C – Starting Classes

Student Exercises – Documentation

# Exercises Overview

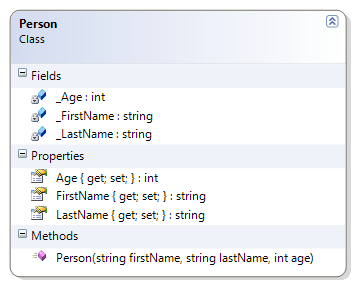
The following exercises are given as practice for this topic. The recommended order of the exercises is as follows.

1. **Person** - Extend the Person class to include a constructor.
2. **Account** – Extend the Account class from the example to include more information. Specifically, include an ***AccountType:String***, ***BankName:String***, ***BranchNumber:Integer***, and ***InstitutionNumber:Integer***.
3. **CanadianAddress** - This class represents an address for some place in Canada.
4. **Course** - This class represents a post-secondary course with a theory (exam) and a lab portion.
5. **ExamResult** - This class represents the results of an exam for a student.
6. **LabResult** - This class represents the results of an lab for a student.

### Person

This simple class illustrates and introduces the idea of encapsulation (private fields with public properties). This example uses a constructor for ensuring the state of an object when it is instantiated (created).

Extend the **Person** class from the example to include a constructor that takes in a first and last name as well as an age in the parameter list.

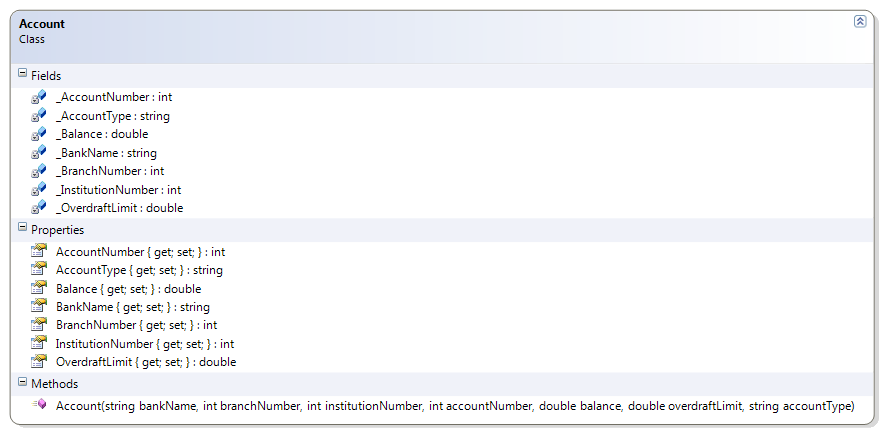


### Account

Extend the Account class from the example to include more information.

## Problem Statement:

Extend the **Account** class from the example to include more information. Specifically, include an **AccountType:String**, **BankName:String**, **BranchNumber:Integer**, and **InstitutionNumber:Integer**. Base your solution on the following class diagram.



Also modify the driver to make use of the added information.

#### Notes

The branch number and the institution number together make up the Transit Number for a bank. "The bank transit number is 8 digits long. This is divided into a 5 digit branch number and 3 digit institution code, for example 10000-200." (See <http://en.wikipedia.org/wikiSort_code>)

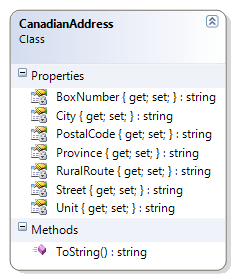
For more information on bank accounts and transit numbers in Canada, see <http://en.wikipedia.org/wiki/Routing_transit_number#Canadian_transit_number>.

### CanadianAddress

This class represents an address for some place in Canada.

## Problem Statement:

Create the **CanadianAddress** class so that it can represent the majority of possible addresses that some place may have in Canada. Design the class to use auto-implemented properties as specified in this class diagram. No constructor is required for this class.



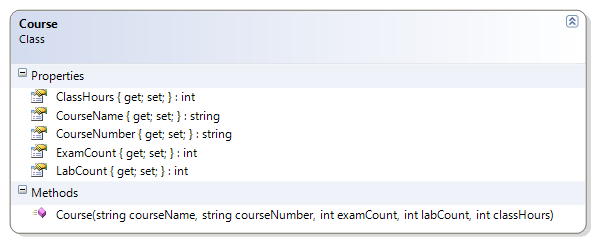
Also create a driver for testing this class; you may use any name for the driver as long as it is not already mentioned in this namespace. In the driver, create two instances of the CanadianAddress class that represent your current address as well as the address of your school (use hard-coded data).

### Course

This class represents a post-secondary course with a theory (exam) and a lab portion.

## Problem Statement:

Create the **Course** class so that it represents a post-secondary course. Design the class to have auto-implemented properties and a constructor as specified in this class diagram. Note that the auto-implemented properties should have their "set" implementations marked as private.



Also create a driver for testing this class; you may use any name for your driver as long as it is not already mentioned in this namespace. In the driver, instantiate all of the first term classes you are taking and populate those objects with data (use hard-coded data).

### LabResult

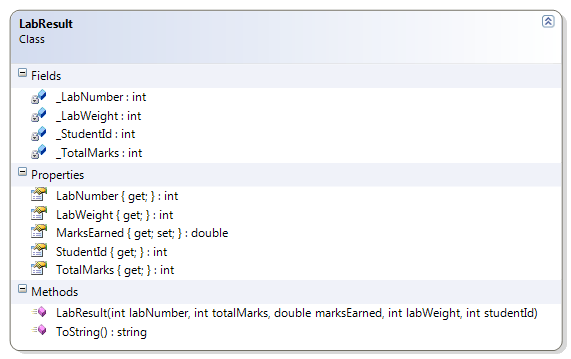
This class represents the results of a lab for a student.

## Problem Statement:

Create the **LabResult** class so that it represents the results of a lab submitted by a student. Design the class to have the fields, properties, and constructor as specified in this document. Also create a ToString() method that describes the information in any given instance of this class.

Use an auto-implemented property for the **MarksEarned**; use fields as the "backing store" for all other properties.

Note that the following properties should ***not*** have "set" implementations: **TotalMarks**, **LabNumber**, **LabWeight**, and **StudentId**.



Here are some XML comments to describe the ToString( ) method.

/// <summary>

/// This method overrides the default ToString() method to display

/// more meaningful information about this object.

/// </summary>

/// <returns>A string displaying the StudentId, MarksEarned, and

/// TotalMarks.</returns>

/// <remarks>

/// A call to this method (such as <c>Lab4.ToString()</c>)

/// would produce the following result:

/// <code>The student (200702694) received 24.5/35 for this lab.</code>

/// </remarks>

Also create a driver for testing this class; you may use any name for the driver as long as it is not already mentioned in this namespace. In the driver, instantiate all of the labs you have submitted to date in any course and populate those objects with data (use hard-coded data); if you haven't had any labs as of yet, then make up some data.

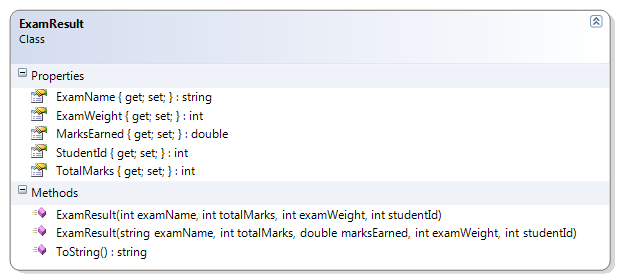
### ExamResult

This class represents the results of an exam for a student.

## Problem Statement:

Create the **ExamResult** class so that it represents the results of an exam written by a student. Design the class to have the properties and constructors as specified in this document. (Note that there are two constructors for this class.) Also create a ToString( ) method that describes the information in any given instance of this class.

Use auto-implemented properties for the entire class. Note that only the following properties should have ***private*** "set" implementations: **TotalMarks**, **ExamName**, **ExamWeight**, and **StudentId**.



Also create a driver for testing this class; you may use any name for the driver as long as it is not already mentioned in this namespace. In the driver, instantiate all of the exams you have taken to date in this course and populate those objects with data (use hard-coded data); if you need to, you may create fake data for the driver.