**­­­­Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное образовательное учреждение

высшего образования «Уральский федеральный университет

имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ

Центр ускоренного обучения

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6**

по дисциплине «Конструирование программного обеспечения»

**Тема:** **Паттерны проектирования**

Студенты гр. РИВ-400027у Р.А.Гиндуллин

А.П.Хромченков

Е.Н.Ботоногов

А.М.Власов

Преподаватель С.И.Тимошенко,

доц., к.т.н

**Екатеринбург 2023**

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146121450)

[2 Результаты проделанной работы 4](#_Toc146121451)

[2.1 Абстрактная фабрика 4](#_Toc146121452)

[2.2 Мост 6](#_Toc146121453)

[2.3 Команда 9](#_Toc146121454)

[3 Анализ полученных результатов 13](#_Toc146121455)

# 1 Постановка задачи

1. Изучить паттерны проектирования по книге Швец А. Погружение в паттерны проектирования / А. Швец // Сайт Sergey Drozdov. – [2022]. – Режим доступа: https://sd.blackball.lv/library/ Pogruzhenie\_v\_patterny\_proektirovanija (2021).pdf (дата обращения 01.07.2022). Книга есть также в папке “Книги\_и\_статьи\Паттерны (шаблоны) проектирования”.
2. Выбрать варианты паттернов для исследования, исходя из номера команды по таблице (вариант 2).
3. Ознакомиться с паттернами по указанной в п. 1 книге.
4. Загрузить в одну из IDE проект с паттернами на Java (лучше взять из папки “Швец А. Погружение в паттерны проектирования. 2021 - Java (проект в Eclipse)”).
5. Выполнить паттерны, убедившись в работоспособности кода и разобравшись в принципах его работы.
6. Выполнить обратное проектирование кода (reverse engineering), построив по коду диаграмму классов.
7. В отчет по лабораторной работе поместить полученные диаграммы классов для каждого паттерна и описать по ним принцип работы паттерна.

# 2 Результаты проделанной работы

Наша команда №2 взяла второй вариант паттернов проектирования из таблицы: абстрактная фабрика, мост, команда. Далее в лабораторной работе будут рассмотрены данные паттерны.

## 2.1 Абстрактная фабрика

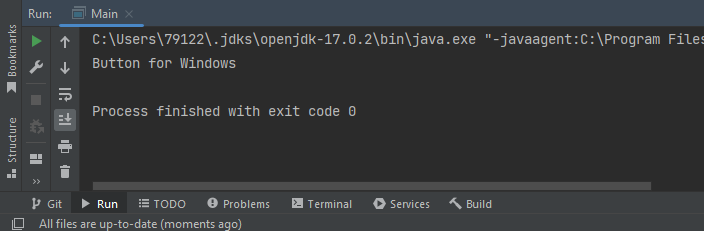
Абстрактная фабрика (Abstract Factory) — это порождающий паттерн, который позволяет создавать родственные объекты, не привязываясь к их конкретным классам. Он предоставляет интерфейс для создания семейства связанных или зависимых объектов, без указания их конкретных классов.

Абстрактная фабрика использует механизм наследования и позволяет создавать семейства объектов, которые связаны между собой. При этом все объекты создаются на основе интерфейсов, что является основным преимуществом этого паттерна перед другими методами создания объектов.

Паттерн Абстрактная фабрика решает следующие проблемы:

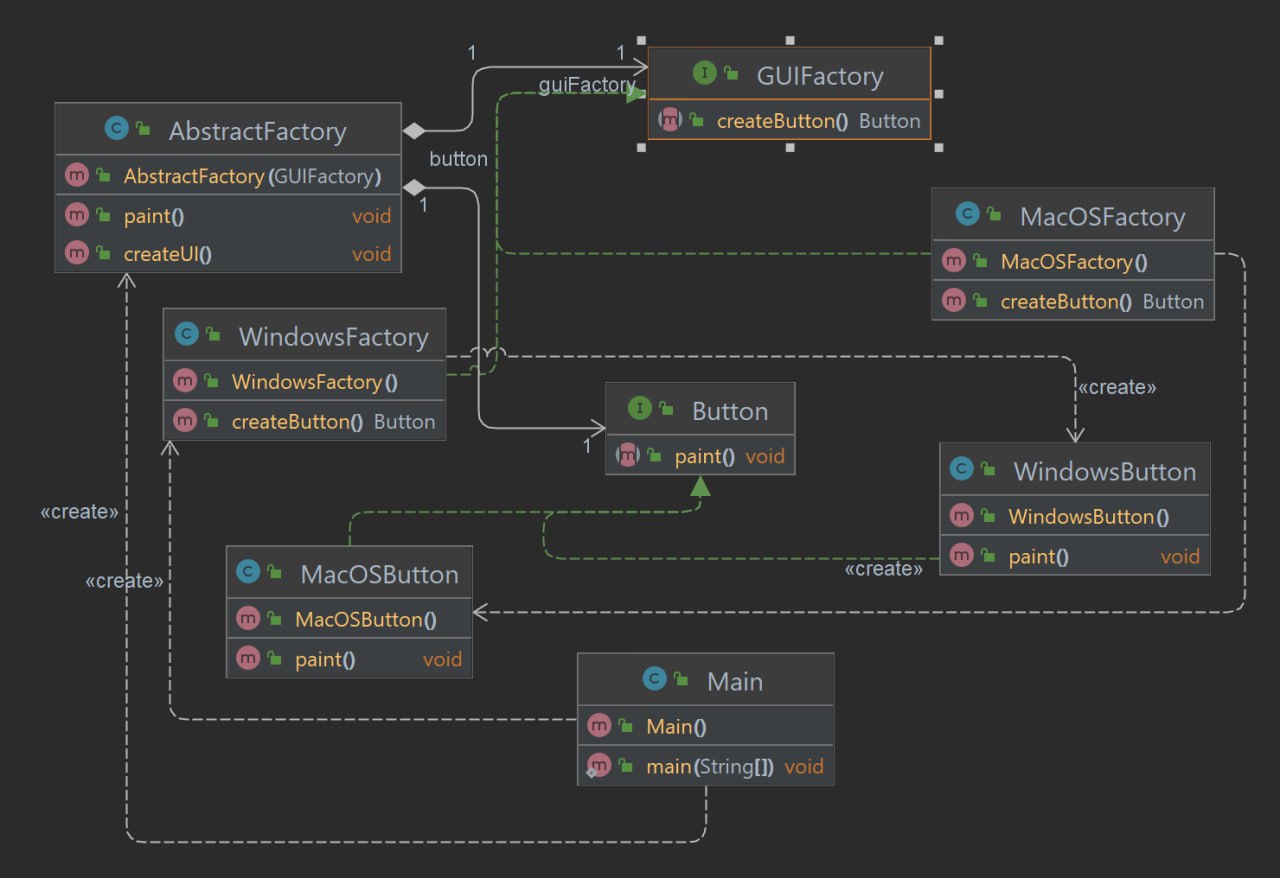
* Обеспечивает создание семейств взаимосвязанных объектов, не привязывая клиентов к конкретным классам объектов.
* Сокращает зависимости между объектами, разделяя их создание и использование.
* Облегчает добавление новых типов объектов в приложение, не приводя к изменению уже существующего кода в приложении.
* Способствует поддерживаемости и расширяемости приложения.
* Обеспечивает контроль над созданием объектов и определением возможности создать объекты новых семейств.

После запуска кода в intellij IDEA с реализованным кодом паттерна абстрактная фабрика в консоль выводится следующий результат, приведённый на рисунке 1:



1. Результат вывода в консоль

Выполним обратное проектирование кода для данного паттерна и разберёмся в принципах его работы. На рисунке 2 представлена диаграмма классов для паттерна асбтрактная фабрика.



1. Диаграмма классов паттерна Асбтрактная фабрика

Main - это класс, который содержит точку входа в программу и нужен для запуска приложения. main(String[] args) - это метод главного класса Main, который запускает приложение

MacOSButton и WindowsButton - это классы, которые реализуют интерфейс Button и определяют метод paint() для отображения кнопки в MacOS и Windows соответственно.

MacOSFactory и WindowsFactory - это классы, которые реализуют интерфейс GUIFactory и определяют метод createButton() для создания экземпляра MacOSButton и WindowsButton соответственно. В данном случае, WindowsFactory создает объекты WindowsButton, а MacOSFactory - MacOSButton.

GUIFactory - это интерфейс, который определяет метод createButton() для создания объекта, реализующего интерфейс Button

AbstractFactory - это класс, который использует фабрику для создания объектов. Он имеет два метода: createUI для создания пользовательского интерфейса и paint для отображения элементов интерфейса.

Button - это интерфейс, который определяет метод paint() для рисования кнопки

Связи между классами определяют зависимости между объектами и приложением в целом. В данном случае, Main зависит от AbstractFactory, а AbstractFactory зависит от интерфейсов GUIFactory и Button. Конкретные реализации фабрик и кнопок зависят от интерфейсов GUIFactory и Button соответственно. Для создания пользовательского интерфейса мы используем фабрику и создаем объекты кнопок через метод createButton(). В методе paint() мы используем полученный объект кнопки для отображения элементов интерфейса.

На диаграмме между классом Main и классами AbstractFactory и WindowsFactory связь «зависимость», между AbstractFactory и интерфейсами GUIFactory, Button связь «агрегация», между WindowsFactory и GUIFactory связь «реализация», между MacOSButton, WindowsButton и интерфейсом Button связь «реализация», между WindowsFactory и WindowsButton связь «зависимость».

## 2.2 Мост

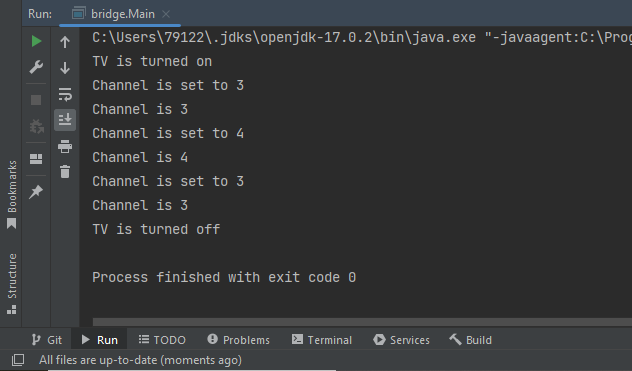
Мост (Bridge) - это структурный паттерн проектирования, который позволяет отделить абстракцию от ее реализации. Он используется в тех случаях, когда нужно обеспечить параллельную работу двух и более иерархий классов, каждая из которых может изменяться независимо друг от друга.

Паттерн Мост является композиционным и в его основе лежит разделение классов на две иерархии - абстракцию и реализацию. Абстракция - это класс, который определяет интерфейс и взаимодействие с пользователем, тогда как реализация - это класс, который реализует этот интерфейс.

Паттерн Мост решает следующие проблемы:

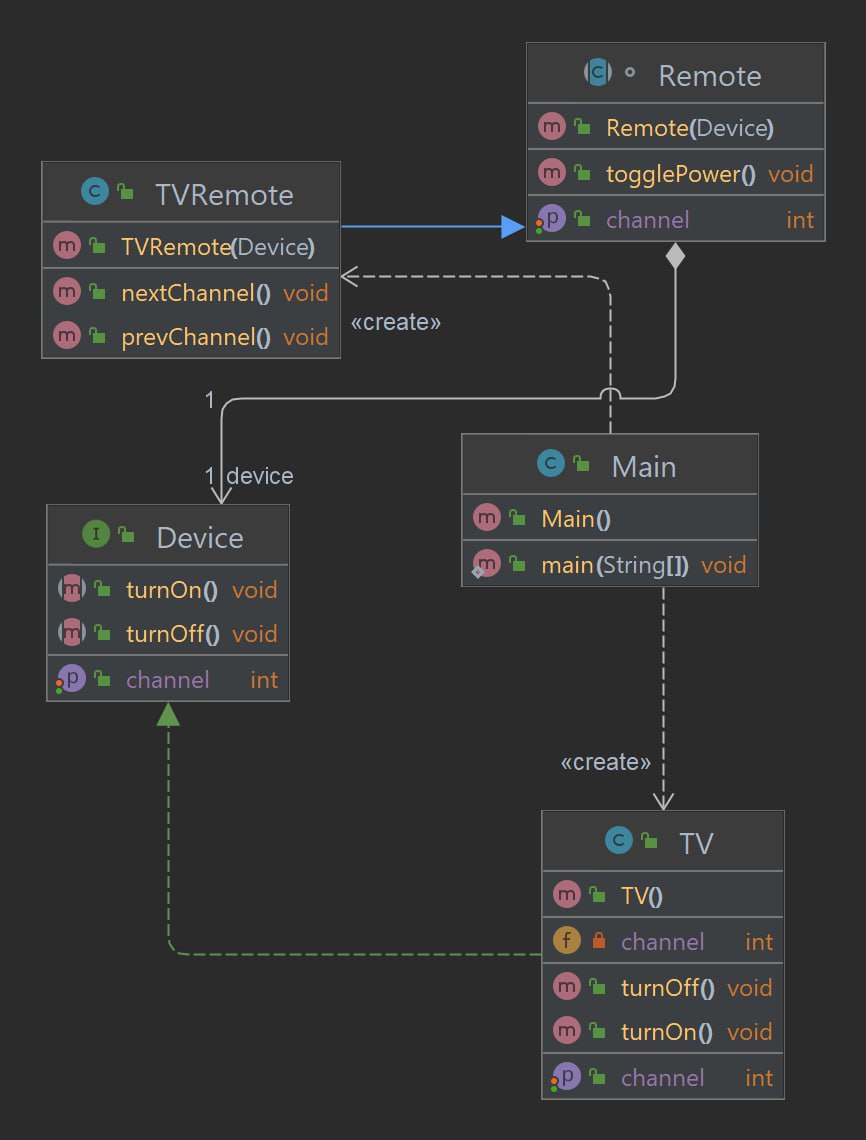
* Устранение жесткой связи между абстракцией и реализацией;
* Разделение классов на две независимые иерархии;
* Поддержка расширяемости и гибкости приложений;
* Увеличение повторного использования кода;
* Сокращение зависимостей в приложении.

После запуска кода в intellij IDEA с реализованным кодом паттерна Мост в консоль выводится следующий результат, приведённый на рисунке 3



1. Результат вывода в консоль

Выполним обратное проектирование кода и разберёмся в принципах его работы. На рисунке 4 представлена диаграмма классов для паттерна Мост.



1. Диаграмма классов паттерна Мост

Main - это класс, который содержит точку входа в программу и нужен для запуска приложения. main(String[] args) - это метод главного класса Main, который запускает приложение

Класс TV - реализует интерфейс Device и представляет собой конкретное устройство - телевизор. У этого класса есть методы turnOn(), turnOff(), setChannel() и getChannel().

Абстрактный класс Remote - служит для реализации абстракции пульта управления. Он содержит ссылку на устройство, которым он управляет, а также методы togglePower(), setChannel() и getChannel().

Класс TVRemote - является конкретной реализацией пульта управления для телевизора. Он наследует абстрактный класс Remote и содержит дополнительные методы nextChannel() и prevChannel().

Интерфейс Device – содержит в себе методы включения и выключения turnOn(), turnOff(), а также методы установки каналов setChannel() и getChannel().

Связь между классами осуществляется при помощи наследования. Класс TVRemote наследуется от абстрактного класса Remote, а класс Remote содержит ссылку на устройство (объект класса TV), которым он управляет. Также объект класса TVRemote передается в качестве параметра в конструктор класса TVRemote при его создании.

Класс TV содержит методы для работы с телевизором: turnOn, turnOff, setChannel, getChannel. Метод turnOn() включает телевизор и выдаёт сообщение на экран "TV is turned on". Метод turnOff() выключает телевизор и выводит сообщение на экран "TV is turned off". Метод setChannel() устанавливает канал на заданный номер, например, "Channel is set to 3". Метод getChannel() выводит на экран номер текущего канала.

Класс Remote содержит методы togglePower(), setChannel() и getChannel(). Метод togglePower() отвечает за включение и выключение телевизора. Если телевизор включен, то метод выключает его. Если телевизор выключен, то метод включает его. Метод setChannel() устанавливает текущий канал для телевизора. Метод getChannel() выводит номер текущего канала на экран.

Класс TVRemote наследует методы класса Remote и добавляет к ним методы nextChannel() и prevChannel() для выставления следующего и предыдущего канала на телевизоре.

На диаграмме указаны следующие виды связей между классами:

Между Main и TV «зависимость»

Между TV и Device «реализация»

Между Main и TVRemote «зависимость»

Между Remote и Device «агрегация»

## 2.3 Команда

Паттерн Команда (Command) – это поведенческий паттерн проектирования, который позволяет инкапсулировать запрос на выполнение определенного действия в виде отдельного объекта. Этот объект может быть запущен в любое время, даже если само действие неизвестно: оно будет определено в зависимости от того, какая команда была выбрана и передана объекту.

Основная задача паттерна Команда заключается в том, чтобы разделить запрос на выполнение некоторого действия и его выполнение. Это позволяет повторно использовать команду или создавать новые команды ориентированными на изменение поведения программы.

Паттерн Команда позволяет:

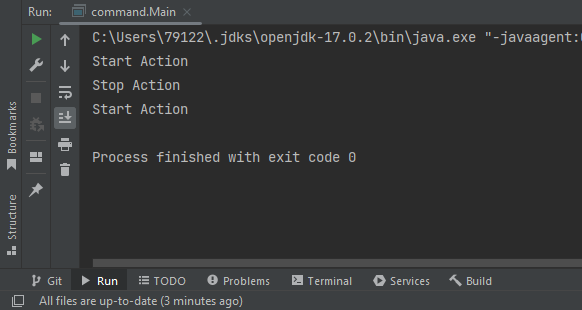
- Убрать прямую зависимость между объектом, который вызывает операцию, и объектом, который знает, как её выполнить.

- Создавать отложенные вызовы функций.

- Комбинировать несколько команд в одну.

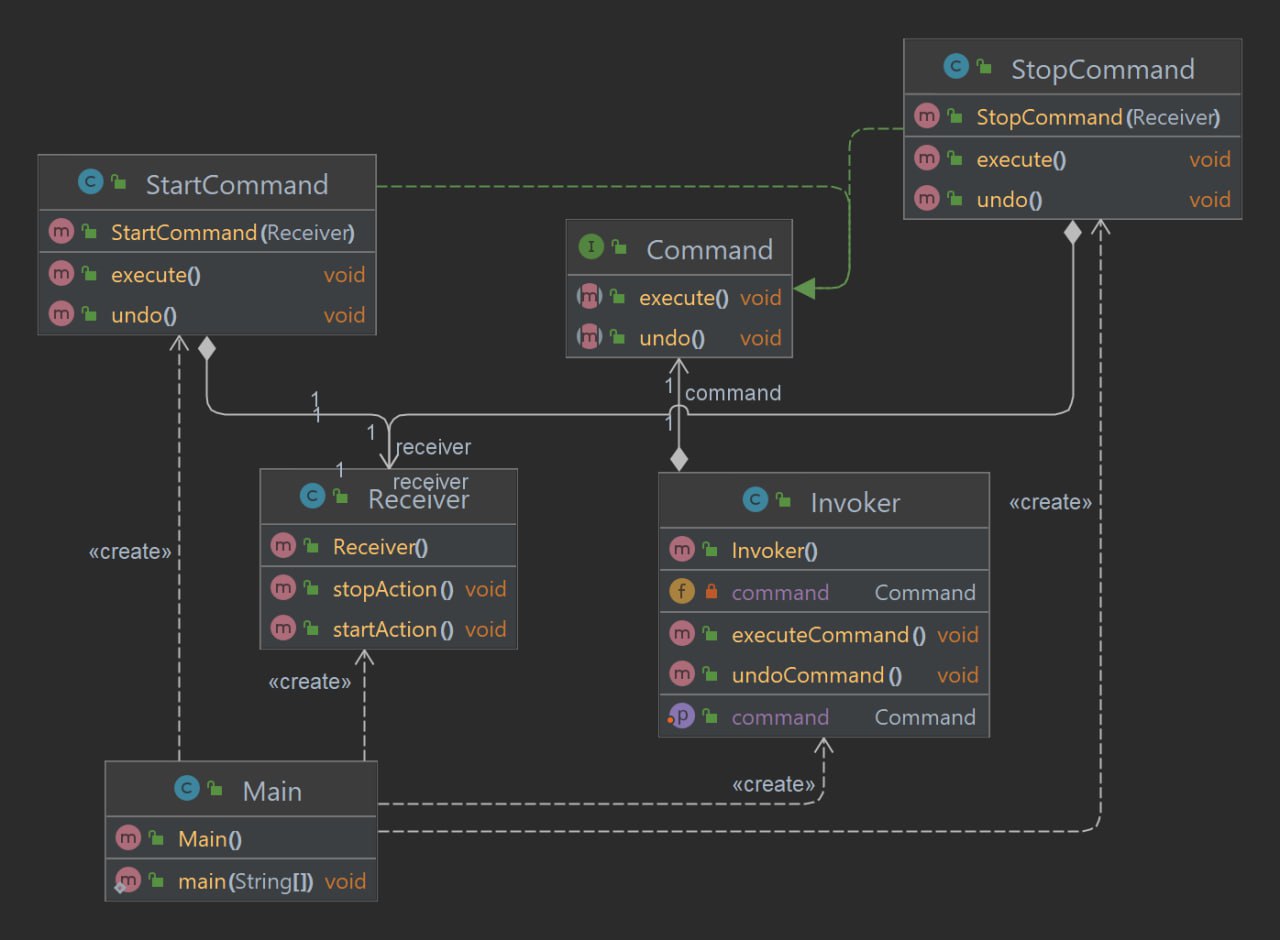
- Предоставлять простой интерфейс для добавления новых команд.

После запуска кода в intellij IDEA с реализованным кодом паттерна Комманда в консоль выводится следующий результат, приведённый ниже на рисунке 5.



1. Результат вывода в консоль

Выполним обратное проектирование кода и разберёмся в принципах его работы. На рисунке 6 представлена диаграмма классов для паттерна Команда.



1. Диаграмма классов паттерна Команда

Класс Main - содержит функцию main, используется для создания объекта Invoker и нескольких команд StartCommand и StopCommand

Интерфейс Command - содержит два метода - execute() и undo()

Класс Invoker, который служит для вызова команды и, если это необходимо, обратного вызова команды

Класс Receiver, который содержит методы startAction() и stopAction(), которые вызываются в зависимости от команды.

Классы StartCommand и StopCommand представляют конкретные команды.

При выполнении программы создается объект Invoker и две команды StartCommand и StopCommand, которые инициализируются объектом Receiver, соответственно. Команды связаны с Invoker, и при необходимости могут исполняться.

Класс Main используется для выполнения команд. В данном примере выполняются две команды, startCommand и stopCommand, и после каждой команды вызывается функция undo(), которая отменяет выполнения последней выполненной команды (путем вызова метода undo() соответствующей команды).

Класс Invoker использует экземпляр класса Command для вызова методов из Receiver. Класс Invoker выступает в роли отделения кода, который инициирует выполнение действия от кода, который знает, как обрабатывать этот запрос.

Класс Receiver содержит методы, которые могут быть вызваны командами, и определяют действия, которые должны быть выполнены.

Классы StartCommand и StopCommand реализуют интерфейс Command. Оба класса инкапсулируют для вызова определенную функцию в объекте Receiver.

Связи между классами осуществляются при помощи агрегации объектов Invoker, Command, Receiver. Объект Invoker содержит объект Command, который в свою очередь ссылается на объект Receiver.

На диаграмме указаны следующие виды связей между классами:

Между Main и StartCommand «зависимость»

Между Main и StopCommand «зависимость»

Между Main и Invoker «зависимость»

Между Main и Receiver «зависимость»

Между StartCommand и Receiver «агрегация»

Между StopCommand и Receiver «агрегация»

Между Command и Invoker «агрегация»

Между StartCommand, StopCommand и Command «реализация»

# 3 Анализ полученных результатов

В результате выполнения лабораторной работы № 6 были описаны следующие паттерны проектирования:абстрактная фабрика, мост, команда. Данные паттерны были выполнены на Java в Intellij idea, было выполнено обратное проектирование кода и построены диаграммы классов.