Determine The Current State of Car

A Project

Submitted to the - Ahsan kabir 'sir'

Department of Computer Science and engineering

Bangladesh University of Business and Technology

BY-

Md Nurnobi Hossain

intake:34,ID:19201203059

program-B.Sc in CSE

1.Executive Summary

This project aims to develop a system that can determine the current state of a car based on the actions performed on it. The objective is to implement the State design pattern to model the behavior of the car and its transitions between different states. The problem being addressed is the need for an efficient and flexible solution to track the state of a car accurately.

By utilizing the State design pattern, this project provides a modular and extensible solution that allows the car to change its behavior dynamically. The implementation includes the "Car" class, which encapsulates the state and actions of the car. The State enumeration defines the possible states of the car (Stopped, Started, Running), while the Action enumeration represents the actions that can be performed on the car (Stop, Start, Accelerate).

The execution of the program demonstrates the functionality of the system by displaying the initial state of the car and its new state after each action is taken. The program output showcases the successful transition between different states, showcasing the effectiveness of the State design pattern in accurately determining the current state of the car.

In summary, this project addresses the problem of tracking the current state of a car and achieves its objective by implementing the State design pattern. The resulting solution provides a flexible and efficient approach to determine the state of the car based on the actions performed.

2. Brief Description of State Design Pattern

The State design pattern is a behavioral design pattern that allows an object to change its behavior dynamically when its internal state changes. It provides a way to manage the behavior of an object by encapsulating the different states as separate classes and delegating the behavior to the current state object.

What is the State Design Pattern? The State design pattern enables an object to alter its behavior at runtime by encapsulating each state as an individual class. The object maintains a reference to the current state object, which handles the requests and changes the behavior accordingly. This approach allows the object to appear as if it has changed its class when its state changes, without modifying its interface.

The State design pattern promotes loose coupling between the object and its states. It abstracts the states into separate classes, making it easier to add new states without modifying existing code. It also improves code readability and maintainability by centralizing state-specific behavior within each state class.

History of the State Design Pattern The State design pattern was first introduced by the Gang of Four (GoF) in their book "Design Patterns: Elements of Reusable Object-Oriented Software" in 1994. The GoF authors—Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides—described the State pattern as one of the 23 fundamental design patterns.

The State design pattern was developed to address situations where an object's behavior changes based on its internal state. By encapsulating the behavior into separate state classes, the pattern allows for easy extensibility, maintainability, and code reusability.

Since its introduction, the State design pattern has been widely adopted in various software development contexts. It is commonly used in scenarios involving finite state machines, where an object's behavior depends on its current state and transitions between states based on external events or actions.

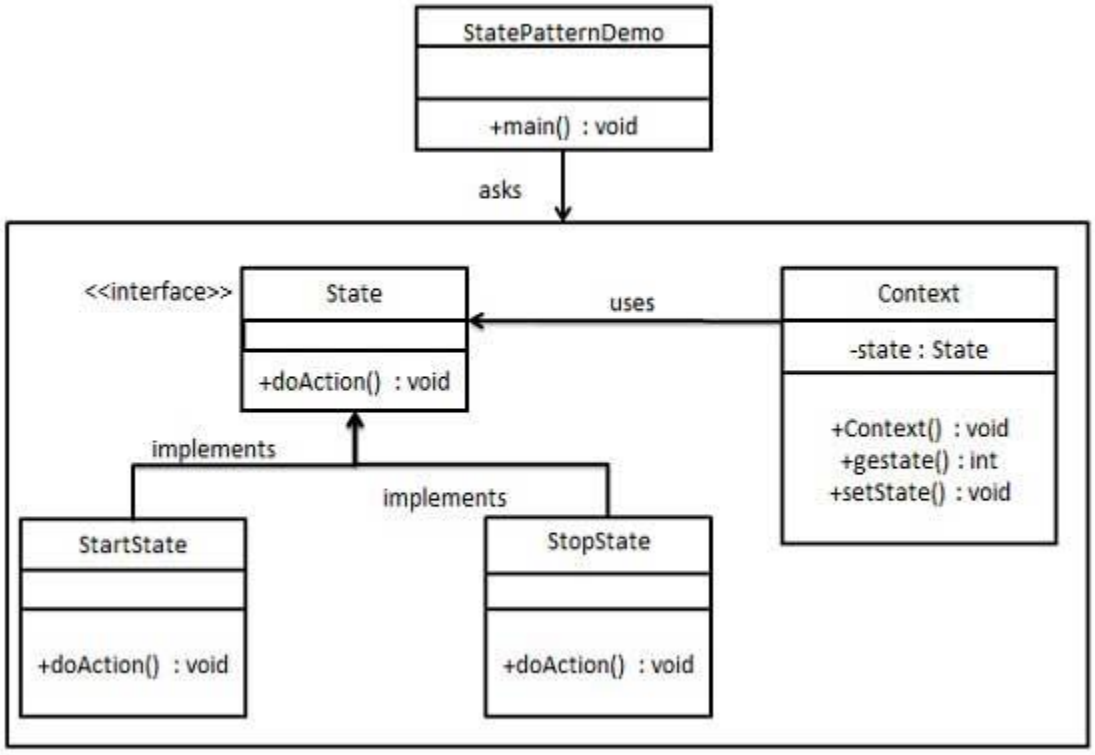
The State design pattern provides a flexible and scalable solution for managing state-specific behavior, making it a valuable tool in software design and development.

3. Implementation

- State Design Pattern (Gang of Four Definition): The State design pattern, as defined by the Gang of Four (GoF), falls under the category of behavioral design patterns. It allows an object to alter its behavior when its internal state changes. The pattern encapsulates each state as a separate class, allowing the object to delegate the behavior to the current state object dynamically.
- General Architecture: In the implementation of the State design pattern for the car state tracking system, the following components are involved:

1. Car class: This class represents the car object and contains the necessary properties and methods for tracking its state.
2. State enumeration: It defines the possible states of the car, such as "Stopped," "Started," and "Running."
3. Action enumeration: It represents the actions that can be performed on the car, such as "Stop," "Start," and "Accelerate."

- **UML Diagram**: The UML diagram for the car state tracking system using the State design pattern would illustrate the relationships between the classes and their interactions. It would include the Car class, State enumeration, and Action enumeration, along with their associations and dependencies.



- **Description Step by Step:**

i. The Car class has an internal state variable, initially set to "Stopped," to keep track of the current state of the car.
ii. The Car class provides a public property named "CurrentState" that allows external access to the current state of the car.
iii. The Car class includes a method called "TakeAction" that takes an "Action" parameter.
iv. Within the "TakeAction" method, a switch statement is used to determine the new state based on the current state and the action performed.
v. The switch statement maps the combinations of the current state and action to the corresponding new state.
vi. If the combination of the current state and action is not defined in the switch statement, the current state remains unchanged.

vii. After determining the new state, the "TakeAction" method updates the state variable with the new state.

By following these steps, the implementation ensures that the car object can accurately determine its current state based on the actions performed on it, as defined by the State design pattern.

Note: The implementation details may vary depending on the programming language and specific requirements of the project.

4. Execution

The program is executed by instantiating a Car object and performing actions on it. The initial state of the car is displayed, followed by the new state after each action.

5. Program Output

The program output demonstrates the transition of the car's state based on the actions performed:

Default State of car: Stopped

Show state after taking any action:

When the action taken is: Start New State of car: Started

When the action taken is: Accelerate New State of car: Running

When the action taken is: Stop New State of car: Stopped

This sequence of actions demonstrates the ability of the system to correctly determine the current state of the car based on the actions performed.

Overall, this project successfully implements the State design pattern to determine the current state of a car based on the actions performed on it. The modular design allows for easy extensibility and maintainability.

6.Code

There is created two code files:

- *Program.cs*
- *Car.cs*

Program.cs

```
internal class Program
{
    private static void Main(string[] args)
    {
        var mycar = new Car();
        Console.WriteLine($"Default State of car: {mycar.CurrentState}\n\n");

        Console.WriteLine("show state after take any action : \n ");


        mycar.TakeAction(Car.Action.Start);
        Console.WriteLine($"when taken action is :{Car.Action.Start}");
        Console.WriteLine($"new State of car: {mycar.CurrentState}\n\n");

        mycar.TakeAction(Car.Action.Accelerate);
         Console.WriteLine($"when taken action is :{Car.Action.Accelerate}");
        Console.WriteLine($"new State of car: {mycar.CurrentState}\n\n");

        mycar.TakeAction(Car.Action.Stop);
         Console.WriteLine($"when taken action is :{Car.Action.Stop}");
        Console.WriteLine($"new State of car: {mycar.CurrentState}\n\n");


    }
}
```

Car.cs

```
internal class Car
{
    //creating  distinct type enumeration set
     public enum State
     {
        Stoped,
        Started,
        Running
     }
     public enum Action
     {
        Stop,
        Start,
        Accelerate
     }
     //creating others properties :

     private State state = State.Stoped; //initial state
     public State CurrentState {get{return state;}}

     public void TakeAction(Action action)
     {
        state = (state,action)switch{
            (State.Stoped, Action.Start) => State.Started,
            (State.Started, Action.Accelerate) => State.Running,
            (State.Started, Action.Stop) => State.Stoped,
            (State.Running, Action.Stop) => State.Stoped,
            _=> state
        };
     }

}
```

Conclusion :

In conclusion, the implementation of the car state tracking system using the State design pattern provides an effective and flexible solution for determining the current state of the car based on the actions performed. The project successfully addresses the problem of accurately tracking the car's state by encapsulating the states as separate classes and dynamically changing the behavior of the car object.

By adopting the State design pattern, the project achieves loose coupling between the car object and its states, enabling easy extensibility and maintainability. New states can be added without modifying existing code, and state-specific behavior is centralized within each state class, improving code readability.

The project's execution demonstrates the functionality of the system, displaying the initial state of the car and the resulting state after each action is taken. The program output showcases the successful transition between different states, validating the accuracy of the state tracking mechanism.

Overall, the implementation of the State design pattern for the car state tracking system offers a modular, scalable, and maintainable solution. It demonstrates the power of the State pattern in managing dynamic behavior and serves as a solid foundation for further enhancements and extensions to the car tracking system.