Exception Handling

An exception is an object that describes an unusual condition occurs in a small piece of code.

When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error. That method may choosed to handle itself or, pass it on. Either way at some point, the exception is caught and processed. Exception can be generated by the Java runtime system (JVM) or it can be manually generated by your code.

Java exception handling is managed via five keywords: try, catch, throw, throws and finally.

try :- Program statements that you want to monitor for exceptions are contained within a try block. If an exception occurs within the try block, it is thrown. The code can catch this exception and handle it.

Catch block :- If an exception occur within a try block, it is thrown. we can catch this exception by using catch block to handle this exception.

throw :- System generated exceptions are automatically thrown by the Java runtime system. (JVM). To throw an exception manually use the keyword 'throw'.

throws :- it used in a method declaration to indicate that the method might throw one or more exceptions during its execution. It allows the method to pass responsibility for handling those exceptions to the method that calls it.

finally :- it is used to create a block of code that is always executed after a try block - regardless of whether an exception is thrown or caught. If an exception is thrown, the finally will execute even if no catch statement matches the exception.

The general form of exception handling is as follows:

```
try
  {
     block of code to monitor exception
  }
catch (Exception type1 exobj1)
  {
     exception handling
  }
catch (Exception type2 exobj2)
  {
     exception handling
  }
  :
finally
  {
     block of code must be executed
  }
```

Here, exception type is the type of exceptions that are occured.

Note:- we can catch all types of exceptions by using the catch block with "Exception" keyword.
i.e catch(Exception e)

```
/* program to handle ArithmeticException */

import java.util.*;

class Exceptdemo
{
    public static void main(String args[])
    {
        int a, b, c;
        Scanner s = new Scanner(System.in);
        System.out.println("enter a, b");
        a = s.nextInt();
        b = s.nextInt();
        try
        {
            c = a / b;
            System.out.println(c);
        }
        catch(ArithmeticException e)
        {
            System.out.println("division by
                                zero");
        }
    }
}
```

single try with multiple catch statements

```
class multiplecatchdemo
{
    public static void main(String args[])
    { try
      {
        int a[] = new int[5];
        a[0] = 10;
        a[1] = 20;
        System.out.println(a[0]);
        System.out.println(a[6]);
      }
    catch(ArrayIndexoutofBoundsException e)
      {
        System.out.println("Accessing array out
                            of its boundary");
      }
    catch(NegativesizeArrayException e)
      {
        System.out.println("Array size
                            should not be negative");
      }
    }
}
```

TO demonstrate finally block

```
import java.util.*;
class finallydemo
{
    public static void main(String args[])
    {
        int a,b,c;
        Scanner s = new Scanner(System.in);
        System.out.println("enter a,b");
        a = s.nextInt();
        b = s.nextInt();
        try
        {
            c = a/b;
            System.out.println(c);
        }
        catch(ArithmeticException e)
        {
            system.out.println("division
                                by zero");
        }
        finally
        {
            System.out.println("executed
                                when exception or no exception");
        }
    }
}
```

nested try block

'try' block inside a another tryblock is called nested try block. Each time a try statement is entered, the context of that exception is pushed on the stack. If an inner try statement does not have a catch handler for a particular exception, the stack is unwound and the next try statements catch handlers are inspected for a match. This continues until one of the catch statement succeeds or until all of the nested try statements are exhausted. If no catch statement matches, then the java run-time system will throw an exception.

```
import java.util.*;
class nestedtry
 {
    public static void main(String args[])
    {
        int a,b,c;
        Scanner s = new Scanner(System.in);
```

```
try
{ System.out.println("enter a, b");
  a = s.nextInt();
  b = s.nextInt();
    try
    {
      c = a | b;
      System.out.println(c);
        {
          int d[] = {10, 20};
          System.out.println(d[0]);
          d[4] = 40;
        }
    Catch (ArrayIndexOutOfBoundsException
    (          e)
      System.out.println("Accessing the
                  array out of its boundary");
    }
}
Catch (ArithmeticException e)
  {
    System.out.println("division by zero");
  }
}
}
```

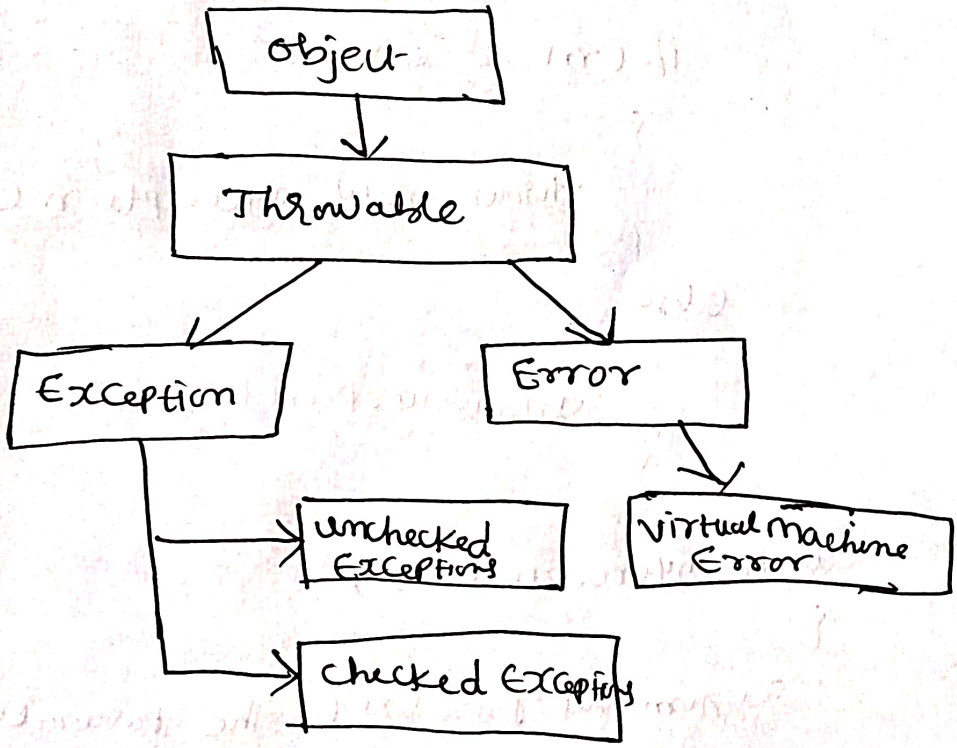
user-defined (customized) Exception

We can create our own exceptions by extending a class called Exception.

```
class myexception extends Exception
{
    void check(int m)
    {
        try
        {
            if(m < 0)
            {
                throw new myexception();
            }
            else
            {
                System.out.println(m);
            }
        }
        catch(myexception e)
        {
            System.out.println("the parameter
                should not be negative");
        }
    }
}
```

```
class myexceptiondemo
{
    Public static void main (String args[])
    {
        myexception e = new myexception ();
        e.check(10);
        e.check(-20);
    }
}
```

Hierarchy of exception



Super class for all the java classes is 'object'. Subclass of object is 'Throwable'. Throwable is the superclass of all the exception. Throwable is having two subclasses called Exception and Error.

Exception class contains checked and unchecked exceptions. Checked exceptions are checked at compilation time. unchecked exceptions are checked during runtime. Exceptions arised by the 'Error' by the virtual machine.

Examples of unchecked exception

1. ArithmeticException
2. ArrayIndexOutofBoundsException
3. NegativeArraySizeException.

The above exceptions are subclasses of 'RuntimeException'.

Examples of checked Exceptions

1. classNotFoundException — class not found
2. NoSuchFieldException — A reqested field does not exist.
3. NoSuchMethodException — A reqested method does not exist.

Benefits of Exception handling

1. Improves the readability of the program.
2. Allows for more accurate error reporting.
3. Improves the security of the program.