Министерство образования и науки Российской Федерации

ФГАОУ ВО «КФУ им. В.И. Вернадского»

физико-технический институт (структурное подразделение)

|  |
| --- |
|  |

Кафедра компьютерной инженерии и моделирования

Пугач Давид Сергеевич

отчет по практической работе №7  
по дисциплине **«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ»**

Направление подготовки:

09.03.04 "Программная инженерия"

Оценка -



Симферополь, 2023

**Практическая работа №7  
Тема: Паттерны проектирования ПО на C#.**

**Цель работы:** Изучить основные паттерны проектирования по книге Эриха Гаммы, Ричарда Хелма, Ральфа Джонсона, Джона Влиссидеса ("Банда четырёх"). "Приемы объектно-ориентированного проектирования. Паттерны проектирования", разобраться с вопросом целесообразности использования паттернов в крупных проектах, научиться реализации паттернов на языке программирования C#.

**Задание 1. Написать в отчет о практической работе небольшой авторский реферат о паттернах проектирования:**Описать структуры паттернов:

1. Порождающие паттерны ­ ­Порождающие паттерны — это паттерны, которые абстрагируют процесс инстанцирования или, иными словами, процесс порождения классов и объектов. Среди них выделяются следующие: ­ ­ Абстрактная фабрика (Abstract Factory) ­ ­ Строитель (Builder) ­ ­ Фабричный метод (Factory Method) ­ ­ Прототип (Prototype) ­ ­ Одиночка (Singleton)
2. Структурные паттерны - рассматривает, как классы и объекты образуют более крупные структуры - более сложные по характеру классы и объекты. К таким шаблонам относятся: ­ ­ Адаптер (Adapter) ­ ­ Мост (Bridge) ­ ­ Компоновщик (Composite) ­ ­ Декоратор (Decorator) ­ ­ Фасад (Facade) ­ ­ Приспособленец (Flyweight) ­ ­ Заместитель (Proxy) ­
3. ­Поведенческие паттерны - определяют алгоритмы и взаимодействие между классами и объектами, то есть их поведение. Среди подобных шаблонов можно выделить следующие: ­ ­ Цепочка обязанностей (Chain of responsibility) ­ ­ Команда (Command) ­ ­ Интерпретатор (Interpreter) ­ ­ Итератор (Iterator) ­ ­ Посредник (Mediator) ­ ­ Хранитель (Memento) ­ ­ Наблюдатель (Observer) ­ ­ Состояние (State) ­ ­ Стратегия (Strategy) ­ ­ Шаблонный метод (Template method) ­ ­ Посетитель (Visitor).

**Паттерны проектирования** – это повторяемые решения для типичных проблем, возникающих при проектировании программного обеспечения. Они представляют собой архитектурные шаблоны, описывающие, как организовать классы, объекты и отношения между ними.

В отличие от готовых функций или библиотек, паттерн нельзя просто взять и скопировать в программу. Паттерн представляет собой не какой-то конкретный код, а общую концепцию решения той или иной проблемы, которую нужно будет ещё подстроить под нужды вашей программы.

**Преимущество знания паттернов:**

1. Улучшение проектирования (Паттерны предлагают оптимальные способы решения распространенных проблем, что помогает создавать качественный и гибкий код.)
2. Лёгкость поддержки и расширения (Паттерны способствуют созданию модульного кода, который легко поддерживать и изменять.)
3. Проверенные решения (Использование паттернов позволяет повторно использовать проверенные решения, экономя время и снижая вероятность ошибок.)
4. Общий программистский словарь (Паттерны обеспечивают общий язык и согласованные решения, что улучшает коммуникацию в команде разработчиков.)

## Основные типы паттернов:

1. Порождающие паттерны — паттерны, которые абстрагируют процесс инстанцирования или, иными словами, процесс порождения классов и объектов.
2. Структурные паттерны – рассматривают, как классы и объекты образуют более крупные структуры - более сложные по характеру классы и объекты.
3. Поведенческие паттерны – определяют алгоритмы и взаимодействие между классами и объектами, то есть их поведение.

## Порождающие паттерны:

* Абстрактная фабрика (Abstract Factory) порождающий шаблон проектирования, предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов. Шаблон реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы (например, для оконного интерфейса он может создавать окна и кнопки). Затем пишутся классы, реализующие этот интерфейс.
* **Строитель (Builder)** — порождающий шаблон проектирования предоставляет способ создания составного объекта. Отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться разные представления.
* **Фабричный метод (Factory Method)** порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.
* Прототип (Prototype) Задаёт виды создаваемых объектов с помощью экземпляра-прототипа и создаёт новые объекты путём копирования этого прототипа. Он позволяет уйти от реализации и позволяет следовать принципу «программирование через интерфейсы». В качестве возвращающего типа указывается интерфейс/абстрактный класс на вершине иерархии, а классы-наследники могут подставить туда наследника, реализующего этот тип. Проще говоря, это паттерн создания объекта через клонирование другого объекта вместо создания через конструктор.
* **Одиночка (Singleton)** порождающий шаблон проектирования, гарантирующий, что в однопоточном приложении будет единственный экземпляр класса с глобальной точкой доступа.

## Структурные паттерны:

* Адаптер (Adapter) структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс.
* Мост (Bridge) структурный шаблон проектирования, используемый в проектировании программного обеспечения чтобы «разделять абстракцию и реализацию так, чтобы они могли изменяться независимо». Шаблон мост использует инкапсуляцию, агрегирование и может использовать наследование для того, чтобы разделить ответственность между классами.
* Компоновщик (Composite) структурный шаблон проектирования, объединяющий объекты в древовидную структуру для представления иерархии от частного к целому. Компоновщик позволяет клиентам обращаться к отдельным объектам и к группам объектов одинаково
* Декоратор (Decorator) структурный шаблон проектирования, предназначенный для динамического подключения дополнительного поведения к объекту. Шаблон Декоратор предоставляет гибкую альтернативу практике создания подклассов с целью расширения функциональности.
* Фасад (Facade) структурный шаблон проектирования, позволяющий скрыть сложность системы путём сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы.
* Приспособленец (Flyweight) структурный шаблон проектирования, при котором объект, представляющий себя как уникальный экземпляр в разных местах программы, по факту не является таковым.
* **Заместитель (Proxy)** структурный шаблон проектирования, который предоставляет объект, который контролирует доступ к другому объекту, перехватывая все вызовы (выполняет функцию контейнера).

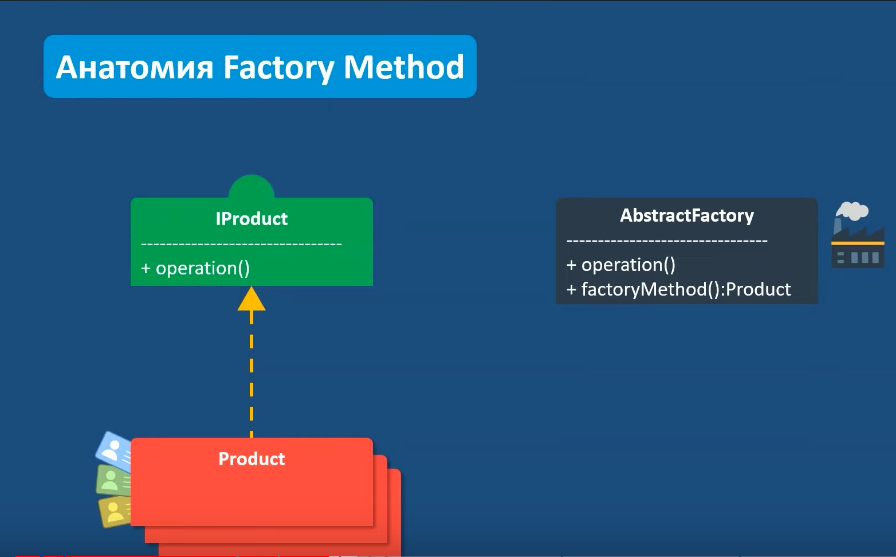
## Поведенческие паттерны:

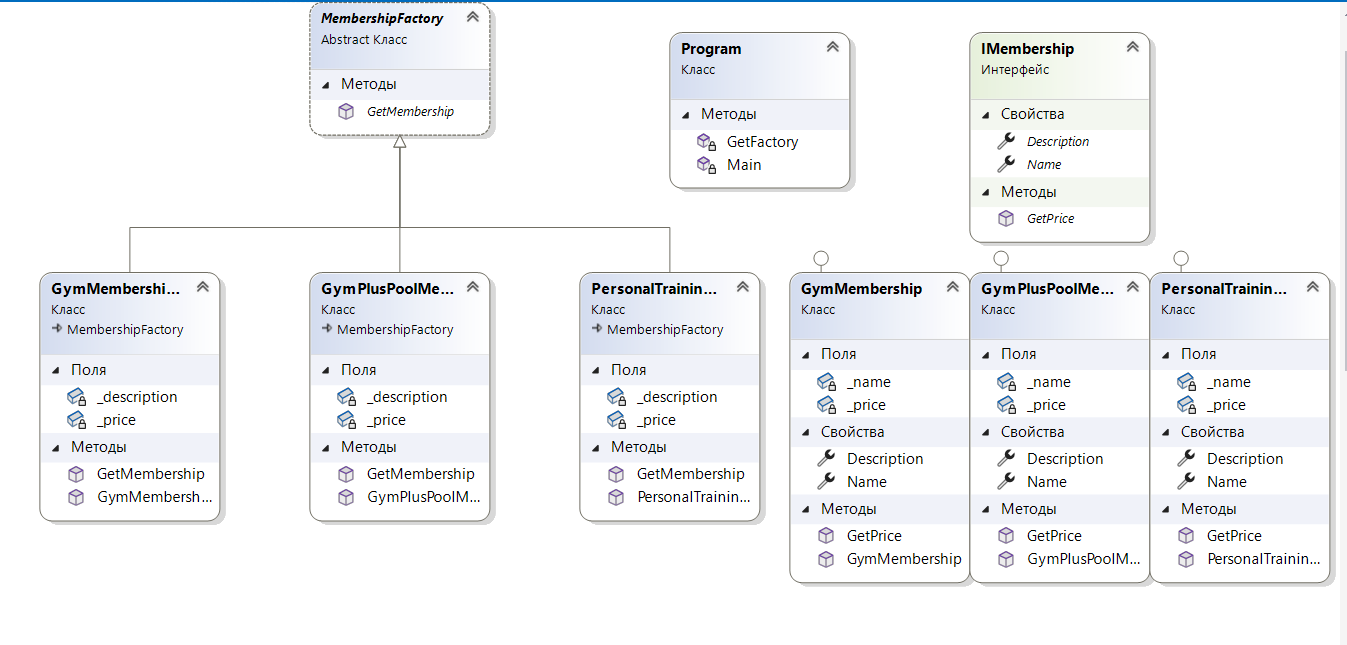
* Цепочка обязанностей (Chain of responsibility) — поведенческий шаблон проектирования, предназначенный для организации в системе уровней ответственности.
* Команда (Command) поведенческий шаблон проектирования, используемый при объектно-ориентированном программировании, представляющий действие. Объект команды заключает в себе само действие и его параметры.
* Интерпретатор (Interpreter) поведенческий шаблон проектирования, решающий часто встречающуюся, но подверженную изменениям, задачу. Также известен как Little (Small) Language
* Итератор (Iterator) интерфейс, предоставляющий доступ к элементам коллекции (массива или контейнера) и навигацию по ним. В различных системах итераторы могут иметь разные общепринятые названия. В терминах систем управления базами данных итераторы называются курсорами.
* Посредник (Mediator) поведенческий шаблон проектирования, обеспечивающий взаимодействие множества объектов, формируя при этом слабую связанность и избавляя объекты от необходимости явно ссылаться друг на друга
* Хранитель (Memento) поведенческий шаблон проектирования, позволяющий, не нарушая инкапсуляцию, зафиксировать и сохранить внутреннее состояние объекта так, чтобы позднее восстановить его в это состояние.
* Наблюдатель (Observer) поведенческий шаблон проектирования. Также известен как «подчинённые» (Dependents). Создает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов об изменении их состояния, тем самым наблюдая за ними.
* Состояние (State) поведенческий шаблон проектирования. Используется в тех случаях, когда во время выполнения программы объект должен менять своё поведение в зависимости от своего состояния.
* **Стратегия (Strategy)** поведенческий шаблон проектирования, предназначенный для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм путём определения соответствующего класса. Шаблон Strategy позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют.
* Шаблонный метод (Template method) поведенческий шаблон проектирования, определяющий основу алгоритма и позволяющий наследникам переопределять некоторые шаги алгоритма, не изменяя его структуру в целом.
* Посетитель (Visitor) поведенческий шаблон проектирования, описывающий операцию, которая выполняется над объектами других классов. При изменении visitor нет необходимости изменять обслуживаемые классы.

**Задание 2. Реализовать 3-4 паттерна проектирования на языке программирования C#.**

1. **Фабричный метод (Factory Method)**

**Фабричный метод (Factory Method)** порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.

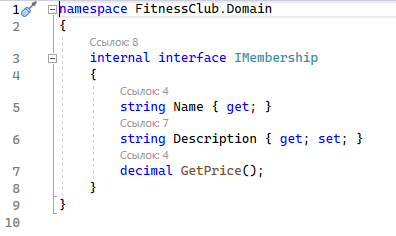




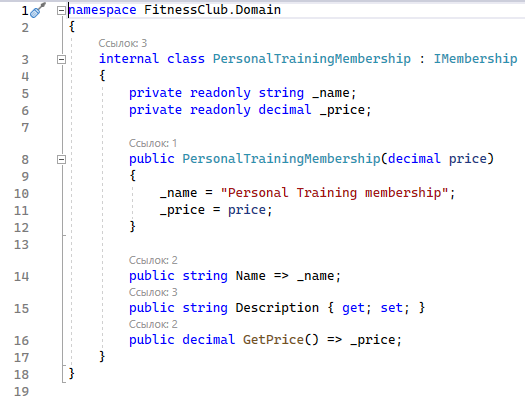
Класс Program



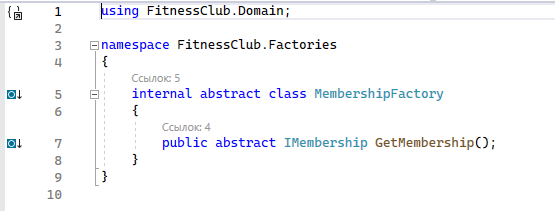
Интерфейс IMembership



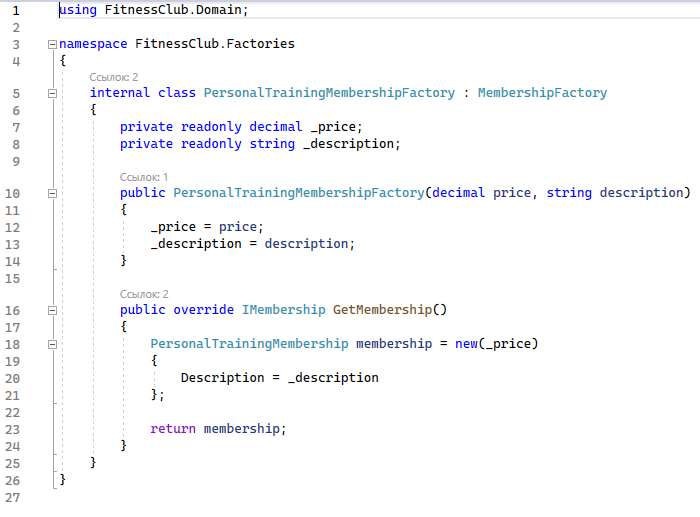
Класс PersonalTrainingMembership, реализующий интерфейс выше.



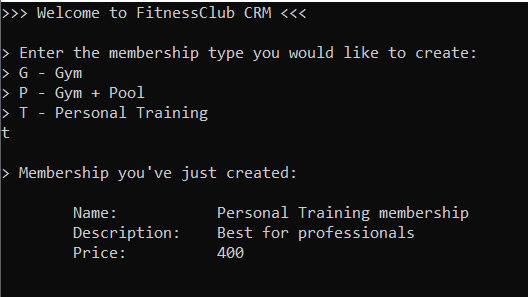
Абстрактный класс MembershipFactory



Наследуемый класс PersonalTrainingMembershipFactory

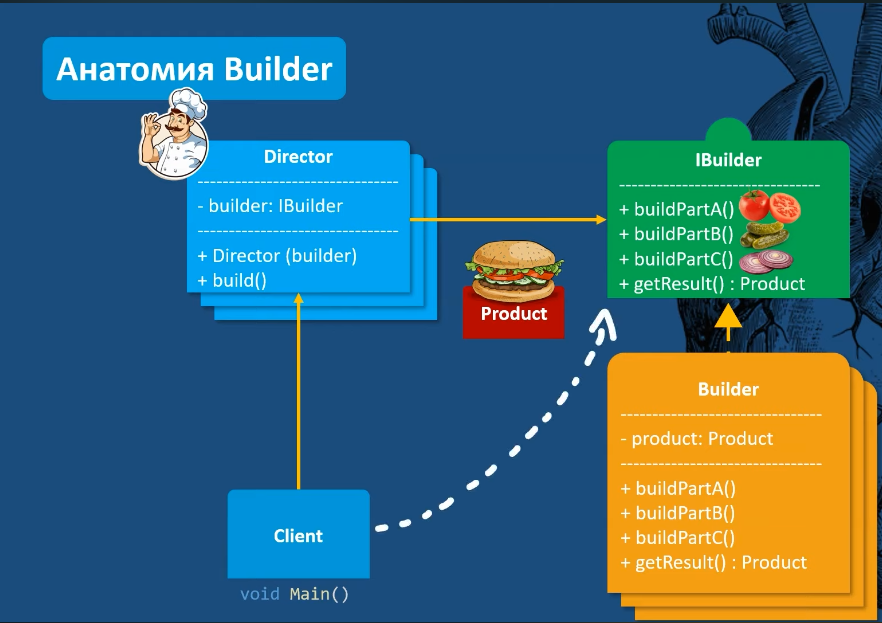


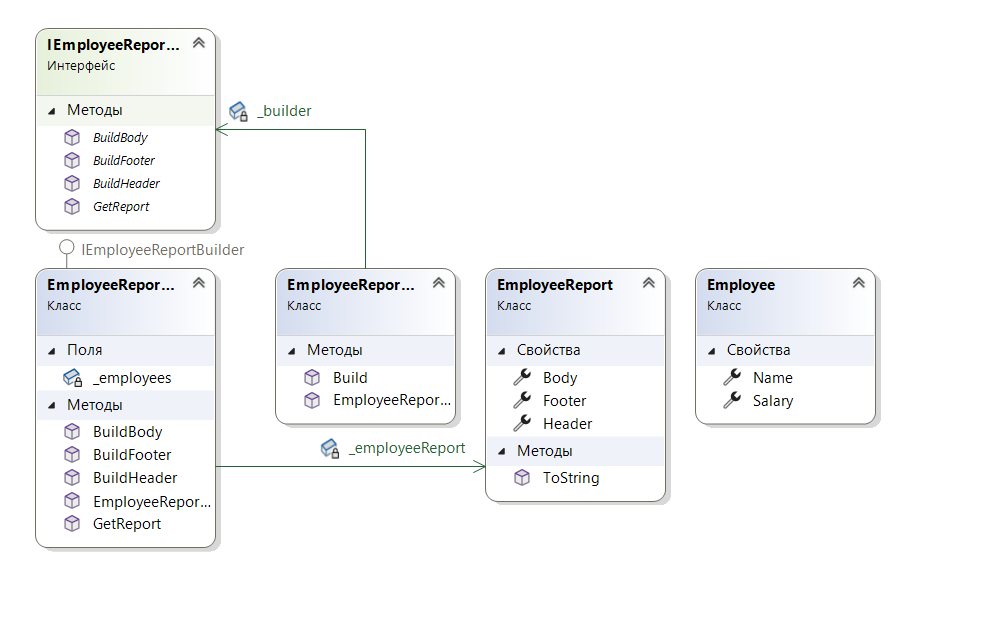
Пример работы программы



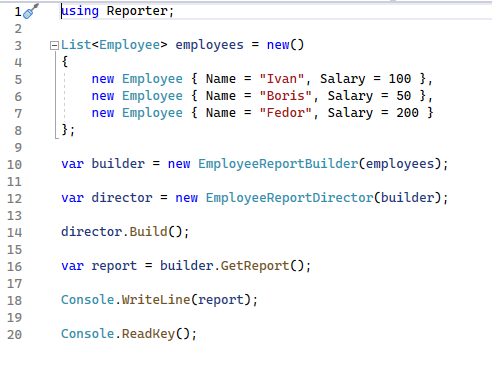
1. **Builder (Строитель)**

**Строитель (Builder)** — порождающий шаблон проектирования предоставляет способ создания составного объекта. Отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться разные представления.

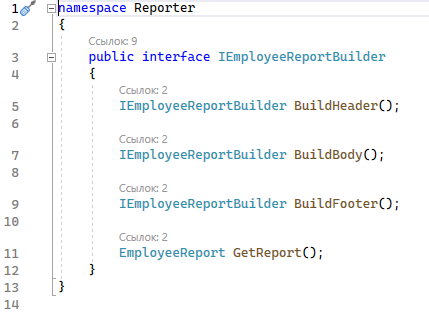




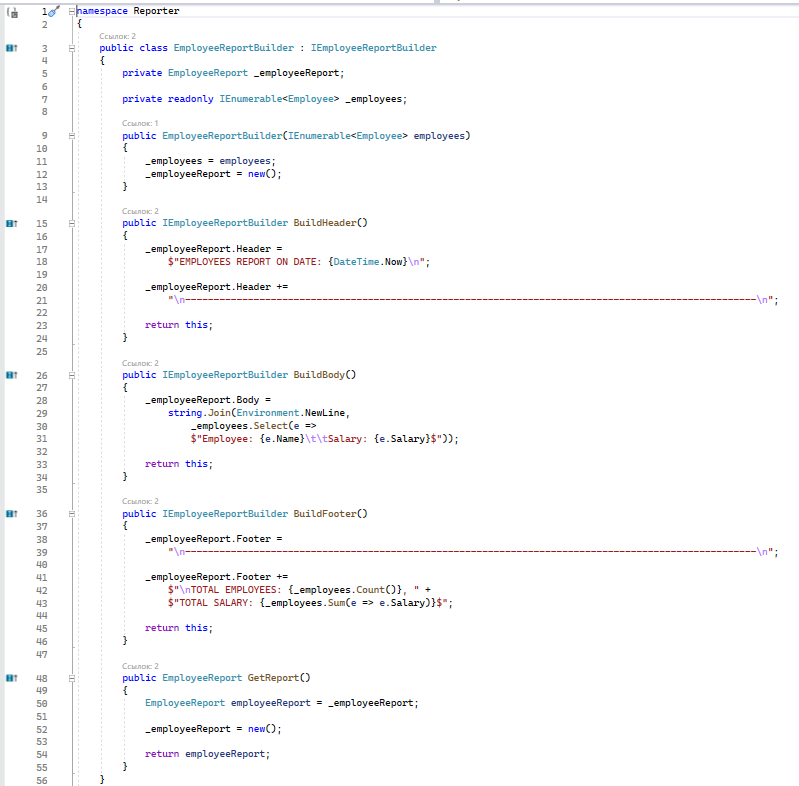
Класс Program



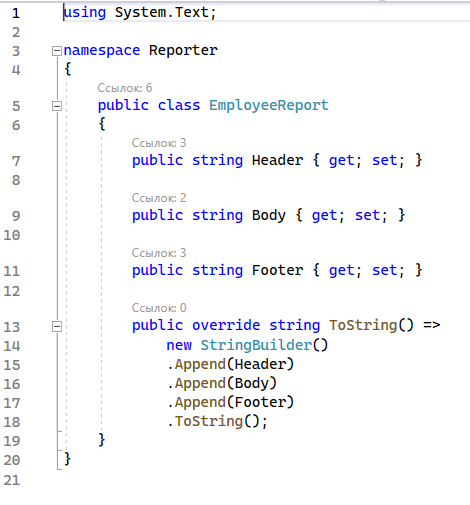
Интерфейс IEmployerReportBuilder



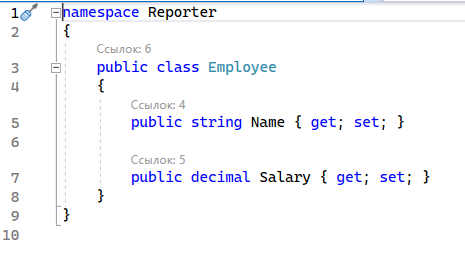
Класс EmployerReportBuilder



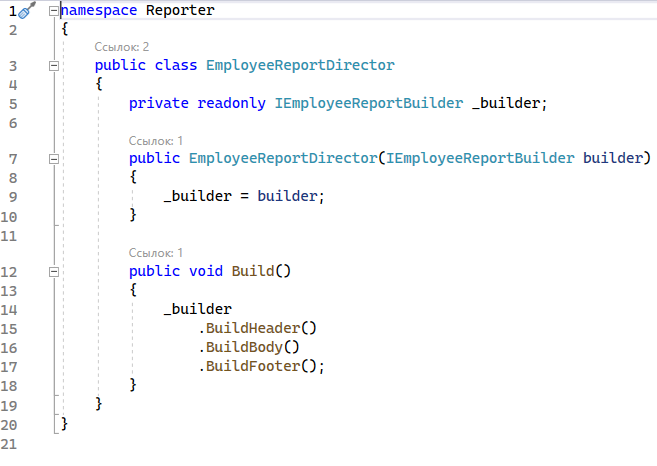
Класс EmployeeReport



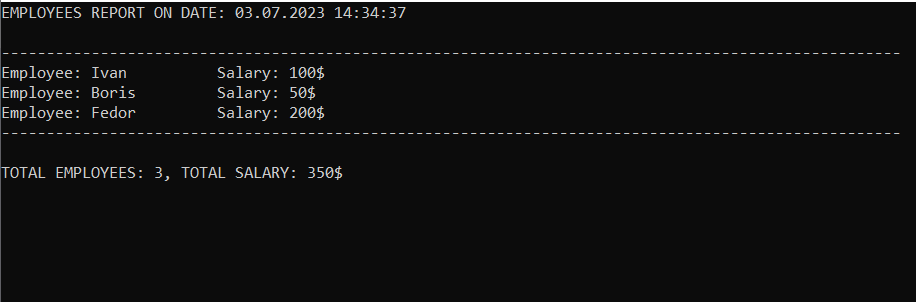
Класс Employee



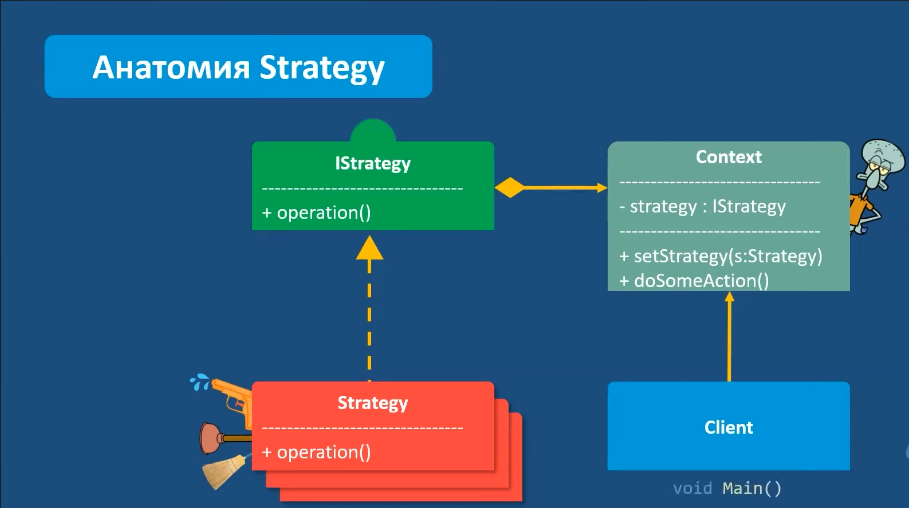
Класс EmployeeReportDirector

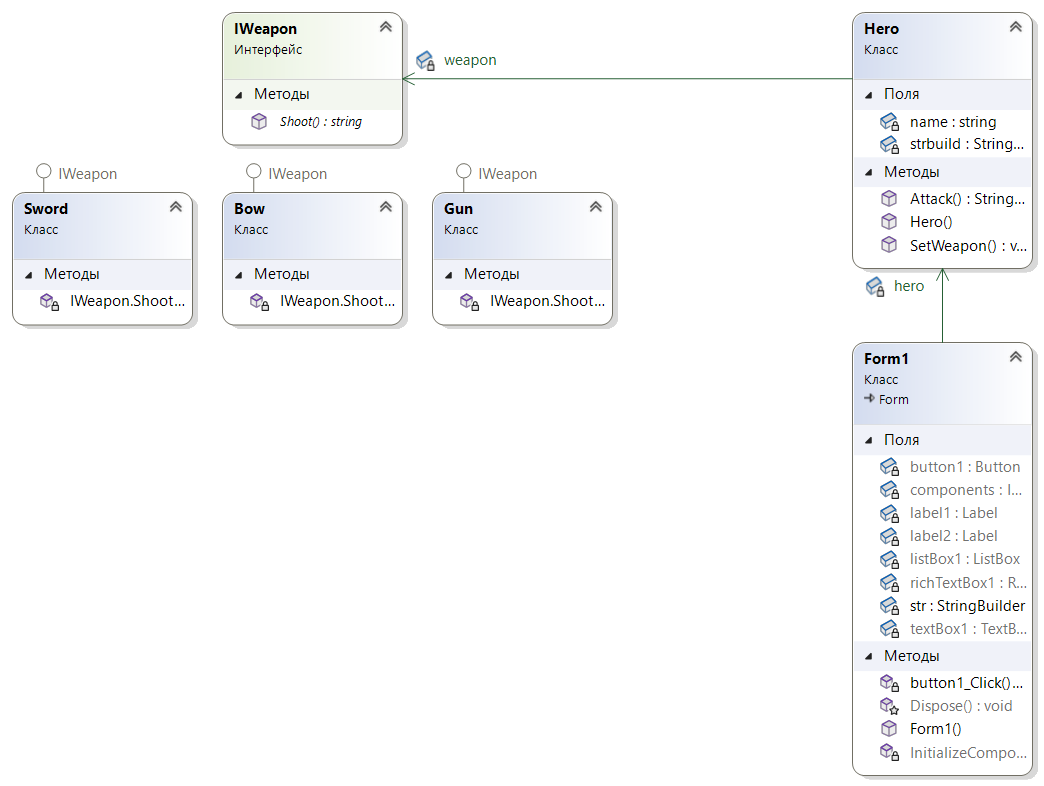


Пример работы программы

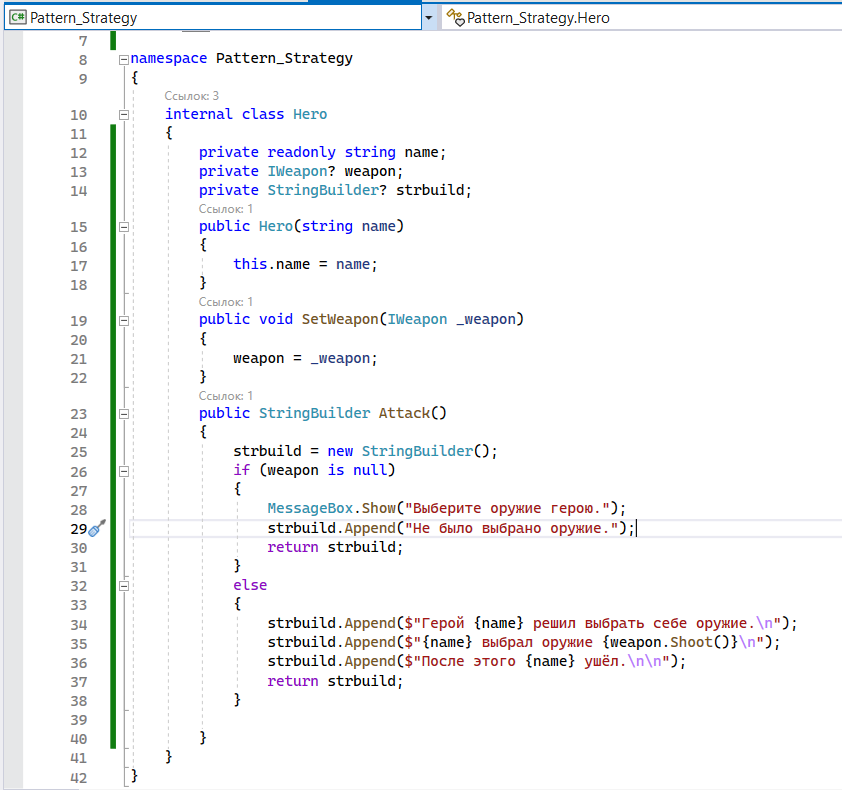


1. **Strategy (Стратегия)**

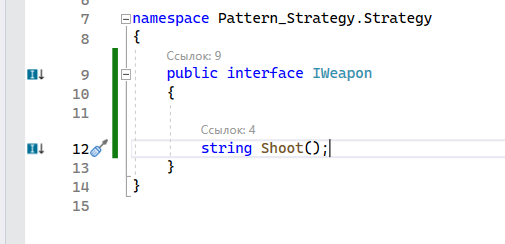




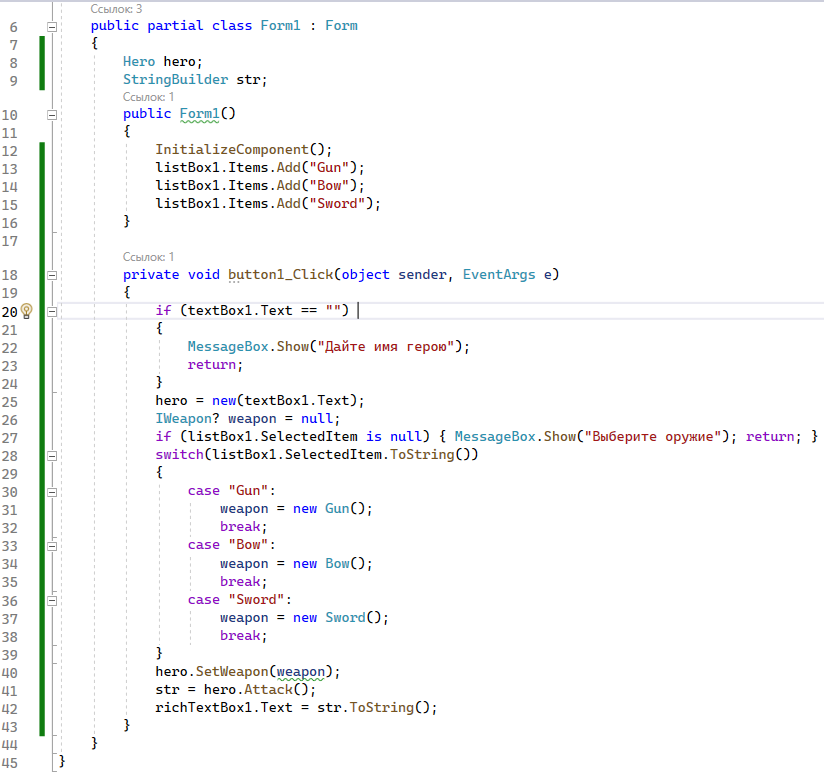
Класс Hero

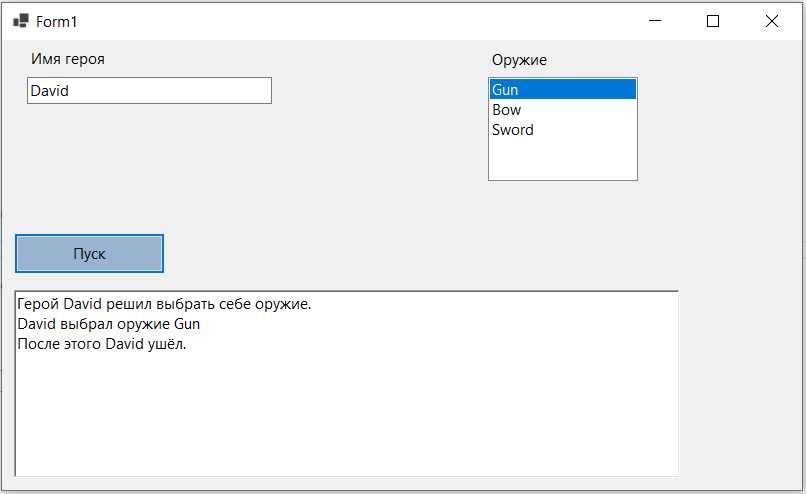


Интерфейс IWeapon



Класс Form1





**Представлены 3 проекта, реализованных в Visual Studio Community 2022.  
Проекты представлены преподавателю в электронной форме, продемонстрирована их работоспособность, разъяснены детали программного кода.**

**Ссылка на репозиторий с программами**

https://github.com/Bananaws/Object-oriented-programming