Exception Handling

```
/*
 * 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 catchExercise{
    public static void main(String[] args) {
        int a[] = new int[5];
        try{
            a[5] = 30/5;
        } catch(ArithmeticException e){
            System.out.println(e);// ArithmeticException occurs
        } catch(ArrayIndexOutOfBoundsException e){
            System.out.println(e); //ArrayIndexOutOfBoundsException occurs
        }
    }
}
```

```
/*
 * 2. Create a program to ask the user for a real number and display its square root. Errors must be trapped using "try..catch".
 */

public class problem2 {
    public static void main(String[] args) {
        try{
            System.out.print("Enter a real number: ");
            double num = Double.parseDouble(System.console().readLine());
            if(num < 0)
                throw new IllegalArgumentException("Unreal Number"); //throwing an exception

            System.out.println("Square root of " + num + " is " + Math.sqrt(num));
        } catch(NumberFormatException e){ //catching the exception thrown
            System.out.println("Invalid input");
        } catch(IllegalArgumentException e){ //catching the exception thr
            System.out.println(e.getMessage());
        } catch(Exception e){ //catching all other exceptions
            System.out.println("Unknown error");
        }
    }
}
```

```
/*
 * 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.
 */

 public class problem3 {
    public static void main(String[] args) {
        try {
            throw new ExceptionC("This is an exception of type ExceptionC");
        } catch (ExceptionA e) {
            System.out.println(e.getMessage());
        }
    }
}

class ExceptionA extends Exception{
    public ExceptionA(String errorMsg) { //constructor
        super(errorMsg); //calling the constructor of the superclass
    }
}

class ExceptionB extends ExceptionA{
    public ExceptionB(String errorMsg) {
        super (errorMsg); //calling the constructor of the superclass
    }
}
```

```
class ExceptionC extends ExceptionB{
    public ExceptionC(String errorMsg) {
        super(errorMsg); //calling the constructor of the superclass
    }
}
```

```
/*
 * 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;
public class problem4 {
    public static void main(String[] args) {
        try {
            //exception of type ExceptionA
            //throw new ExceptionA("This is an exception of type ExceptionA");

            //exception of type ExceptionB
            //throw new ExceptionB("This is an exception of type ExceptionB");

            //exception of type ExceptionC
            //throw new ExceptionC("This is an exception of type ExceptionC");

            //exception of type NullPointerException
            //String str = null;
            //System.out.println(str);
            //throw new NullPointerException("This is an exception of type NullPointerException");

            //exception of type IOException
            throw new IOException("This is an exception of type IOException");

        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

class ExceptionA extends Exception{
    public ExceptionA(String errorMsg) { //constructor
        super(errorMsg); //calling the constructor of the superclass
    }
}

class ExceptionB extends ExceptionA{
    public ExceptionB(String errorMsg) {
        super (errorMsg); //calling the constructor of the superclass
    }
}

class ExceptionC extends ExceptionB{
    public ExceptionC(String errorMsg) {
        super(errorMsg); //calling the constructor of the superclass
    }
}
```

```
/*
 * 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 problem5 {
    public static void main(String[] args) {
        try{
            System.out.println(55/0);
        //} catch (Exception e) {
            //System.out.println(e.getMessage());
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
        //} catch (Exception e) {
            //System.out.println(e.getMessage());
        }
    }
}
```

```
/*
 * 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.
 */

public class problem6 {
    public static void main(String[] args) {
        try {
            SomeClass obj = new SomeClass();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

class SomeClass {
    public SomeClass() throws Exception {
        throw new Exception("Constructor failure: Unable to create object of SomeClass");
    }
}
```

```
/*
 * 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 problem7 {
    public static void someMethod() throws Exception { // someMethod throws an exception
        try {
            someMethod2(); // calling someMethod2
        } catch (Exception e) {
            System.out.println("Exception caught in someMethod, rethrowing...");
            throw e; // rethrowing the exception
        }
    }

    public static void someMethod2() throws Exception { // someMethod2 throws an exception
        throw new Exception("Exception thrown from someMethod2"); // throwing an exception from someMethod2
    }

    public static void main(String[] args) { // main method
        try {
            someMethod(); // calling someMethod
        } catch (Exception e) {
            System.out.println("Exception caught in main: " + e.getMessage());
            System.out.println("Printing stack trace of the rethrown exception: ");
            e.printStackTrace(); // printing the stack trace of the rethrown exception
        }
    }
}
```

```
/*
 * 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 problem8 {
    public static void main(String[] args) {
        try {
            try {
                method(); // calling method that throws an exception
            } catch (ArithmeticException e) { // catching ArithmeticException
                System.out.println("ArithmeticException caught in inner try block");
            }
        } catch (Exception e) { // catching other Exceptions
            System.out.println("Exception caught in outer try block: " + e.getMessage());
        }
    }

    public static void method() throws Exception { // method throws an exception
        try {
            System.out.println(10/0);
        } catch (NullPointerException e) { // catching NullPointerException
```

```
            System.out.println("NullPointerException caught in method");
        }
    }
}
```