# Unit 2 - Assignment

This assignment will test your knowledge of the following areas:

* Coding classes with properties
* Coding a class that inherit from another
* Coding abstract classes
* Throwing exceptions
* Coding static classes

## Introduction

This phase of the project will have you begin development on a library

## Requirements

### Visual Studio Solution

Create a new Visual Studio Solution named RRCAGLibraryFirstLast, where First and Last correspond to your first and last names.

Create a Class Library project named RRCAGLibrary in the new solution named.

### SalesQuote Class

Add a class called SalesQuote to the RRCAGLibrary project. Copy your code from the previous assignment into this file.

Update the SalesQuote class to replace methods with properties.

#### **Properties**

+ VehicleSalePrice : decimal - Gets and sets the sale price of the vehicle.

+ TradeInAmount : decimal - Gets and sets the trade in amount.

+ AccessoriesChosen : Accessories - Gets and sets the accessories that were chosen.

+ ExteriorFinishChosen : ExteriorFinish - Gets and sets the exterior finish that was chosen.

+ AccessoryCost : decimal - Gets the cost of accessories chosen.

+ FinishCost : decimal - Gets the cost of the exterior finish chosen.

+ SubTotal : decimal - Gets the sum of the vehicle’s sale price and the Accessory and Finish Cost.

+ SalesTax : decimal - Gets the amount of tax to charge based on the subtotal.

+ Total : decimal - Gets the sum of the subtotal and taxes.

+ AmountDue : decimal - Gets the result of subtracting the trade-in amount from the total.

#### **Exceptions**

Update the SalesQuote class to include the following exceptions:

VehicleSalePrice Property

* ArgumentOutOfRangeException - When the property is set to less than or equal to 0. Message: “The value cannot be less than or equal to 0.”

TradeInAmount Property

* ArgumentOutOfRangeException - When the property is set to less than 0. Message: “The value cannot be less than 0.”

SalesQuote(decimal, decimal, decimal, Accessories, ExteriorFinish)

* ArgumentOutOfRangeException - When the vehicle sale price is less than or equal to 0. Message: “The argument cannot be less than or equal to 0.”
* ArgumentOutOfRangeException - When the trade in amount is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the sales tax rate is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the sales tax rate is greater than 1. Message: “The argument cannot be greater than 1.”

SalesQuote(decimal, decimal, decimal)

* ArgumentOutOfRangeException - When the vehicle sale price is less than or equal to 0. Message: “The argument cannot be less than or equal to 0.”
* ArgumentOutOfRangeException - When the trade in amount is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the sales tax rate is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the sales tax rate is greater than 1. Message: “The argument cannot be greater than 1.”

### Invoice Class

This abstract class contains functionality that supports the business process of creating an invoice.

**Note**: Do not add any additional fields, properties or methods to this class. Additions to the class diagram below will result in deductions during evaluation of your assignment.

Create the Invoice class in the Library project. This class must be defined under the Last.First.Business namespace.

#### **Fields**

- provincialSalesTaxRate : decimal - The provincial sales tax rate applied to the invoice.

- goodsAndServicesTaxRate : decimal - The goods and services tax rate applied to the invoice.

#### **Properties**

+ ProvincialSalesTaxRate : decimal - Gets and sets the provincial sales tax rate.

Exceptions:

* ArgumentOutOfRangeException - When the property is set to less than 0. Message: “The value cannot be less than 0.”
* ArgumentOutOfRangeException - When the property is set to greater than 1. Message: “The value cannot be greater than 1.”

+ GoodsAndServicesTaxRate : decimal - Gets and sets the goods and services tax rate.

Exception:

* ArgumentOutOfRangeException - When the property is set to less than 0. Message: “The value cannot be less than 0.”
* ArgumentOutOfRangeException - When the property is set to greater than 1. Message: “The value cannot be greater than 1.”

+ ProvincialSalesTaxCharged : decimal (Abstract) - Gets the amount of provincial sales tax charged to the customer.

+ GoodsAndServicesTaxCharged : decimal (Abstract) - Gets the amount of goods and services tax charged to the customer.

+ SubTotal : decimal (Abstract) - Gets the subtotal of the Invoice.

+ Total : decimal - Gets the total of the Invoice (Subtotal + Taxes).

#### **Methods**

+ Invoice(decimal, decimal) - Initializes an instance of Invoice with a provincial and goods and services tax rates.

Parameters:

* provincialSalesTaxRate - The rate of provincial tax charged to a customer.
* goodsAndServicesTaxRate - The rate of goods and services tax charged to a customer.

Exceptions:

* ArgumentOutOfRangeException - When the provincial sales tax rate is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the provincial sales tax rate is greater than 1. Message: “The argument cannot be greater than 1.”
* ArgumentOutOfRangeException - When the goods and services tax rate is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the goods and services tax rate is greater than 1. Message: “The argument cannot be greater than 1.”

### ServiceInvoice Class

This class contains functionality that supports the business process of creating an invoice for the service department. The ServiceInvoice class derives from the Invoice class.

**Note**: Do not add any additional fields, properties or methods to this class. Additions to the class diagram below will result in deductions during evaluation of your assignment.

Create the ServiceInvoice class in the Library project. This class must be defined under the Last.First.Business namespace.

#### **Properties**

+ LabourCost : decimal - Gets the amount charged for labour.

+ PartsCost : decimal - Gets the amount charged for parts.

+ MaterialCost : decimal - Gets the amount charged for shop materials.

+ ProvincialSalesTaxCharged : decimal - Gets the amount of provincial sales tax charged to the customer. Provincial Sales Tax is not applied to labour cost.

+ GoodsAndServicesTaxCharged : decimal - Gets the amount of goods and services tax charged to the customer.

+ SubTotal : decimal - Gets the subtotal of the Invoice.

+ Total : decimal - Gets the total of the Invoice.

**Note**: The LabourCost, PartsCost and MaterialCost properties have a private set accessor.

#### **Methods**

+ ServiceInvoice(decimal, decimal) - Initializes an instance of ServiceInvoice with a provincial and goods and services tax rates.

Parameters:

* provincialSalesTaxRate - The rate of provincial tax charged to a customer.
* goodsAndServicesTaxRate - The rate of goods and services tax charged to a customer.

Exceptions:

* ArgumentOutOfRangeException - When the provincial sales tax rate is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the provincial sales tax rate is greater than 1. Message: “The argument cannot be greater than 1.”
* ArgumentOutOfRangeException - When the goods and services tax rate is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the goods and services tax rate is greater than 1. Message: “The argument cannot be greater than 1.”

+ AddCost(CostType, decimal) : void - Increments a specified cost by the specified amount.

Parameters:

* costType - The type of cost being incremented.
* amount - The amount the cost is being incremented by.

Exceptions:

* ArgumentOutOfRangeException - When the amount is less than or equal to 0. Message: “The argument cannot be less than or equal to 0.”

### CarWashInvoice Class

This class contains functionality that supports the business process of creating an invoice for the car wash department. The CarWashInvoice class derives from the Invoice class.

**Note**: Do not add any additional fields, properties or methods to this class. Additions to the class diagram below will result in deductions during evaluation of your assignment.

Create the CarWashInvoice class in the Library project. This class must be defined under the Last.First.Business namespace.

#### **Fields**

- packageCost : decimal - The amount charged for the chosen package.

- fragranceCost : decimal - The amount charged for the chosen fragrance.

#### **Properties**

+ PackageCost : decimal - Gets and sets the amount charged for the chosen package.

Exception:

* ArgumentOutOfRangeException - When the property is set to less than 0. Message: “The value cannot be less than 0.”

+ FragranceCost : decimal - Gets and sets the amount charged for the chosen fragrance.

Exception:

- ArgumentOutOfRangeException - When the property is set to less than 0. Message: “The value cannot be less than 0.”

+ ProvincialSalesTaxCharged : decimal - Gets the amount of provincial sales tax charged to the customer. No provincial sales tax is charged for a car wash.

+ GoodsAndServicesTaxCharged : decimal - Gets the amount of goods and services tax charged to the customer.

+ SubTotal : decimal - Gets the subtotal of the Invoice.

+ Total : decimal - Gets the total of the Invoice.

#### **Methods**

+ CarWashInvoice(decimal, decimal) - Initializes an instance of CarWashInvoice with a provincial and goods and services tax rates. The package cost and fragrance cost are zero.

Parameters:

* provincialSalesTaxRate - The rate of provincial tax charged to a customer.
* goodsAndServicesTaxRate - The rate of goods and services tax charged to a customer.

Exceptions:

* ArgumentOutOfRangeException - When the provincial sales tax rate is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the provincial sales tax rate is greater than 1. Message: “The argument cannot be greater than 1.”
* ArgumentOutOfRangeException - When the goods and services tax rate is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the goods and services tax rate is greater than 1. Message: “The argument cannot be greater than 1.”

+ CarWashInvoice(decimal, decimal, decimal, decimal) - Initializes an instance of CarWashInvoice with a provincial and goods, services tax rate, package cost and fragrance cost.

Parameters:

* provincialSalesTaxRate - The rate of provincial tax charged to a customer.
* goodsAndServicesTaxRate - The rate of goods and services tax charged to a customer.
* packageCost - The cost of the chosen package.
* fragranceCost - The cost of the chosen fragrance.

Exceptions:

* ArgumentOutOfRangeException - When the provincial sales tax rate is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the provincial sales tax rate is greater than 1. Message: “The argument cannot be greater than 1.”
* ArgumentOutOfRangeException - When the goods and services tax rate is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the goods and services tax rate is greater than 1. Message: “The argument cannot be greater than 1.”
* ArgumentOutOfRangeException - When the package cost is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the fragrance cost is less than 0. Message: “The argument cannot be less than 0.

### CostType Enumeration

Namespace: Last.First.Business

The Accessories enumeration has the following values:

* Labour
* Part
* Material

### Financial Class

This static class contains functionality that includes financial functions.

**Note:** Do not add any additional fields, properties or methods to this class. Additions to the class diagram below will result in deductions during evaluation of your assignment.

Create the Financial class in the API project. This class must be defined under the Last.First.Business namespace.

#### **Methods**

+ GetPayment(decimal, int, decimal) : decimal - Returns the payment amount for an annuity based on periodic, fixed payments and a fixed interest rate.

Parameters:

* rate - the interest rate per period. For example, if the rate is an annual percentage rate (APR) of 10 percent and the customer makes monthly payments, the rate per period is 0.1/12, or 0.0083.
* numberOfPaymentPeriods - the total number of payment periods in the annuity. For example, if you make monthly payments on a four-year car loan, your loan has a total of 4 × 12 (or 48) payment periods.
* presentValue - the present value (or lump sum) that a series of payments to be paid in the future is worth now. For example, when a customer finances a car, the loan amount is the present value to the lender of the car payments the customer will make.

Exceptions:

* ArgumentOutOfRangeException - When the rate is less than 0. Message: “The argument cannot be less than 0.”
* ArgumentOutOfRangeException - When the rate is greater than 1. Message: “The argument cannot be greater than 1.”
* ArgumentOutOfRangeException - When the number of payments is less than or equal to zero. Message: “The argument cannot be less than or equal to 0.”
* ArgumentOutOfRangeException - When the present value is less than or equal to zero. Message: “The argument cannot be less than or equal to 0.”

The following is the code for calculating the payment:

**decimal** futureValue = 0;

**decimal** type = 0;

**return** (rate == 0) ? presentValue / numberOfPaymentPeriods : rate \* (futureValue + presentValue \* (**decimal**)Math.**Pow**((**double**)(1 + rate),

(**double**)numberOfPaymentPeriods)) / (((**decimal**)Math.**Pow**((**double**)(1 + rate), (**double**)numberOfPaymentPeriods) - 1) \* (1 + rate \* type));

## Program

Create a new Visual Studio Solution called Assignment2FirstLast, where First and Last correspond to your first and last names. The solution will contain a Console Application project named LibraryTest.

Copy the RRCAGLibrary.dll from the other Visual Studio solution (RRCAGLibraryFirstLast - created above) and put it in the Debug/bindirectory of your Console Application. Add a reference to the library (RRCAGLibrary.dll) in order to access the library classes.

Verify that all classes function correctly by testing all behaviors (including properties) for all their outcomes. Print the results to the console in this format:

### Sample Output

Testing method GetVehicleSalesPrice()

Test 1

Expected: 5000.00

Actual: 5000.00

Testing method SetVehicleSalesPrice(decimal)

Test 1

Expected: 9000.00

Actual: 5000.00

**Note**: DO NOT use keyboard input. Hard code all test values.