# // Exception Hadling

# input/Problem\_01

// 1.Write a java program using multiple catch blocks. Create a class CatchExercise inside the  
// try block declare an array a[] and initialize with value a[5] =30/5; . In each catch block  
// show Arithmetic exception and ArrayIndexOutOfBoundsException.  
// Test Data:  
// a[5] =30/5;  
// Expected Output :  
// ArrayIndexOutOfBoundsException occurs  
// Rest of the code  
  
public class Problem\_01 {  
 public static void main(String[] args) {  
 try {  
 int a[] = new int[5];  
 a[5] = 30 / 5;  
 } catch (ArithmeticException e) {  
 System.out.println(e + " occurs");  
 } catch (ArrayIndexOutOfBoundsException e) {  
 System.out.println(e + " occurs");  
 }  
 System.out.println("Rest of the code");  
 }  
}

# input/Problem\_02

// 2. Create a program to ask the user for a real number and display its square root. Errors  
// must be trapped using "try..catch".  
  
import java.util.Scanner;  
  
public class Problem\_02 {  
 public static void main(String[] args) {  
 float real\_num;  
 System.out.print("Enter a real number: ");  
  
 Scanner scanner = new Scanner(System.in);  
 real\_num = scanner.nextFloat();  
 scanner.close();  
  
 try {  
 if (real\_num < 0)  
 throw new IllegalArgumentException("The number must not be negative");  
 System.out.println("Square root is " + Math.sqrt(real\_num));  
 } catch (IllegalArgumentException e) {  
 System.out.println(e);  
 } catch (Exception e) {  
 System.out.println(e);  
 }  
  
 System.out.println("rest of the code...");  
 }  
}

# input/Problem\_03

// 3. (Catching Exceptions with Superclasses) Use inheritance to create an exception  
// superclass (called ExceptionA) and exception subclasses ExceptionB and ExceptionC,  
// where ExceptionB inherits from ExceptionA and ExceptionC inherits from ExceptionB.  
// Write a program to demonstrate that the catch block for type ExceptionA catches  
// exceptions of types ExceptionB and ExceptionC.  
  
class ExceptionA extends Exception {  
 public ExceptionA(String message) {  
 super(message);  
 }  
}  
  
class ExceptionB extends ExceptionA {  
 public ExceptionB(String message) {  
 super(message);  
 }  
}  
  
class ExceptionC extends ExceptionB {  
 public ExceptionC(String message) {  
 super(message);  
 }  
}  
  
public class Problem\_03 {  
 public static void main(String[] args) {  
 try {  
 throw new ExceptionB("Exception of type ExceptonB");  
 } catch (ExceptionA e) {  
 System.out.println(e);  
 }  
  
 try {  
 throw new ExceptionC("Exception of type ExceptonC");  
 } catch (ExceptionA e) {  
 System.out.println(e);  
 }  
 }  
}

# input/Problem\_04

// 4. (Catching Exceptions Using Class Exception) Write a program that demonstrates how  
// various exceptions are caught with catch (Exception exception ) This time, define classes  
// ExceptionA (which inherits from class Exception) and ExceptionB (which inherits from  
// class ExceptionA). In your program, create try blocks that throw exceptions of types  
// ExceptionA, ExceptionB, NullPointerException and IOException. All exceptions should  
// be caught with catch blocks specifying type Exception.  
  
import java.io.IOException;  
  
class ExceptionA extends Exception {  
 public ExceptionA(String message) {  
 super(message);  
 }  
}  
  
class ExceptionB extends ExceptionA {  
 public ExceptionB(String message) {  
 super(message);  
 }  
}  
  
public class Problem\_04 {  
 public static void main(String[] args) {  
 try {  
 throw new ExceptionA("Exception type A");  
 } catch (Exception exception) {  
 System.out.println(exception);  
 }  
  
 try {  
 throw new ExceptionB("Exception type B");  
 } catch (Exception exception) {  
 System.out.println(exception);  
 }  
  
 try {  
 throw new NullPointerException("Null pointer exception");  
 } catch (Exception exception) {  
 System.out.println(exception);  
 }  
  
 try {  
 throw new IOException("IO Exception");  
 } catch (Exception exception) {  
 System.out.println(exception);  
 }  
 }  
}

# input/Problem\_05

// 5. (Order of catch Blocks) Write a program that shows that the order of catch blocks is  
// important. If you try to catch a superclass exception type before a subclass type, the  
// compiler should generate errors.  
  
public class Problem\_05 {  
 public static void main(String[] args) {  
 try {  
 int a = 50 / 0;  
 System.out.println(a);  
 } catch (Exception e) {  
 System.out.println(e);  
 }  
   
 // catch (ArithmeticException e) {  
 // System.out.println(e);  
 // }  
  
 // Uncommenting these 3 lines will cause compile time error  
 }  
}

# input/Problem\_06

// 6. (Constructor Failure) Write a program that shows a constructor passing information  
// about constructor failure to an exception handler. Define class SomeClass, which throws  
// an Exception in the constructor. Your program should try to create an object of type  
// SomeClass and catch the exception that’s thrown from the constructor.  
  
class SomeClass {  
 public SomeClass() throws Exception {  
 throw new Exception("SomeClass constructor exception");  
 }  
}  
  
public class Problem\_06 {  
 public static void main(String[] args) {  
 try {  
 SomeClass someClass = new SomeClass();  
 } catch (Exception e) {  
 System.out.println(e);  
 }  
 }  
}

# input/Problem\_07

// 7. (Rethrowing Exceptions) Write a program that illustrates rethrowing an exception. Define  
// methods someMethod and someMethod2. Method someMethod2 should initially throw  
// an exception. Method someMethod should call someMethod2, catch the exception and  
// rethrow it. Call someMethod from method main, and catch the rethrown exception. Print  
// the stack trace of this exception.  
  
public class Problem\_07 {  
 static void someMethod2() throws Exception {  
 throw new Exception("An exception");  
 }  
  
 static void someMethod() throws Exception {  
 someMethod2();  
 }  
  
 public static void main(String[] args) {  
 try {  
 someMethod2();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 System.out.println("rest of the code...");  
 }  
}

# input/Problem\_08

// 8. (Catching Exceptions Using Outer Scopes) Write a program showing that a method with  
// its own try block does not have to catch every possible error generated within the try.  
// Some exceptions can slip through to, and be handled in, other scopes.  
  
public class Problem\_08 {  
 static void another\_method() {  
 try {  
 System.out.println(1 / 0);  
 } catch (ArrayIndexOutOfBoundsException e) {  
 System.out.println(e + " - Array Index");  
 }  
 }  
  
 public static void main(String[] args) {  
 try {  
 try {  
 another\_method();  
 } catch (ArithmeticException e) {  
 System.out.println(e + " - Arithmetic");  
 }  
 } catch (Exception e) {  
 System.out.println(e + " - General");  
 }  
 }  
}