Programming II – Review

# Final Test (??? minutes)

## Prohibited Resources:

* Any online resources (websites, forums, AI tools , etc.).
* Code shared with or by others (including peers).

## Instructions:

* You must use the machine that is provided by the college i.e the computers in the labs.
* You will be monitored continuously by the instructor. Any violation will result in struct disciplinary action.
* There are ??? questions in this examination.
* Upload your work to Luminate.

Please read the instructions carefully before attempting this evaluation.

## Warning:

You may not use ANY techniques that was not covered in COMP123 so far.

## Vehicle class

You will implement the **Vehicle** Class in Visual Studio. The purpose of this abstract class is to serve as the parent of the two other main types in this application:

|  |
| --- |
| ***Vehicle***  Abstract Class |
| **Fields** |
|  |
| **Properties** |
| + «prop, private set» Make : string  + «prop, private set» Model : string  + «prop, private set» Year : int  + «virtual prop, no set» FullDescription : string  + «abstract prop, only get» Value : double |
| **Methods** |
| + «Constructor» Vehicle(  make : string,  model : bool,  year : int)  + «abstract» StartEngine : string |

### Fields:

There are no fields

### Properties:

There are three auto-implemented public properties.

1. Make – This string represents the manufacturer of the vehicle.
2. Model – This string represents the model of the vehicle.
3. Year – This int represents the year of manufacture of the vehicle.
4. FullDescription – This string represents the make, model and year of the vehicle.
5. Value – This double object will represents the retail price of the vehicle.

### Constructors:

1. public Vehicle(string make, string model, int year) – this public constructor takes two strings and a int argument and assign them to the relevant properties.

### Methods:

There is a single method that is also abstract.

1. public abstract string StartEngine() – this method will be implemented in a child class.

## Car class

Again, you will implement the **Car** Class in Visual Studio. This inherits from the **Vehicle** Class to provide a custom object for this application. It consists of three properties and a single a constructor and a single method:

|  |
| --- |
| ***Car***  Class  → Vehicle |
| **Fields** |
|  |
| **Properties** |
| + «prop, private set» DoorCount : int  + «override prop, no set» FullDescription : string  + «override prop, only get» Value : double |
| **Methods** |
| + «Constructor» Car(  make : string,  model : bool,  year : int)  + «override» StartEngine : string |

### Fields:

There are no fields

### Properties:

There is a single explicitly define property and five implicitly (comes from parent) defined properties.

1. DoorCount – This int represents the number of doors this car has.
2. FullDescription – This string represents the information from the parent, as well as the number of doors and the value of this car.
3. Value – This double represents the retail price of the vehicle. It is calculated with the formula below:  
   20\_000 - 1\_000 \* (DateTime.Now.Year + 1 - Year) + 10\_000 \* (6 - DoorCount)

### Constructors:

1. public Car(string make, string model, int year, int doorCount) – this public constructor takes two string, a bool and two int arguments and does the following:
   1. Invokes the base constructor with the appropriate arguments
   2. Assigns the last argument to the appropriate property.

### Methods:

There are no explicitly defined methods.

1. public override string StarEngine() – this public method returns the string takes two string, "Car engine started: Vroom!" .

## MotorCycle class

Again, you will implement the **MotorCycle** Class in Visual Studio. This inherits from the **Vehicle** Class. It consists of three properties and a single a constructor and a single method:

|  |
| --- |
| ***Car***  Class  → Vehicle |
| **Fields** |
|  |
| **Properties** |
| + «prop, private set» HasSideCar : bool  + «override prop, no set» FullDescription : string  + «override prop, only get» Value : double |
| **Methods** |
| + «Constructor» MotorCycle(  make : string,  model : bool,  year : int)  + «override» StartEngine : string |

### Fields:

There are no fields

### Properties:

There is a single explicitly define property and five implicitly (comes from parent) defined properties.

1. HasSideCar – This bool indicates the presence of a side car.
2. FullDescription – This string represents the information from the parent, as well as the presence of a side car and the value of this car.
3. Value – This double represents the retail price of the vehicle. It is calculated with the formula below:  
   15\_000 - 1\_000 \* (DateTime.Now.Year + 1 - Year) + 1\_000 \* (HasSideCar ? 1 : 0)

### Constructors:

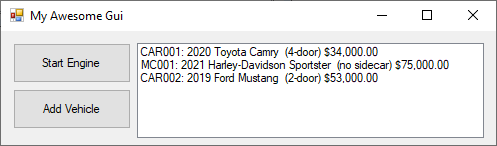
1. public MotorCycle(string make, string model, int year, bool hasSideCar) – this public constructor takes two string, an int and a bool arguments and does the following:
2. Invokes the base constructor with the appropriate arguments
3. Assigns the last argument to the appropriate property.

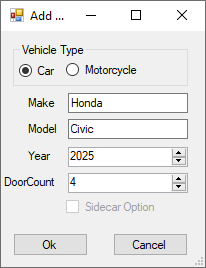
### Methods:

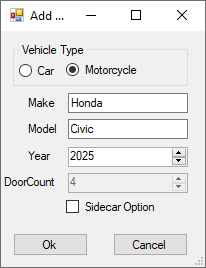
There are no explicitly defined methods.

1. public override string StarEngine() – this public method returns the string takes two string, "Motorcycle engine started: Vroom Vroom!" .

## Graphical User Interface

This application has two forms:





## Main Form

You need to build a GUI that satisfy the following:

1. A Dictionary is the data source that drives this UI.
2. There are wo buttons and a listbox.
3. The listbox is populated with three vehicles at startup
4. When the **Start Engine** button is clicked it does the following:
   1. If a vehicle is selected then the StartEngine is invoke on that object
   2. If a vehicle is not selected an appropriate a message box is displayed.
5. When the **Add Vehicle** button is clicked, it launches the second form.

## AddVehicle Form

You need to build a second GUI that satisfy the following:

1. Declare six properties with public getters and private setter that will store all the information of a Vehicle.
2. A private method that will initialize the gui to suitable states.
3. When the RadioButtons are checked, it enables or disables some control on the UI.
4. A Dictionary is the data source that drives this UI.
5. There are wo buttons and a listbox.
6. The listbox is populated with three vehicles at startup
7. When the **Ok** button is clicked it does the following:
   1. Performs some basic error checking: such as empty fields etc.
   2. Sets the above properties to reflect the values entered by the user.
   3. Set the **DialogResult** property of the form to **Dialog.OK** (this closes the form).
8. When the **Cancel** button is clicked, it closes the form.