**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики» Рефакторинг баз данных и приложений**

Лабораторная работа №4

Выполнил: Студент группы

БПИ2301

Быков Максим Эдуардович

Москва

2025

# Цель:

1. Ознакомиться с основными принципами и задачами код-ревью.
2. Научиться проводить перекрестное код-ревью, выявлять проблемы в коде и предлагать улучшения.
3. Применить полученные рекомендации для рефакторинга кода.
4. Развить навыки командной работы и взаимодействия в процессе разработки.

# Задачи:

1. Ознакомьтесь с теоретическими основами код-ревью, его целями и преимуществами. Изучите лучшие практики проведения код-ревью, такие как фокус на читаемости, тестируемости и соблюдении стандартов кодирования.
2. Разделитесь на команды по 2 человека. Обменяйтесь кодом проектов

(можно использовать проект с учебной практики).

1. Проведите анализ кода коллеги, обращая внимание на читаемость и понятность кода, наличие дублирующегося или избыточного кода, соответствие стандартам кодирования, возможности для улучшения структуры или производительности, составьте список

рекомендаций для улучшения кода.

1. Получив рекомендации от коллеги, внесите изменения в свой код, применяя техники рефакторинга (например, устранение дублирования, улучшение именования, разделение

больших функций). Убедитесь, что после рефакторинга программа сохраняет свою функциональность.

1. Подготовьте отчет, включающий описание исходного состояния кода, рекомендации, полученные в процессе код-ревью, внесенные изменения и их обоснование, итоговое состояние кода.
2. Обсудите результаты код-ревью с коллегой. Укажите, какие рекомендации были полезны, а какие могли быть улучшены.

До рефакторинга:

# Ход работы

1. **Анализ исходного кода**

* Весь код находился в одном исходном классе mainSpeed, который содержал всю логику кода, что делало его перегруженным.
* Использовались static переменные GUI напрямую из внутреннего класса, что нарушало инкапсуляцию и делало код уязвимым к ошибкам при масштабировании.
* Жесткая связь между логикой и UI – вся логика была внутри одного внутреннего класса. Не было абстракции параметров, возможности повторно использовать/тестировать логику отдельно от интерфейса.
* Большое количество дублирующих фреймов.

# Рекомендации по итогам код-ревью

1. Ввести модульную структуру. Сделать несколько классов, каждый из которых будет отвечать за свою логику
2. Cделать упрощённую анимацию передачи параметров, чтобы не было зависимости от UI.
3. Использовать Enums вместо строк, что устранит ошибки, связанные с опечатками в строках, а так же облегчит читаемость автодополнение и переключение логики
4. Всю логику UI вынести в отдельный класс, чтобы вся логика взаимодействия с пользователем находилась в одном классе
5. Отделение View от Model. Сделать чистую архитектуру

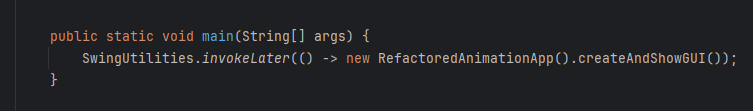
# Выполненные изменения

В ходе рефакторинга были выполнены **следующие ключевые улучшения**:

# 3.1. Введение модульной структуры.

Единственный класс mainSpeed был разделен на несколько модулей, у каждого из которых было своё назначение

* Main – отвечает за запуск программы



* AnimationPanel - Отвечает за отрисовку и анимацию

Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

* ControlPanel - Панель с кнопками и выпадающими меню

Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

* AnimationSettings - Объект с настройками: форма, скорость, направление.
* ShapeType (enum), Direction (enum) - Возможные формы, направления движения

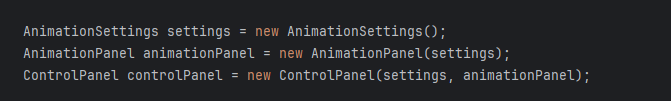
Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

# . Упрощённая анимация передачи параметров.

# В старом коде MyPanel напрямую обращался к static JComboBox, JRadioButton и т.п., что плохо тестируется, делает код негибким, затрудняет масштабирование

# С AnimationSettings всё централизовано: один объект хранит всё, легко подставить другой набор значенийю, легко тестировать (можно создать объект settings и передать в AnimationPanel)



# Использование Enums вместо строк.

# Использование enums(набор логически связанных констант) устраняет ошибки, связанные с опечатками в строках и облегчает читаемость, автодополнение и переключение логики

# Изображение выглядит как текст, снимок экрана, Шрифт Контент, сгенерированный ИИ, может содержать ошибки.

# Вынести всю логику в отдельный класс ControlPanel.

# Создание и обработка кнопок Start, Stop, Show/Hide

# Создание выпадающих списков: форма, направление, скорость

# При изменении пользовательского выбора — изменения сразу сохраняются в AnimationSettings

# 

# Отделение View от Model.

# AnimationSettings — это Model, который не содержит логики отображения, не зависит от GUI (Swing), просто хранит значения (модель состояний анимации)

# 

# AnimationPanel — это View. Всё что нужно для его работы — данные из AnimationSettings. Если settings изменится, AnimationPanel просто его перечитает.

# Это и есть View, зависящий только от Model, а не от интерфейса

# Изображение выглядит как текст, снимок экрана, Шрифт Контент, сгенерированный ИИ, может содержать ошибки.

# ControlPanel — это "посредник" (Controller/часть View)

# Пользователь выбирает значение, ControlPanel изменяет только settings, он не трогает AnimationPanel напрямую

# Таким образом, ControlPanel — это пользовательский интерфейс, а не часть логики отрисовки.

# ССЫЛКА НА РЕПОЗИТОРИЙ

https://github.com/A1e0nd/Refactoring4

# ВЫВОД

# Результат рефакторинга:

# Повышена читаемость

# Упростилось добавление новых параметров (новые формы, режимы движения и т.д.)

# Архитектура стала гибкой и масштабируемой

# Код стал соответствовать принципам SOLID, в частности:

# S — Single Responsibility (каждый класс делает одно)

# O — Open/Closed (новая логика добавляется без изменения старого кода)

Мы ознакомились с основными принципами и задачами код-ревью. Научились проводить перекрестное код-ревью, выявлять проблемы в коде и предлагать улучшения. Применять полученные рекомендации для рефакторинга кода. Развивать навыки командной работы и взаимодействия в процессе разработки.