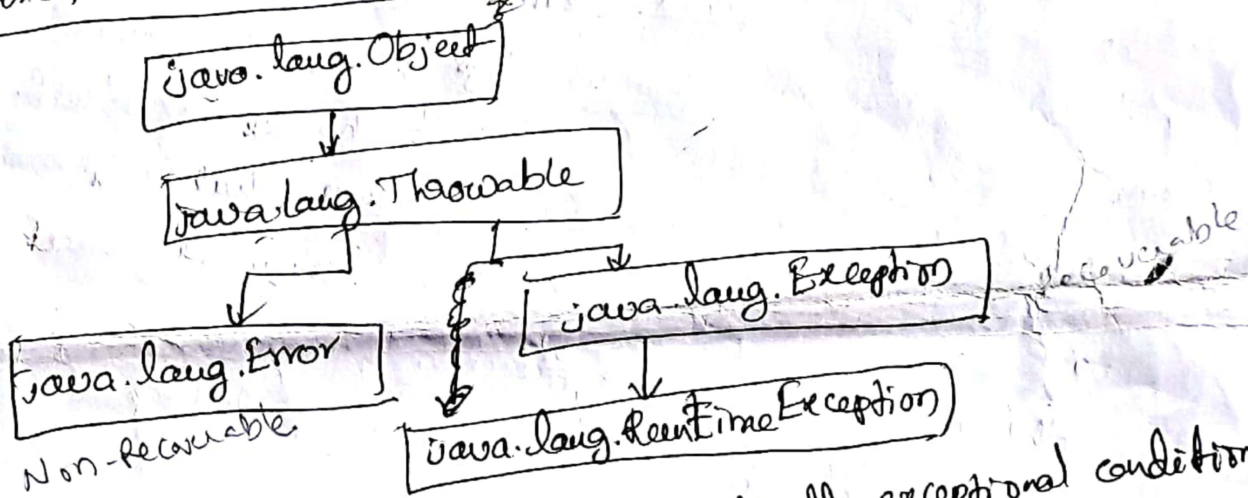
UNIT-III

Exception Handling

Exception is an abnormal condition that can occur during the execution of a program.
If these exceptions are not prevented or at least handled properly, either the program will be aborted abnormally, or the incorrect result will be carried ~~out~~ on.

Exception class Hierarchy



→ The class Throwable is used to represent all exceptional conditions.
→ Two immediate subclasses of Throwable are Exceptions & Errors.
→ The class Exceptions are used for exceptional conditions that user programs can catch.
→ The subclass of Exception is RuntimeException is for Exceptional conditions that are created during the runtime
→ The class Error, which defines the conditions that should not be expected to be caught under normal circumstances.

Unchecked :- (Not checked by compiler)
Java Application is not connected to outer resources
↳ Handling is optional

/ do
No abnormal termination

\ don't
Abnormal termination

Checked :- (Checked by compiler)
↳ Java application is connected to outer resources.
Handling is mandatory
| don't handle
Than .class file is not created
✗

compiler is involved in checking

Exceptions

Exceptions are of 2 types

→ Checked Exceptions (Exceptions)
→ Unchecked Exception (Runtime Exceptions)

Unchecked Exceptions :- Unchecked Exceptions are those exceptions that can be handling by java's default exception handlers. If we try to throw unchecked exception inside a function, we don't need to put try and catch block nor the throws.

→ The ~~un~~checked exceptions defined by java.lang that must be included in method's throws list of that method can generate one of these exceptions and doesn't handle it itself

→ 'Checked Exceptions are those exceptions that can't handle by java's default exception handlers when a checked exception is throws from inside a function we are bound to put the try and catch block or the throws declaration.

List of Checked Exceptions (·java.lang.Exception)
↳ checked by compiler at runtime

ClassNotFoundException → class not found

CloneNotSupportedException → Attempt to clone an object that doesn't implement the cloneable interface

IllegalAccessException → Access to a class is denied

InstantiationException → Attempt to create an object of an abstract or interface

InterruptedException → One thread can be interrupted by another thread

NoSuchFieldException → A requested field doesn't exist.

NoSuchMethodException → A requested method doesn't exist

Unche

Arit

Unchecked Exceptions :- Subclasses of RuntimeException.
Not checked by compiler
Java application is not connected to outer resource

ArithmeticException → Arithmetic error, such as devide by zero.

ArrayIndexOutOfBoundsException → Array index is Out of bounds.

ArrayStoreException → Assignment to an array element of an incompatible type.

ClassCastException → invalid cast.

EnumConstantNotPresentException → An Attempt is made to use an undefined enumeration value

IllegalArgumentException → Illegal argument used to invoke a method.

IllegalMonitorStateException → Illegal monitor operation, such as waiting on an unlocked thread.

IllegalStateException → Environment or application is in incorrect state.

IllegalThreadStateException → Requested operation not compatible with current thread state.

IndexOutOfBoundsException → Some type of index out-of bounds

NegativeArraySizeException → Array created with a -ve size.

NullPointerException → Invalid use of ~~an~~ null reference.

NumberFormatException → Invalid conversion of a string to a numeric format.

SecurityException → Attempt to violate security.

StringIndexOutOfBounds → Attempt to index outside the bounds of a string.

TypeNotPresentException → Type not found.

UnsupportedOperationException → An unsupported operation encountered.

→ Problem with default exception handling mechanism of Java

Ex:-
```
public class Exception_Demo
{   public static void main(String args[])
    {     int ~~i,j,k;~~ i, j = 1, k = 0;

              i = j/k;
          System.out.println("After devision by 0");

    }
}
```

→ Here by doing j/k (1/0), our program will stop abruptly.

And it will give a weird error message like.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Exception_Demo.main(Exception_Demo.java:6)
```

Exception Handling Mechanisms in Java

}

{

→ Java Exception handling is managed via 5 keywords. (3)

1) try
2) catch
3) throw
4) throws
5) finally

→ Basic form of Exception handling block

```
try
  {   //block of code
  }
  catch(ExceptionType1  e)
  {
     //Exception handling routine for ExceptionType1(optional)
  }
  catch(ExceptionType2  e)
  {  //Exception handling routine for ExceptionType2(optional)
  }
       :
       :
   catch (ExceptionTypen  e)
   {
       //Exception handling routine for ExceptionTypen(optional)
   }
  finally
  {
       //program code for exit (optional)
  }
```

This structure implements that when a block of code is executed, and if an error occurs, you may catch based on what type of exception it is, or ultimately this will be dealt with by default handler.

ample:- me
lass Exception
{
publi

→ Simple example of Exception handling

```
class DevideZero
{
    static int anyFunction (int x, int y)
    {   int a=x/y;
        return(a);
    }
    public static void main (String args[])
    {   int result=anyfunction(20,0);
        System.out.println("Result:"+result);
    }
}
```

→ Here, java provides a default runtime handler. In this case, When the java run time tries to execute the devision, it notices that the denominator is zero and then instantiates an Exception object (namely ArithmeticException) to cause this code to stop and deal with this error condition. The ~~default~~ handler prints out the exception message.

O/p:-

```
java.lang.ArithmeticException: /by zero
    at DevideZero.anyFunction (DevideZero.java:3)
    at DivideZero.main (DevideZero.java:7)
```

Example program for try-catch

try:- The try keyword can be used to specify a block of code that should be guarded against all exceptions.

catch:- Immediately following a try-block, you should include a catch ~~block~~ clause which specifies the exception type that you which to catch.

Example :- multiple Errors - only one catch

```
class ExceptionTest
{ public int j;
  static void main(String args[])
  { for(int i=0; i<4; i++)
      try
      { switch(i)
        { case 0: int zero=0;
                  j= 999/zero;    //AE
                  break;
          case 1: int b[]=null;
                  j= b[0];        //NPE
                  break;
          case 2: int c= new int[2];
                  j= c[10];       //AIOOB
                  break;
          case 3: char ch="java".charAt(9);
                                  //SIOOB
                  break;

        }
      }
      catch(Exception e)
      {
       System.out.println("In Test case #"+i+"\n");
       System.out.println(e);
      }
  }
}
```

```
O/P:- In Test case #0
        Divide by zero
        In TestCase #1
        Null Pointer error
        In Test Case #2
        ArrayIndexOut-of-bound
        In TestCase #3
        String Index is Out of Bound
```

try-catch-finally :-

The finally clause defines a block of code which will always be executed irrespective of whether any exception occurs or not.

Example:-

```
class FinallyDemo
{ public static void main(String args[])
   { int i=0;
      String greeting[] = { "Hello Twinkle", "Hello
                      Java!", "Hello world!" };
      while(i<4)
     { try
        { System.out.println(greetings[i]);
        }
        catch(Exception e)
        { System.out.println(e.toString);
          System.out.println("Resetting index value");
        }
        finally
        { System.out.println("Hi!");
            i++;
        } }
} }
```

O/P:

```
Hello Twinkle
 Hi!
 Hello Java
  Hi!
 Hello world!
  Hi!
  Array index is out-of-bound
  Resetting index value
  Hi!
```

If you run this pgm, you will see that the code in finally block will be executed always when the loop is iterated.

Throw :-

In Java, the throw keyword is used by which the user can throw an exception of his own instead of the automatic exception object generated by Java during run time. To use this, we have to create an instance of a Throwable object.

The throw keyword in java is used to explicitly throw an exception from a method or any block of code. General form of throw is

→ throw Instance

Here instance must be of type Throwable or a subclass of Throwable.

The flow of execution of the program stops immediately after the throw statement is executed and the nearest enclosing try block is checked and to see if it finds a match, controlled is transferred to that statement otherwise next enclosing try block is checked and so on. If no matching catch is found then the default exception handler will halt the program.

Example :-

```
class ThrowExce
{   static void func()
    {   try
        {  throw new NullPointerException("demo");
        }
        catch(NullPointerException e)
        {  s.o.pln("caught inside func().");
           throw e;
        }
    }
    public static void main(String args[])
     {   try
         {  func();
         }
         catch(NullPointerException e)
         {     System.out.pln("caught in main.");
         }
     }
}
```

o/p: caught inside func().
       caught in main.

throws :-

throws keyword in java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

general form

type method-name(parameters) throws exception-list

Example :-

```
class ThrowsExcep
{  static void fun() throws IllegalAccessException
   {
        s.o.pln("Inside func().");
        throw new IllegalAccessException("demo");
   }
   public static void main(String args[])
   {     try
         { fun();
         }
         catch(IllegalAccessException e)
         { s.o.pln(" caught in main.");
         }
   }
}
```

o/p: Inside fun().
     caught in main.

Nested try/catch block:-

Example

```
class NestingDemo
{
  p s r m(String args[])
  {
   try
   {
     try
     {
       try
       {
         int arr[]={1,2,3,4}
         s.o.pln (arr[10]);
       }
       catch(ArithmeticException e)
       {
         s.o.pln("ArithmeticException");
         s.o.pln("handled in try-block3");
       }
     }
     catch(ArithmeticException e)
     {
       s.o.pln("ArithmeticException");
       s.o.pln("handled in try block2");
     }
   }
   catch(ArithmeticException e)
   {
     s.o.pln("ArithmeticException");
     s.o.pln ("handled in main try block");
   }
   catch(ArrayIndexOutOfBoundsException e4)
   {
     s.o.pln ("AIOOB Exception");
     s.o.pln("handled in main tryblock");
   }
   catch(Exception e5)
   {
     s.o.pln("Exception");
     s.o.pln("handled in main try-block");
   }
  }
}
```

o/p ArryIndexOutOfBoundsException handled ⑤
in main try-block.

Throws keyword :- used to declare the exception. It provide information to the programmer that there may occur an exception so during call of that method, programmer must use exception handling mechanism.

→ Throws keyword is used to propagate <u>checked</u>
<u>Exception</u>.

→ <u>diff b/w throw & throws</u>

throw keyword is used to ~~one exception~~ one exception explicitly
~~explicitly~~ throw
throws is used to declare the exception

$$\begin{tikzcd}
P1 \arrow[r] & m1 \\
P_2 \arrow[r] & m2 \arrow[u, bend right, "call"'] \\
P3 & \\
\vdots & m4 \arrow[uu, bend right, "call"'] \\
P.4 \arrow[ru] & \\
\vdots & \\
Pn &
\end{tikzcd}$$