Министерство образования Республики Беларусь
Учреждение образования
«Гомельский государственный технический университет
имени П.О.Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ПРАКТИЧЕСКИЕ, СЕМИНАРСКИЕ ЗАНЯТИЯ
по дисциплине
«ПРОГРАММНАЯ ИНЖЕНЕРИЯ»
для студентов специальности
1-40 05 01 «Информационные системы и технологии (по направлениям)»
направление специальности 1-40 05 01-01 «Информационные системы и технологии (в проектировании и производстве)»

Трубенок Д.Н.

Гомель 2018

Библиотека ГГТУ им.П.О.Сухого

СОДЕРЖАНИЕ

Библиотека ГГТУ им.П.О.Сухого

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №1

«ОСНОВНЫЕ ШАБЛОНЫ ПРОЕКТИРОВАНИЯ»

Цель занятия: изучить основные (*fundamental*) паттерны проектирования.

Основные шаблоны проектирования [1]:

- Шаблон Делегирования (*Delegation pattern*) – объект внешне выражает некоторое поведение, но в реальности передаёт ответственность за выполнение этого поведения связанному объекту.
- Шаблон функционального дизайна (*Functional design*) – гарантирует, что каждый модуль компьютерной программы имеет только одну обязанность и исполняет её с минимумом побочных эффектов на другие части программы.
- Неизменяемый интерфейс (*Immutable interface*) – создание неизменяемого объекта.
- Интерфейс (*Interface*) – общий метод для структурирования компьютерных программ для того, чтобы их было проще понять.
- Интерфейс-маркер (*Marker interface*) – в качестве атрибута (как пометки объектной сущности) применяется наличие или отсутствие реализации интерфейса-маркера. В современных языках программирования вместо этого могут применяться атрибуты или аннотации.
- Контейнер свойств (*Property container*) – позволяет добавлять дополнительные свойства для класса в контейнер (внутри класса), вместо расширения класса новыми свойствами.
- Канал событий (*Event channel*) – расширяет шаблон *Publish*/*Subscribe*, создавая централизованный канал для событий. Использует объект-представитель для подписки и объект-представитель для публикации события в канале.

Делегирование (англ. *Delegation*) – основной шаблон проектирования, в котором объект внешне выражает некоторое поведение, но в реальности передаёт ответственность за выполнение этого поведения связанному объекту.

Пример реализации шаблона Делегирование представлен на UML диаграмме на рисунке 1.1.

Библиотека ГГТУ им. П.О. Сухого

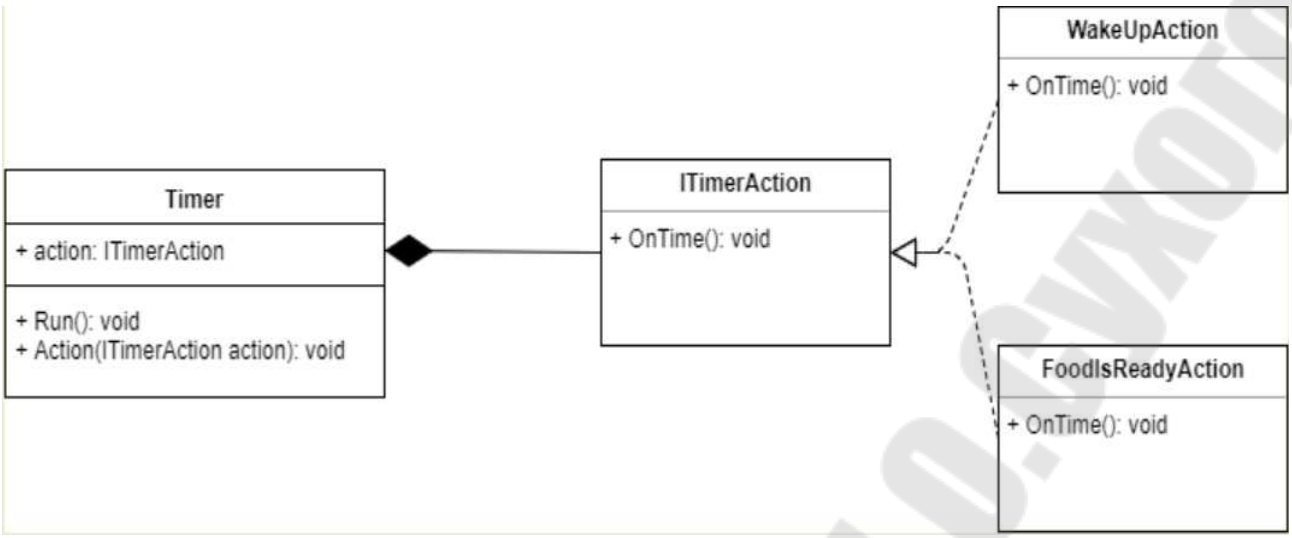

Рисунок 1.1 – Реализация паттерна Делегирование

Библиотека ГГТУ им.П.О.Сухого

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №2

«ПОРОЖДАЮЩИЕ ШАБЛОНЫ ПРОЕКТИРОВАНИЯ»

Цель занятия: изучить порождающие шаблоны проектирования.

Порождающие шаблоны (*Creational*) – шаблоны проектирования, которые абстрагируют процесс инстанцирования. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов. Шаблон, порождающий классы, использует наследование, чтобы изменять инстанцируемый класс, а шаблон, порождающий объекты, делегирует инстанцирование другому объекту [2, 3].

Порождающие шаблоны проектирования:

- Абстрактная фабрика (*Abstract factory*) – класс, который представляет собой интерфейс для создания компонентов системы.
- Строитель (*Builder*) – класс, который представляет собой интерфейс для создания сложного объекта.
- Фабричный метод (*Factory method*) – определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать.
- Отложенная инициализация (*Lazy initialization*) – объект, инициализируемый во время первого обращения к нему.
- Пул одиночек (*Multiton*) – гарантирует, что класс имеет поименованные экземпляры объекта и обеспечивает глобальную точку доступа к ним.
- Объектный пул (*Object pool*) – класс, который представляет собой интерфейс для работы с набором инициализированных и готовых к использованию объектов.
- Прототип (*Prototype*) – определяет интерфейс создания объекта через клонирование другого объекта вместо создания через конструктор.
- Одиночка (*Singleton*) – класс, который может иметь только один экземпляр.

Строитель (*Builder*) – порождающий шаблон проектирования предоставляет способ создания составного объекта [2, 3].

Цель. Отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться разные представления.

Пример использования паттерна «Строитель» приведен на рисунке 2.1.

Библиотека ГГТУ им. П.О. Сухого

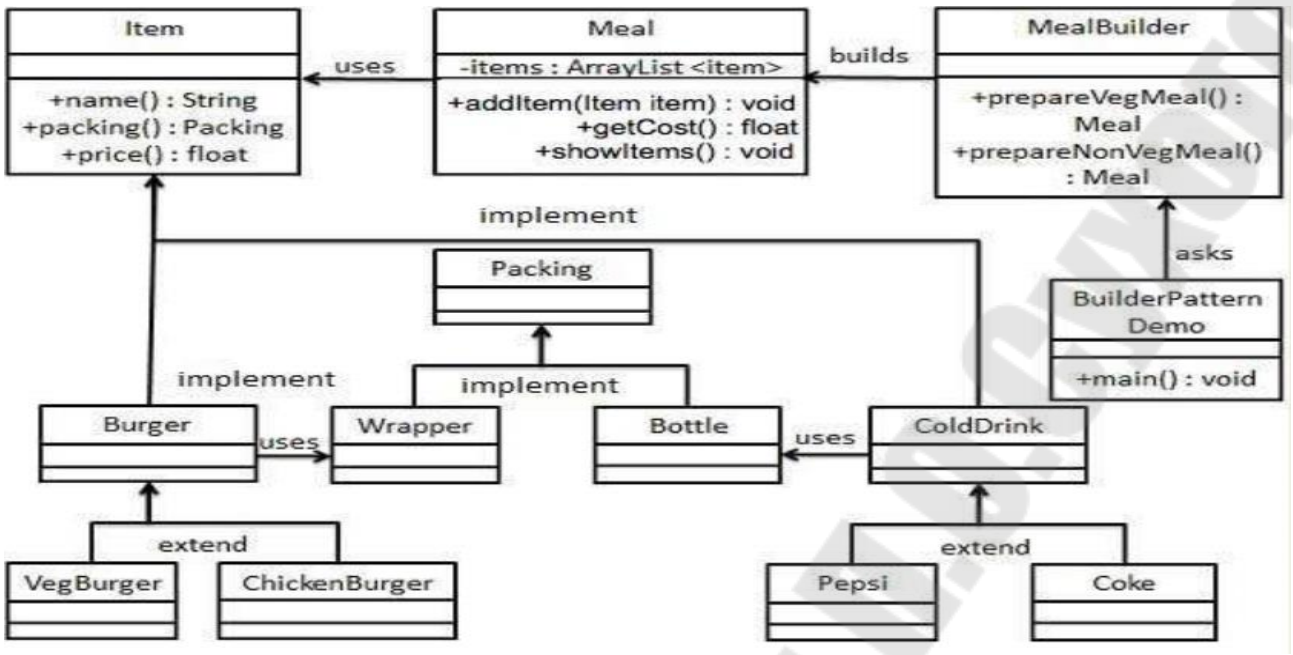

Рисунок 2.1 – Использование паттерна «Строитель»

Фабричный метод (англ. *Factory Method*) – порождающий шаблон проектирования, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, данный шаблон делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне [2, 3].

Есть несколько подходов использования фабричного метода [2]. Использование одного из подходов приведено на рисунке 2.2.

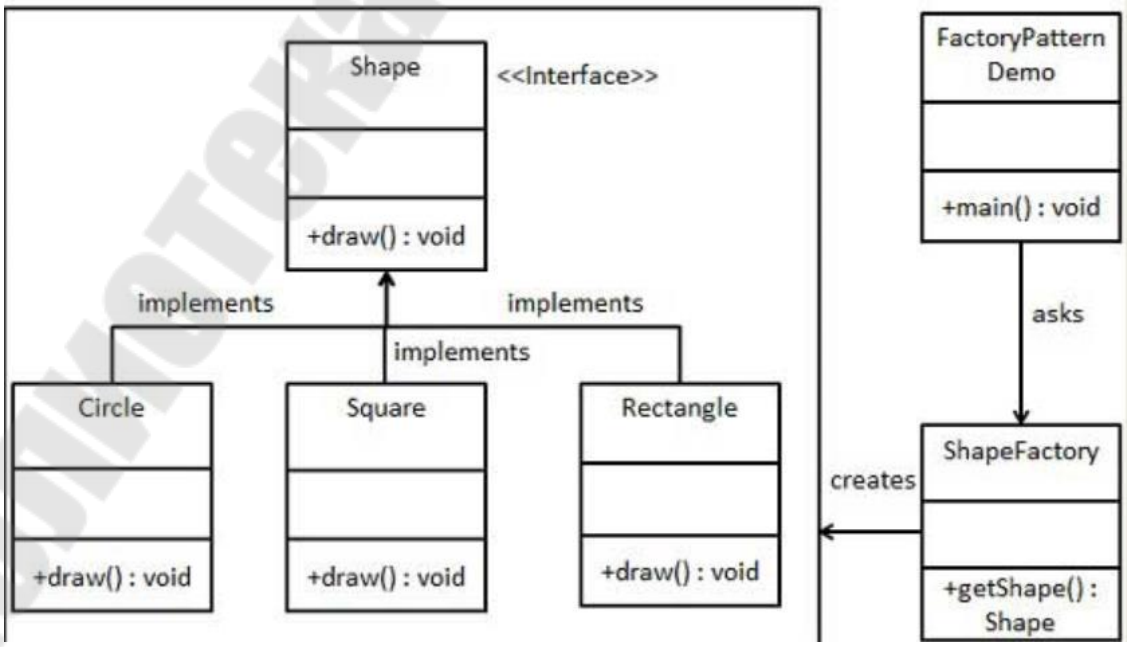

Рисунок 2.2 – Использование паттерна «Фабричный метод»

Библиотека ГГТУ им. П.О.Сухого

Одиночка (*Singleton*) – порождающий шаблон проектирования, гарантирующий, что в однопроцессном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру [2, 3].

Цель. У класса есть только один экземпляр, и он предоставляет к нему глобальную точку доступа. Существенно то, что можно пользоваться именно *экземпляром* класса, так как при этом во многих случаях становится доступной более широкая функциональность.

Пример использования паттерна «Одиночка» приведен на UML-диаграмме 2.3.

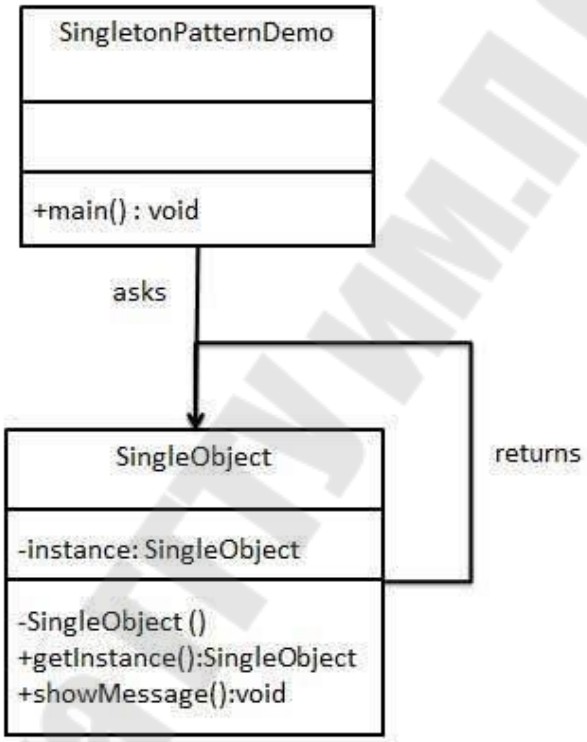

Рисунок 2.3 – Паттерн «Одиночка»

Абстрактная фабрика (англ. *Abstract factory*) – порождающий шаблон проектирования, предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов [2, 3].

Плюсы

- изолирует конкретные классы;
- упрощает замену семейств продуктов;
- гарантирует сочетаемость продуктов.

Минусы

- сложно добавить поддержку нового вида продуктов.

Пример использования паттерна «Абстрактная фабрика» приведен на рисунке 2.4.

Библиотека ГГТУ им. П.О. Сухого

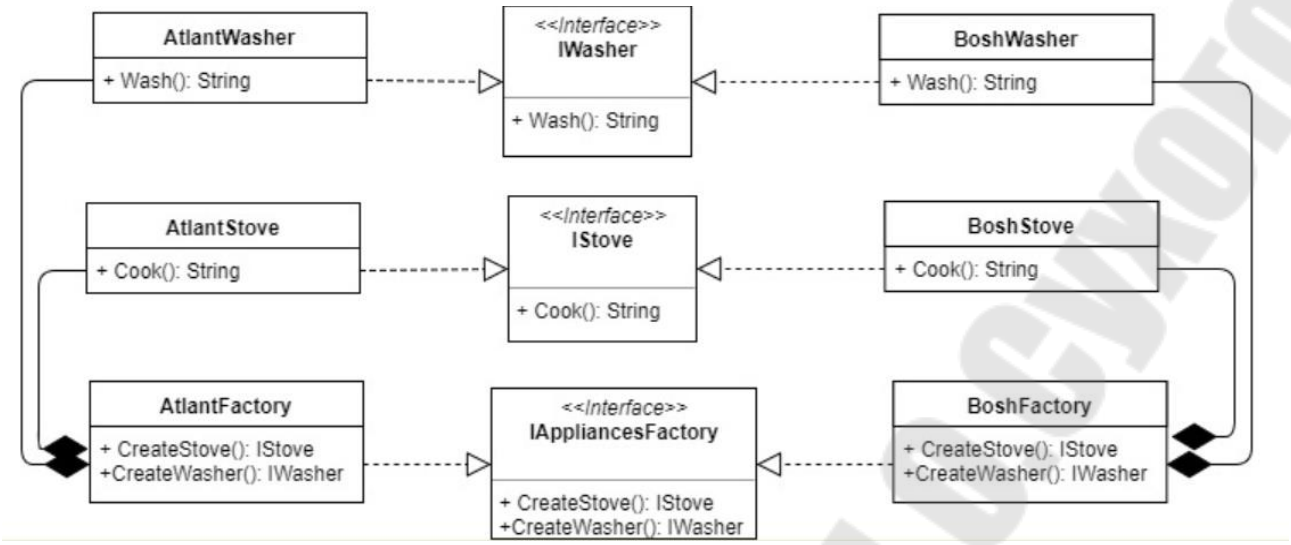

Рисунок 2.4 – Использование паттерна «Абстрактная фабрика»

Библиотека ГГТУ им.П.О.Сухого

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №3

«СТРУКТУРНЫЕ ШАБЛОНЫ ПРОЕКТИРОВАНИЯ»

Цель занятия: изучить структурные шаблоны проектирования.

Структурные шаблоны (*Structural*) – определяют различные сложные структуры, которые изменяют интерфейс уже существующих объектов или его реализацию, позволяя облегчить разработку и оптимизировать программу.

Структурные шаблоны проектирования:

- Адаптер (*Adapter*) – объект, обеспечивающий взаимодействие двух других объектов, один из которых использует, а другой предоставляет несовместимый с первым интерфейс.
- Мост (*Bridge*) – структура, позволяющая изменять интерфейс обращения и интерфейс реализации класса независимо.
- Компоновщик (*Composite*) – объект, который объединяет в себе объекты, подобные ему самому.
- Декоратор (*Wrapper* или *Decorator*) – класс, расширяющий функциональность другого класса без использования наследования.
- Фасад (*Facade*) – объект, который абстрагирует работу с несколькими классами, объединяя их в единое целое.
- Единичная точка входа (*Front controller*) – обеспечивает унифицированный интерфейс для интерфейсов в подсистеме. *Front Controller* определяет высокоуровневый интерфейс, упрощающий использование подсистемы.
- Приспособленец (*Flyweight*) – это объект, представляющий себя как уникальный экземпляр в разных местах программы, но фактически не являющийся таковым.
- Заместитель (*Proxy*) – объект, который является посредником между двумя другими объектами, и который реализует/ограничивает доступ к объекту, к которому обращаются через него.

Декоратор (англ. *Decorator***)** – структурный шаблон проектирования, предназ-наченный для динамического подключения дополнительного поведения к объекту.

Цель. Объект, который предполагается использовать, выполняет основные функции. Однако может потребоваться добавить к нему некоторую дополнительную функциональность, которая будет выполняться до, после или даже вместо основной функциональности объекта.

Реализация паттерна «Декоратор» на примере пиццы и начинок к ней показана на рисунке 3.1.

Библиотека ГГТУ им. П.О. Сухого

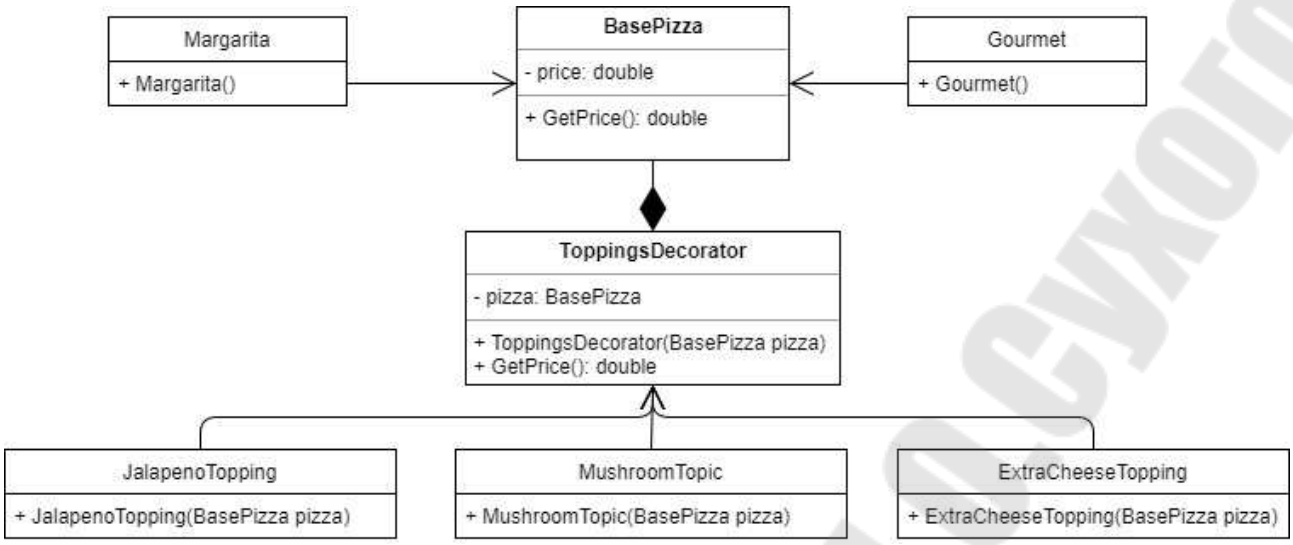

Рисунок 3.1 – Реализация паттерна Декоратор

Компоновщик (англ. *Composite pattern*) – 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.

Цель. Паттерн определяет иерархию классов, которые одновременно могут состоять из примитивных и сложных объектов, упрощает архитектуру клиента, делает процесс добавления новых видов объекта более простым.

Реализация паттерна «Компоновщик» показана на примере иерархии сотрудников компании на рисунке 3.2.

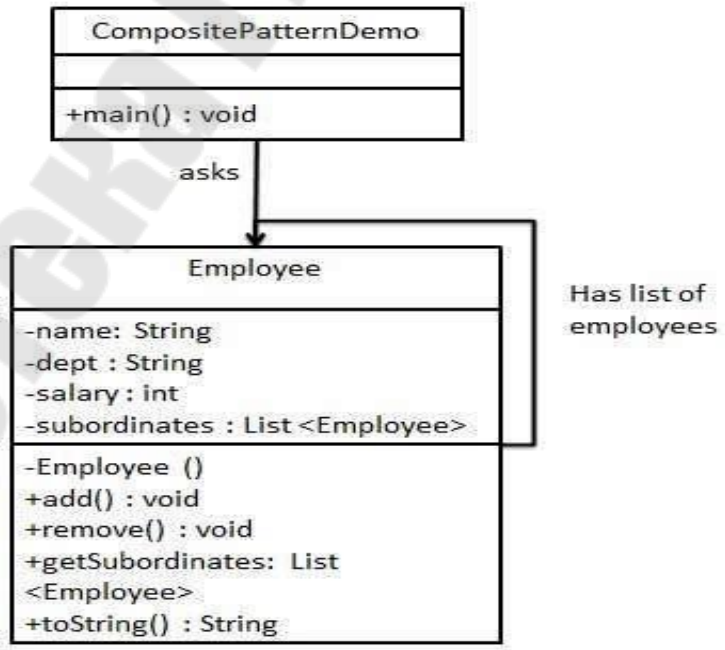

Рисунок 3.2 – Реализация паттерна «Компоновщик»

Библиотека ГГТУ им. П.О. Сухого

Адаптер (англ. *Adapter*) – структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс.

Задача. Система поддерживает требуемые данные и поведение, но имеет неподходящий интерфейс.

Способ решения. Адаптер предусматривает создание класса-оболочки с требуемым интерфейсом.

Реализация паттерна «Адаптер» показана на примере адаптации животного под средство передвижения на рисунке 3.3.

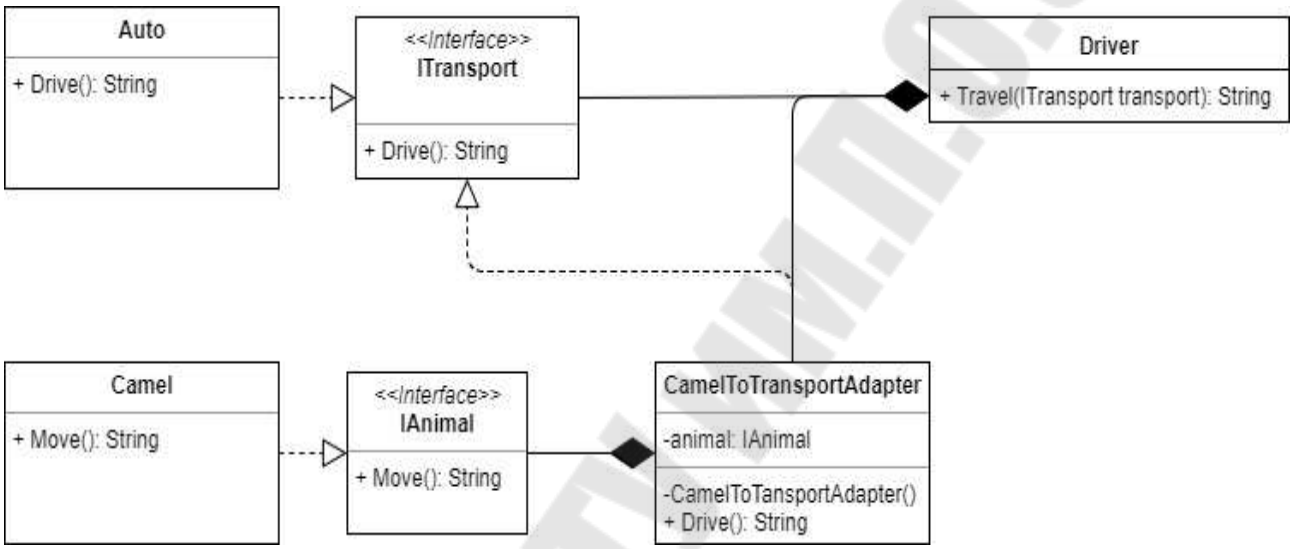

Рисунок 3.3 – Паттерн «Адаптер»

Фасад (англ. *Facade*) – структурный шаблон проектирования, позволяющий скрыть сложность системы путём сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы.

Особенности применения. Шаблон применяется для установки некоторого рода политики по отношению к другой группе объектов. Если политика должна быть яркой и заметной, следует воспользоваться услугами шаблона Фасад. Если же необходимо обеспечить скрытность и аккуратность (прозрачность), более подходящим выбором является шаблон Заместитель (*Proxy*).

Реализация паттерна «Фасад» показана на рисунке 3.4.

Библиотека ГГТУ им. П.О. Сухого

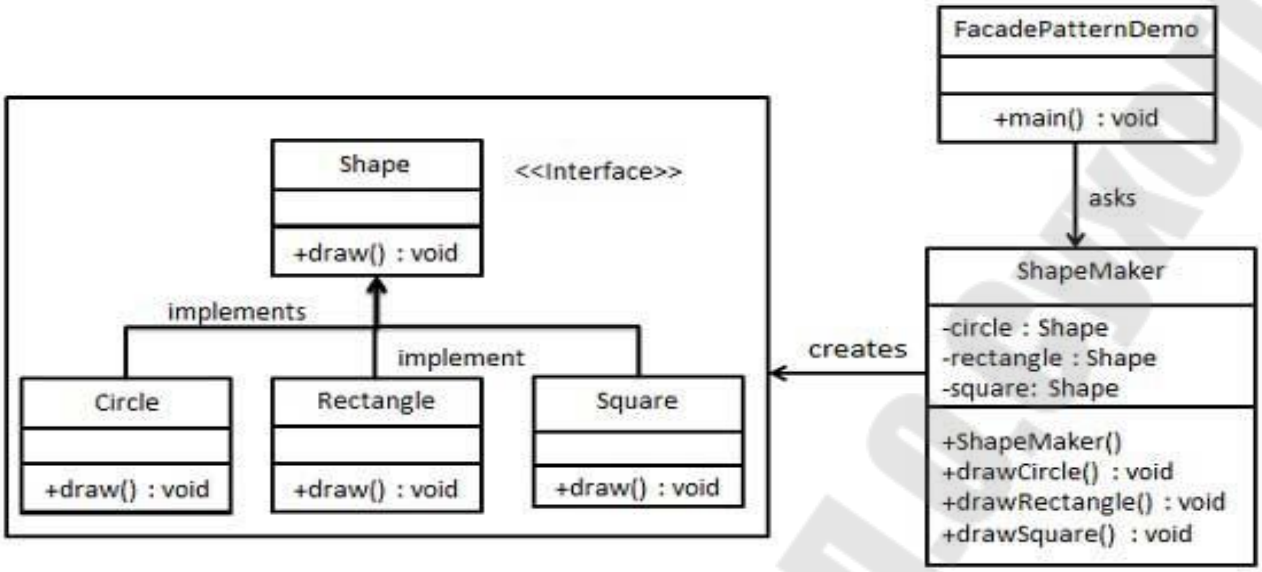

Рисунок 3.4 – Паттерн «Адаптер»

Библиотека ГГТУ им. П.О. Сухого

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №4

«ПОВЕДЕНЧЕСКИЕ ШАБЛОНЫ ПРОЕКТИРОВАНИЯ»

Цель занятия: изучить поведенческие шаблоны проектирования.

Поведенческие шаблоны (англ. *behavioral patterns*) – шаблоны проектирования, определяющие алгоритмы и способы реализации взаимодействия различных объектов и классов.

Поведенческие шаблоны проектирования:

- Цепочка обязанностей (*Chain of responsibility*) – предназначен для организации в системе уровней ответственности.
- Команда (*Command*) – представляет действие. Объект команды заключает в себе само действие и его параметры.
- Интерпретатор (*Interpreter*) – решает часто встречающуюся, но подверженную изменениям, задачу.
- Итератор (*Iterator*) – представляет собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из объектов, входящих в состав агрегации.
- Посредник (*Mediator*) – обеспечивает взаимодействие множества объектов, формируя при этом слабую связанность и избавляя объекты от необходимости явно ссылаться друг на друга.
- Хранитель (*Memento*) – позволяет, не нарушая инкапсуляцию зафиксировать и сохранить внутренние состояния объекта так, чтобы позднее восстановить его в этих состояниях.
- Наблюдатель (*Observer*) – определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии.
- Состояние (*State*) – используется в тех случаях, когда во время выполнения программы объект должен менять своё поведение в зависимости от своего состояния.

Интерпретатор (англ. *Interpreter*) – поведенческий шаблон проектирования, решающий часто встречающуюся, но подверженную изменениям, задачу. Также известен как *Little* (*Small*) *Language*.

Проблема. Имеется часто встречающаяся, подверженная изменениям задача.

Решение. Создать интерпретатор, который решает данную задачу.

Реализация паттерна «Интерпретатор» показана на UML-диаграмме на рисунке 4.1.

Библиотека ГГТУ им. П.О. Сухого

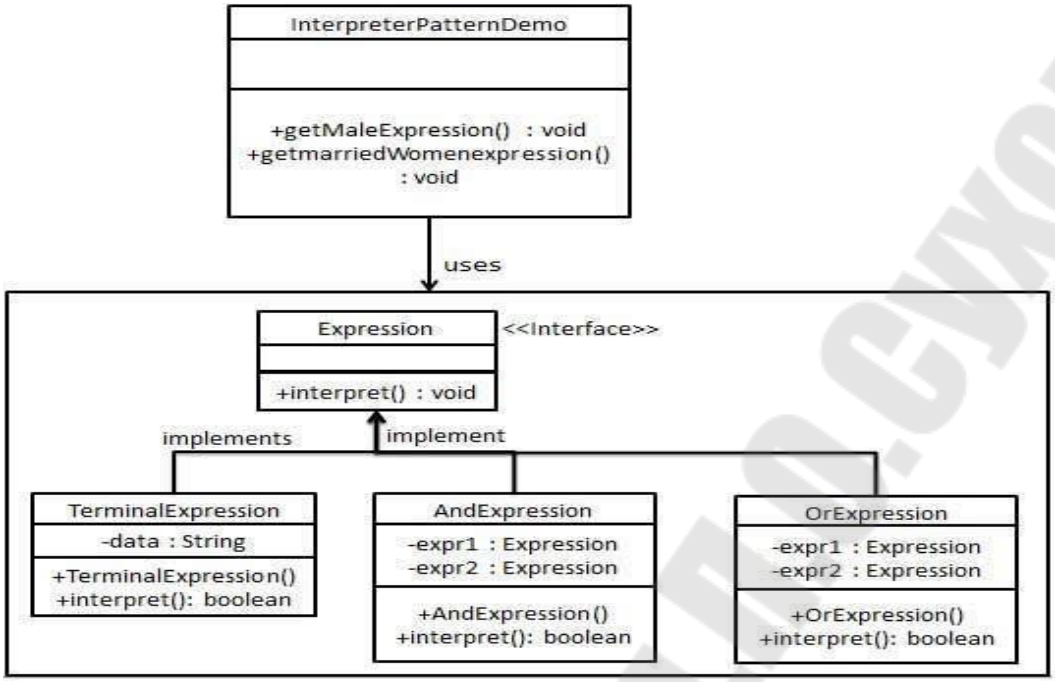

Рисунок 4.1 – Паттерн «Интерпретатор»

Команда (англ. *Command*) – поведенческий шаблон проектирования, используемый при объектно-ориентированном программировании, представляющий действие. Объект команды заключает в себе само действие и его параметры.

Цель. Создание структуры, в которой класс-отправитель и класс-получатель не зависят друг от друга напрямую. Организация обратного вызова к классу, который включает в себя класс-отправитель.

Реализация паттерна «Команда» показана на рисунке 4.2.

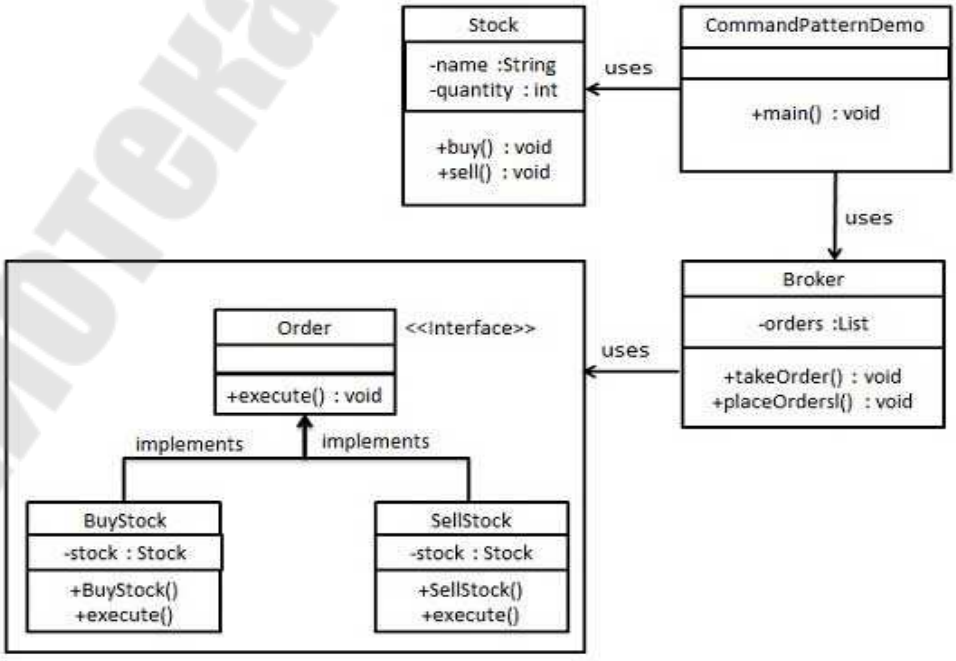

Рисунок 4.2 – Паттерн «Команда»

Библиотека ГГТУ им. П.О.Сухого

Наблюдатель (англ. *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.

Назначение. Определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии.

Реализация паттерна «Наблюдатель» показана на UML-диаграмме на рисунке 4.3.

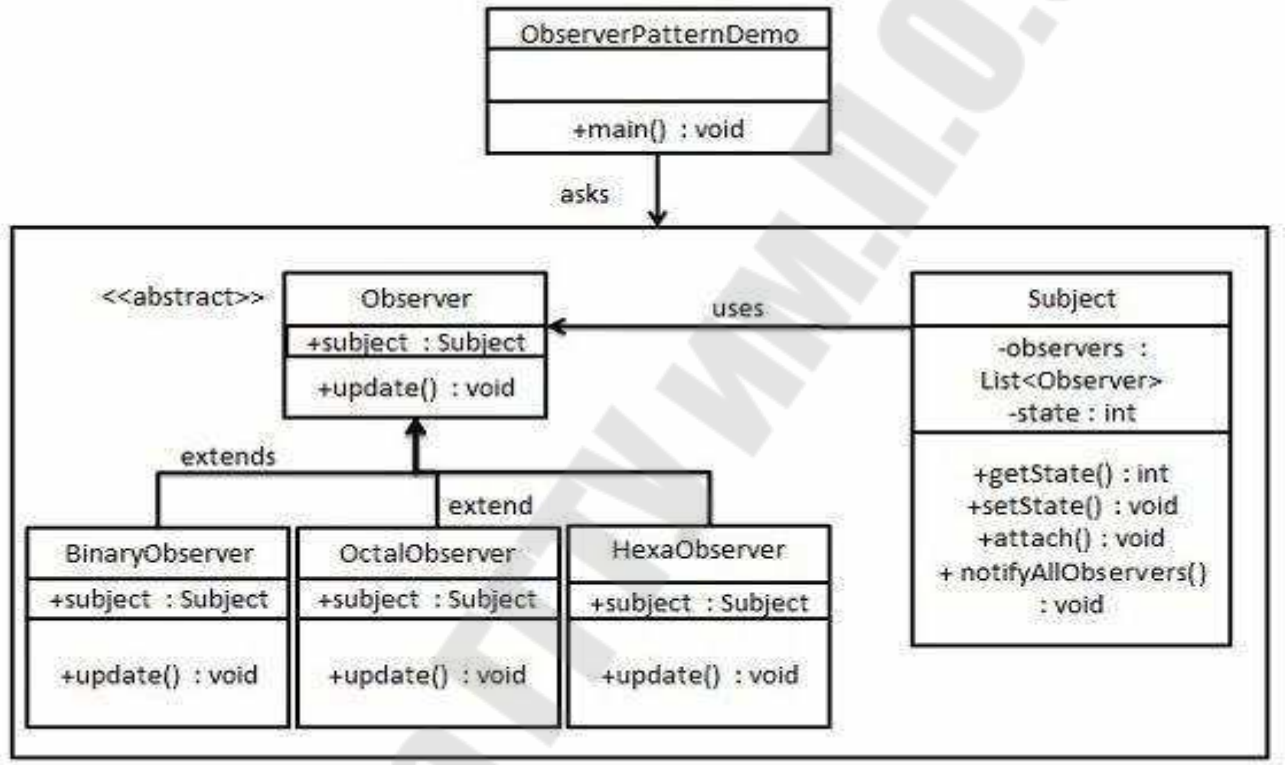

Рисунок 4.3 – Паттерн «Наблюдатель»

Библиотека ГГТУ им. П.О. Сухого

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №5

«АРХИТЕКТУРНЫЕ ШАБЛОНЫ ПРОЕКТИРОВАНИЯ»

Цель занятия: изучить архитектурные шаблоны проектирования.

Архитектурный шаблон проектирования является общим многоразовым решением общей проблемы в архитектуре программного обеспечения в рамках данного контекста.

Архитектурные шаблоны проектирования:

- *Model View Controller* (*MVC*) – схема использования нескольких шаблонов проектирования, с помощью которых модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные.
- *Model View View-Model* (*MVVM*) – *MVVM* удобно использовать вместо классического *MVC* и ему подобных в тех случаях, когда в платформе, на которой ведётся разработка, присутствует «связывание данных». В шаблонах проектирования *MVC/MVP* изменения в пользовательском интерфейсе не влияют непосредственно на Модель, а предварительно идут через Контроллер или *Presenter*.
- *Model View Presenter* (*MVP*) – шаблон проектирования, производный от *MVC*, который используется в основном для построения пользовательского интерфейса. Элемент *Presenter* в данном шаблоне берёт на себя функциональность посредника (аналогично контроллеру в *MVC*) и отвечает за управление событиями пользовательского интерфейса (например, использование мыши) так же, как в других шаблонах обычно отвечает представление.

Model View Controller (***MVC***, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

Схема работы *MVC* приведена на рисунке 5.1.

Библиотека ГГТУ им. П.О. Сухого

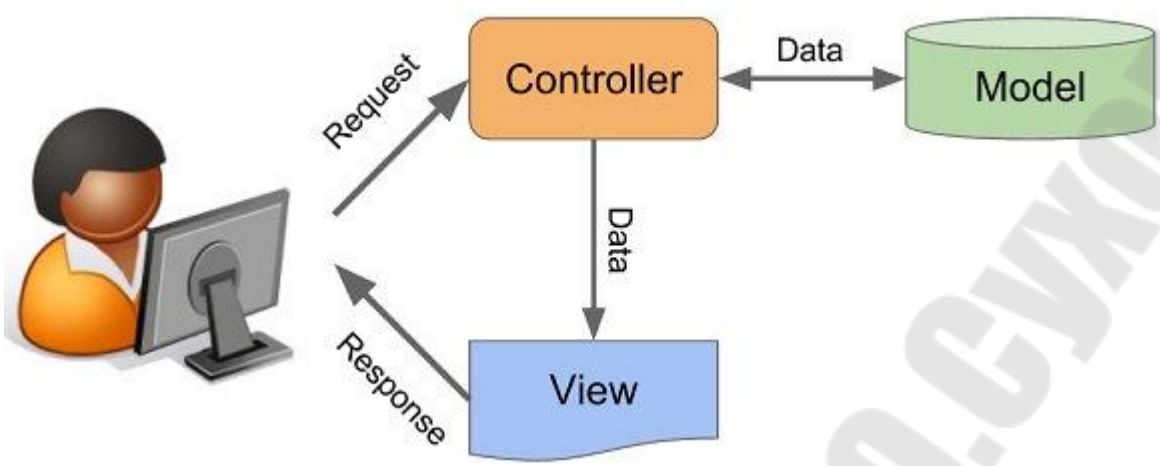

Рисунок 5.1 – Паттерн *MVC*

MVVM (*Model-View-ViewModel*) – это шаблон, который появился для обхода ограничений паттернов *MVC* и *MVP*, и объединяющий некоторые из их сильных сторон. Эта модель впервые появилась в составе фреймворка *Small Talk* в 80-х, и была позднее улучшена с учетом обновленной модели презентаций (*MVP*).

Шаблон *MVVM* имеет три основных компонента: модель, которая представляет бизнес-логику приложения, представление пользовательского интерфейса *XAML*, и представление-модель, в котором содержится вся логика построения графического интерфейса и ссылка на модель, поэтому он выступает в качестве модели для представления.

На рисунке 5.2 представлена диаграмма, которая показывает, как реализовать шаблон *MVVM*.

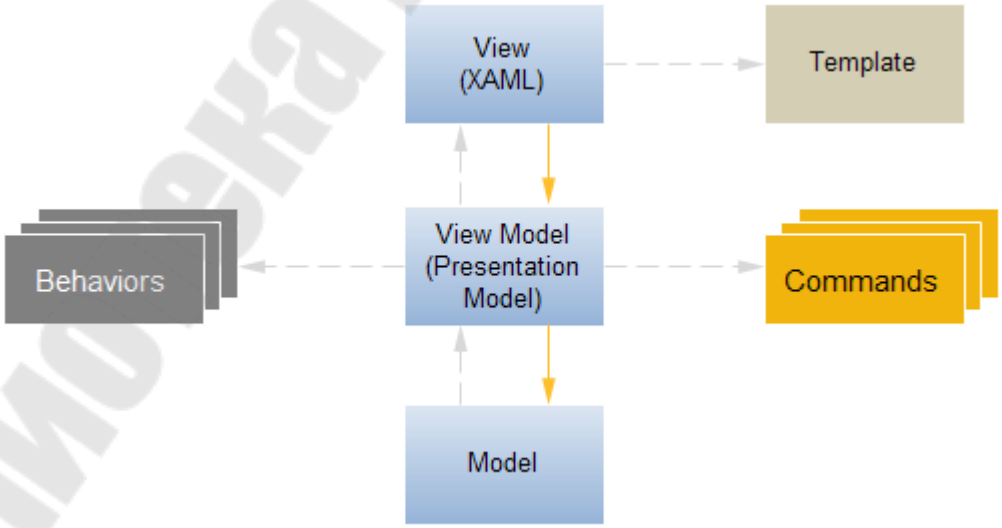

Рисунок 5.2 – Паттерн *MVVM*

Библиотека ГГТУ им. П.О.Сухого

Список использованных источников

1. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.] ; пер. с англ. А. Слинкина. - Санкт-Петербург [и др.] : Питер, 2014. – 366 с.
2. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. – Санкт-Петербург [и др.] : Питер, 2017. – 366 с.
3. Фаулер, М. Шаблоны корпоративных приложений / М. Фаулер. – М. «Вильямс», 2012. – 544 с.

Библиотека ГГТУ им.П.О.Сухого