ПЗ 03.01 Работа с поведенческими паттернами
Тамбовцева Александра, 3пк2

Задание: реализовать проекты по предметной области в соответствии с вариантом.

Содержание отчета:

1. Описание предметной области
2. UML диаграмма классов реализованного проекта
3. Краткий вывод по использованию паттерна в текущей задаче
4. Листинг кода
5. Отчет в виде doc или pdf загурзить в репозиторий.

Вариант:

	1 проект. (Шаблонный метод)	2 проект. (Состояние)	3 проект (команда).
5	Напишите приложение, которое отправляет уведомления различными способами (по электронной почте, SMS, через push-уведомления), используя шаблонный метод для общего алгоритма отправки.	Спроектируйте систему обработки заказов с состояниями, такими как "Новый заказ", "В обработке", "Отправлено", "Доставлено", "Отменено". Изменение состояния должно определять доступные действия.	Реализуйте симуляцию магазина, где пользователи могут выполнять команды "Добавить товар в корзину", "Удалить товар из корзины" и "Оформить заказ" с возможностью отмены операций.

1 проект: Шаблонный метод

1. Описание предметной области

Проект реализует систему отправки уведомлений различными способами: по электронной почте, SMS и через push-уведомления. Используется паттерн «Шаблонный метод», который определяет общий алгоритм отправки уведомлений, но позволяет подклассам переопределять отдельные шаги (подготовка уведомления, отправка и подтверждение доставки).

2. UML диаграмма классов

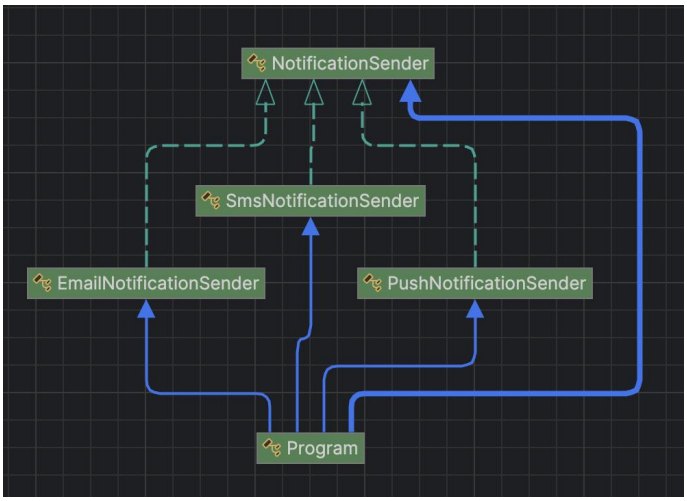


Рисунок 1 - UML диаграмма классов проекта «TemplateMethod»

3. Краткий вывод по использованию паттерна

Паттерн «Шаблонный метод» позволяет выделить общий алгоритм отправки уведомлений в базовом классе «NotificationSender», оставляя реализацию конкретных шагов на усмотрение подклассов «EmailNotificationSender», «SmsNotificationSender», «PushNotificationSender». Это упрощает добавление новых способов отправки уведомлений и обеспечивает единообразие в выполнении алгоритма.

4. Листинг кода

```
internal abstract class NotificationSender
{
    1 usage  3 overrides
    public abstract void PrepareNotification();
    1 usage  3 overrides
    public abstract void Send(string message);
    1 usage  3 overrides
    public abstract void ConfirmationSending();

    /// <summary>
    /// шаблонный метод
    /// </summary>
    3 usages
    public void SendNotification(string message)
    {
        PrepareNotification();
        Send(message);
        ConfirmationSending();
    }
}
```

Рисунок 2 - Класс «NotificationSender»

```
internal class EmailNotificationSender : NotificationSender
{
    0+1 usages
    public override void PrepareNotification()
    {
        Console.WriteLine("подготовка email уведомления...");
    }

    0+1 usages
    public override void Send(string message)
    {
        Console.WriteLine($"отправка уведомления по email: \"{message}\"");
    }

    0+1 usages
    public override void ConfirmationSending()
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("email уведомление отправлено");
        Console.ForegroundColor = ConsoleColor.White;
    }
}
```

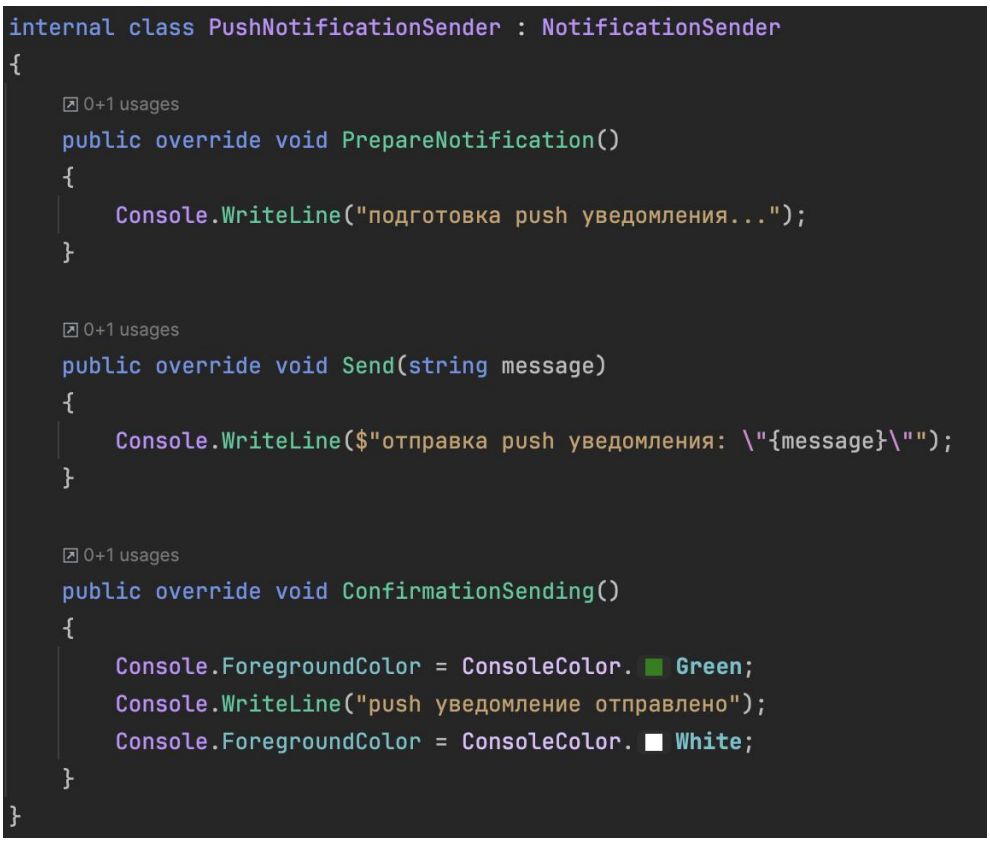
Рисунок 3 - Класс «EmailNotificationSender»

```
internal class SmsNotificationSender : NotificationSender
{
    0+1 usages
    public override void PrepareNotification()
    {
        Console.WriteLine("подготовка SMS уведомления...");
    }

    0+1 usages
    public override void Send(string message)
    {
        Console.WriteLine($"отправка уведомления по SMS: \"{message}\"");
    }

    0+1 usages
    public override void ConfirmationSending()
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("SMS уведомление отправлено");
        Console.ForegroundColor = ConsoleColor.White;
    }
}
```

Рисунок 4 - Класс «SmsNotificationSender»



```
internal class PushNotificationSender : NotificationSender
{
    0+1 usages
    public override void PrepareNotification()
    {
        Console.WriteLine("подготовка push уведомления...");
    }

    0+1 usages
    public override void Send(string message)
    {
        Console.WriteLine($"отправка push уведомления: \"{message}\"");
    }

    0+1 usages
    public override void ConfirmationSending()
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("push уведомление отправлено");
        Console.ForegroundColor = ConsoleColor.White;
    }
}
```

Рисунок 5 - Класс «PushNotificationSender»

```
internal class Program
{
    static void Main(string[] args)
    {
        NotificationSender emailSender = new EmailNotificationSender();
        NotificationSender smsSender = new SmsNotificationSender();
        NotificationSender pushSender = new PushNotificationSender();

        emailSender.SendNotification( message: "Вы можете забрать свою посылку");
        Console.WriteLine();
        smsSender.SendNotification( message: "Вы можете забрать свою посылку");
        Console.WriteLine();
        pushSender.SendNotification( message: "Вы можете забрать свою посылку");
    }
}
```

Рисунок 6 - Класс «Program»

2 проект: Состояние

1. Описание предметной области

Проект реализует систему обработки заказов с состояниями: "Новый заказ", "В обработке", "Отправлено", "Доставлено", "Отменено". Используется паттерн «Состояние», который позволяет объекту изменять своё поведение в зависимости от внутреннего состояния.

2. UML диаграмма классов

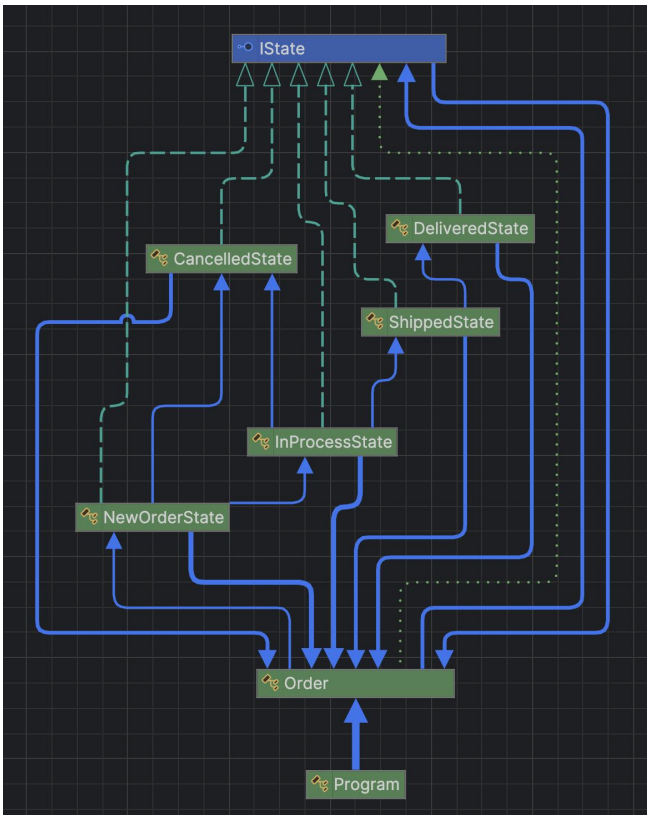


Рисунок 7 - UML диаграмма классов проекта «StatePattern»

3. Краткий вывод по использованию паттерна

Паттерн «Состояние» позволяет инкапсулировать поведение, связанное с каждым состоянием, в отдельные классы «NewOrderState», «InProcessState», «ShippedState», «DeliveredState», «CancelledState». Это упрощает добавление новых состояний и делает код более читаемым и поддерживаемым. Класс «Order» делегирует выполнение операций текущему состоянию, что делает систему гибкой и легко расширяемой.

4. Листинг кода

```
7 usages  5 inheritors
internal interface IState
{
    1 usage  5 implementations
    void ProcessOrder(Order order);
    1 usage  5 implementations
    void ShipOrder(Order order);
    1 usage  5 implementations
    void DeliverOrder(Order order);
    1 usage  5 implementations
    void CancelOrder(Order order);
}
```

Рисунок 8 - Интерфейс «IState»

```
30 usages
internal class Order
{
    private IState _state;

    3 usages
    public Order()
    {
        _state = new NewOrderState();
    }

    5 usages
    public void SetState(IState state)
    {
        _state = state;
    }

    3 usages
    public void ProcessOrder()
    {
        _state.ProcessOrder(this);
    }

    3 usages
    public void ShipOrder()
    {
        _state.ShipOrder(this);
    }

    3 usages
    public void DeliverOrder()
    {
        _state.DeliverOrder(this);
    }

    3 usages
    public void CancelOrder()
    {
        _state.CancelOrder(this);
    }
}
```

Рисунок 9 - Класс «Order»

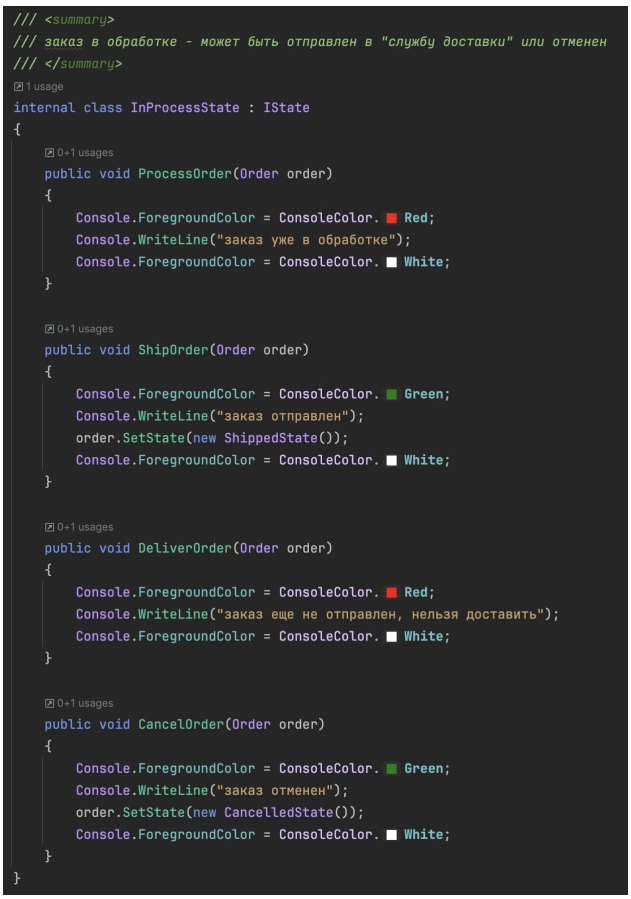
```
/// <summary>
/// новый заказ - может быть отправлен в обработку или отменен
/// </summary>
1 usage
internal class NewOrderState : IState
{
    0+1 usages
    public void ProcessOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("заказ передан в обработку");
        order.SetState(new InProcessState());
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void ShipOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("заказ еще не обработан, нельзя отправить");
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void DeliverOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("заказ еще не обработан, нельзя доставить");
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void CancelOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("заказ отменен");
        order.SetState(new CancelledState());
        Console.ForegroundColor = ConsoleColor.White;
    }
}
```

Рисунок 10 - Класс «NewOrderState»



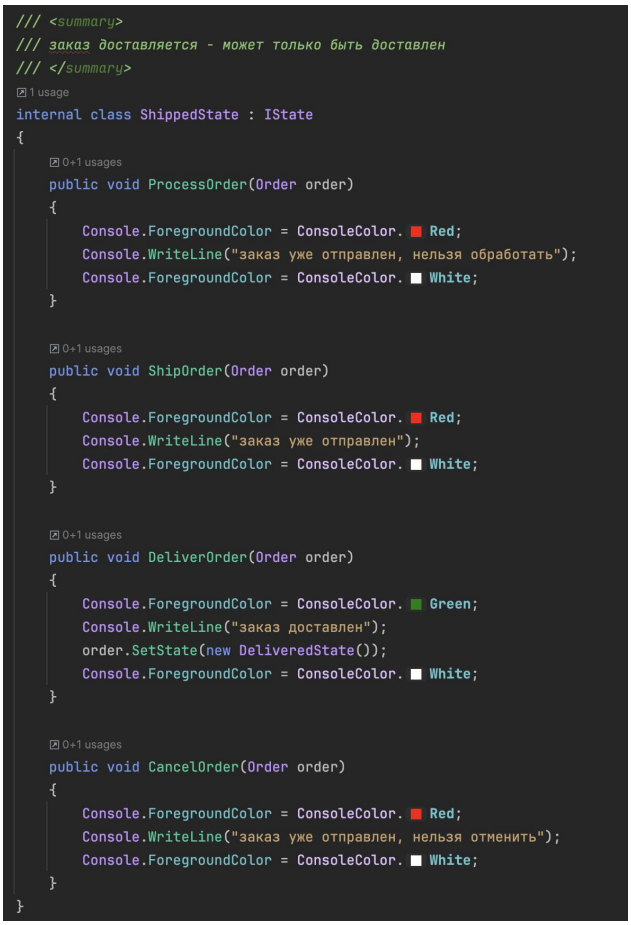
```
/// <summary>
/// заказ в обработке - может быть отправлен в "службу доставки" или отменен
/// </summary>
1 usage
internal class InProcessState : IState
{
    0+1 usages
    public void ProcessOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("заказ уже в обработке");
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void ShipOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("заказ отправлен");
        order.SetState(new ShippedState());
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void DeliverOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("заказ еще не отправлен, нельзя доставить");
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void CancelOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("заказ отменен");
        order.SetState(new CancelledState());
        Console.ForegroundColor = ConsoleColor.White;
    }
}
```

Рисунок 11 - Класс «InProcessState»



```
/// <summary>
/// заказ доставляется - может только быть доставлен
/// </summary>
1 usage
internal class ShippedState : IState
{
    0+1 usages
    public void ProcessOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("заказ уже отправлен, нельзя обработать");
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void ShipOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("заказ уже отправлен");
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void DeliverOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("заказ доставлен");
        order.SetState(new DeliveredState());
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void CancelOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("заказ уже отправлен, нельзя отменить");
        Console.ForegroundColor = ConsoleColor.White;
    }
}
```

Рисунок 12 - Класс «ShippedState»

```
/// <summary>
/// заказ доставлен - никаких действий, он уже доставлен
/// </summary>
1 usage
internal class DeliveredState : IState
{
    0+1 usages
    public void ProcessOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("заказ уже доставлен, нельзя обработать");
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void ShipOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("заказ уже доставлен, нельзя отправить");
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void DeliverOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("заказ уже доставлен");
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void CancelOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("заказ уже доставлен, нельзя отменить");
        Console.ForegroundColor = ConsoleColor.White;
    }
}
```

Рисунок 13 - Класс «DeliveredState»

```
/// <summary>
/// заказ отменен - никаких действий, он уже отменен
/// </summary>
2 usages
internal class CancelledState : IState
{
    0+1 usages
    public void ProcessOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("заказ отменен, нельзя обработать");
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void ShipOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("заказ отменен, нельзя отправить");
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void DeliverOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("заказ отменен, нельзя доставить");
        Console.ForegroundColor = ConsoleColor.White;
    }

    0+1 usages
    public void CancelOrder(Order order)
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("заказ уже отменен");
        Console.ForegroundColor = ConsoleColor.White;
    }
}
```

Рисунок 14 - Класс «CancelledState»

```
internal class Program
{
    static void Main(string[] args)
    {
        Order order1 = new Order();
        Order order2 = new Order();
        Order order3 = new Order();

        Console.WriteLine("Первый заказ:");
        order1.ProcessOrder();
        order1.ShipOrder();
        order1.DeliverOrder();
        order1.CancelOrder();

        Console.WriteLine("\nВторой заказ:");
        order2.ProcessOrder();
        order2.CancelOrder();
        order2.ShipOrder();
        order2.DeliverOrder();

        Console.WriteLine("\nТретий заказ:");
        order3.ProcessOrder();
        order3.ShipOrder();
        order3.CancelOrder();
        order3.DeliverOrder();
    }
}
```

Рисунок 15 - Класс «Program»

3 проект: Команда

1. Описание предметной области

Проект реализует симуляцию магазина, где пользователи могут добавлять товары в корзину, удалять их и оформлять заказы. Используется паттерн Команда, который инкапсулирует запросы как объекты, позволяя поддерживать отмену операций.

2. UML диаграмма классов

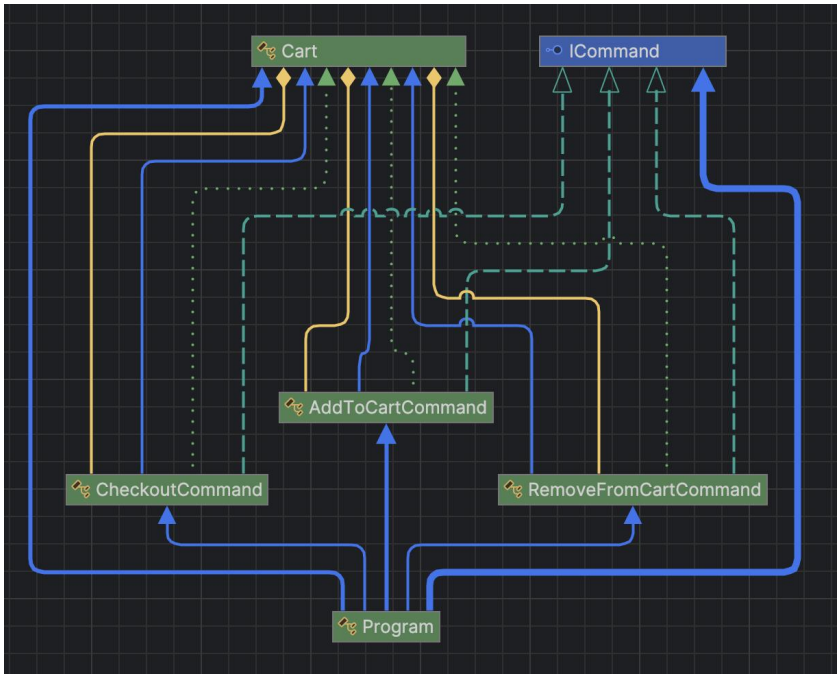


Рисунок 16 - UML диаграмма классов проекта «ShopSimulation»

3. Краткий вывод по использованию паттерна

Паттерн «Команда» позволяет инкапсулировать запросы как объекты «AddToCartCommand», «RemoveFromCartCommand», «CheckoutCommand», что делает систему более гибкой.

4. Листинг кода

```
7 usages  3 inheritors
internal interface ICommand
{
    4 usages  3 implementations
    void Execute();
    1 usage  3 implementations
    void Undo();
}
```

Рисунок 17 - Интерфейс «ICommand»

```
8 usages
internal class Cart
{
    private List<string> _orders = new List<string>();

    2 usages
    public void AddOrder(string order)
    {
        _orders.Add(order);
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine($"товар \"{order}\" добавлен в корзину");
        Console.ForegroundColor = ConsoleColor.White;
    }

    2 usages
    public void RemoveOrder(string order)
    {
        if (_orders.Contains(order))
        {
            _orders.Remove(order);
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine($"товар \"{order}\" удален из корзины");
            Console.ForegroundColor = ConsoleColor.White;
        }
        else
        {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine($"товар \"{order}\" не найден в корзине");
            Console.ForegroundColor = ConsoleColor.White;
        }
    }

    1 usage
    public void Checkout()
    {
        Console.WriteLine("оформление заказа...");
        foreach (var order :string in _orders)
        {
            Console.WriteLine($"товар: \"{order}\"");
        }
        _orders.Clear();
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("заказ оформлен");
        Console.ForegroundColor = ConsoleColor.White;
    }
}
```

Рисунок 18 - Класс «Cart»

```
/// <summary>
/// команда для добавления товара в корзину
/// </summary>
2 usages
internal class AddToCartCommand : ICommand
{
    private Cart _cart;
    private string _order;

    2 usages
    public AddToCartCommand(Cart cart, string order)
    {
        _cart = cart;
        _order = order;
    }

    0+4 usages
    public void Execute()
    {
        _cart.AddOrder(_order);
    }

    0+1 usages
    public void Undo()
    {
        _cart.RemoveOrder(_order);
    }
}
```

Рисунок 19 - Класс «AddToCartCommand»

```
/// <summary>
/// команда для удаления товара в корзину
/// </summary>
1 usage
internal class RemoveFromCartCommand : ICommand
{
    private Cart _cart;
    private string _order;

    1 usage
    public RemoveFromCartCommand(Cart cart, string order)
    {
        _cart = cart;
        _order = order;
    }

    0+4 usages
    public void Execute()
    {
        _cart.RemoveOrder(_order);
    }

    0+1 usages
    public void Undo()
    {
        _cart.AddOrder(_order);
    }
}
```

Рисунок 20 - Класс «RemoveToCartCommand»

```
/// <summary>
/// команда для оформления заказа
/// </summary>
1 usage
internal class CheckoutCommand : ICommand
{
    private Cart _cart;

    1 usage
    public CheckoutCommand(Cart cart)
    {
        _cart = cart;
    }

    0+4 usages
    public void Execute()
    {
        _cart.Checkout();
    }

    0+1 usages
    public void Undo()
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("нельзя отменить оформление заказа");
        Console.ForegroundColor = ConsoleColor.White;
    }
}
```

Рисунок 21 - Класс «CheckoutCommand»

```
internal class Program
{
    static void Main(string[] args)
    {
        Cart cart = new Cart();

        ICommand addOrder1Command = new AddToCartCommand(cart, order: "футболка");
        addOrder1Command.Execute();

        ICommand addOrder2Command = new AddToCartCommand(cart, order: "джинсы");
        addOrder2Command.Execute();

        Console.WriteLine();
        addOrder2Command.Undo();

        ICommand removeOrder1Command = new RemoveFromCartCommand(cart, order: "джинсы");
        removeOrder1Command.Execute();

        Console.WriteLine();
        ICommand checkoutCommand = new CheckoutCommand(cart);
        checkoutCommand.Execute();
    }
}
```

Рисунок 22 - Класс «Program»