**Министерство образования и науки Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)**

Кафедра компьютерных систем в управлении и проектировании (КСУП)

ПРИНЦИПЫ SOLID

Лабораторная работа по предмету «Вычислительные системы»

|  |  |
| --- | --- |
|  | Студент группы 586-M1  Орлова К.О.  « » 2018  канд. техн. наук каф. КСУП  \_\_\_\_\_\_\_\_\_\_\_\_\_ Калентьев А. А.  « » 2018 |

Томск 2018

Содержание

[1 Введение 3](#_Toc514171864)

[2 Основная часть 4](#_Toc514171865)

[2.1 Принцип единственной ответственности(SRP) 4](#_Toc514171866)

[2.2 Принцип открытости/закрытости(OCP) 5](#_Toc514171867)

[2.3 Принцип подстановки Барбары Лисков(LSP) 6](#_Toc514171868)

[2.4 Принцип разделения интерфейсов (ISP) 7](#_Toc514171869)

[2.5 Принцип инверсии зависимостей (PDI) 8](#_Toc514171870)

[3 Заключение 9](#_Toc514171871)

[Список источников 10](#_Toc514171872)

# 1 Введение

Любая полезная программа постепенно изменяется, в нее вносятся новые возможности. Очевидно, что с кодом что-то не так, если для небольшого изменения приходится переписывать значительную часть системы. Под чистым кодомпонимают код, который обеспечивает удобство сопровождения и безболезненность внесения изменений.

Чистый код должен быть не только оформлен определенным образом, но и иметь определенную структуру. Существует множество принципов, которым должен соответствовать хороший программный проект, а также ряд техник, позволяющих привести его в соответствие с этими принципами.

В отчете описаны пять основных принципов **SOLID**, которым должен соответствовать хороший объектно-ориентированный проект.

# 2 Основная часть

Аббревиатура SOLID была предложена Робертом Мартином и обозначает пять основных принципов объектно-ориентированного проектирования описывающих архитектуру программного обеспечения.

Аббревиатура пяти основных принципов дизайна классов в объектно-ориентированном проектировании — это Single responsibility, Open-closed, Liskov substitution, Interface segregation и Dependency inversion.

Таким образом, мы имеем 5 принципов, которые и рассмотрим ниже:

* Принцип единственной ответственности (Single responsibility);
* Принцип открытости/закрытости (Open-closed);
* Принцип подстановки Барбары Лисков (Liskov substitution);
* Принцип разделения интерфейса (Interface segregation);
* Принцип инверсии зависимостей (Dependency Invertion).

## 2.1 Принцип единственной ответственности(SRP)

Принцип единой ответственности заключается в следующем: На каждый объект должна быть возложена одна единственная обязанность.

Этот принцип означает, что каждый класс или подобная структура в коде должна отвечать только за одну цель. Все члены этого класса должны быть связаны одной целью. Класс должен быть реализован таким образом, чтобы при изменении одного из членов не нужно было изменять весь инструментарий. Это не означает, что классы должны содержать только один метод или свойство. Может быть много членов, если они относятся к единой ответственности[1].

Принцип единой ответственности дает хороший способ определения классов на этапе проектирования приложения и заставляет думать сразу обо всех способах изменения класса. Хорошее разделение обязанностей выполняется только тогда, когда имеется полная картина того, как приложение должно работать.

В моём приложении данный принцип нарушается в классе CutFigure. Класс управляет не только вырезание фигуры, но и вставкой фигуры после вырезания.

Для соблюдения принципа единственной ответственности необходимо реализацию вставки фигуры поручить новому классу.

На рисунке 2.1 и 2.3 представлена диаграмма классов до и после внесения изменений.

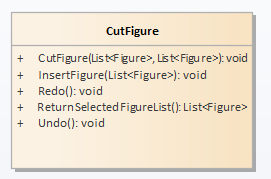


Рисунок 2.1 – Диаграмма класса CutFigure

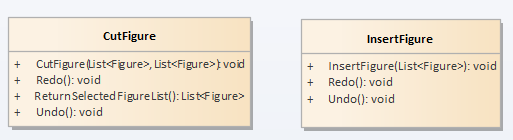


Рисунок 2.2 – Диаграмма классов CutFigure и InsertFigure

## 2.2 Принцип открытости/закрытости(OCP)

Данный принцип гласит — Программные сущности должны быть открыты для расширения, но закрыты для изменения. Это означает, что класс должен быть открыт для расширений, но закрыт от модификаций. То есть, модуль должен быть разработан так, чтобы новая функциональность могла быть добавлена только при создании новых требований. «Закрыт для модификации» означает, что класс уже разработан, и прошел модульное тестирование. Класс нельзя менять, пока не найдутся ошибки. Класс должен быть открытым только для расширений и в C# можно использовать для этого наследование.

В моём приложении есть класс Selection. В методе этого класса есть условие, в котором представлен выбор между двумя действиями: выделение фигуры мышкой и выделение областью (SelectionPoint и SelectionRectangle). Проблема в этом классе заключается в том, что если необходимо будет внести новый вид выделения (выделение эллипсом иди другой областью), понадобится добавить новое условие. Но согласно принципу OCP, класс должен быть закрыт от модификаций и открыт для расширений. Чтобы этого добиться, нужно выделить два класса PointSelection и RectangleSelection и создать интерфейс ISelection.

Теперь, если необходимо будет добавить новое выделение областью, достаточно просто создать новый класс и унаследовать его от ISelection. Таким образом, класс ISelection закрыт от модификаций, но доступен для расширений.

Ниже представлена диаграмма классов после внесения изменений.

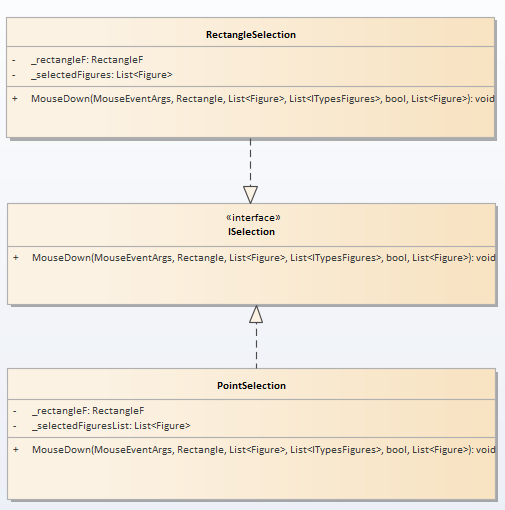


Рисунок 2.3 – Диаграмма классов PointSelection, RectangleSelection и ISelection

## 2.3 Принцип подстановки Барбары Лисков(LSP)

Этот принцип является вариацией принципа открытости/закрытости, суть принципа заключается в следующем: Объекты в программе могут быть заменены их наследниками без изменения свойств программы.

Это означает, что класс, разработанный на основании базового класса путем расширения, должен работать в приложении без сбоев. То есть, если разработчик расширяет класс и использует его в приложении, он не должен нарушать работу приложения или создавать фатальные ошибки для всего приложения.

Этого легко добиться, если помнить одно простое правило: Если базовый класс делает строго одно дело, разработчик получит при использовании класса только одну проблему. Это может привести к некоторым ошибкам в одной области, но все приложение не будет работать неправильно[2].

В моей программе подклассы переопределяют методы родительского класса так, что не нарушается основная функциональность. Данный принцип можно рассмотреть на примере выделения фигуры прямоугольной областью. Родительский класс может быть заменен наследником без нарушения логики работы программы. На рисунке 2.4 представлена диаграмма классов после изменений.

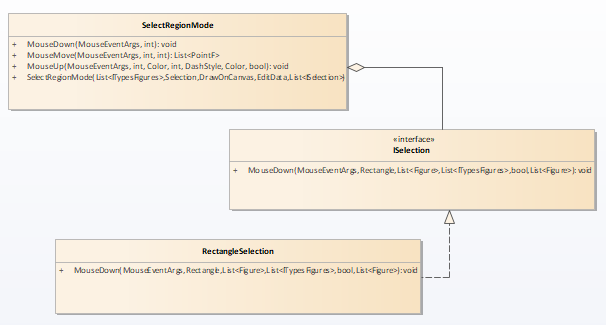


Рисунок 2.4 – Диаграмма классов RectangleSelection, SelectRegionMode и ISelection

## 2.4 Принцип разделения интерфейсов (ISP)

Принцип представляет собой следующее: Клиенты не должны быть вынуждены реализовывать ненужные методы, которые они не будут использовать.

Принцип разделения интерфейсов говорит о том, что слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические, чтобы клиенты маленьких интерфейсов знали только о методах, которые необходимы им в работе. В итоге, при изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют[3].

Например, у нас в приложении есть интерфейс ITypesFigures(рисунок 2.5), в него добавлены методы, отвечающие за изменение масштаба фигуры. Но эти методы нужны не всем классам данного интерфейса, следовательно, класс будет вынужден реализовывать ненужный метод, и здесь нарушен принцип разделения интерфейсов.

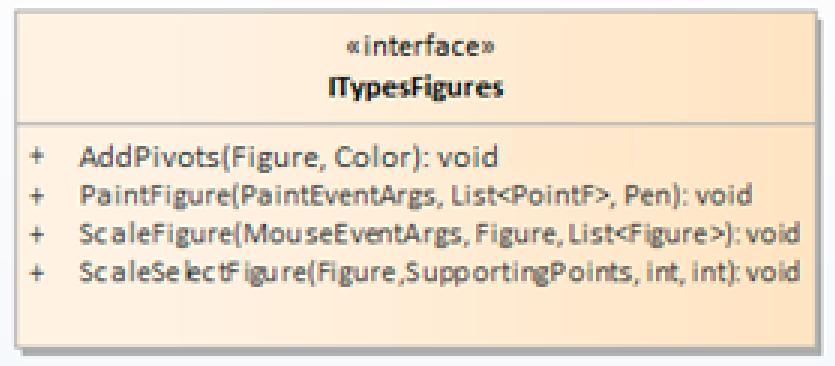


Рисунок 2.5 – Диаграмма интерфейса ITypesFigures

Для соблюдения принципа необходимо выделить два интерфейса. Один из которых будет отвечать за изменение размеров фигуры, а другой – за отрисовку. Диаграммы интерфейсов представлены на рисунке 2.6.

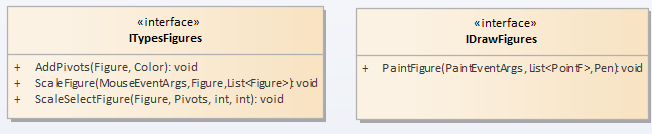


Рисунок 2.6 – Диаграмма интерфейсов ITypesFigures и IDrawFigures

## 2.5 Принцип инверсии зависимостей (PDI)

Суть принципа в следующем: Зависимости внутри системы строятся на основе абстракций. Модули верхнего уровня не зависят от модулей нижнего уровня. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

В моей программе принцип инверсии зависимостей можно увидеть при выделении фигуры. Класс SelectPointoMode отвечает за выбор режима выделение фигуры мышью, но он не знает о том, как выделить фигуру. Этими данными обладает ISelection. Его наследник проверяет, совпадают ли координаты нажатия мыши с точками расположения фигур.

На рисунке 2.7 представлена диаграмма классов SelectPointoMode, PointSelection и ISelection.

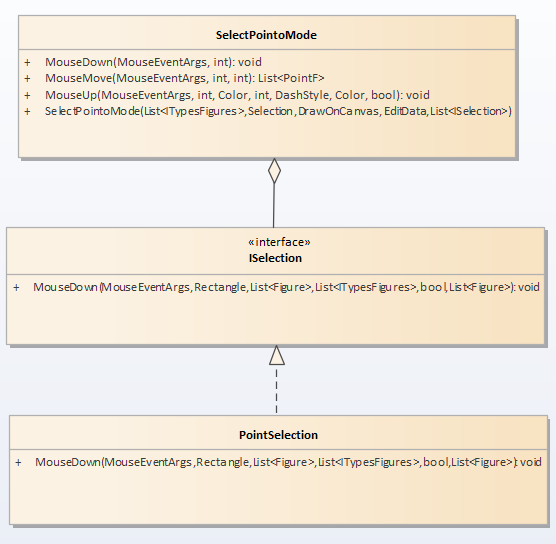


Рисунок 2.7 – Диаграмма классов SelectPointoMode, PointSelection и ISelection

# 3 Заключение

В ходе лабораторной работы были изучены пять основных принципов дизайна классов в объектно-ориентированном проектировании (SOLID), а также на основании данных принципов были исправлены ошибки, допущенные при проектировании графического редактора.

# Список источников

1. Принципы SOLID в C#, [Электронный ресурс] – Режим доступа: https://professorweb.ru/my/it/blog/net/solid.php (дата обращения 10.03.2018)
2. Пять основных принципов дизайна классов (S.O.L.I.D.), [Электронный ресурс] – Режим доступа: http://info.javarush.ru/translation.html (дата обращения 20.03.2018)
3. ISP (Interface segregation principle) - Принцип разделения интерфейса, [Электронный ресурс] – Режим доступа: https://makedev.org/principles/solid/isp.html (дата обращения 22.03.2018)