Web Programming III (420-5W6-AB)  
Fall 2022

**Assignment # 1 (worth 6%)  
Due: Saturday, September 10th, 2022 at 23:59 on Lea**

*Deductions for late submissions is 10%/day, up to a maximum of 3 days.   
Better late than never.*

# Instructions

* Use the provided Visual Studio .Net Core C# Project and add all classes in MyClass project. MyClassTest are the unit tests.
* For each class you are creating, look at the code in the Unit Tests. It provides essential hints on the data types.
* Submit the provided project with your classes inside and your Main() method as Program.cs
* Do not modify the provided Unit Tests. You must add more unit tests for code coverage >= 95%.
* Create a file for each Class and add in its header:

/\*

\* Course: Web Programming 3

\* Assessment: Assignment 1

\* Created by: STUDENT NAME

\* Date: X September 202x

\* Class Name: Employee.cs

\* Description: Explain what the class stores and its functionality.

\*/

**Robust programming**:

1. All your classes must have the namespace Assignment1
2. The Unit tests do not have user input but still use a Try-Catch block.
3. Look at the Unit Tests to see the exact name of the Classes, constructors’ parameters, and Properties name. You will find in bold the **Class Name** or **Method Name** 🡺 use these exact names to facilitate unit testing.
4. Write more Unit Tests for code coverage of 100%. This is calculated by Visual Studio Enterprise edition (not available in Community edition).
5. Program.cs should contain code that tests all your code with user input. It does not contain unit tests.
6. For all Classes class overrides the ToString(), Equals() and GetHashCode() methods.
7. Points are awarded for excellent OOP design and efficient algorithms, otherwise points are deducted.
8. **Note**: if your unit tests do not run, delete the packages folder. VS will redownload all the packages and it works.
9. **Create a program that holds at least 25 employee’s information and randomly picks a set of 3 winners and a set of 5 winners**:   
   Platinum (first), Gold (second), Silver (third), Bronze (fourth), Fifth, Sixth, etc.   
   Avoid employee repetition.

An Employee is identified with:   
 Employee ID, First Name, Last Name, Department, Position, Yearly Salary.

Output all the information of the 3 and 5 winning employees.

Write a C# console application that demonstrates this program.

The Employee class is not static. Put these two methods in Util.cs Class:

//GenerateEmployees() should be implemented efficiently:

//avoid hard codding all 25 employees. Use random and loops to randomize data from //a set of first names, last names, departments, and positions.

public static List<Employee> GenerateEmployees()

//PickWinners implementation must be efficient

//and not affect the original Employee List

//Comment the algorithm used: how did you picked the winners?   
public static List<Employee> PickWinners(List<Employee> list, int winnersNum = 3)

Constructors’ parameters order:

public Employee(string name\_, uint idNumber\_)

public Employee(uint idNumber\_, string firstName\_, string lastName\_, string department\_, string position\_, decimal yearlySalary\_)

The exact Properties names can be found in the UnitTest.cs

1. **A company wants to keep track of the letters they send to clients.**

All letters have a date and recipient, but certified letters also have a tracking number. Create the Letter class and a child class CertifiedLetter with appropriate fields/properties. Include a default constructor and a constructor with parameters.

Override the ToString() method to return the type of letter and all its fields.

Write a C# console application that demonstrates your use of these classes and methods.

Constructors’ parameters order:

public Letter(DateTime date\_, string recipient\_)

public CertifiedLetter(long trackingNumber\_, DateTime date\_, string recipient\_)

The exact Properties names can be found in the UnitTest.cs

1. **A cookie baker wants to store cookie orders placed by customers.**

Each order contains a customer name, order number, quantity, type of cookie and total price. The price is calculated at $2.25 per cookie for any orders below two dozen, and $1.50 per cookie for any orders of two dozen or more.

Create a **CookieOrder** class and appropriate constructors/methods as required (including a **CalculatePrice**() method).

Occasionally special orders are placed (**SpecialCookieOrder**) and a description must be included in these to show why the order is special, e.g. gluten-free. This class should inherit from the **CookieOrder** class but also contain the field for the description. It should also override the **CalculatePrice** method for special orders where a handling fee of $5.00 must be added to the price.

Write a C# console application that demonstrates your use of these classes and methods.

Constructors’ parameters order:

public CookieOrder(string customerName\_, uint orderNumber\_, uint quantity\_, string cookieType\_)

public SpecialCookieOrder(string customerName\_, uint orderNumber\_, uint quantity\_, string cookieType\_, string description\_)

The exact Properties names can be found in the UnitTest.cs

1. Create a housing application for a property manager. Include a base class named

**Housing**. Include data characteristics such as address, year built, type of construction, cleaning crew name and if they ever had an insurance claim. Include a virtual method that returns.

Define an interface named **IUnits** that has a method that returns the total projected rental amount for the year: **decimal ProjectedRentalAmt()**.   
Create subclasses: **MultiUnit** and **SingleFamily**. The **MultiUnit** class implements the **IUnits** interface.

**SingleFamily** include characteristics such as size in square feet, number of garages, number of bedrooms and number of bathrooms and rent amount.

**MultiUnit** include characteristics such as the number of units, complex name, rent Amount Per Unit (assume it’s the same for all units).

Code the default constructor and 2 other constructors for each class.

Write a C# console application that demonstrates your use of these classes and methods.

public Housing(int year\_, string addr\_, string contructionType\_, string cleaningCrew\_, bool insuranceClaimHistory\_)

public SingleFamily(string addr\_, int beds\_, int baths\_, decimal rent\_)

public MultiUnits(string addr\_, int numberUnits\_, decimal rentAmt\_)

1. Write Unit Tests

For the teacher provided Unit Test to work, you must make sure:

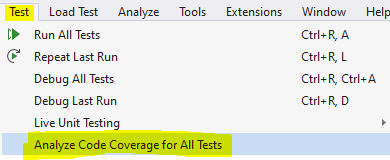
* All your classes to be public. Ex: public class CookieOrder
* All your classes must have the namespace Assignment1
* Incorporate all your classes in the provided Unit Test solution and make sure all tests passe.
* Look at the provided Units to make sure the constructer parameters types and numbers match, the Properties names match and the ToString() method is displaying the appropriate values.
* You must write Unit tests to have 100% code coverage.
* Follow Unit Tests Naming Convention
  + ClassNameAction. Ex: EmployeeIsEqual, EmployeeDifferentWinner
* Your code must pass all Unit tests, otherwise deductions will be imposed for each failed unit test.

When all tests pass:

Graphical user interface, text, application, chat or text message

Description automatically generated

* Verify the Code Coverage (Visual Studio Enterprise is required):



* Currently the tests cover is around 60% of the classes. Write more unit tests for code coverage 100%.

Graphical user interface

Description automatically generated