Лабораторная работа «Шаблоны проектирования»

Цель работы: ознакомиться с основными шаблонами проектирования, научиться применять их при проектировании и разработке ПО.

Рекомендуемое ПО для практической части: JDK 8; IntelliJ IDEA 14 Community Edition; плагин PlantUML integration; Graphviz.

Теоретические сведения

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектноориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие именно конечные классы или объекты приложения будут использоваться.

Сообразное использование паттернов проектирования дает разработчику ряд неоспоримых преимуществ. Приведем некоторые из них. Модель системы, построенная в терминах паттернов проектирования, фактически является структурированным выделением тех элементов и связей, которые значимы при решении поставленной задачи. Помимо этого, модель, построенная с использованием паттернов проектирования, более проста и наглядна в изучении, чем стандартная модель. Тем не менее, несмотря на простоту и наглядность, она позволяет глубоко и всесторонне проработать архитектуру разрабатываемой системы с использованием специального языка. Применение паттернов проектирования повышает устойчивость системы к изменению требований и упрощает неизбежную последующую доработку системы. Кроме того, трудно переоценить роль использования паттернов при интеграции информационных систем организации. Также следует упомянуть, что совокупность паттернов проектирования, по сути, представляет собой единый словарь проектирования, который, будучи унифицированным средством, незаменим для общения разработчиков друг другом.

Но самое главное любой шаблон проектирования может стать палкой о двух концах: если он будет применен не к месту, это может обернуться катастрофой и создать вам много проблем в последующем. В то же время, реализованный в нужном месте, в нужное время, он может стать для вас настоящим спасителем.

Есть три основных вида шаблонов проектирования:

- структурные;

- порождающие;
- поведенческие.

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

Порождающие шаблоны 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.

Поведенческие шаблоны определяют взаимодействие между объектами, увеличивая таким образом его гибкость.

Нужно учесть, что шаблонов очень много, и рассмотрение всех шаблонов может занять целый учебный курс. Поэтому ниже кратко рассмотрены лишь ***некоторые*** из существующих паттернов. Если будет необходима дополнительная информация о конкретном шаблоне, то обращайтесь к литературе.

Структурные шаблоны

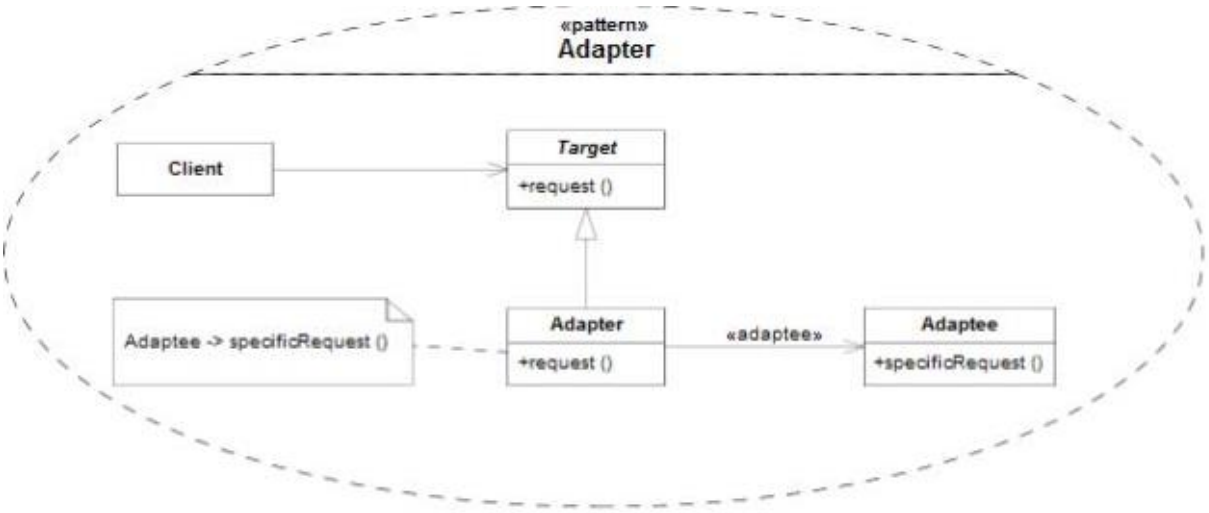
Адаптер (Adapter)

Проблема: необходимо обеспечить взаимодействие несовместимых интерфейсов или создать единый устойчивый интерфейс для нескольких компонентов с разными интерфейсами.

Решение: преобразовать исходный интерфейс компонента к другому виду с помощью промежуточного объекта адаптера, то есть, добавить специальный объект с общим интерфейсом в рамках данного приложения и перенаправить связи от внешних объектов к этому объекту адаптеру.

Класс ***Adapter*** приводит интерфейс класса***Adaptee*** в соответствие с интерфейсом класса***Target*** (наследником которого является***Adapter***). Это позволяет объекту***Client*** использовать объект***Adaptee*** (посредством адаптера***Adapter***) так, словно он является экземпляром класса***Target.***

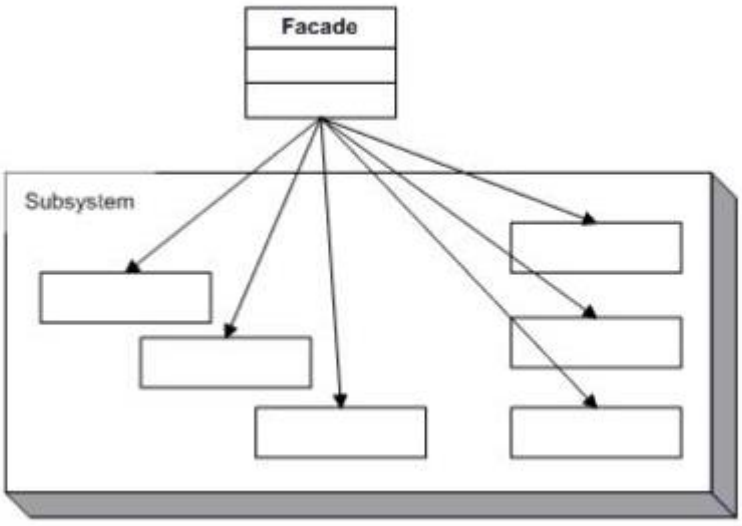
Таким образом ***Client*** обращается к интерфейсу***Target***, реализованному в наследнике***Adapter***, который перенаправляет обращение к***Adaptee***.



Шаблон Адаптер позволяет включать уже существующие объекты в новые объектные структуры, независимо от различий в их интерфейсах.

Этот шаблон позволяет в процессе проектирования не принимать во внимание возможные различия в интерфейсах уже существующих классов. Если есть класс, обладающий требуемыми методами и свойствами (по крайней мере, концептуально), то при необходимости всегда можно воспользоваться шаблоном Адаптер для приведения его интерфейса к нужному виду.

Фасад (Facade)



Шаблон “фасад” структурный шаблон проектирования, позволяющий скрыть сложность системы путем сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы.

Проблема: как обеспечить унифицированный интерфейс с набором разрозненных реализаций или интерфейсов, например, с подсистемой, если нежелательно высокое связывание с этой подсистемой или реализация подсистемы может измениться?

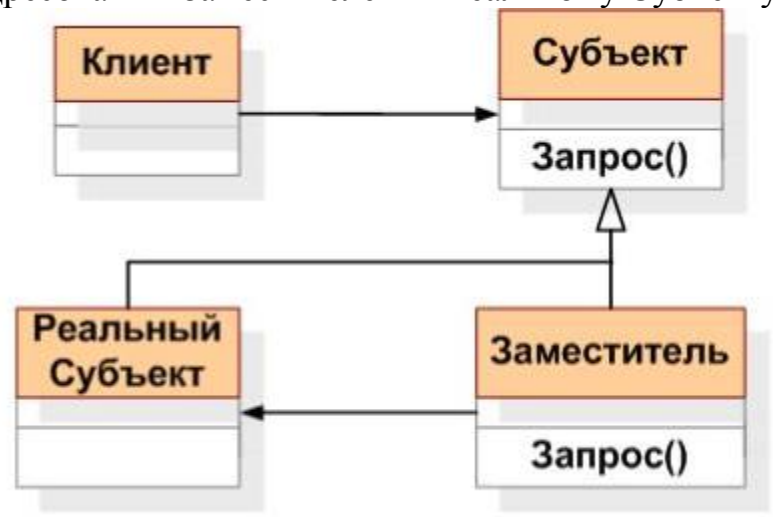
Решение: определить одну точку взаимодействия с подсистемой — фасадный объект, обеспечивающий общий интерфейс с подсистемой, и возложить на него обязанность по взаимодействию с её компонентами. Фасад — это внешний объект, обеспечивающий единственную точку входа для служб подсистемы. Реализация других компонентов подсистемы закрыта и не видна внешним компонентам.

Заместитель (Proxy)

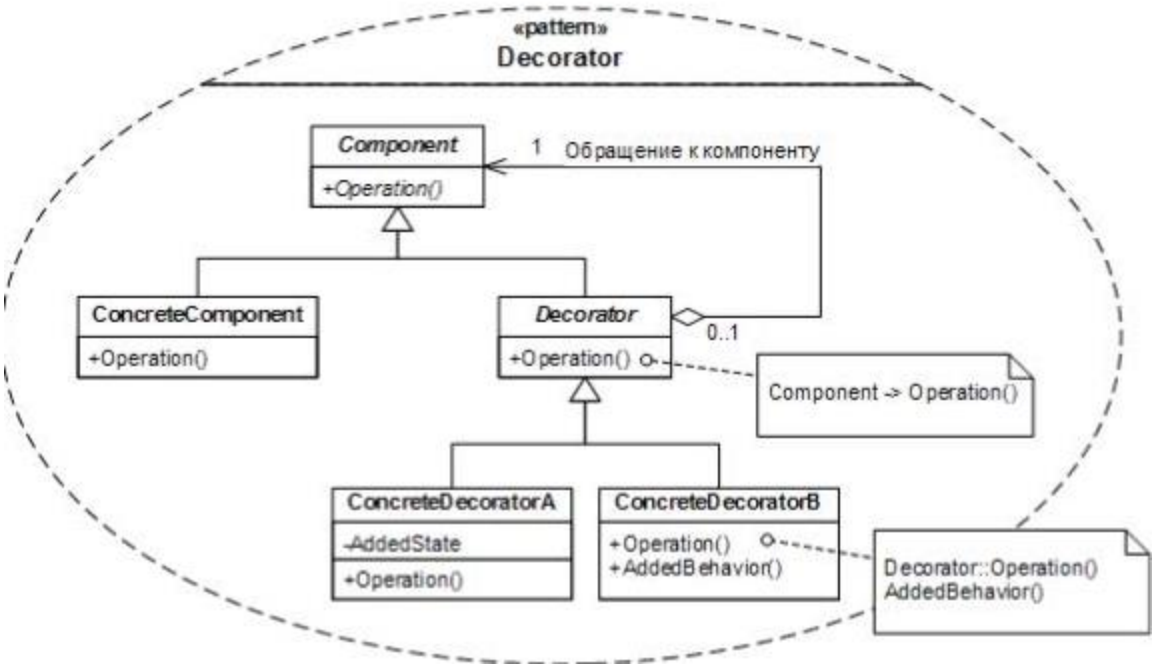
Заместитель — структурный шаблон проектирования, который предоставляет объект, который контролирует доступ к другому объекту, перехватывая все вызовы.

Проблема: необходимо управлять доступом к объекту так, чтобы не создавать громоздкие объекты «по требованию».

Решение: создать суррогат громоздкого объекта. «Заместитель» хранит ссылку, которая позволяет заместителю обратиться к реальному субъекту (объект класса «Заместитель» может обращаться к объекту класса «Субъект», если интерфейсы «Реального Субъекта» и «Субъекта» одинаковы). Поскольку интерфейс «Реального Субъекта» идентичен интерфейсу «Субъекта», так, что «Заместителя» можно подставить вместо «Реального Субъекта». «Заместитель» контролирует доступ к «Реальному Субъекту», может отвечать за создание или удаление «Реального Субъекта». «Субъект» определяет общий для «Реального Субъекта» и «Заместителя» интерфейс, так, что «Заместитель» может быть использован везде, где ожидается «Реальный Субъект». При необходимости запросы могут быть переадресованы «Заместителем» «Реальному Субъекту».



Декоратор (Decorator)



Декоратор — структурный шаблон проектирования, предназначенный для динамического подключения дополнительного поведения к объекту. Шаблон Декоратор предоставляет гибкую альтернативу практике создания подклассов с целью расширения функциональности.

Задача: 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.

Решение: шаблон “декоратор” предусматривает расширение функциональности объекта без определения подклассов.

Класс ***ConcreteComponent*** — класс, в который с помощью шаблона Декоратор добавляется новая функциональность. В некоторых случаях базовая функциональность предоставляется классами, производными от класса***ConcreteComponent***. В подобных случаях класс***ConcreteComponent*** является уже не конкретным, а абстрактным. Абстрактный класс***Component*** определяет интерфейс для использования всех этих классов.

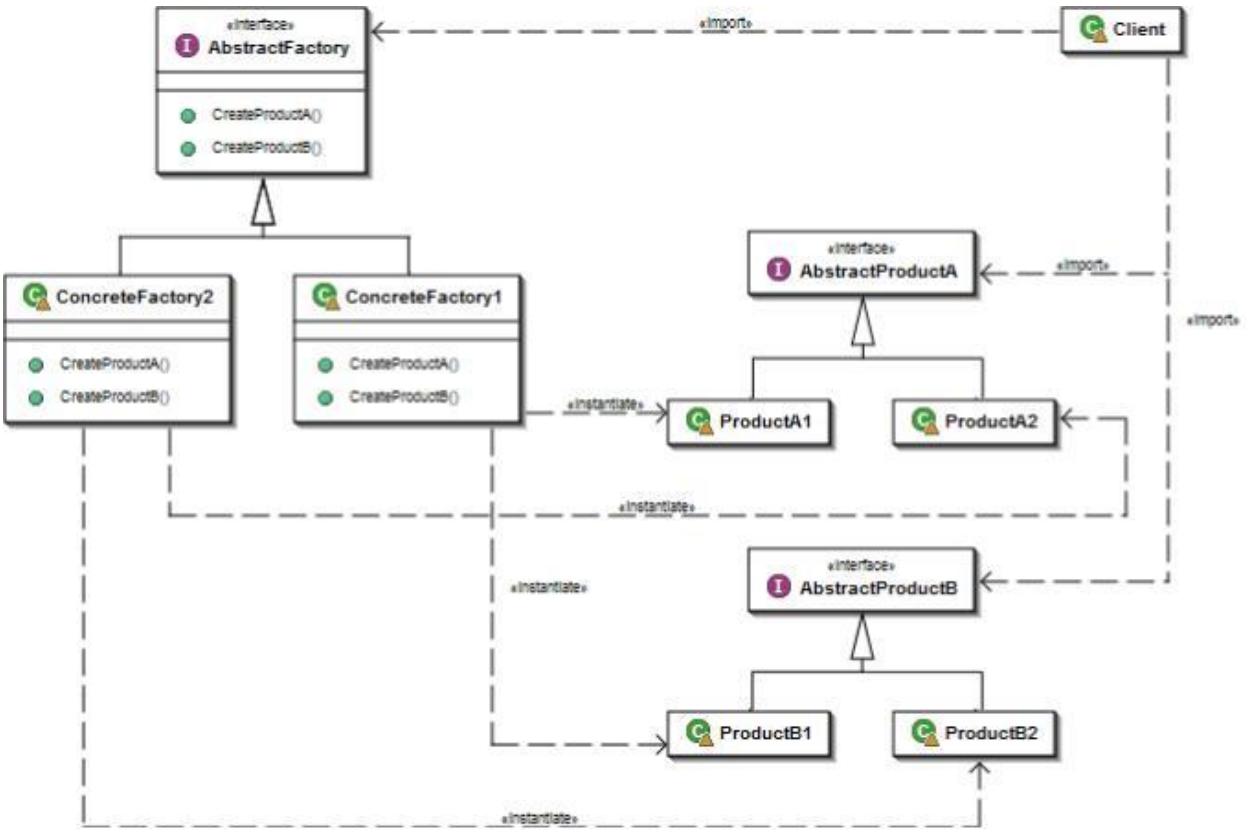
Порождающие шаблоны

Абстрактная фабрика (Abstract Factory)

Абстрактная фабрика — порождающий шаблон проектирования, позволяющий изменять поведение системы, варьируя создаваемыми

объектами, при этом сохраняя интерфейсы. Он позволяет создавать целые группы взаимосвязанных объектов, которые, будучи созданными одной фабрикой, реализуют общее поведение. 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.

Этот шаблон предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.



Фабричный метод (Factory Method)

Фабричный метод — порождающий шаблон проектирования, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, Фабрика делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

Шаблон определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать. Фабричный метод позволяет классу делегировать создание подклассов. Используется, когда:

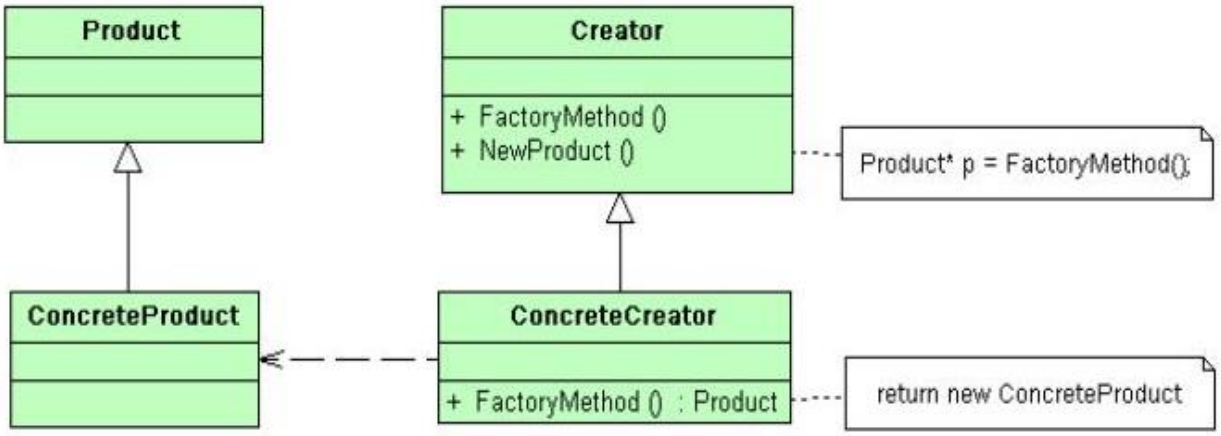
- классу заранее неизвестно, объекты каких подклассов ему нужно создавать.
- класс спроектирован так, чтобы объекты, которые он создаёт, специфицировались подклассами.
- класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и планируется локализовать знание о том, какой класс принимает эти обязанности на себя.

Структура:

Product — продукт; определяет интерфейс объектов, создаваемых абстрактным методом.***ConcreteProduct*** — конкретный продукт, реализует интерфейс***Product.***

Creator — создатель; объявляет фабричный метод, который возвращает объект типа***Product***. Может также содержать реализацию этого метода «по умолчанию»; может вызывать фабричный метод для создания объекта типа Product.

ConcreteCreator — конкретный создатель; переопределяет фабричный метод таким образом, чтобы он создавал и возвращал объект класса***ConcreteProduct.***



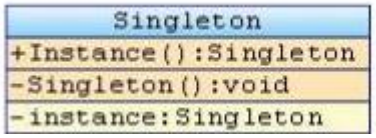
Одиночка (Singleton)

Одиночка — порождающий шаблон проектирования, гарантирующий, что в однопоточном приложении будет единственный экземпляр класса с глобальной точкой доступа.

Существенно то, что можно пользоваться именно ***экземпляром*** класса, так как при этом во многих случаях становится доступной более широкая функциональность.

Глобальный «одинокий» объект — именно объект, а не набор процедур, не привязанных ни к какому объекту — бывает нужен:

- если используется существующая объектноориентированная библиотека;
- если есть шансы, что один объект когданибудь превратится в несколько;
- если интерфейс объекта (например, игрового мира) слишком сложен, и не стоит засорять основное пространство имён большим количеством функций;
- если, в зависимости от каких-нибудь условий и настроек, создаётся один из нескольких объектов. Например, в зависимости от того, ведётся лог или нет, создаётся или настоящий объект, пишущий в файл, или «заглушка», ничего не делающая.



Поведенческие шаблоны

Стратегия (Strategy)

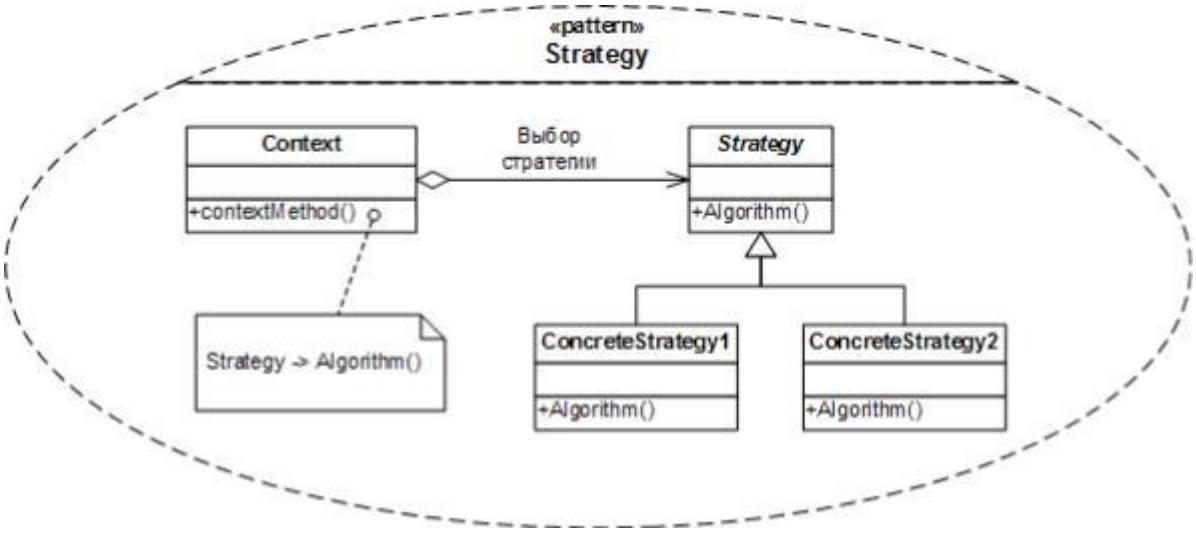
Стратегия — поведенческий шаблон проектирования, предназначенный для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм путем определения соответствующего класса. Шаблон Strategy позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют.

Проблема: по типу клиента (или по типу обрабатываемых данных) выбрать подходящий алгоритм, который следует применить. Если используется правило, которое не подвержено изменениям, нет необходимости обращаться к шаблону «стратегия».

Решение: отделение процедуры выбора алгоритма от его реализации. Это позволяет сделать выбор на основании контекста.

Класс Strategy определяет, как будут использоваться различные алгоритмы. Конкретные классы ConcreteStrategy реализуют эти различные алгоритмы. Класс Context использует конкретные классы ConcreteStrategy посредством ссылки на конкретный тип абстрактного класса Strategy. Классы Strategy и Context взаимодействуют с целью реализации выбранного

алгоритма (в некоторых случаях классу Strategy требуется посылать запросы классу Context). Класс Context пересылает классу Strategy запрос, поступивший от его классаклиента.



Наблюдатель (Observer)

Наблюдатель — поведенческий шаблон проектирования. Создает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов об изменении их состояния, тем самым наблюдая за ними.

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

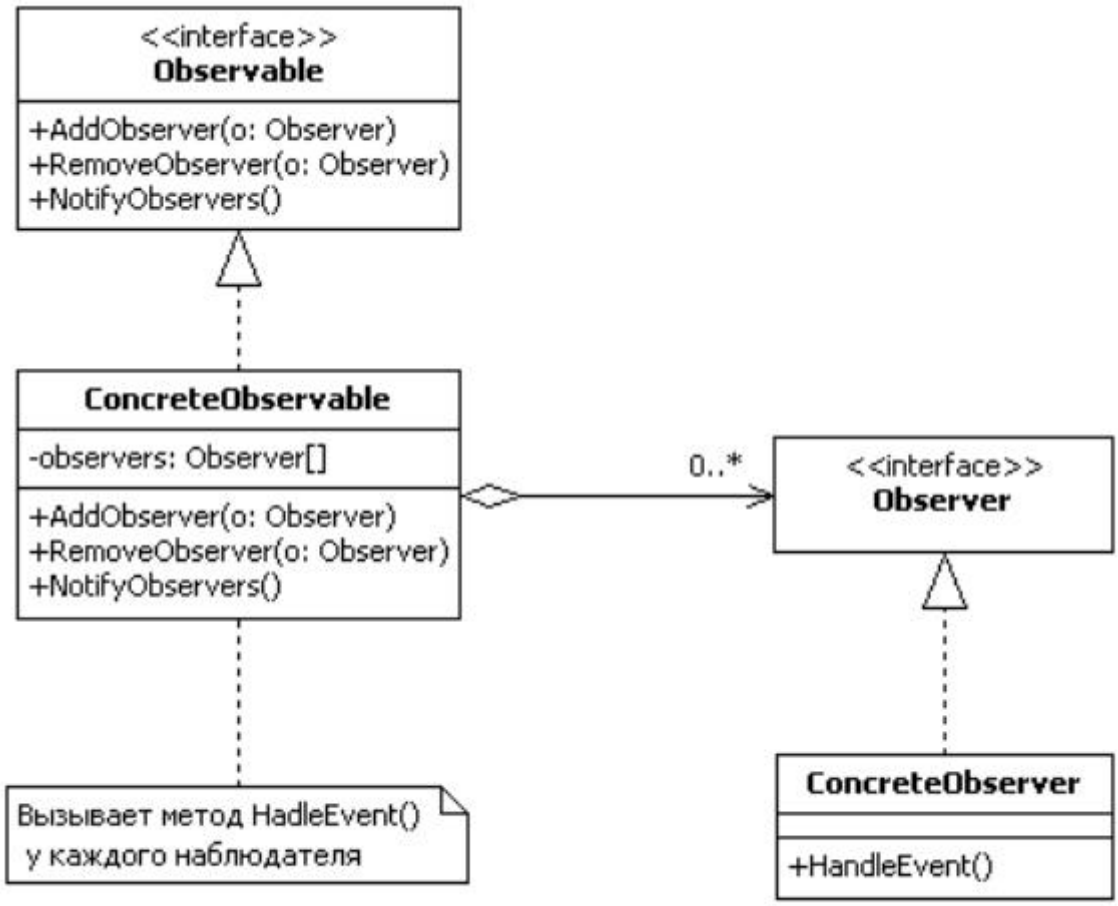
При реализации шаблона «наблюдатель» обычно используются следующие классы:

- **Observable** — интерфейс, определяющий методы для добавления, удаления и оповещения наблюдателей;
- **Observer** — интерфейс, с помощью которого наблюдатель получает оповещение;
- **ConcreteObservable** — конкретный класс, который реализует интерфейс Observable;
- **ConcreteObserver** — конкретный класс, который реализует интерфейс Observer.

Шаблон «наблюдатель» применяется в тех случаях, когда система обладает следующими свойствами:

- существует, как минимум, один объект, рассылающий сообщения;
- имеется не менее одного получателя сообщений, причём их количество и состав могут изменяться во время работы приложения;
- нет надобности очень сильно связывать взаимодействующие объекты, что полезно для повторного использования.

Данный шаблон часто применяют в ситуациях, в которых отправителя сообщений не интересует, что делают получатели с предоставленной им информацией.

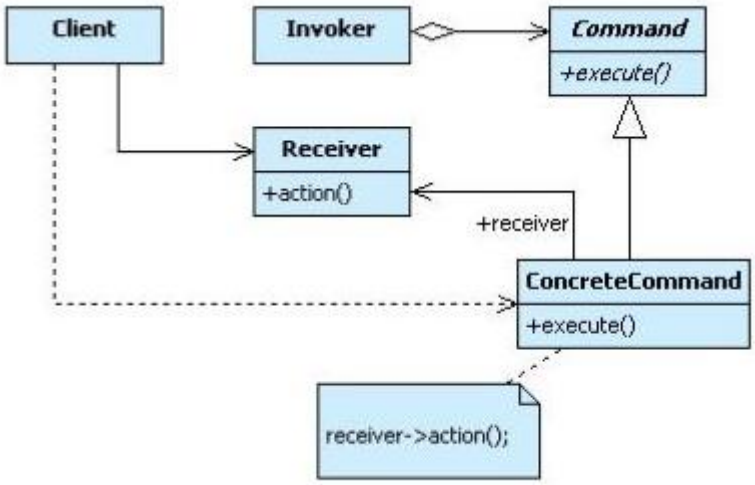


Команда (Command)

Команда — поведенческий шаблон проектирования, используемый при объектноориентированном программировании, представляющий действие. Объект команды заключает в себе само действие и его параметры.

Паттерн обеспечивает обработку команды в виде объекта, что позволяет сохранять её, передавать в качестве параметра методам, а также возвращать её в виде результата, как и любой другой объект.

Паттерн Command преобразовывает запрос на выполнение действия в отдельный объекткоманду. Такая инкапсуляция позволяет передавать эти действия другим функциям и объектам в качестве параметра, приказывая им выполнить запрошенную операцию. Команда – это объект, поэтому над ней допустимы любые операции, что и над объектом.



Интерфейс командного объекта определяется абстрактным базовым классом Command и в самом простом случае имеет единственный метод execute(). Производные классы определяют получателя

запроса (указатель на объектполучатель) и необходимую для выполнения операцию (метод этого объекта). Метод execute() подклассов Command просто вызывает нужную операцию получателя.

Впаттерне Command может быть до трех участников:

●Клиент, создающий экземпляр командного объекта.

●Инициатор запроса, использующий командный объект.

●Получатель запроса.

Сначала клиент создает объект ConcreteCommand, конфигурируя его получателем запроса. Этот объект также доступен инициатору. Инициатор использует его при отправке запроса, вызывая метод execute(). Этот алгоритм напоминает работу функции обратного вызова в процедурном программировании

– функция регистрируется, чтобы быть вызванной позднее.

Паттерн Command отделяет объект, инициирующий операцию, от объекта, который знает, как ее выполнить. Единственное, что должен знать инициатор, это как отправить команду. Это придает системе гибкость:

позволяет осуществлять динамическую замену команд, использовать сложные составные команды, осуществлять отмену операций.

Задания для самостоятельной работы

Общее задание:

Разработать каркас системы в соответствии с выданным вариантом задания.

При проектировании архитектуры **необходимо** использовать обозначенный в задании шаблон проектирования, иные варианты реализации задания **не принимаются.**

Программа должна функционировать в соответствии с приведённым в варианте задания примером работы.

Выбранный язык программирования должен поддерживать объектно-ориентированную парадигму. Выбор остального окружения производится студентом самостоятельно.

Требования к отчету:

1. Краткая характеристика приведённого в варианте задания паттерна (решаемая проблема проектирования, применимость, плюсы, минусы)
2. Шаги реализации приведённого в варианте задания паттерна
3. UML-диаграмма классов полученная в ходе проектирования каркаса системы с **обязательным** наличием пояснений (роль класса в паттерне).
4. Результаты работы полученной программы в виде скриншотов
5. Исходный код

При защите лабораторной работы быть готовым к ответу на вопросы, приведенные в конце данного документа.

1 Вариант. Паттерн «Фабрика».

Компания «Грузовоз» оказывает услуги по грузоперевозке. В настоящий момент компания занимается только частными перевозками, однако, планируется расширение и внедрение коммерческих перевозок.

Необходимо написать программу, которая будет принимать на вход идентификатор вида перевозки и строку с адресами отправления/назначения и выводить сообщение о том перевозка какого вида по каким адресам была осуществлена. Программа должна быть закрыта для изменения, но открыта для расширения.

Пример работы программы:

```
Выберите тип грузоперевозки:
1. Частная
2. Коммерческая
>> 1
Введите адреса отправления/назначения:
>> Иркутск, ул. Лермонтова, 83 – Иркутск ул. Чернышевского, 15
Была осуществлена частная грузоперевозка «Иркутск, ул. Лермонтова, 83 – Иркутск ул. Чернышевского, 15».
```

2 Вариант. Паттерн «Абстрактная фабрика».

Компания «Пиджачок» занимается пошивом костюмов, в комплект входят пиджак и брюки. В данный момент компания выпускает линейку костюмов «Котофей», однако планируется запустить в производство вторую линейку «Кембриджский профессор». Важно, чтобы при формировании заказа на партию костюмов костюмы принадлежали одной линейке.

Необходимо написать программу, которая будет принимать на вход идентификатор линейки костюмов и размер партии и выводить сообщение о том какие и в каком количестве элементы костюма были произведены. Программа должна быть закрыта для изменения, но открыта для расширения.

Пример работы программы:

```
Выберите линейку костюмов:
1. Котофей
2. Кембриджский профессор
>> 1
Введите размер партии:
>> 1500
Был осуществлен заказ на 1500 брюк линейки «Котофей» и 1500 пиджаков линейки «Котофей».
```

3 Вариант. Паттерн «Строитель».

Организация АО «Компот» работает в сфере общественного питания, специализируясь на пельменях. Широкий ассортимент включает в себя такие виды пельменей как «Охотничьи», которые состоят из толстого теста, оленины, в качестве специй используются пряные таежные травы, и «Изысканные», которые состоят из тонкого теста, крольчатины, из специй добавляется душистый перец. Лепка пельменей состоит из операций раскатывания теста, укладки начинки и добавления специй.

Необходимо написать программу, которая будет принимать на вход идентификатор вида пельменей и выводить сообщение о составе созданного продукта. В архитектуре программы предполагается наличие класса с ролью Director. Программа должна быть закрыта для изменения, но открыта для расширения.

Пример работы программы:

```
Выберите вид пельменей:
1. Охотничьи
2. Изысканные
>> 1
Была слеплена порция пельменей «Охотничьи». Описание: толстое тесто, оленина, пряные таежные травы.
```

4 Вариант. Паттерн «Адаптер».

Вы принимаете участие в проекте по разработке системы распознавания текста с бланков в качестве системного архитектора. Ситуация такова, что в текущем году форма заполнения бланков была изменена.

На текущей стадии решено, что главный класс вашей системы будет иметь метод «распознать текст», в который через параметры передается экземпляр класса «Бланк с новой формой». Необходимо доработать архитектуру системы так, чтобы она поддерживала устаревшие бланки со

старой формой. Программа должна быть закрыта для изменения, но открыта для расширения.

Пример работы программы:

```
Выберите тип бланка:
1. Бланк со старой формой
2. Бланк с новой формой
>> 1
Был распознан бланк со старой формой через адаптер «Имя адаптера».
>> 2
Был распознан бланк с новой формой штатными методами системы.
```

5 Вариант. Паттерн «Мост».

Компания АО «Автомеш» занимается установкой систем сигнализации моделей «Защита-200» и «Защита-400» на автомобили марок «Audi» и «Volkswagen». Известно, что у всех автомобилей интерфейс подключения систем сигнализации примерно одинаковый.

Вам необходимо реализовать каркас системы, связывающей системы сигнализации с автомобилями. Программа будет принимать на вход идентификатор системы сигнализации и идентификатор марки и возвращать сообщение о том, система сигнализации какой модели на автомобиль какой марки была установлена. Программа должна быть закрыта для изменения, но открыта для расширения.

Пример работы программы:

```
Выберите модель сигнализации:
1. Защита-200
2. Защита-400
>> 1
Выберите марку автомобиля:
1. Audi
2. Volkswagen
>> 1
Была установлена система сигнализация модели «Защита-200» на автомобиль марки «Audi».
```

6 Вариант. Паттерн «Декоратор».

Вы задействованы в создании программного продукта для сети кондитерских мастерских «Пирожокъ» в качестве системного архитектора. Данная организация занимается приемом заказов не изготовление тортов. Известно, что при оформлении заказа на торт можно выбрать:

- классические или шоколадные коржи;
- фруктовую или ягодную пропитку;
- сливочную или шоколадную глазурь.

Вам необходимо реализовать каркас системы оформления заказа. Программа будет принимать на вход идентификатор составляющих торта и возвращать сообщение о том, какой торт был заказан. Программа должна быть закрыта для изменения, но открыта для расширения.

Пример работы программы:

```
Выберите тип коржа:
1. Классический
2. Шоколадный
>> 1
```

```
Выберите тип пропитки:
1. Фруктовая
2. Ягодная
>> 1
Выберите тип глазури:
3. Сливочная
4. Шоколадная
>> 1

Был заказан следующий торт: классический корж, фруктовая пропитка, сливочная глазурь. Все верно?
```

7 Вариант. Паттерн «Компоновщик».

Вы проектируете структуру файлового хранилища в некотором программном продукте. Известно, что в системе могут храниться два типа файлов: текстовые и файлы-изображения. У каждого файла есть размер (в мегабайтах), который может быть изменен и показан. Файлы могут быть вложены в папки.

Необходимо реализовать каркас файлового хранилища. Программа будет принимать на вход данные о том, какой элемент файлового хранилища должен быть создан. В случае, если выбран файл, то дополнительно указывается его размер. В случае, если выбрана папка, то последующие создаваемые файлы считаются вложенными в нее. После окончания ввода программа должна вывести объем занимаемой файловым хранилищем памяти, а также количество хранимых файлов и папок. Программа должна быть закрыта для изменения, но открыта для расширения.

Пример работы программы:

```
Выберите тип создаваемого элемента:
1. Текстовый файл
2. Файл-изображение
3. Папка
4. Конец
>> 1 1 //(второй параметр – размер файла)
>> 3
>> 2 4
>> 1 2
>> 3
>> 2 2

Объем памяти, занимаемый файловым хранилищем составил 9 мегабайт. Хранится: 4 файла и 2 папки.
```

8 Вариант. Паттерн «Фасад».

Вы принимаете участие в создании пакета для статистического анализа данных. В ваши задачи входит создание архитектуры модуля проведения описательного анализа. В ходе анализа производится расчёт среднего выборочного, медианы и моды (отдельный класс «средние величины»), расчет значений квартилей (отдельный класс «процентили распределения»), а также расчет дисперсии и среднеквадратического отклонения (отдельный класс «показатели рассеивания»).

Необходимо реализовать каркас модуля. Для обеспечения удобного интерфейса доступа ко всему описательному анализу главный класс модуля должен иметь метод «Произвести описательный анализ», который

производит все вышеперечисленные операции. При запуске программа должна вызывать этот метод главного класса модуля и выводить названия операций, которые были произведены. Программа должна быть закрыта для изменения, но открыта для расширения. **Вычислений вышеприведенных характеристик производить не требуется, выводятся только названия операций.**

Пример работы программы:

```
Было рассчитано среднее выборочное
Была рассчитана медиана
Была рассчитана мода
Были рассчитаны значения квартилей
Были рассчитаны дисперсия и среднеквадратическое отклонение.
```

9 Вариант. Паттерн «Легковес»

Вы принимаете участие в разработке видеоигры в качестве системного архитектора. В разрабатываемой игре предполагается наличие большого количества однотипных объектов. В контексте игры эти объекты представляют собой виртуальных котов. Объекты обладают определенной моделью, текстурой для этой модели, координатами в пространстве и некоторым поведением. Всего предполагается три вида поведения: спать, бежать и лакать. У всех объектов предполагается наличие одной модели и текстуры.

Необходимо разработать архитектуру данного фрагмента приложения и запрограммировать полученное решение. Спроектированная архитектура должна обеспечивать оптимизацию используемой памяти. В результате работы программа должна выводить сообщения с описанием состояния объектов класса «Кот». Состояние задается случайно при создании объекта. Программа должна быть закрыта для изменения, но открыта для расширения.

Пример работы программы:

```
Объект #1: {модель: cat.3dm; текстура cat.png; координаты: {x:274;y:2;z:23}; состояние: спит;}
                                    ...
Объект #N: {модель: cat.3dm; текстура cat.png; координаты: {x:23;y:12;z:263}; состояние: лакает;}
```

10 Вариант. Паттерн «Заместитель»

Вы занимаетесь разработкой службы по сбору статистики о результативности работы сотрудников компании «Искусственные Сети Предикатов». Информация о результативности работы сотрудников хранится в базах стороннего сервиса. Доступ к ней возможен только путем удаленного вызова процедур. Вызываемые процедуры возвращают следующую информацию: количество отработанных сотрудником часов, количество решенных сотрудником трудовых задач. Однако, при интенсивных обращениях к данному сервису начинают возникать временные задержки с получением ответа на запрос, а также могут возникать сбои в его работе.

Необходимо разработать архитектуру службы так, чтобы нивелировать недостатки стороннего сервиса, но при этом использовать для доступа к данным те же методы. Программа должна быть закрыта для изменения, но

открыта для расширения. При работе программы необходимо имитировать работу стороннего сервиса путем создания объекта класса «сторонний сервис» с соответствующими процедурами. Обращаться к методам объекта класса «сторонний сервис» напрямую не допускается. Возвращаемые процедурами данные могут быть жестко прописаны прямо в коде.

Пример работы программы:

```
>>Запрос
Количество отработанных часов: 40; Количество решенных трудовых задач: 18. Данные получены от стороннего сервиса.
>>Запрос
Количество отработанных часов: 40; Количество решенных трудовых задач: 18. Данные получены из кеша.
```

11 Вариант. Паттерн «Цепочка обязанностей»

Компания N занимается доставкой посылок по всему миру воздушным путем. Так как некоторые города не связаны авиамаршрутом, то доставка осуществляется через транзитные города. Известно, что авиамаршрутами связаны следующие города:

Энск⇔Ивентск

Ивентск⇔Шаблонск

Шаблонск⇔Диаграмск

Шаблонск⇔Атрибутск (предполагается, что на момент тестирования системы авиамаршрут временно не функционирует)

Вам поручена задача по разработке прототипа сервиса по обеспечению доставки посылок. В случае, если город, в который прибыла посылка не является пунктом назначения, то система должна самостоятельно определить какой город будет являться следующим в пути посылки, руководствуясь хранимой в ней информации о авиамаршрутах. В случае, если необходимый авиамаршрут в момент регистрации посылки временно не функционирует, то система выводит соответствующее сообщение (не волнуйтесь, посылки будут отправлены, как только снабжение восстановится). Предполагается, что о наличии и функционировании конкретного авиамаршрута известно только в городе, который является его частью (пунктом отправления или прибытия). Программа должна принимать на входе данные о пункте отправления и возвращать пройденный посылкой маршрут или сообщение о том, в каком городе остановилась посылка. Программа должна быть закрыта для изменения, но открыта для расширения.

Пример работы программы:

```
Индексы городов:
1. Энск
2. Ивентск
3. Шаблонск
4. Диаграмск
5. Атрибутск
Укажите индекс города отправления
>> 1
Укажите индекс города назначения
>> 4
Пройденный маршрут: Энск->Ивентск->Шаблонск->Диаграмск
Посылка доставлена в пункт назначения.
```

```
Укажите индекс города отправления
>> 1
Укажите индекс города назначения
>> 5
Пройденный маршрут: Энск->Ивентск->Шаблонск
В настоящее авиамаршрут Шаблонск->Атрибутск временно не функционирует. Не волнуйтесь, посылка будет отправлена, как только снабжение восстановится.
```

12 Вариант. Паттерн «Стратегия»

Вы принимаете участие в разработке прикладного программного комплекса для обеспечения работы кофейного автомата. Кофе можно сделать по-разному, а разный кофе по-разному действует на живые организмы. В вашем распоряжении имеются следующие рецепты кофе:

– Кофе «Интроспекция» – улучшает догадливость при распознавании неизвестного;

– Кофе «Абстракция» – позволяет видеть важных для вас людей и не замечать остальных;

– Кофе «Инкапсуляция» – Вы теряете желание делиться ходом выполнения работы с остальными;

API ОС кофейного автомата при загрузке создает объект класса «Кофевар» автоматически и использует только его, в виду чего использовать паттерн «Фабрика» не представляется возможным. Объект класса «Кофевар» в методе «приготовитьПоРецепту()» принимает на вход объект класса «Рецепт», который обязательно должен иметь метод «варитьКофе()».

Необходимо разработать архитектуру данного фрагмента приложения и запрограммировать полученное решение. Подразумевается, что в ходе промышленного шпионажа за другими кофейными автоматами могут быть открыты новые рецепты. В виду этого программа должна быть закрыта для изменения, но открыта для расширения. На вход программе подается название кофе, на выходе программа должна напечатать свойство, которое обретет клиент в результате употребления кофе.

Пример работы программы:

```
Коды кофе:
1. Интроспекция
2. Абстракция
3. Инкапсуляция
Выберите кофе
>> 3
Предполагаемый эффект: вы теряете желание делиться ходом выполнения работы с остальными;
```

13 Вариант. Паттерн «Наблюдатель»

Вы принимаете участие в разработке программного комплекса системы пожарной сигнализации в качестве системного архитектора. В помещении установлен датчик дыма, в случае регистрации им количества дыма сверх предельно-допустимой концентрации, датчик должен отправить сигналы на пульт пожарной охраны, владельцам помещения, а также системе автоматического пожаротушения. Все три вышеперечисленных объекта реагируют на данное сообщение. Пожарная охрана высылает пожарный

расчёт, владельцы помещения отправляются к помещению, системе автоматического пожаротушения включает звуковое оповещение и начинает тушить пожар.

Так как разрабатываемая программа является прототипом и реального датчика задымления нет, то программа должна опрашивать пользователя о возникновении пожара с частотой раз в 3 секунды. В случае положительного ответа программа должна вывести действие, предпринятое каждым из вышеперечисленных объектов. Программа должна быть закрыта для изменения, но открыта для расширения.

Пример работы программы:

```
У нас пожар?
>>нет
У нас пожар?
>>нет
У нас пожар?
>>да
Пожарный расчет выехал на борьбу с огнем
Владельцы помещения выехали
Система автоматического пожаротушения оповестила жильцов и начала тушить пожар.
```

14 Вариант. Паттерн «Команда»

Вы принимаете участие в разработке программного комплекса устройства для помощи на экзаменах (Списывать плохо!). Устройство представляет собой микроконтроллер и множество кнопок, взаимодействующих с ним по беспроводному протоколу. Соответственно спрятать кнопку можно туда, где вам будет удобнее её нажимать. Некоторые кнопки устройства имеют фиксированную функцию, а остальные могут менять своё назначение в зависимости от пожеланий пользователя. Опустим реализацию функциональности, привязанной к конкретным кнопкам т.к. это тривиальная задача. Функциональность устройства представлена тремя функциями: вывести ответ на беспроводной дисплей, озвучить ответ в беспроводном наушнике и передать ответ азбукой Морзе через вибродинамик. Соответственно необходимо и три кнопки.

Программа должна в режиме диалога с пользователем назначать функциональность конкретной кнопке, а затем опрашивать пользователя какую кнопку он нажал и выводить описание произведенного действия, тем самым имитируя работу устройства. Программа должна быть закрыта для изменения, но открыта для расширения.

Пример работы программы:

```
Режим назначения кнопок.
Виды кнопок:
1. Кнопка в ботинке
2. Кнопка между пальцев руки
3. Кнопка, прикрепленная к зубу
Действия:
1. Вывести ответ на беспроводной дисплей
2. Озвучить ответ в беспроводном наушнике
3. Передать ответ азбукой Морзе через вибродинамик
Формат описания назначений ВидКнопки:Действие;ВидКнопки:Действие...
>>1:3;2:2;3:1;
```

```
Кнопка в ботинке позволяет вывести ответ на беспроводной дисплей.
Кнопка между пальцев руки позволяет озвучить ответ в беспроводном наушнике.
Кнопка, прикрепленная к зубу позволяет передать ответ азбукой Морзе через вибродинамик.
```

15 Вариант. Паттерн «Итератор»

Вы принимаете участие в разработке программного обеспечения для телевизоров. Предпочтения клиентов в вопросе выбора телевизоров – чем больше, тем лучше. Однако, когда дело доходит до пультов управления, предпочтения клиентов расходятся. В ходе исследований были выявлены две большие группы клиентов: любители пультов с кнопками и любитель старинных пультов с потенциометрами. Что те, что те умеют переключать каналы, но делают это по-разному.

Телевизоры с пультами с кнопками при настройке сканирует частоты и формируют список каналов. При нажатии кнопки на пульте с кнопками происходит моментальный переход к уже хранимой частоте следующего или предыдущего канала.

Телевизоры с пультами с потенциометрами при вращении человеком ручки потенциометра постепенно изменяют частоту. То есть сканирование частот происходит в режиме реального времени.

Необходимо написать прототип программы для системы управления телевизором. Программа принимает на вход тип телевизора (с пультом с кнопками или с пультом с потенциометром), а дальше в режиме диалога с пользователем происходит имитация переключения каналов. После каждого переключения программа должна выводить каким образом был переключен канал. Программа должна быть закрыта для изменения, но открыта для расширения.

Пример работы программы:

```
Выберите тип телевизора:
1. С пультом с кнопками
2. С пультом с потенциометром
>> 2
Выберите команду:
1. Следующий канал
2. Предыдущий
>> 1
Был найден следующий канал путем сканирования частот в режиме реального времени
Выберите тип телевизора:
1. С пультом с кнопками
2. С пультом с потенциометром
>> 1
Выберите команду:
1. Следующий канал
2. Предыдущий
>> 1
Был найден следующий канал путем выбора следующей частоты из сформированной заранее таблицы частот
```

16 Вариант. Паттерн «Состояние»

Вы принимаете участие в разработке программного обеспечения для мультиинструмента. Известно, что мультиинструмент может работать в одном из трех состояний: перфоратор, дрель и гайковерт. В зависимости от состояния мультиинструмент при нажатии на кнопку пуска выполняет разную работу. При активном состоянии «перфоратор» инструмент совершает как вращательные, так и ударные движения рабочим инструментом (буром). В состоянии «дрель» инструмент производит только вращательные движения рабочим инструментом (сверлом) на высоких оборотах. В состоянии «гайковерт» инструмент производит только вращательные движения рабочим инструментом (головкой) исключительно на низких оборотах.

Вам необходимо написать прототип программы, реализующей поведение мультиинструмента. На вход программа принимает номер состояния, которое должен принять инструмент, на выходе выводит какие движения производит инструмент. Программа должна быть закрыта для изменения, но открыта для расширения.

Пример работы программы:

```
Выберите режим работы мультиинструмента:
1. Перфоратор
2. Дрель
3. Гайковерт
>> 3
Мультиинструмент совершает только вращательные движения рабочим инструментом (головкой) исключительно на низких оборотах
>> 1
Мультиинструмент совершает как вращательные, так и ударные движения рабочим инструментом (буром).
```

Вопросы для подготовки к защите:

1. Что такое паттерн проектирования?
2. Классификация паттернов проектирования. Назначение каждого класса паттернов.
3. Что подразумевает под собой принцип DRY?
4. Что подразумевает под собой принцип KISS?
5. Что подразумевает под собой принцип YAGNI.
6. Перечислите и характеризуйте принципы SOLID.
7. Перечислите и характеризуйте шаблоны GRASP.
8. Что такое антипаттерны проектирования? Каково их назначение
9. Примеры антипаттернов проектирования.
10. Паттерн «одиночка». Польза и вред.
11. Что такое идиома, пример идиом.
12. Шаблоны архитектуры систем. Назначение. Примеры.