Philip Petrosino

Object Oriented Programming

Prerequisites

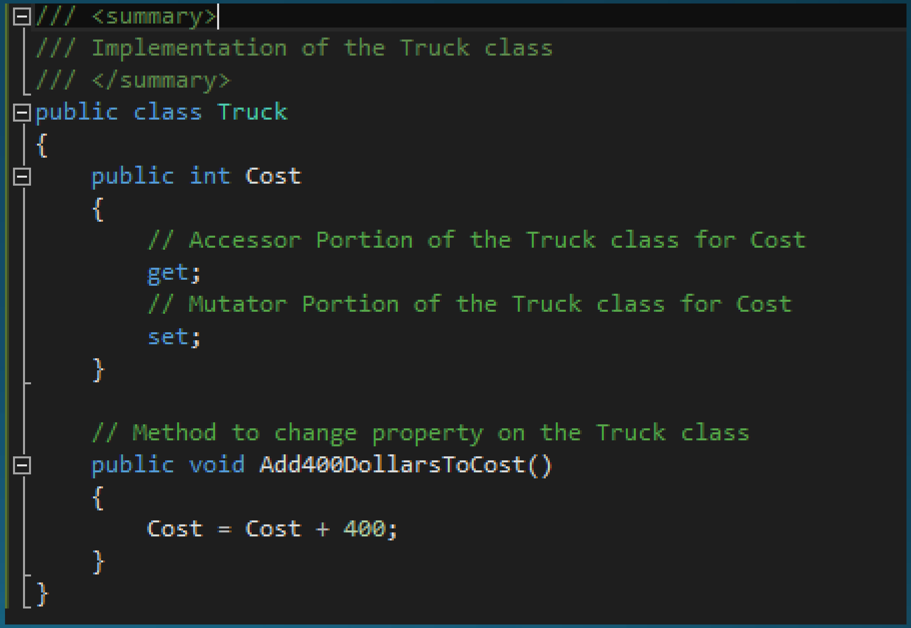
- In order to follow this tutorial you will need Visual Studio installed
- Visual Studio is free for students and can be downloaded at dreamspark.com

OOP (Object Oriented Programming)

- OOP is a programming language structure that uses objects instead of actions and data rather than logic.
- Most of the popular languages in OOP are class-based
    - Classes are objects that can have attributes, accessors and mutators
        - Properties can be anything from a description or an identifier on the object or class
        - Accessors allows things outside of the class to access information on the object or class
        - Mutators allows properties of the object to be changed outside of the class to change information on the object or class

Creating a class in C#

1. Create new class file
   1. File -> New -> File -> Select C# Class
2. Change the name of the class to Truck
3. Add the Cost property to Truck
   1. This allows you to change and get the Cost for your truck
4. Add the method Add400DollarsToCost
   1. When this method is used the Trucks Cost property will be 400 dollars more



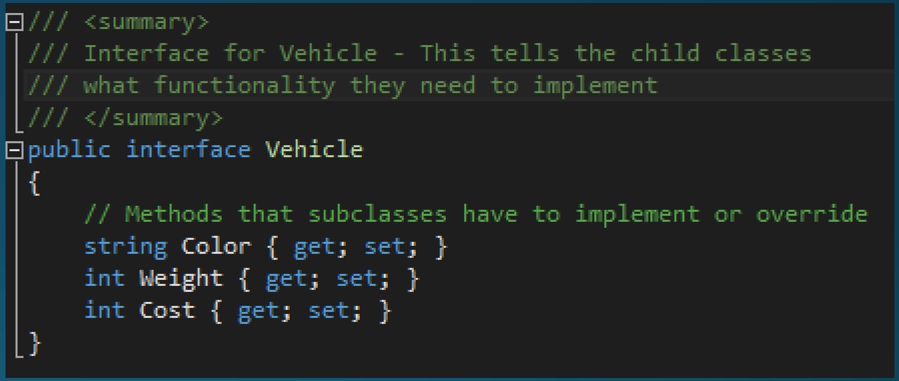
```
/// <summary>
/// Implementation of the Truck class
/// </summary>
public class Truck
{
    public int Cost
    {
        // Accessor Portion of the Truck class for Cost
        get;
        // Mutator Portion of the Truck class for Cost
        set;
    }

    // Method to change property on the Truck class
    public void Add400DollarsToCost()
    {
        Cost = Cost + 400;
    }
}
```

Abstraction

- Abstraction generalizes a concept to allow child classes to define the concepts in more detail
- Abstraction makes managing complex computer systems easier
- This is usually seen as an interface
  - Interfaces are used as a guideline class to ensure child classes implement all of the functionality
- Classes that use this interface must implement all of the functionality of the interface
- This ensures functionality is always there and very reusable making the process of adding a class a fraction of the time

Creating an Interface in C#

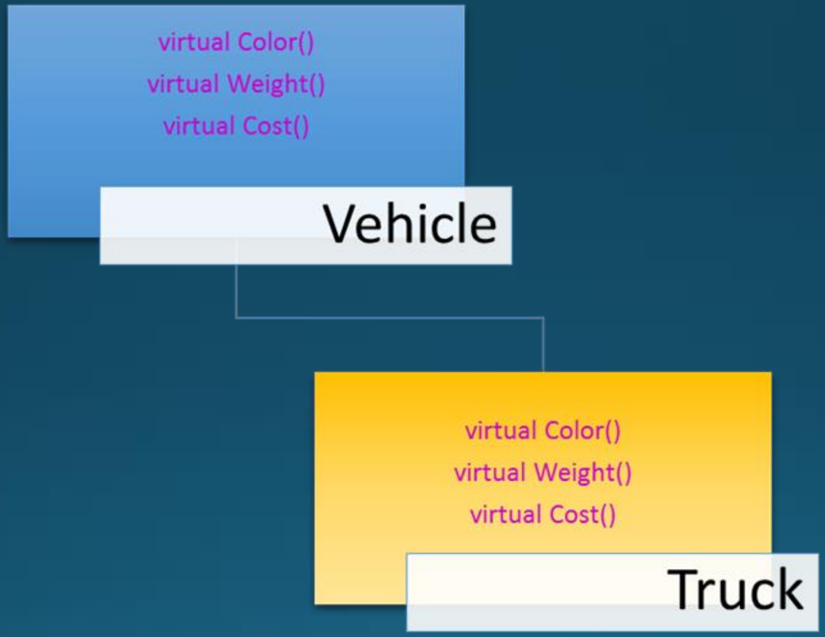
1. Create a file for the Vehicle interface
2. Unlike the previous class you mark this as a interface instead of class
3. Add the properties Color, Weight and Cost



```
/// <summary>
/// Interface for Vehicle - This tells the child classes
/// what functionality they need to implement
/// </summary>
public interface Vehicle
{
    // Methods that subclasses have to implement or override
    string Color { get; set; }
    int Weight { get; set; }
    int Cost { get; set; }
}
```

Abstraction Diagram

Abstraction class



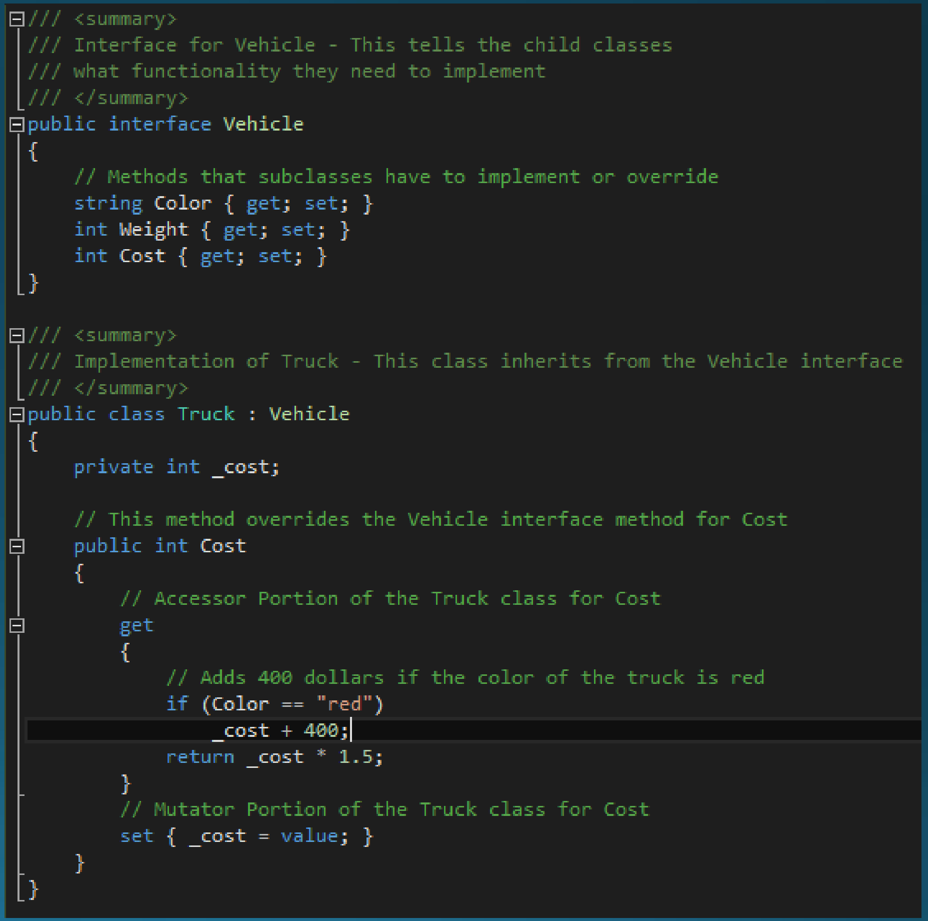
Normal class that implements the abstract class

Polymorphism

- Polymorphism allows a class to implement a parent class but have it’s own distinct functionality.
- Polymorphism can be used with abstract classes to implement subtyping
- Subtyping is where a child class inherits from a parent class but can also overwrite functionality inherited from the parent
- This allows multiple classes that are similar, like vehicles, to look very similar in code but can have different end results

Override Cost Accessor Portion

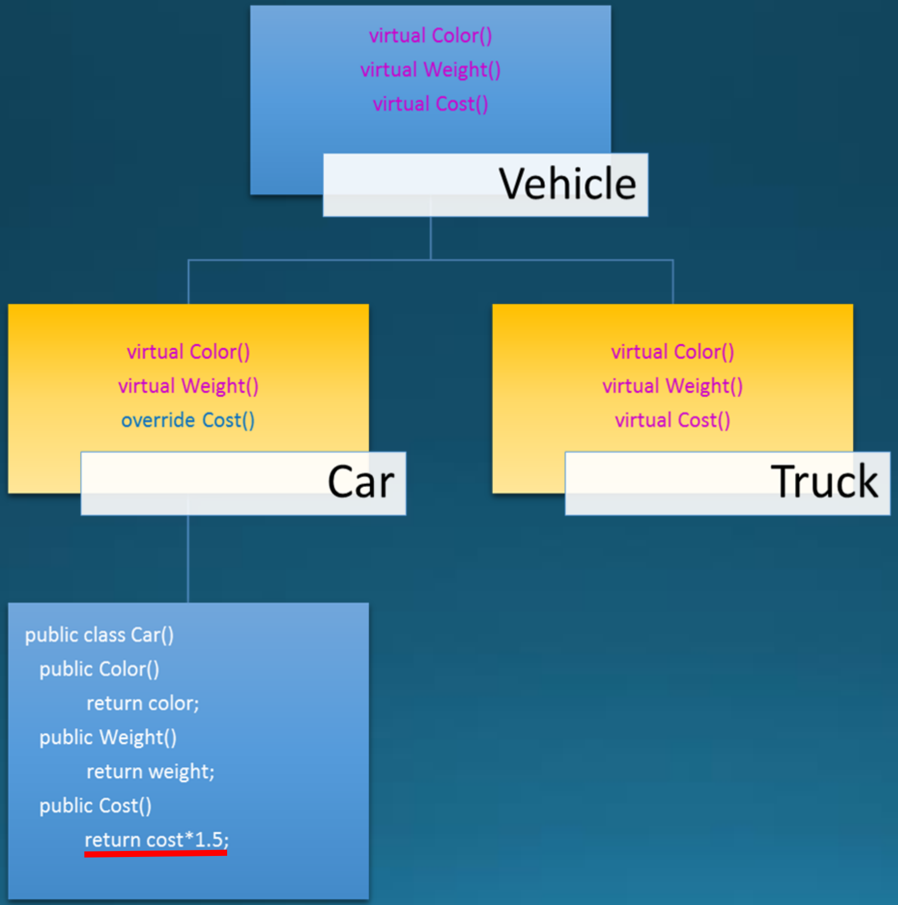
1. Add " : Vehicle" after Truck to inherit the Vehicle class
2. Add a private variable (that cannot be accessed outside of the class) to Truck
3. Add a condition that if the car's color is red then add $400 to the Cost
4. Multiply the cost by 1.5 for tax



```
/// <summary>
/// Interface for Vehicle - This tells the child classes
/// what functionality they need to implement
/// </summary>
public interface Vehicle
{
    // Methods that subclasses have to implement or override
    string Color { get; set; }
    int Weight { get; set; }
    int Cost { get; set; }
}

/// <summary>
/// Implementation of Truck - This class inherits from the Vehicle interface
/// </summary>
public class Truck : Vehicle
{
    private int _cost;

    // This method overrides the Vehicle interface method for Cost
    public int Cost
    {
        // Accessor Portion of the Truck class for Cost
        get
        {
            // Adds 400 dollars if the color of the truck is red
            if (Color == "red")
                _cost + 400;
            return _cost * 1.5;
        }
        // Mutator Portion of the Truck class for Cost
        set { _cost = value; }
    }
}
```

Polymorphism Diagram

Parent Abstract Class



```
public class Car()
  public Color()
          return color;
  public Weight()
          return weight;
  public Cost()
          return cost*1.5;
```

Normal inheritance

Implements parent class but differs in how much the vehicle costs