreadDemo2    t2=new ThreadDemo2();
                t1.start();
                t2.start();
    }
}
```

OUTPUT:

```
Active Thread=Thread[Thread-0,5,main]
11
12
13
14
15
Active Thread=Thread[Thread-1,5,main]
11
12
13
14
15
```

Thread Communication: (Inter Thread Communication)

- Inter-thread communication in Java is a technique through which multiple threads communicate with each other to achieve specific task.
- It provides an efficient way through which more than one thread communicate with each other by reducing CPU idle time. (CPU idle time is a process in which CPU cycles are not wasted.)
- When more than one threads are executing simultaneously, sometimes they need to communicate with each other by exchanging information with each other. A thread exchanges information before or after it changes its state.
- There are several situations where communication between threads is important.
- For example, suppose that there are two threads A and B. Thread B uses data produced by Thread A and performs its task.
- If Thread B waits for Thread A to produce data, it will waste many CPU cycles. But if threads A and B communicate with each other when they have completed their tasks, they do not have to wait and check each other's status every time.
- Thus, CPU cycles will not waste. This type of information exchanging between threads is called "inter-thread communication" in Java
- Inter thread communication is used when an application has two or more threads that exchange same information.
- Inter thread communication helps in avoiding thread pooling.

- Inter thread communication in Java can be achieved by using three methods provided by Object class of java.lang package. They are:
  1. wait()
  2. notify()
  3. notifyAll()
- These methods can be called only from within a synchronized method or synchronized block of code otherwise, an exception named *IllegalMonitorStateException* is thrown.
- All these methods are declared as final. Since it throws a checked exception, therefore, you must be used these methods within Java try-catch block.
- Let's see these methods in details-

1. wait() Method:

- wait() method in Java keeps current thread in waiting state.
- This method will throws InterruptedException.
- Various forms of wait() method allow us to specify the amount of time a thread can wait. They are as follows:

Syntax:

public final void wait()
public final void wait(long millisecond) throws InterruptedException
public final void wait(long millisecond, long nanosecond) throws InterruptedException

- All overloaded forms of wait() method throw InterruptedException.
- If time is not specified in the wait() method, a thread can wait for maximum time.

2. notify() Method:

- The notify() method wakes up a single thread that which is already in wait() state. If more than one thread is waiting, this method will awake one of them.
- The general syntax to call notify() method is as follows:

Syntax:

public final void notify()

3. notifyAll() Method:

- The notifyAll() method is used to wake up all threads that are already in wait() state.
- The general syntax to call notifyAll() method is as follows:

Syntax:

public final void notifyAll()

Following program shows inter thread communication between producer and consumer thread

```
class producer extends Thread
{
        StringBuffer sb;
        producer()
        {
          sb=new StringBuffer(); //allot memory
        }
        public void run()
        {
          synchronized(sb)
          {
                try
                {
                   for (int i=1;i<=10 ;i++ )
                  {
                        Thread.sleep(100);    //that will causes InterruptedException
                        sb.append(i+" : ");
                        System.out.println("Producing...."+i);
                  }
                }
                catch (InterruptedException e)
                {

                }
              System.out.println("Production Is over..");
              sb.notify();  //production is Over & notify to consumer thread
```

```
                }
            }
}
class  consumer  extends  Thread
{
            producer    p;
            consumer(producer    t)
            {
              p=t;
            }
            public  void  run()
            {
              synchronized(p)
              {
                    try
                    {
                          p.wait();  // Wait till notification is received
                    }
                    catch (Exception e)
                    {

                    }
                    System.out.println("Consumer Receive Production= "+p.sb);
              }
            }
}
class  ThreadCommunicate
{
            public static void main(String []as)
            {
              producer    t1=new  producer();
              consumer    t2=new  consumer(t1);
              t1.start();
              t2.start();
            }
}
```

OUTPUT:

```
Producing....1
Producing....2
Producing....3
Producing....4
Producing....5
Producing....6
Producing....7
Producing....8
Producing....9
Producing....10
Production Is over..
Consumer Receive Production= 1 : 2 : 3 : 4 : 5 : 6 : 7 : 8 : 9 : 10 :
```

Networking in Java:

- Networking is the concept of connecting multiple remote or local networking devices (computer, mobiles, printer, scanner etc.) together. Java program communicates over the network at application layer.
- All the Java networking classes and interfaces use **java.net** package to implement networking application. These classes and interfaces provide the functionality to develop system-independent network communication system.
- The java.net package provides the functionality for two common protocols and they are-

TCP (Transmission Control Protocol)

- TCP is a connection based protocol that provides a reliable flow of data between two devices.
- This protocol provides the reliable connections between two applications so that they can communicate easily.
- It is a connection based protocol.

UDP (User Datagram Protocol)

- UDP protocol sends independent packets of data, called datagram from one computer to another with no guarantee of arrival.
- It is not connection based protocol.

Networking Terminology:

i) Request and Response:

- When an input data is sent to an application via network, it is called request.
- The output data coming out from the application back to the client program is called response.

ii) Protocol:

- A protocol is basically a set of rules and guidelines which provides the instructions to send request and receive response over the network.
- For example: TCP, UDP, SMTP, FTP etc.

iii) IP Address:

- IP Address stands for Internet protocol address. It is an identification number that is assigned to a node of a computer in the network.
- For example: 192.168.2.01
- Range of the IP Address
  0.0.0.0 to 255.255.255.255

iv) Port Number:

- The port number is an identification number of server software. The port number is unique for different applications. It is a 32-bit positive integer number having between ranges 0 to 65535.

v) Socket

- Socket is a listener through which computer can receive requests and responses. It is an endpoint of two way communication link. Every server or programs runs on the different computers that has a socket and is bound to the specific port number.

Java URL Class

- While dealing with network programming in Java, it has URL class.
- URL stands for Uniform Resource Locator.
- The URL provides the logically understandable form of uniquely identifying the resource or address information on the Internet.
- The URL class provides a simple, concise API to access information over the Internet.
- URL Class Methods:

Method	Description
String getAuthority()	Returns the authority part of the URL.
int defaultPort()	Returns the default port number of the protocol associated with given URL.
String getFile()	Returns the file name of the given URL.
String getHost()	Returns the host name of the URL.
String getPath()	Returns the complete path of the given URL.
int getPort()	Returns the port number of the given URL. If it is not applicable, it returns -1.
String getQuery()	Returns the query part of the given URL.
String getProtocol()	Returns the name of the protocol of the given URL.

Following program shows use of different methods of URL class:

```
import    java.net.*;
public   class   urlDemo
{
   public   static   void    main(String args[]) throws     MalformedURLException
   {
      URL    url = new     URL("http://www.google.com/java.htm");
      System.out.println("URL: " + url.toString());
      System.out.println("Protocol: "+url.getProtocol());
      System.out.println("path: " + url.getPath());
      System.out.println("Port: "+url.getPort());
      System.out.println("Host: "+url.getHost());
      System.out.println("Authority: " + url.getAuthority());
      System.out.println("File: "+url.getFile());
      System.out.println("Query: "+url.getQuery());
      System.out.println("HashCode: "+url.hashCode());
      System.out.println("External form: "+url.toExternalForm());
   }
}
```

OUTPUT:

```
URL: http://www.google.com/java.htm
Protocol: http
path: /java.htm
Port: -1
Host: www.google.com
Authority: www.google.com
File: /java.htm
Query: null
HashCode: -1526831008
External form: http://www.google.com/java.htm
```

Theory Assignment No: 05

1) What is Exception? Explain types of exceptions in Java.
2) Write difference between Error and Exception
3) How exceptions are handle in Java? Explain with example.
4) How exceptions are handled with multiple catch blocks.
5) Give the difference between throw and throws.
6) What is Thread? Explain Multithreading.
7) How we can create thread in Java?
8) What is synchronization? How thread is synchronized in Java?
9) Write short note on:
   1) finally block 2) throw keyword 3) throws keyword 4) Custom exception
   5) Thread life cycle 6) Thread Priority 7) Thread communication 8) URL class in Java.

Practical Assignment No: 07

1) Write a program that demonstrate use of checked exception.
2) Write a program that demonstrate use of unchecked exception.
3) Write a program that shows use of ArrayIndexOutOfBoundsException.
4) Write a program that shows use of ArithmeticException.
5) Write a program that shows use of NullPointerException.
6) Write a program that shows use of NumberFormatException.
7) Write a program that shows use of FileNotFoundException.
8) Write a program that demonstrate use of multiple catch blocks to handle different exceptions.
9) Write a program that demonstrate use of finally block.
10) Write a program that demonstrate use of throw keyword.
11) Write a program that demonstrate use of throws keyword.
12) Write a program that creates user defined exception. (Custom exception)
12) Write a program that shows creating thread by extending Thread class.
13) Write a program that shows creating thread by implementing Runnable interface.
14) Write a program that shows use of MAX_PRIORITY, MIN_PRIORITY and NORM_PRIORITY.
15) Write a program that set user defined priority for thread and retrieve it.
16) Write a program that demonstrate multi-threading using synchronized block.
17) Write a program that demonstrate multi-threading using synchronized method.
18) Write a program that shows inter thread communication.
19) Write a program that shows use of different methods of URL class.