D – Testing and Debugging

Code Samples – Documentation

# Examples Overview

The following examples are used to illustrate this topic.

1. **Person** - This simple class was used to introduce the idea of encapsulation (private fields with public getters and setters). This example uses a constructor for ensuring the state of an object when it is instantiated (created). In this topic, it is used to demonstrate unit testing.
2. **Account** - This simple class also illustrates encapsulation, but with some of the fields being read-only. This class uses a constructor (which is also necessary for getting state into fields which do not have corresponding setter methods).
3. **Student** - This class reinforces the idea of encapsulation and constructors. It also demonstrates the idea of overloading the default ToString() method that every class inherits from the Object class.
4. **Employee** – The Employee class represents basic information about an employee of a company.
5. **Company** - These are other classes similar to the Person and Student classes. These classes, however, "share" a driver, illustrating the fact that any given "program" typically uses more than one class.

### Test Drivers (Console I/O)

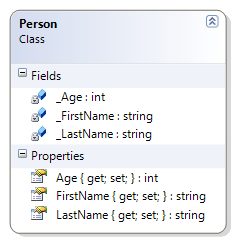
TBA

### Unit Tests (nUnit)

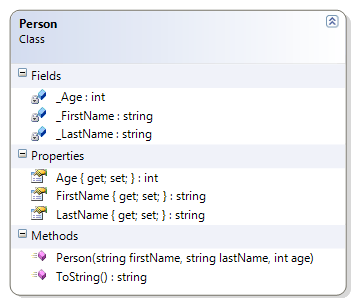
The unit tests for these classes are included in the solution folder under the "Demos + Practice" folder. Simply double-click the NUnit Test Project (.nunit) and the tests will open in the NUnit GUI.

#### Person

This simple class was used to introduce the idea of encapsulation (private fields with public getters and setters). This example uses a constructor for ensuring the state of an object when it is instantiated (created). In this topic, it is used to demonstrate unit testing. The following diagram represents the last design for the Person class, as completed in the exercises portion of the previous topic.



The supplied unit tests check all of this previously documented behaviour, but they also check to see if the Person class has overridden the ToString() method. Because this requirement is new, and has not yet been implemented, the unit tests show this as a “failed test.” The following diagram is the design that the unit tests are evaluating.

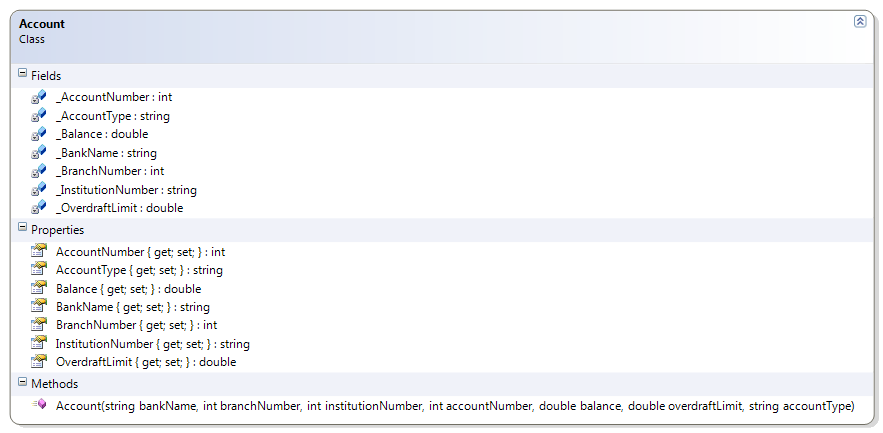


Should…

* Instantiate (build) from constructor
* Get/Set First Name
* Get/Set Last Name
* Get/Set Age
* **[*NOT-YET-IMPLEMENTED*]** Override ToString() to get the person’s full name (as first name then last name)  
  ***The test for this method will report as “failed” because it has not yet been implemented.***

#### Account

This simple class also illustrates encapsulation, but with some of the fields being read-only. This class uses a constructor (which is also necessary for getting state into fields which do not have corresponding setter methods).

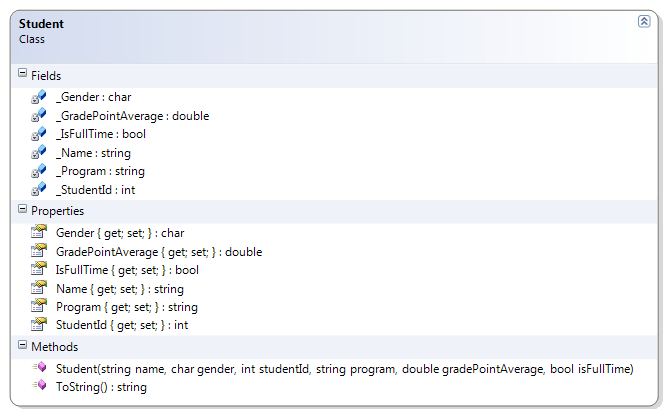


Should…

* Get Bank Name
* Get Branch Number
* Get Institution Number
* Get Account Number
* Get Account Type
* Get/Set Balance
* Get/Set Overdraft Limit

#### Student

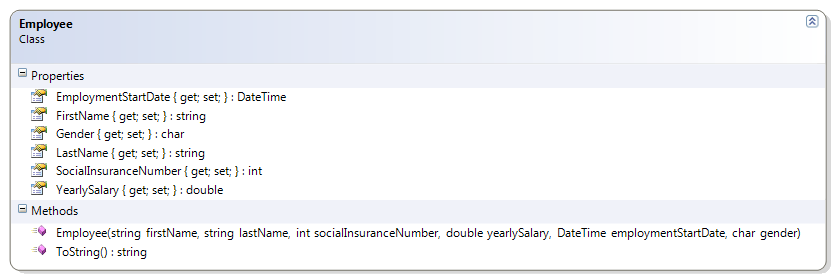
This class reinforces the idea of encapsulation and constructors. It also demonstrates the idea of overloading the default ToString() method that every class inherits from the Object class.



Should…

* Get/Set Name
* Get/Set Gender
* Get/Set Student Id
* Get/Set GPA
* Get/Set Program
* Get/Set Full-Time
* Override ToString() to get the student’s ID and name

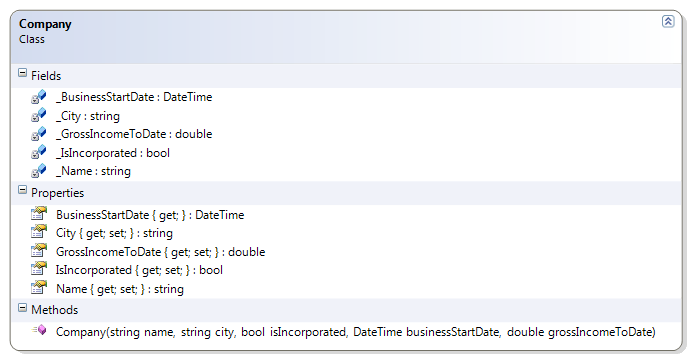
#### Employee



Should…

* Get/Set First Name
* Get/Set Last Name
* Get/Set Gender
* Get/Set Employment Date
* Get/Set Salary
* Get Social Insurance Number

#### Company



Should…

* Get/Set Name
* Get/Set City
* Get/Set Incorporated
* Get/Set Number of Employees
* Get/Set Gross Income to Date
* Get Business Start Date