Unit - IV (Exception Handling and Multithreading)

1 Exception Handling

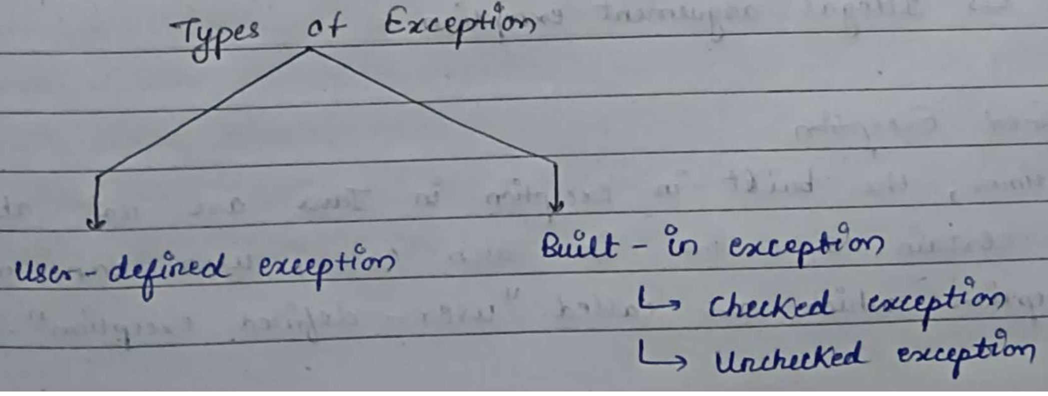
, An exception is an unwanted or unexpected event that occurs during the execution of program and disrupts the normal flow of program's instruction.

, Exception is an error condition that occurs when something wrong happens during the program execution.

, Reason why an exception occur
↳ Invalid user input
↳ Device failure
↳ Code error
↳ Loss of network Connection

, Exception handling is a mechanism to handle run time error such as
↳ Class not found exception
↳ Input / Output exception
↳ SQL exception
↳ Arithmetic exception

Types



① Built -in Exception

Build -in exception are pre defined exception provided by Java to handle common errors during program execution.

- Checked Exception

Checked exception are called compile - time exception because these exception are checked at compile - time by compiler either by catching them or declaring them in method signature using the throws Keyword.

Ex:-
↳ class not found exception
↳ Interrupted exception
↳ I/O exception
↳ SQL exception

- Unchecked Exception

Unchecked expection are just opposite to checked exception. The compile will not check these exception at compile time

Ex:-
↳ Arithmetic exception
↳ Null pointer exception
↳ Array index Out of bond exception
↳ Illegal arguement exception

② User - defined Exception

Sometimes, the built -in exception in Java are not able to describe a certain situation. in such cases, user can also create exception which are called "user - defined exception".

Ex:-

```
import java.io.* ;

class Exception {
    public static void main (String [] args)
    {
        int  n = 10 ;
        int  m = 0 ;

        int  ans = n/m ;

        System. out. println ("Answer :" + ans );
    }
}.
```

Try - Catch Block

- try - catch block is a mechanism to handle exception. This ensure that the application continues to run even if an error occurs.
- The code inside the try block is executed, and if any exception occur then it is caught by catch block.

try

The try block contains a set of statement where an exception can occur.

catch

The catch block is used to handle the uncertain condition of a try block.

Syntax of try - catch block

```
try {
    // code that might throw an exception
  }
   catch {
        // code that handle the exception
   }
```

Ex :-

```
Public class Arithmetic Exception {
       public static Vaid main ( String [ ] args ) {
              try {
                   int a = 10 ;
                   int b = 0 ;
                   int c = a/b ;
                   System. out. println ( "Result : " ) ;
              } catch ( Arithmetic Exception e ) {
                   system. out. print ln ( " Division by zero is not
                                                  allowed" ) ;
              }
              System. out. println ("Program continues.... " ) ;
       }
 }
```

Multiple - Catch

. A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. if we have to perform different task at the occurrence of different exception we use multiple - catch block

Ex :-

```
public class Multiple Catch Example {
      public static void main ( string [ ] args ) {
            try {
                 int [ ] numbers = { 1, 2, 3 } ;
                 System. out. println ( number [5] ) ;

                 string str = null ;
                 system . out . println ( str. length ( ) ) ;
            } catch ( Array index Out of Bound Exception e ) {
                    System. out. print ln ( "Caught an Array Index Out
                                              of Bound Exception" ) ;
            } catch ( Null pointer Exception e ) {
                    System. out. print ln ( " Caught a Null pointer
                                                 expection " ) ;
            } catch ( Exception e ) {
                    System. out. print ln ( "Caught a genral Exception" );

            } finally {
                  System. out. println ( " This block always
                                                executes " ) ;
            }
      }
}
```

Nested - try statement

In Java, using a try block inside another try block is permitted. it is called as nested try block

Sometimes a situation may arise where a part of block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested

Syntax :-

```
 // main try block

try
   {
      Statement 1;
      Statement 2;
  // try catch block within another try block

     try
     {
        Statement 3;
        Statement 4;
  // try catch block within nested try block

      try {
         Statement 5;
         Statement 6;
      }
catch (Exception e1)
    {
    // exception message
    }
}
 catch (Exception e2)
    {
    //exception message
   }
}
 catch (Exception e3)
     {
      //exception message
     }
```

Ex:-

```
Public class Nested Try {
     public static void main (String [] args) {
          try {
               try {
                    int [] number = {1, 2, 3};
                    System. out. println (number [5]);
               } catch (Array Index out of Bound Exception e) {
                    System. out. println ("Caught an array index out
                                             of Bound exception");
               }

          try {
               String str = null;
```

```
            System.out.println (Str. length);
        } catch (Null pointer Exception e) {
            System.out.print ln ("caugh a null pointer exception");
        }
    } catch (Exception e) {
          System. out. print ln ("Caugh a general exception");
    } finally {
          System.out.print ln ("This block always executes");
      }
    }
}
```

Throws and finally

- The throws keyword is used in a method signature to declare that the method can throw one or more exception.

-
  ```
  Public void my Method ( )
          throws IO Exception , SQL Exception {
              // Code that may throw IO Exception , SQL Exception
          }
  ```

- The finally block is used to execute important code such as closing resource , even if an exception is thrown.

```
public void My method ( ) {
        try {
            // code that may throw an exception
        } catch (Exception e) {
            // Exception handling code
        } finally {
            // Code that will always execute
        }
}
```

combining throws and finally

```
public void My method ( ) ,
          throws IO Exception {
              try {
                // Code that may throw an exception
              } catch ( IO Exception e) {
                  // Exception handling code
                    throw e ;
              } finally {
                    // code that will always execute
              }
          }
```

Uncaught Exception

Uncaught exception in Java are exception that occur during the execution of program and are not caught by any catch block. when this happen , the Java Virtual Machine (JVM) terminates the program and print a stack trace to help you identify the cause and location of exception

Common ways to handle uncaught exception :-

1) Unchecked Exception

These include Null Pointer Exception, Array Index Out of Bound Exception. they often occur due to logic error or runtime condition

2) Thread Exception

Uncaught exception in one thread might not be caught by the main thread.

3) Incorrect Exception Handling

Missing catch block or inadequate error handling mechanism can lead to uncaught exception.

Java thread

Threads are lightweight subprocess, representing the smallest unit of execution with separate paths. The main advantage of multiple thread is efficiency.

Multithreading

This is the ability of a CPU to provide multiple thread of execution concurrently, this allows different parts of program to run simultaneously. improving the efficiency and performance of application.

Threads can be used to perform complicated task in background without interrupting the main program

Life Cycle of thread

There are different state thread transfer during its lifetime.

① New State

By default, a thread will be in new state, in this state code has not yet been run and execution process is not yet initiated.

② Active State

A thread that is new state by default get transferred to active state when it invokes the start () method.

The two sub - state are

↳ Runnable state

↳ Running state

③ Waiting / Blocked State

when a thread is waiting ~~for~~ ~~another~~ ~~thread~~ ~~to~~ indefinitely for another thread to perform a particular action. or when a thread is waiting for a resource that is currently held by another thread.

④ Timed Waiting

when a thread is waiting for another thread to perform an action for a specified waiting time

⑤ Terminated

when a thread has completed its execution.

Creating a thread

In Java, we can create thread in two ways :-

1. By Extending the 'Thread' class
2. By Implementing a 'Runnable' interface.

- class My thread extend thread

```
    {
       Public vaid run ( )
        {
          String str = "Thread started running" ;
          System. out. print ln (str);
        }
    }
```

- Class My thread implments Runnable

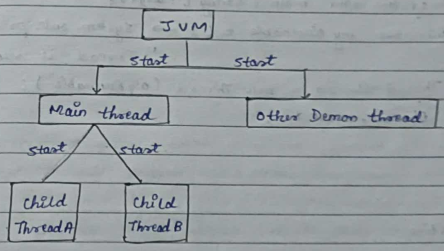
```
    {
       Public vaid run ( )
        {
          String str = "Thread is running Successfully";
          System. out. print ln (str);
        }
    }
```

Main thread

- When a Java program starts up, one thread begins running immediately. This is usually called the main thread of our program because it is one that is executed when our program begins.

- It is the thread from which other "child" thread will be spawned.

Flow diagram



Control Main thread

The main thread is created automatically when our program is started. To control it we must obtain a reference to it. This can be done by calling the method current Thread ( ) which is present in Thread class.

Creating Threads in Java

There are multiple ways to create threads in Java

① Extending thread class

② Implementing Runnable interface

③ Using Lamda Expression

```
public class MyLamdaThread {
        public static void main (string [] args) {
                Runnable myRunnable = () -> System.out.println (
                                                "Thread is running...");
                Thread thread = new Thread (myRunnable);
                 thread.start ();
        }
   }
```

Thread Priorities

- Thread priorities in java determine the order in which threads are scheduled for execution. Each thread is assigned a priority, an integer value between 1 (minimum priority) and 10 (maximum priority).

- By default, a new thread inherits the priority of the thread that created it, but you can change a thread's priority using the set Priority () method.

Thread Priority Level

- Thread.MIN_PRIORITY
  Set the minimum priority for the thread (Priority 1)

- Thread.NORM_PRIORITY
  set the default priority for the thread (Priority 5)

- Thread.MAX_PRIORITY
  Set the maximum priority for the thread (Priority 10)

Synchronization

- Synchronization is crucial for ensuring that multiple thread operate safely on shared resources. without Synchronization data incosistency or corruption can occur when multiple thread try to access and modify shared variable simultaneously.

- It is a mechanism that ensure that only one thread can access a resource at any given time. This process helps prevent issue such as data incosistency and race condition

Synchronized Block

A synchronized block in Java is synchronized on some object. Synchronized block in Java are marked with synchronized Keyword.

Syntax

```
Synchronized
  {
     // Access shared variable
     // shared resource
  }
```

- Synchronization is implemented in Java with a concept called monitors or lock. 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.

Types

① Process Synchronization

Process Synchronization is a technique used to coordinate the execution of multiple process. It ensure that shared resources are safe and in order.

② Thread Synchronization

Thread Synchronization is used to coordinate and odering of execution of the threads in a multi-threaded program.

Ex:-

```
class Counter
    private int count = 0;

  // synchronized method to ensure one thread can access at a time.
public synchronized void increment () {
        count ++ ;
     }
     public int get Count () {
           return count ;
      }

  }

Class Counter Thread extend Thread {
        private Counter ;

      public Counter Thread ( Counter ) {
              this . counter = counter ;
      }
       public void run () {
              for (int i = 0 ; i < 1000 ; i++ ) {
                     Counter . increment () ;
              }
          }
      }

public Class synchronization Example {
       public static void main (String [] args ) {
                  Counter counter = new Counter () ;

          // creating multiple thread
            Counter thread 1 = new Counter Thread (Counter) ;
            Counter thread 2 = new Counter Thread (counter) ;
            Counter thread 3 = new Counter Thread (counter) ;
    // starting the thread
        thread 1. start ();
        thread 2. start ();
        thread 3. start ();

  try {
       // waiting for all thread
        thread 1. join ();
        thread 2. join ();
        thread 3. join ();
    } catch ( Interrupted Exception e) {
          e. print stack trace ();
    }

    // Displaying final count
    System. out. println ("final count : ");

   }

}
```

Inter - thread Communication

- Interthread communication in java is a meachanism in which a thread is paused running in its critical section and another thread is allowed to enter in the same critical section to be executed.

- Inter - thread communication is also Known as Cooperation in Java.

Polling

The process of testing a condition repeatedly till it becomes true is Known as polling.

. Polling is usually implemented with the help of loops to check wheather a particular condition is true or not.

Ex- when one thread is producing data, then other thread is consuming it.

. To avoid polling, Java uses three methods, namely :-

↳ wait ()

It tells the calling thread to give up the lock and go to sleep until some other thread enters the same monitor.

↳ notify ()

It wakes up one single thread called wait () on the same object.

↳ notify All ()

It wakes up all the thread called wait () on the same object.

Ex :-

```
    class Customer {
        int amount = 10000;
```

```
Synchronized void withdraw (int amount) {
        System. out. println ("going to withdraw");
      }
  if ( this. amount < amount ) {
      System. out. println ("Less balance ; waiting for deposit");
      }
      try {
          wait ( ) ;
        } catch ( Exception e ) {

      }
      this . amount - = amount {
          system. out. println (" withdraw completed ") ;
        }

   Synchronized void deposit ( int amount )
      {
       System. out. println ("going to deposit");
      }
     this . amount + = amount {
        System. out. println ("deposit completed");
        }
         notify ( ) ;

  Class Test {
   public static void main (String [ ]args) {
        final Customer = new Customer ( );
         new Thread ( ) {
               public void run ( ) { c.withdraw (15000) ; }
```

```
} start ( ) ;
    new Thread ( ) {
        public void run ( ) { c. deposit (10000) ; }
   } . start ( ) ;


}

}
```

Suspending Thread in Java

In Java, a thread can be suspended by using the wait ( ) method on an object. This method suspend thread execution until it is notified by another thread using the notify ( ) method.

Ex:-
```
     Object lock = new Object ( ) ;
      Thread my Thread = new Thread ( ) → {
              Synchronized (lock) {
                     try {
                          lock. wait ( ) ;
                     } catch ( Interrupted Exception e) {
                            e. print Stack Trace ( ) ;
                     }
             }
             my Thread. start ( ) ;          // Start thread
              Synchronized (lock) {
                     lock. notify ( ) ;
               }
```

Resuming Thread in Java

An interrupted thread execution can be picked back up by notifying the waiting thread using notify () method.

Ex:-

```
Object lock = new Object () ;
 Thread my Thread  = new Thread () → {
        Synchronized (lock ) {
              try {
                   lock . wait () ;
              } catch ( Interrupted Exception e) {
                     e. print stack Trace () ;
              }
        }
 }
 my Thread . start () ;            // Start thread
 my Thread. interrupt ();          // interrupt thread
    Synchronized (lock ) {
         lock. notify ();          // Resume thread
    }
```

Stopping Thread in Java

To stop a running thread, use a boolean flag to signal the thread to stop gracefully.

Ex-

```
class My Thread extend Thread {
         private boolean running = true ;

   public vaid stop Thread () {
            running = false ;
      }
      public vaid run () {
              while (running ) {
         }
      }
}
My Thread my Thread = new My Thread () ;
   my Thread . Start () ;                 // start thread
   my Thread . Stop Thread () ;           // stop thread
```