# Module-Exception Handling Exercises

## **Handling Runtime Exceptions**

Exception handling using try-catch-finally blocks

### Objective:

Understand what type of runtime exceptions might occur for a given scenario and how to handle them.

### Problem Statement:

Take the data from command line arguments and pass that data to a method to compute.

The method should divide the array element chosen using the index number passed as second argument with the number passed as the third argument.

The method should return the result. The method should also set the array reference to null before returning irrespective of if exception occurs or not.

Syntax:

public int MethodName (int[] data,int index, int divisor) { }

Sample input:

If the Command line arguments are entered as:

3 5 6 7 2 5 0

Then int[] data contains {3,5,6,7,2} and index will be 5 and the divisor will be 0

If the Command line arguments are entered as:

8 3

int[] data will be null and index will be 8 and the divisor will be 3

The above problem statement should also be solved using traditional approach of conditional statements and without using try catch blocks.

## **Banking Application**

A very simple banking application should allow account creation, transfer funds between accounts.

While doing these operations many abnormalities may occur and to make your application a robust one is a challenging task.

### Objective

Write a hierarchy of custom exception class and to funnel the exceptions.

### Problem Statement

Create an Account class that contains the following information:

Account number, name of account holder and balance.

The Account class should allow the following operations:

1. Deposit
   1. It should throw InvalidAmountException, if user tries to deposit amount less than or equal to zero.
   2. The balance should get updated if user enters a valid amount
2. Withdraw
   1. Should update the balance if an valid amount is entered
   2. Should throw InvalidAmountException, if user tries to withdraw amount less than or equal to zero.
   3. Should throw InsufficientBalanceException, if the user tries to withdraw an amount which is greater than existing balance.
3. Balance inquiry.
   1. Should return the existing balance in the account

Create a Bank Service class with following functionalities

1. Create an Account
   1. On successful creation the account should be stored into a static data.
   2. If account creation fails should throw AccountCreationException.
   3. Reasons why account creation should fail
      1. Initial amount for account creation is less than 500.
      2. Account Number already exists
      3. Name not provided
2. Transfer funds between two accounts
   1. On successful transfer of funds the balance should get updated in both the accounts.
   2. On failure should throw AccountTransactionException

Reasons why Transaction fails

* + 1. InvalidAmountException
    2. InsufficientBalanceException
  1. Need to log all the transaction attempts (i.e, Debit from account, Credit to account and the amount) irrespective of successful transaction or even for failure cases also.

The main application using these two classes should work with the following try, catch

1. try {

// invoke Bank Service code here

} catch(BankingApplicationException exception) { }

The main code should also work for the following try, catch blocks

1. try {

// invoke Bank Service code here

} catch(AccountCreationException exception) { }

1. try {

// invoke Bank Service code here

} catch(AccountTransactionException exception) { }