**Assignment: Exception Handling**

**1. Write a program to throw a checked exception explicitly using 'throw' keyword and  
  
a) Handle the exception in the same method.**

**Answer:**

import java.io.IOException;

public class Checked\_a {

public static void main(String[] args) {

try {

throw new IOException("IOException");

} catch (IOException e) {

System.out.println(e.getMessage());

}

}

}

**b) use throws clause and handle the exception in some other method (calling method)**

**Answer:**

import java.io.IOException;

public class Checked\_b {

static void method() throws IOException {

throw new IOException("IOException");

}

public static void main(String[] args) {

try {

method();

} catch (IOException e) {

System.out.println(e.getMessage());

}

}

}

**c) Don't either handle or use the throws clause.**

**Answer:**

import java.io.IOException;

public class Checked\_c {

public static void main(String[] args) {

throw new IOException("IOException");

}

}

**2. Repeat program 1 with unchecked Exception and demonstrate the difference in both program.**

**Answer a:**

public class Unchecked\_a {

public static void main(String[] args) {

try {

throw new ArithmeticException("ArithmeticException");

} catch (ArithmeticException e) {

System.out.println(e.getMessage());

}

}

}

**Answer b:**

public class Unchecked\_b {

static void method() throws ArithmeticException {

throw new ArithmeticException("ArithmeticException");

}

public static void main(String[] args) {

try {

method();

} catch (ArithmeticException e) {

System.out.println(e.getMessage());

}

}

}

**Answer c:**

public class Unchecked\_c {

public static void main(String[] args) {

throw new ArithmeticException("ArithmeticException");

}

}

NOTE: The only difference between the first two questions is that in (b) part of checked exception, we have to use throws keyword for handling exception in other methods. But in unchecked exception, we can handle exception in other methods without using throws also.

**3.Create a user defined exception to check whether your employee exist in your data structure (use any data structure to store the employees - like array, ArrayList etc) and throw exception if name is not in the employees list. Use the catch and finally block to make an appropriate handling.**

**Answer:**

import java.util.ArrayList;

import java.util.Scanner;

class NameNotFoundException extends Exception

{

NameNotFoundException(String s)

{

super(s);

}

}

class Employee

{

String name;

int id;

Employee(String name, int id)

{

this.name = name;

this.id = id;

}

}

public class Q3{

public static void main(String[] args){

ArrayList<Employee> employees = new ArrayList<Employee>();

employees.add(new Employee("Tushar", 1));

employees.add(new Employee("Rishabh", 2));

employees.add(new Employee("Saksham", 3));

employees.add(new Employee("Bhavya", 4));

employees.add(new Employee("Yogesh", 5));

Scanner sc = new Scanner(System.in);

System.out.println("Enter the name of the employee: ");

String input = sc.nextLine();

try

{

boolean flag = false;

for(Employee e: employees)

{

if(e.name.equals(input))

{

flag = true;

break;

}

}

if(flag)

System.out.println("Employee found");

else

throw new NameNotFoundException("Employee not found");

}

catch(NameNotFoundException e)

{

System.out.println(e);

}

finally

{

System.out.println("Finally block executed");

sc.close();

}

}

}

**4.The assignment requirements follow:  
  
a. Create your exception in a file called ScoreException.java. Create the UseScoreException.java class to use this exception. The UseScoreException.java file will have the main() method.  
  
b. Prompt the user to enter a test score. Read this score using a dialog box.  
  
c. Any input value greater than 100 or less than 0 should generate the ScoreException exception. You must throw a ScoreException exception at least once and you must have a catch block that catches a ScoreException exception. Use the getMessage method of the Throwable class to display the message associated with this exception. You may do other processing also if you wish. This catch block must display a dialog box with "The score must be >= 0 and <= 100!".  
  
d. Display a return message with "Do you want to enter another score?" after the user inputs 'Yes' or 'Y'.  
  
e. Use the "parseXXXX" methods to convert the String read from the input score. This method will throw a NumberFormatException if it is unable to convert the String object to a valid number. Any input value that has characters other than numbers, a decimal, or a negative sign will generate the NumberFormatException exception. You must have a catch block that catches a NumberFormatException exception. This catch block must display a dialog box with "You must enter a number for the score!".  
  
f. Return step d.  
  
g. Display "That is a valid score." if a valid score is entered..  
  
h. Continue the steps above until the user enters No. ("Do you want to enter another score?").**

**Answer:**

**UserScoreException.java**

import javax.swing.\*;

public class UserScoreException{

public static void main(String[] args){

JFrame f = new JFrame();

String scoreString = JOptionPane.showInputDialog(f,"Enter your test score: ");

try{

int score = Integer.parseInt(scoreString);

try{

if(score > 100 || score < 0){

throw new ScoreException("The score must be >= 0 and <= 100!");

}

else{

JOptionPane.showMessageDialog(f, "That is a valid score.");

}

} catch (ScoreException e){

JOptionPane.showMessageDialog(f, e.getMessage());

}

} catch (NumberFormatException e){

JOptionPane.showMessageDialog(f, "You must enter a number for the score!");

} finally {

String answer = JOptionPane.showInputDialog(f,"Do you want to enter another score? (Yes or Y)");

if(answer.equalsIgnoreCase("Yes") || answer.equalsIgnoreCase("Y")){

main(args);

}

}

}

}

**ScoreException.java**

public class ScoreException extends Exception{

public ScoreException(String message){

super(message);

}

}