Experiment No.6

Title: Implementing the concept of Exception Handling
Aim: To Study 1. How to monitor code for Exception 2. How to Catch exception 3. How to use throws and finally clauses 4. how to create our own exception class

Theory:

- An *exception* is an abnormal condition that arises in a code sequence at run time.
- A Java exception is an object that describes an exceptional condition that has occurred in a piece of code
- When an exceptional condition arises, an object representing that exception is created and *thrown* in the method that caused the error
- An exception can be caught to handle it or pass it on
- Exceptions can be generated by the Java run-time system, or they can be manually generated by your code
- Java exception handling is managed by via five keywords: **try, catch, throw, throws,** and **Finally**
- Program statements to monitor are contained within a **try** block
- If an exception occurs within the **try** block, it is thrown
- Code within **catch** block catch the exception and handle it
- System generated exceptions are automatically thrown by the Java run-time system
- To manually throw an exception, use the keyword **throw**
- Any exception that is thrown out of a method must be specified as such by a **throws** clause

1. Any code that absolutely must be executed before a method returns is put in a **finally** block
2. General form of an exception-handling block

```
Try

{
    // block of code to monitor for errors
}
catch (ExceptionType1 exOb){
    // exception handler for ExceptionType1
}
catch (ExceptionType2 exOb){
    // exception handler for ExceptionType2
}
//…
```

```
finally{
    // block of code to be executed before try block ends
}
```

Exception Types

- All exception types are subclasses of the built-in class **Throwable**
- Throwable has two subclasses, they are
    - o Exception (to handle exceptional conditions that user programs should catch)
        - An important subclass of Exception is **RuntimeException**, that includes division by zero and invalid array indexing
    - o Error (to handle exceptional conditions that are not expected to be caught under normal circumstances). i.e. stack overflow
- finally
- It is used to handle premature execution of a method (i.e. a method open a file upon entry and closes it upon exit)
- **finally** creates a block of code that will be executed after **try/catch** block has completed and before the code following the **try/catch** block
- **finally** clause will execute whether or not an exception is thrown

Creating your Own Exception Classes

- You may not find a good existing exception class
- Can subclass Exception to create your own
- Give a default constructor and a constructor that takes a message

Example :

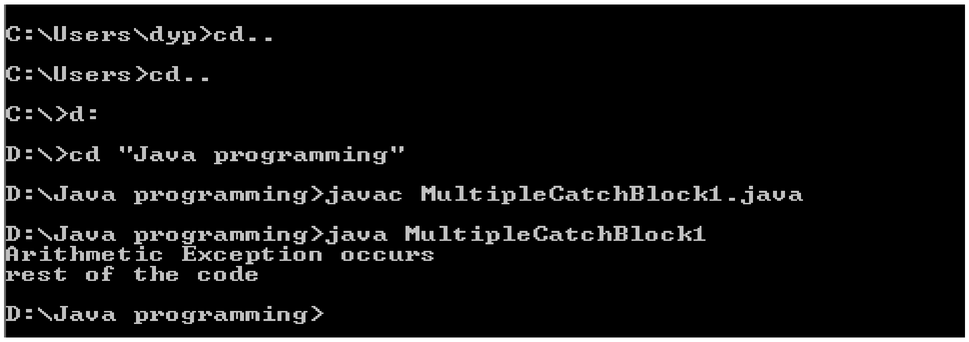
```
public class MultipleCatchBlock1 {


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("reset of the code");
        }
     }
```

Output:



```
C:\Users\dyp>cd..

C:\Users>cd..

C:\>d:

D:\>cd "Java programming"

D:\Java programming>javac MultipleCatchBlock1.java

D:\Java programming>java MultipleCatchBlock1
Arithmetic Exception occurs
rest of the code

D:\Java programming>
```

Problem Statement:

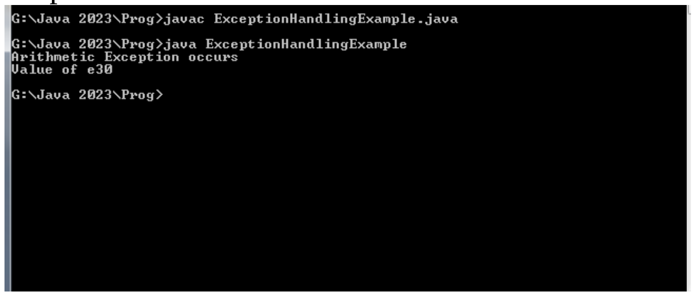
Develop application which can handle any 5 combination of predefined compile time and runtime exceptions using multiple catch blocks. Use throws and finally keywords as well.

```
public class ExceptionHandlingExample
{
   public static void main(String[] args)
{
       try
          {
          // Code that may throw exceptions
                  int a[]=new int[5];
                  a[5]=30/0;
       }
          catch (NullPointerException e)
          {
          // Handle NullPointerException
                  System.out.println("NullPointerException Exception occurs");
       }
          catch (ArrayIndexOutOfBoundsException e)
          {
          // Handle ArrayIndexOutOfBoundsException
                  System.out.println("ArrayIndexOutOfBounds Exception occurs");
       }
          catch (NumberFormatException e)
```

```
        {
        // Handle NumberFormatException
      }
        catch (ArithmeticException e)
        {
        // Handle ArithmeticException
               System.out.println("Arithmetic Exception occurs");
        }

        catch (Exception e)
        {
        // Handle any other exception
               System.out.println("Exception occurs");
      }
        finally
        {
        // Code that should always run
        int c=10,d=20;
               int e=d+c;
               System.out.println("Value of e:\t"+e);
      }
   }
}
```

Output:



```
G:\Java 2023\Prog>javac ExceptionHandlingExample.java

G:\Java 2023\Prog>java ExceptionHandlingExample
Arithmetic Exception occurs
Value of e30

G:\Java 2023\Prog>
```

Conclusion: Thus we have studied Exception handling in different ways.