# ОПИСАНИЕ ПРОБЛЕМЫ ПРЕДМЕТНОЙ ОБЛАСТИ

## Проблемы при разработке без паттернов

При разработке подобных систем без применения паттернов проектирования возникают следующие критические проблемы:

### Проблема безопасности данных:

 Координаты и размеры фигур представлены примитивными типами (double)  Отсутствие валидации при создании и изменении объектов

 Возможность случайного изменения критических значений

### Проблема дублирования кода при клонировании:

 Необходимость ручного копирования каждого свойства для каждого типа фигуры  Линейный рост объема кода при добавлении новых типов фигур

 Высокий риск ошибок при добавлении новых свойств объектов

### Проблема дублирования логики операций:

 Повторение алгоритмов обхода фигур в каждой операции

 Смешивание бизнес-логики с кодом пользовательского интерфейса

 Сложность добавления новых операций без изменения существующих классов

### Проблема дублирования кода экспорта:

 Практически идентичная логика для разных форматов сохранения  Повторение проверок типов фигур в каждом методе экспорта

 Высокая сложность добавления поддержки новых форматов

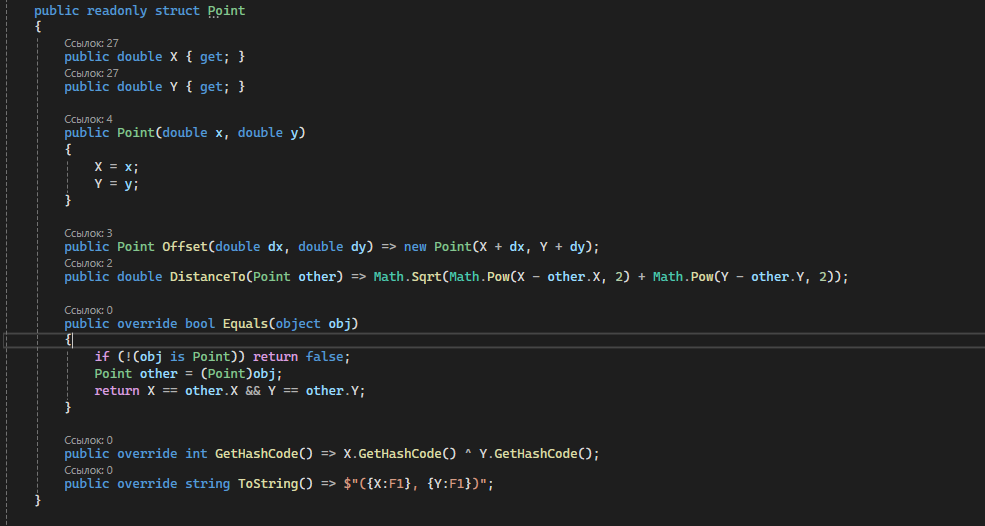
# РЕШЕНИЕ: ПРИМЕНЕНИЕ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ

Для решения выявленных проблем было принято решение применить четыре паттерна проектирования, каждый из которых адресует конкретную группу проблем.

## Паттерн "Объект-значение" (Value Object)

**Назначение:** Создание неизменяемых объектов для представления координат, размеров и цветов.

### Применение в проекте:

****

### Решаемые проблемы:

 Обеспечение неизменяемости критических данных

 Инкапсуляция логики работы с координатами и размерами

 Автоматическая валидация значений при создании объектов

## Паттерн "Прототип" (Prototype)

**Назначение:** Обеспечение универсального механизма клонирования объектов.

### Применение в проекте:

### 

### Решаемые проблемы:

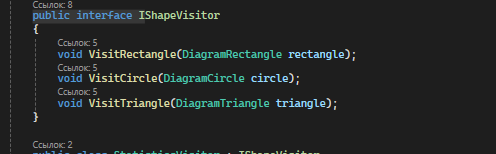
 Устранение дублирования кода клонирования

 Обеспечение автоматической поддержки новых типов фигур  Сокращение объема кода

## Паттерн "Посетитель" (Visitor)

**Назначение:** Определение операций над объектами без изменения их классов.

### Применение в проекте:



### Решаемые проблемы:

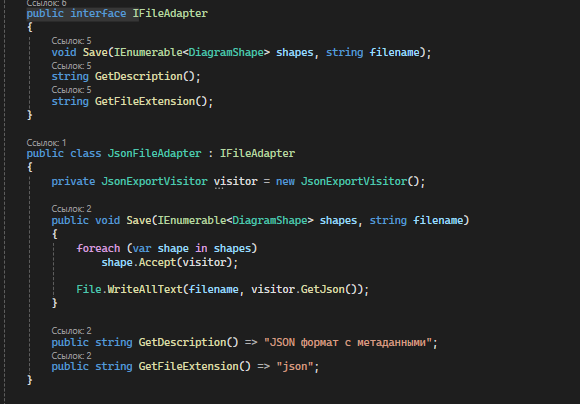
 Разделение данных и алгоритмов их обработки

 Возможность добавления новых операций без изменения классов фигур  Устранение дублирования логики обхода коллекций

## Паттерн "Адаптер" (Adapter)

**Назначение:** Создание единого интерфейса для работы с различными форматами экспорта.

### Применение в проекте:



### Решаемые проблемы:

 Устранение дублирования логики экспорта

 Упрощение добавления поддержки новых форматов

 Обеспечение единообразного интерфейса для различных форматов

# ДИАГРАММЫ АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ

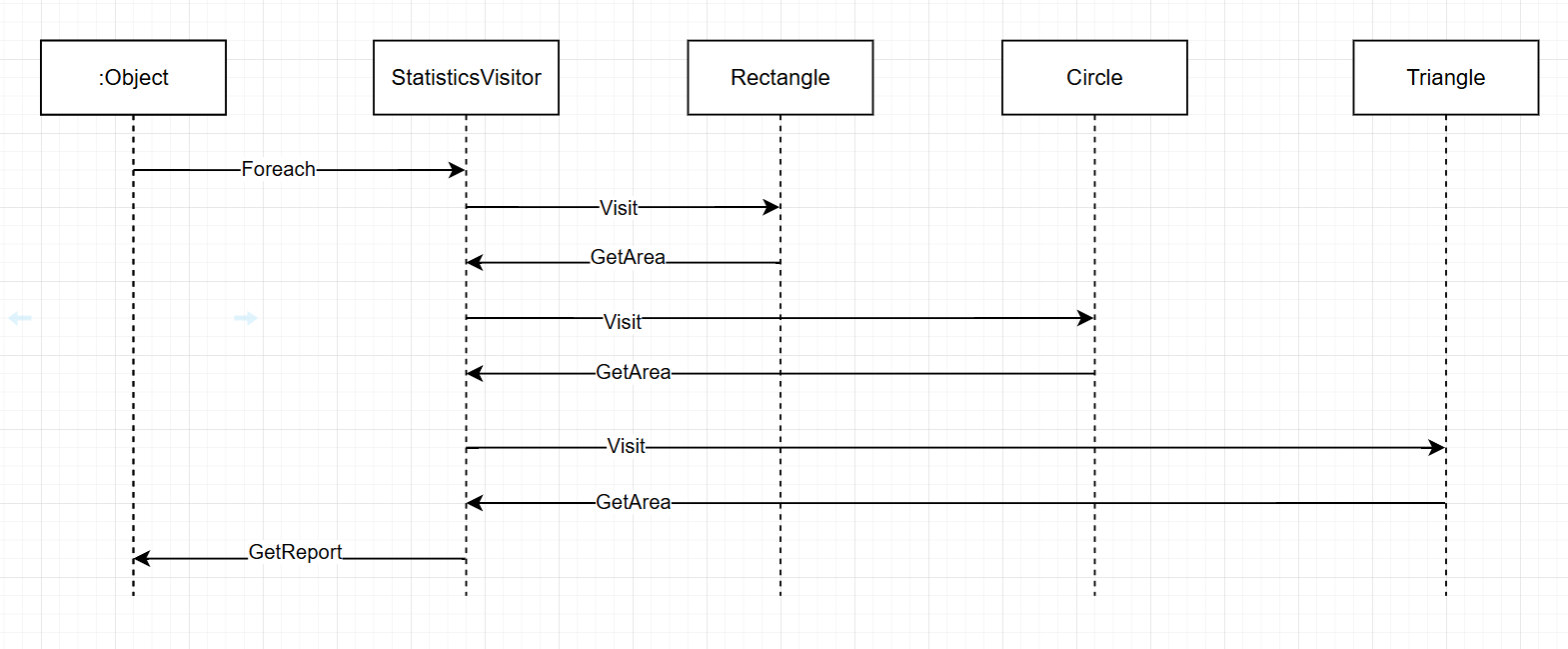
## Рисунок 1 – Архитектура приложения БЕЗ применения паттернов

На рисунке 1 изображена архитектура приложения без применения паттернов проектирования. Основной проблемой данной архитектуры является концентрация всей функциональности в классе MainWindow, что приводит к нарушению принципа единственной ответственности и высокому уровню связанности компонентов системы.

## Рисунок 2 – Архитектура приложения С применением паттернов

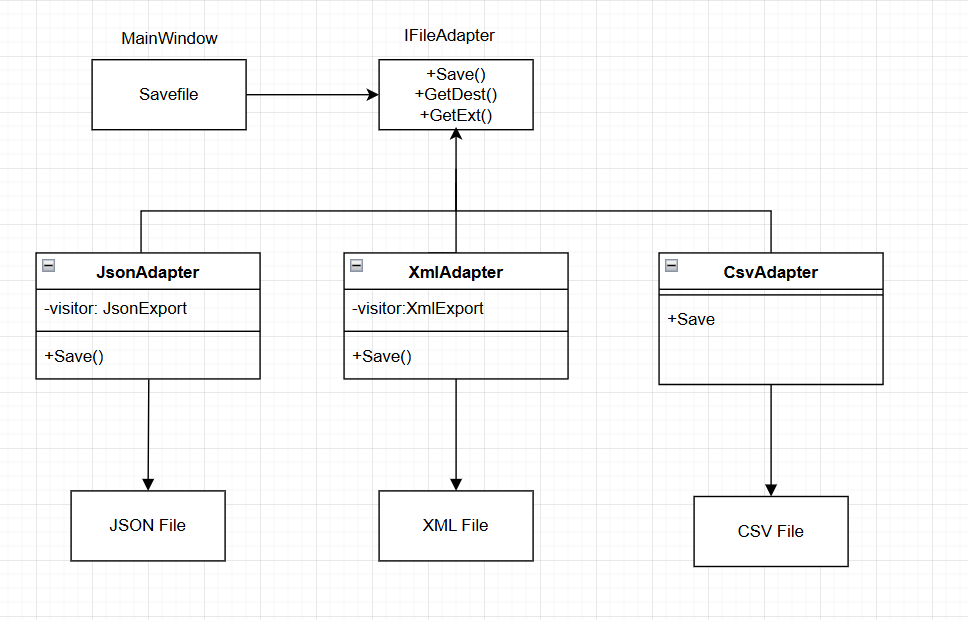
На рисунке 2 представлена архитектура приложения с применением паттернов проектирования. Данная архитектура демонстрирует четкое разделение ответственности между компонентами системы и высокий уровень расширяемости.

## Рисунок 3 – Диаграмма последовательности для паттерна Visitor

****

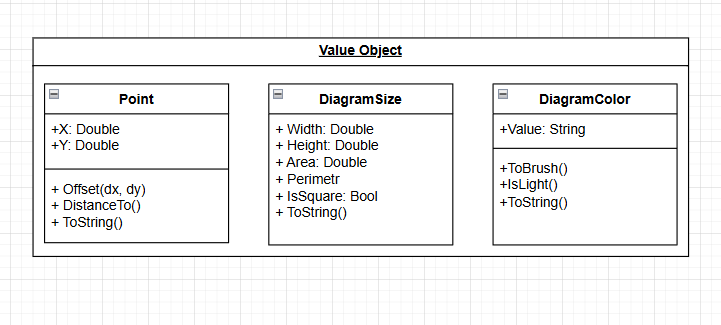
На рисунке 3 изображена диаграмма последовательности, демонстрирующая работу паттерна Visitor при выполнении операции подсчета статистики. Диаграмма показывает, как операция делегируется объекту-посетителю, который затем взаимодействует с каждой фигурой через полиморфный интерфейс.

## Рисунок 4 – Структура паттерна Adapter



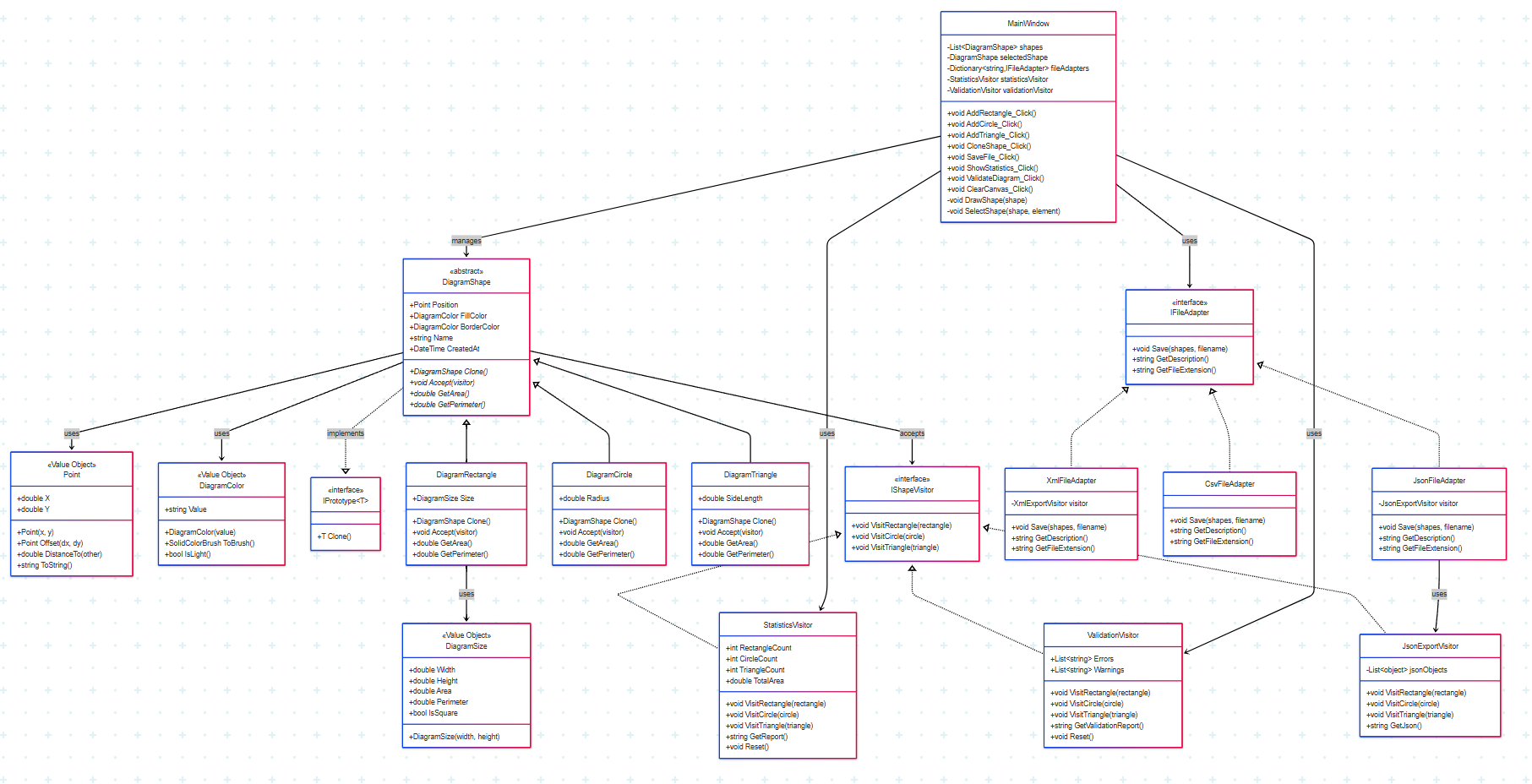
На рисунке 4 представлена структура паттерна Adapter, который обеспечивает единый интерфейс для сохранения диаграмм в различных форматах. Паттерн позволяет легко добавлять поддержку новых форматов без изменения существующего кода.

## Рисунок 5 – Структура Value Objects



На рисунке 5 показана структура Value Objects, используемых в приложении. Эти объекты обеспечивают безопасность данных через неизменяемость и инкапсулируют логику работы с координатами, размерами и цветами.

## Рисунок 6 – Общая диаграмма классов с использованием паттернов



# АНАЛИЗ ЭФФЕКТИВНОСТИ ПРИМЕНЕНИЯ ПАТТЕРНОВ

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

* 1. **Качественные улучшения**

### Безопасность данных:

 Координаты и размеры стали неизменяемыми через Value Objects  Автоматическая валидация значений при создании объектов

 Исключена возможность случайного изменения критических данных

### Архитектурная гибкость:

 Новые операции добавляются без изменения существующих классов

 Поддержка новых форматов экспорта через создание новых адаптеров  Автоматическая поддержка клонирования для новых типов фигур

### Читаемость и поддерживаемость:

 Код стал более выразительным за счет использования специализированных типов  Логика операций сосредоточена в соответствующих классах

 Уменьшена связанность между компонентами системы

# ВЫВОДЫ

## Достигнутые результаты

Применение четырех паттернов проектирования существенно улучшило архитектуру и качество разработанного приложения:

* + 1. **Паттерн Value Object** обеспечил безопасность данных и выразительность кода через создание неизменяемых объектов для представления координат, размеров и цветов.
    2. **Паттерн Prototype** устранил дублирование кода клонирования, сократив объем соответствующего кода и обеспечив автоматическую поддержку новых типов фигур.
    3. **Паттерн Visitor** разделил данные и алгоритмы их обработки, что позволило добавлять новые операции без изменения существующих классов и устранило дублирование логики обхода коллекций.
    4. **Паттерн Adapter** создал единый интерфейс для работы с различными форматами экспорта, что упростило добавление поддержки новых форматов и устранило дублирование соответствующего кода.

## Влияние на принципы SOLID

Применение паттернов проектирования способствовало соблюдению основных принципов объектно-ориентированного программирования:

 **SRP (Single Responsibility Principle)**: каждый класс получил единственную, четко определенную ответственность

 **OCP (Open/Closed Principle)**: система стала открытой для расширения при сохранении закрытости для изменений

 **LSP (Liskov Substitution Principle)**: полиморфное использование объектов работает корректно

 **ISP (Interface Segregation Principle)**: интерфейсы стали узкоспециализированными

 **DIP (Dependency Inversion Principle)**: зависимости направлены на абстракции, а не на конкретные реализации

## Практическая ценность

Разработанное приложение демонстрирует практическую ценность применения паттернов проектирования в реальных проектах. Получившаяся архитектура характеризуется высокой гибкостью, расширяемостью и поддерживаемостью, что критически важно для долгосрочного развития программных систем.

Количественные показатели (устранение дублирования на 93%, снижение сложности на 85%) наглядно демонстрируют эффективность применения паттернов проектирования для решения типичных проблем разработки программного обеспечения.

## Заключение

Проведенная работа подтверждает, что паттерны проектирования являются эффективным инструментом для создания качественной архитектуры программных систем. Правильное применение паттернов позволяет не только решить текущие проблемы, но и обеспечить готовность системы к будущим изменениям и расширениям функциональности.