Object Oriented Development

# Bank Design Exercise

# What does this exercise cover?

This exercise will give you a chance to practise applying Object Oriented Programming concepts to a system design.

# How long will the exercise take to complete?

2-3 hours

# What should you have already completed?

Training in Object Oriented Concepts and Java Syntax

# What do you need?

In order to complete this tutorial exercise you will need:

* Java Development Kit 1.6 or above
* Apache Maven
* Eclipse IDE Kepler or above
* Subversion

# Introduction

The task is to design and implement classes for the business logic layer for a Bank application as described below.

These requirements are neither complete nor perfect; the challenge is to identify elements that you think it needs to have, and try to model them with Object Oriented Programming.

Consider what you have learned about:

* Basic Java Syntax
  + Using access modifiers to control visibility
  + Instance vs. static members
    - Think about the need (or lack of) to have static members in your classes
* Inheritance and Polymorphism
* Abstract Classes/Interfaces
  + Decide when and if to use each
* Collections
  + You will need to use a collection class, which will be covered in depth later. If this is new to you, research proper syntax and use an ArrayList.

# System Description

This program will be used to generate and organize data for a bank. We want to create, manipulate, and track different types of bank accounts, different types of customers, and the connections between accounts and users. There will be no persistence or storage (data will have to be recreated each time the program is executed).

Bank Accounts have the following specifications:

* Bank Accounts have Account Identification Numbers and Balances.
* The ID Number is set when the Bank Account is first created and never modified.
* ID number (System generated) must be unique across all accounts regardless of type or owner. (Starts at 1000 and counts by 5)
* The Balance can be increased with a Deposit, decreased with a Withdraw, or completely changed with a Correction.

Bank Account categories:

* Savings Accounts vs. Checking Accounts
* Business Accounts vs. Personal Accounts

Accounts can fall into different combinations:

* Business Savings
* Business Checking
* Personal Savings
* Personal Checking

Checking Accounts:

* Allowed to be overdrawn
* Track the number of the next check to be issued (can be updated by requesting Next Check)
* First check number to be issued is set on creation

Savings Accounts:

* Can never be overdrawn
* Stores an interest rate (do not calculate interest – just store the value)
* Interest Rate can be accessed and changed. Initialized to 0%.

Business Accounts:

* Must belong to a company
  + Implement this requirement without using if or switch statements

Personal Accounts:

* Must belong to a person
  + Implement this requirement without using if or switch statements

Bank Customers:

* Bank Customers have ID numbers, Names, addresses, Tax Id Numbers (not internally generated)
* ID numbers are unique and auto-generated, starting at 2000000 and counting by 7s

Customers come in two flavours:

* Company vs. Person

Company

* Can add the same amount (specified by user) to all of the accounts it owns at once

Person

* Can reset all of his/her accounts' balances back to 0 in one step

Managing data

* Accounts and customers can be created in any order at any time
* Any number of accounts and/or customers may exist in the system at once
* Each account must belong to a customer
* A customer can own any number of accounts
* Accounts and customers are never to be duplicated -- each new object should represent a completely new item
* If a company or person is removed from the system, the attached accounts must be removed as well

## **User Interface**

You are allowed to implement a simple user interface in a main method, but it is not a requirement and should be your lowest priority. The goal of this exercise is to try your hand at designing a system with many elements that interact with one another.

## **UML**

Please create a UML Class diagram displaying your design. Make sure you utilise correct syntax.