 *DEPARTMENT OF INFORMATION TECHNOLOGY*

Experiment No.14

|  |  |  |
| --- | --- | --- |
| Experiment No | 14 | |
| Experiment Title | Write programs based on exception handling in java. . | |
| Resources / Apparatus Required | Java SE(JDK)8u102 ,  gedit text editor | PC |
| Objectives | The objective of this experiment is to learn programs based on exception handling (try ,catch & finally) in java. | |
| Theory | Exception Handling An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: **try, catch,** and **throw**.   * **throw:** A program throws an exception when a problem shows up. This is done using a **throw** keyword. * **catch:** A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The **catch** keyword indicates the catching of an exception. * **try:** A **try** block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.   **Java try-catch**  **Java try block**  Java try block is used to enclose the code that might throw an exception. It must be used within the method. Java try block must be followed by either catch or finally block.  **Syntax of java try-catch**   1. **Try** 2. { 3. //code that may throw exception 4. } 5. **catch**(Exception\_class\_Name ref){}  Java catch blockJava catch block is used to handle the Exception. It must be used after the try block only.You can use multiple catch block with a single try.Throwing Exceptions If a method needs to be able to throw an exception, it has to declare the exception(s) thrown in the method signature, and then include a throw-statement in the method. Here is an example:  public void divide(intnumberToDivide, intnumberToDivideBy)  throws BadNumberException{  if(numberToDivideBy == 0){  throw new BadNumberException("Cannot divide by 0");  }  return numberToDivide / numberToDivideBy;  }  When an exception is thrown the method stops execution right after the "throw" statement. Any statements following the "throw" statement are not executed. In the example above the "return numberToDivide / numberToDivideBy;" statement is not executed if a BadNumberException is thrown. The program resumes execution when the exception is caught somewhere by a "catch" block. Catching exceptions is explained later.  You can throw any type of exception from your code, as long as your method signature declares it. You can also make up your own exceptions. Exceptions are regular Java classes that extends java.lang.Exception, or any of the other built-in exception classes. If a method declares that it throws an exception A, then it is also legal to throw subclasses of A.  **Finally**  You can attach a finally-clause to a try-catch block. The code inside the finally clause will always be executed, even if an exception is thrown from within the try or catch block. If your code has a return statement inside the try or catch block, the code inside the finally-block will get executed before returning from the method. . | |
| Program & output | A) /\*W.A.P to perform division of two numbers accepted from the user and handle1) ArithmeticException - for division by zero error2) NumberFormatException - for non number value\*/   A)  import java.io.\*;  class EH4  {  public static void main(String args[]) throws IOException  {  int a=0,b=0,res;  String str;  BufferedReader br = new BufferedReader (new inputStreamReader (System.in));  System.out.println("Enter two numbers:");  try  {  str=br.readLine();  a=Integer.parseInt(str);  str=br.readLine();  b=Integer.parseInt(str);  res=a/b;  System.out.println("The Quotient=" + res);  }  catch(ArithmeticException ae)  {  System.out.println("Exception has occurred as entered divisior is 0");  }  catch (NumberFormatException ne)  {  System.out.println("Invalid number");  }  finally  {  System.out.println("Its Finally Block");  }  }  }    C:\Users\Mahesh\Desktop\New folder (2)\Untitled 5.png  B)  **/\***  **W.A.P to accept and display the Month Number.**  **Throw an Exception if improper month is entered. Make your own exception class to handle this exception\*/**  import java.io.\*;  class MonthNumberException extends Exception { public MonthNumberException(String str)  { System.out.println(str); } } class EH9{ public static void main(String args[]) throws IOException { int m; String str; BufferedReader br = new BufferedReader (new InputStreamReader (System.in)); System.out.println("Enter month number:"); str=br.readLine(); m=Integer.parseInt(str); try { if(m>12 || m<1)  throw new MonthNumberException("Invalid Month Number");  System.out.println("Month number entered is "+m); } catch(MonthNumberException me) { System.out.println("\*\*\* Invalid month number \*\*\*\*\*"); } } }  C:\Users\Mahesh\Desktop\New folder (2)\6.png | |
| Conclusion | Thus, we have learnt programs based on exception handling in java. | |