**Final Project: Polymorphism, Interfaces, Virtual and Override Methods, Stacks, and Queues**

**Objective:  
In this assignment, you will apply the concepts of polymorphism, interfaces, virtual and override methods, stacks, and queues to create a race simulation program in C#.**

**Instructions:**

1. **Create a Superclass or Interface:**
   * **Define a Vehicle superclass with the following properties and methods:**
     + **Name (string)**
     + **Color (string)**
     + **Speed (int)**
     + **A method named Move() to simulate the vehicle's movement.**
2. **Create Derived Classes:**
   * **Implement four classes that inherit from the Vehicle superclass. These classes should represent different types of vehicles: Car, Bicycle, Motorbike, and Truck.**
   * **Override the Move() method in each derived class to define specific movement behavior as needed.**
   * **The Bicycle, Car, and Motorbike classes should have an additional property Acceleration (int):**
     + **Bicycle: Acceleration is negative (simulating deceleration).**
     + **Car: Positive acceleration, less than 10.**
     + **Motorbike: Positive acceleration, between 10 and 15.**
3. **Simulate a Race:**
   * **Create a queue or stack that holds instances of these vehicle objects.**
   * **Instantiate a couple of objects of each vehicle type and place them in the queue or stack.**
   * **Provide an option for the user to select 4 objects to participate in the race.**
   * **Each vehicle object should have a name, color, and speed.**
   * **The Car, Motorbike, and Bicycle objects should also have acceleration.**
   * **Use a progress bar to represent the movement of each vehicle:**
     + **The progress bar’s color should match the vehicle’s color.**
     + **The progress bar’s label should be set to the vehicle’s name.**
     + **The progress bar’s progress should increase according to the vehicle's speed.**
     + **For vehicles with acceleration, the progress bar's value should change according to the formula: value += speed + ½ \* acceleration.**
     + **For vehicles without acceleration, the progress bar's value should change according to the formula: value += speed.**
4. **Start and Reset the Race:**
   * **Add a button to the form that, when clicked, starts the race. The vehicles will move based on their speed and acceleration, and the first vehicle to reach the end of the progress bar wins the race.**
   * **Add a reset button that, when clicked, will reset all progress bars and allow the race to be started again.**
5. **Submission:**
   * **Submit the complete C# project, including all source files and a brief explanation of your implementation.**

**Evaluation Criteria:**

* **Correct implementation of the classes.**
* **Appropriate overriding of the Move() method in each class.**
* **Smooth and realistic movement of each vehicle.**
* **Correct functionality and error-free execution of the program.**
* **Proper use of WinForms to display the race.**
* **Attention to SWAG (Style, Aesthetics, and General Appeal)**

Today, I am going to present my C# project which is my Final project Race Simulation program. This will demonstrate the use of object oriented programming principles such as inheritance, polymorphism, and method overriding. It also incorporates GUI elements like progress bars to visually simulate a race between different types of vehicles.

I’ve created a simple Windows Forms application where users can select vehicles to participate in a race. The vehicles include Cars, Motorbikes, Bicycles, and Trucks, each with unique properties like speed and acceleration. The race is visually represented using progress bars, where the progress of each vehicle is shown, and the first to reach 100% wins the race.

At the core of this program is the Vehicle super class, which defines common properties like Name, Color, and Speed. It also has a virtual Move method, which is overridden by the derived classes.

Each specific type of vehicle—Car, Motorbike, Bicycle, and Truck—inherit from the Vehicle class. Cars and Motorbikes have an additional Acceleration property, which affects how quickly they move. The Truck and Bicycle classes also override the Move method to account for their unique behaviors

The main logic of the program is implemented in Form1.cs

First, I initialize all available vehicles in the InitializeVehicles method. This method creates instances of different vehicles like cars, bicycles, motorbikes, and trucks. Each vehicle has a name, color, speed, and, for some vehicles, an acceleration property.

These vehicles are then added to specific ListBox controls—one for each type of vehicle. This allows the user to choose which vehicles will participate in the race.

Vehicles are stored in a list called availableVehicles.

The user can select vehicles from the list boxes, which are populated with vehicle names during initialization.

When a user selects a vehicle and clicks 'Add to Race,' the selected vehicles are added to a queue called raceQueue. This queue represents the vehicles that will compete in the race.

For each vehicle added, we also update labels (labelCar, labelBicycle, etc.) to display the names of the selected vehicles. This makes it clear to the user which vehicles are queued for the race. If fewer than four vehicles are selected, a message is displayed asking the user to select exactly four vehicles.

Once four vehicles are selected, the user can click 'Start Race,' which triggers the race. A timer ticks, and each vehicle’s progress bar is updated based on its speed and acceleration.

The timer triggers the timerRace\_Tick event at regular intervals (every 100 milliseconds), which in turn calls the UpdateProgressBars method to update the progress of each vehicle.

The UpdateProgressBars method is where the actual race logic happens. For each vehicle in the queue, the method updates its progress bar based on the vehicle's speed and, if applicable, acceleration.

The progress bars also change color to match the vehicle’s color, making the race visually intuitive.

The race continues until one of the vehicles reaches the end of its progress bar, at which point the race stops, and the winner is displayed.

After the race, the user can click the 'Reset' button to clear the selected vehicles, reset the progress bars, and start a new race.

The reset functionality ensures that the user can easily set up a new race without any residual effects from the previous race.