**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 GRASP

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](#_Toc514778327)

[2 Основная часть 4](#_Toc514778328)

[2.1 Информационный эксперт (Information Expert) 4](#_Toc514778329)

[2.2 Создатель (Creator) 5](#_Toc514778330)

[2.3 Контроллер (Controller) 5](#_Toc514778331)

[2.4 Слабая связанность (Low Coupling) 6](#_Toc514778332)

[2.5 Высокая сцепленность (High Cohesion) 7](#_Toc514778333)

[2.6 Чистая выдумка или чистое синтезирование (Pure Fabrication) 8](#_Toc514778334)

[2.7 Посредник (Indirection) 9](#_Toc514778335)

[2.8 Сокрытие реализации (Protected Variations) 10](#_Toc514778336)

[2.9 Полиморфизм (Polymorphism) 11](#_Toc514778337)

[3 Заключение 13](#_Toc514778338)

[Список источников 14](#_Toc514778339)

# 1 Введение

**GRASP** (General Responsibility Assignment Software Patterns) — шаблоны проектирования, используемые для решения общих задач по назначению обязанностей классам и объектам.

Известно девять GRAPS шаблонов, изначально описанных в книге [Крейга Лармана](http://en.wikipedia.org/wiki/Craig_Larman) «Применение UML и шаблонов проектирования». GRAPS паттерны не имеют выраженной структуры, четкой области применения и конкретной решаемой проблемы, а лишь представляют собой обобщенные подходы/рекомендации/принципы, используемые при проектировании дизайна системы[1].

GRASP выделяет следующие 9 принципов-шаблонов:

* Information Expert (Информационные эксперт);
* Creator (Создатель);
* Controller (Контроллер);
* Low Coupling (Слабая связанность);
* High Cohesion (Высокая сцепленность)
* Pure Fabrication (Чистая выдумка или чистое синтезирование);
* Indirection (Посредник);
* Protected Variations (Сокрытие реализации или защищенные изменения);
* Polymorphism (Полиморфизм).

Далее будет рассмотрен каждый из них по порядку.

# 2 Основная часть

## 2.1 Информационный эксперт (Information Expert)

Шаблон информационный эксперт является базовым и в то же время самым очевидным из девяти. Информационный эксперт описывает основополагающие принципы назначения обязанностей классам и объектам. Согласно описанию, информационным экспертом (объектом, наделенным некоторыми обязанностями) является объект, обладающий максимумом информацией, необходимой для выполнения назначенных обязанностей[1].

В качестве примера можно рассмотреть сохранения файла в формате PNG, Jpeg, GIF. Класс Canva обладает всей необходимой информацией для сохранения проекта в данных форматах. Поэтому именно он занимается данной обязанностью. На рисунке 2.1 представлена диаграмма класса.

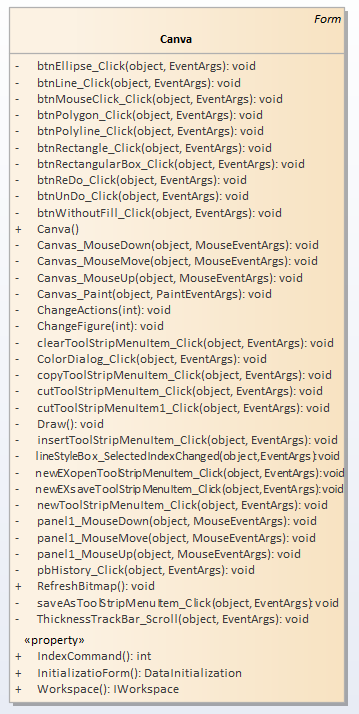


Рисунок 2.1 – Диаграмма класса Canva

Применение шаблона информационный эксперт повышает связность модулей и не противоречит свойству инкапсуляции.

## 2.2 Создатель (Creator)

Шаблон Creator решает проблему о том, кто должен создавать экземпляры новых классов. Решение состоит в назначении классу B обязанностей создавать экземпляры класса A, если выполняется одно из условий[2]:

* Класс B содержит или агрегирует объекты A
* Класс B записывает создаваемые объекты A
* Класс B использует объекты A
* Класс B обладает данными инициализации, которые будут передаваться объектам A при их создании (т.е. при создании объектов А класс В является экспертом).

Примером служит класс SelectRegionFigure, который отвечает за создание класса `MoveFigure, перемещающего фигуры при их выделении. Диаграмма классов представлена на рисунке 2.2.

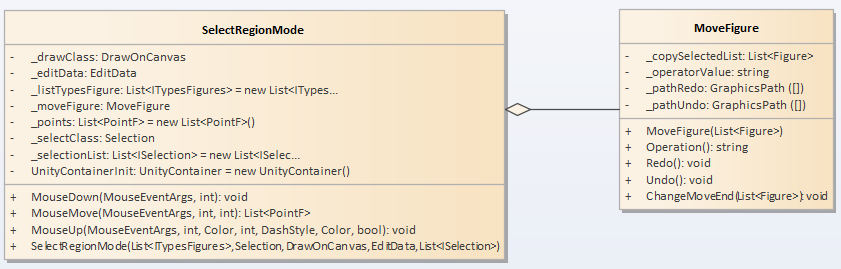


Рисунок 2.2 – Диаграмма классов SelectRegionFigure и MoveFigure

## 2.3 Контроллер (Controller)

Контроллер отвечает за обработку входных системных событий, делегируя обязанности по их обработке компетентным классам. В общем случае, контроллер реализует один или несколько сценариев использования.

В разрабатываемом приложении класс Workspace передает обязанности по обработки событий, поступающий с формы, компонентным классам.

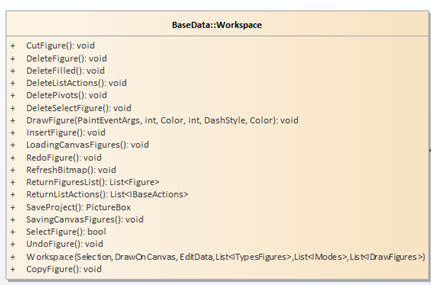


Рисунок – Диаграмма класса Workspace

## 2.4 Слабая связанность (Low Coupling)

Если объекты в приложении сильно связанны, то любое изменение приводит к изменениям во всех связанных объектах. А это неудобно и порождает баги. Вот поэтому необходимо, чтобы код был слабо связан и зависел от абстракций. Например, если класс PointSelection реализует интерфейс ISelection и другие объекты зависят именно от ISelection, т.е. от абстракции, то когда мы захотим внести изменения касательно PointSelection – нам нужно будет всего лишь подменить реализацию. Low Coupling встречается и в SOLID принципах в виде – Dependency Injection. Сейчас можно часто услышать такой принцип. Но суть остается прежней: “Программировать нужно на основе абстракций (интерфейс, абстрактный класс и т.п.), а не реализаций”[3].

На рисунке 2.4 представлены диаграммы классов.

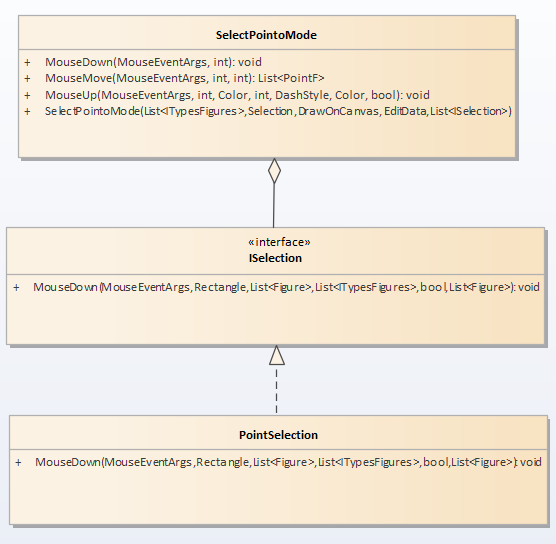


Рисунок 2.4 – Диаграммы классов PointSelection, SelectPointMode и ISelection

## 2.5 Высокая сцепленность (High Cohesion)

Этот принцип тесно соотносится со слабой связанностью, они идут в паре, и одно всегда приводит к другому. Когда их задумывают, имеют какую-то одну ответственность (Single resposibility principle), например класс SaveFigures обладает всеми ответственностями, которые касаются сохранения проекта (Save). Но в классе есть еще одна ответственность такая, как Load, которая отвечает за загрузку файла. Получится, что члены класса, которые касаются Save, будут тесно связанны, также как и класса которые оперируют с Load между собой - будут также тесно связаны, но в целом сцепленность класса SaveAndLoad будет низкой. Следовательно, необходимо разделить класс SaveAndLoad на SaveFigures и LoadFigures, которые внутри будут тесно связанны или по другому сцеплены. Так что высокая сцепленность — это как мера того, что не нарушается single resposibility principle. Вернее сказать, высокая сцепленность получается в результате соблюдения такого принципа из SOLID, как single resposibility principle[3] (SRP). Диаграммы классов до и после разделения представлены на рисунке 2.5 и 2.6.

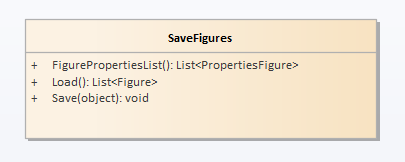


Рисунок 2.5 – Диаграмма класса SaveFigures

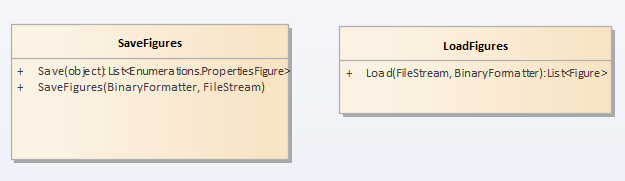


Рисунок 2.6 – Диаграммы классов SaveFigures и LoadFigures

## 2.6 Чистая выдумка или чистое синтезирование (Pure Fabrication)

Здесь суть в выдуманном объекте. Присвоить группу обязанностей с высокой степенью зацепления искусственному классу, не представляющему конкретного понятия из предметной области, т.е. синтезировать искусственную сущность для поддержки высокого зацепления, слабого связывания и повторного спользования.

Такой класс является продуктом нашего воображения и представляет собой синтетику (fabrication). В идеале присвоенные этому классу обязанности поддерживают высокую степень зацепления и низкое связывание, так что cтруктура этого синтетического класса является очень прозрачной или чистой (pure). Отсюда и название: Pure Fabrication (чистая синтетика).

Примером данного принципа является сохранение файла. Если бы данная функция была возложена на сам проект, мы бы получили слабую сцепленность членов класса внутри Canva. Поэтому, сохранение проекта занимается отдельная сущность ISave[4].

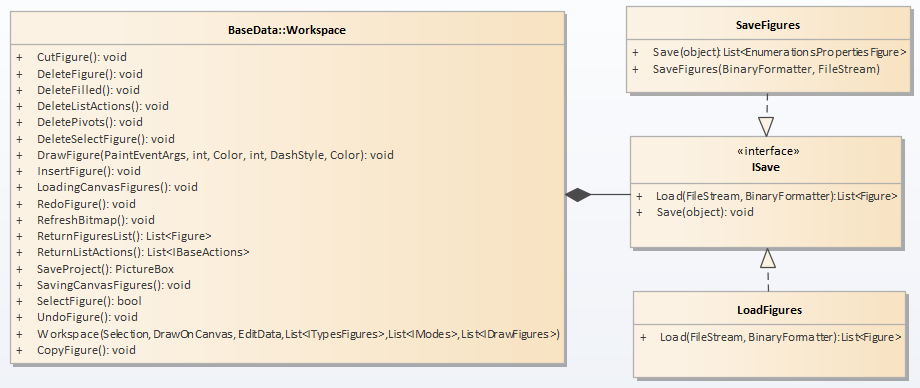


Рисунок 2.7 – Диаграмма классов Workspace, SaveFigure, LoadFigure и ISave

## 2.7 Посредник (Indirection)

Indirection (Перенаправление) - принцип, определяющий объект, который является промежуточным звеном между двумя другими. Он нужен для уменьшения связанности.

Пример, рассмотренный выше, с введением класса Workspace является также и примером шаблона Indirection, т.к. дополнительный класс будет являться промежуточным звеном между Canva и компонентным классом.

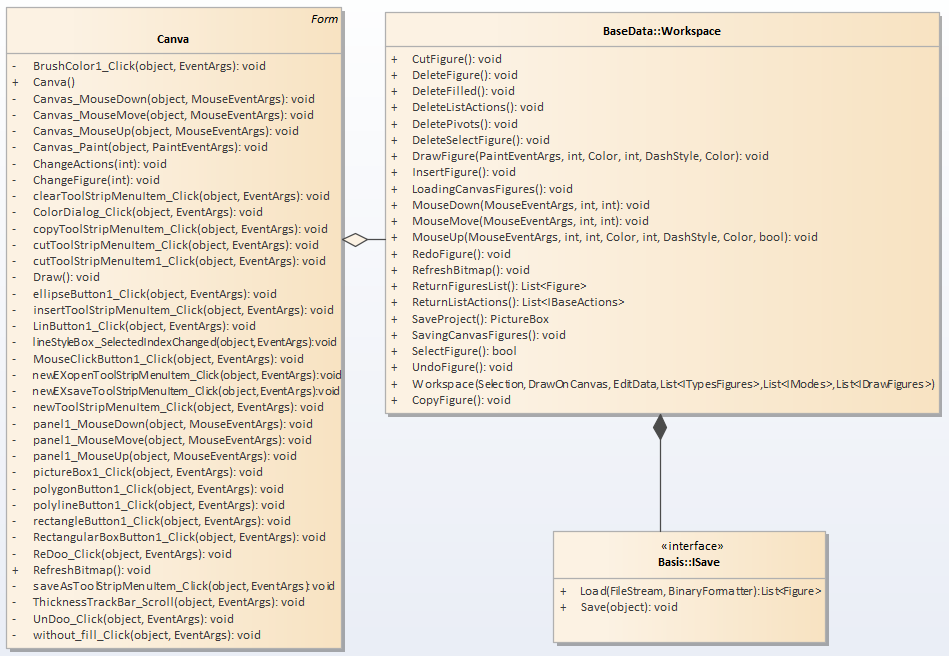


Рисунок 2.8 – Диаграмма классов

## 2.8 Сокрытие реализации (Protected Variations)

Очень похож на инкапсуляцию - еще один принцип ООП. Он защищает объекты от изменения другими элементами, путем вынесения взаимодействия в фиксированный интерфейс.

Суть в том, чтобы идентифицировать точки возможных вариаций или неустойчивости, распределить обязанности таким образом, чтобы обеспечить устойчивый интерфейс То есть принцип в том, чтобы определить места в системе, где поведение может измениться, и выделить абстракцию, на основе которой и будет происходить дальнейшее программирование с использованием этого объекта. Все это делается для того лишь, чтобы обеспечить устойчивость интерфейса. Если будет много изменений, связанных с объектами, он, в таком случае считается неустойчивым, и тогда нужно выносить его в абстракцию, от которой он и будет зависеть, либо же распределять обязанности и ответственность в коде иным образом[5].

Примером данного принципа может служить рабочая область. Определив интерфейс IWorkspace, мы скрываем реализацию конкретного интерфейса. А также делаем его устойчивым к изменениям самой канвы.

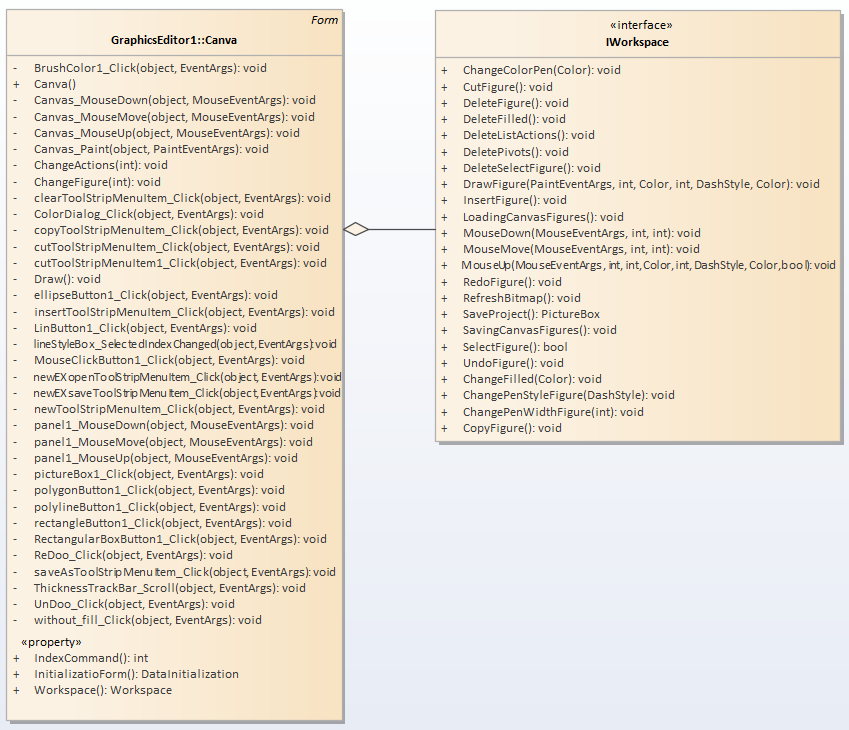


Рисунок 2.9 – Диаграмма классов

## 2.9 Полиморфизм (Polymorphism)

Этот принцип, является основным в концепции объектно-оринтированного программирования. Он дает возможность трактовать однообразно разные объекты с одинаковым интерфейсом (спецификацией). Он обозначает, способность некоторого объекта изменять свое поведение. А именно, есть объект класс Animal, данный класс реализует два интерфейса - Predator и Herbivore, таким образом, в зависимости от типа интерфейсной ссылки объект может обладать поведением хищника или травоядного, волком или безобидной овечкой[6].

В разрабатываемом приложении примером могут служить классы реализующие интерфейс IModes. Мы используем эти классы с одинаковым интерфейсом без информации о их типе и внутренней структуре.

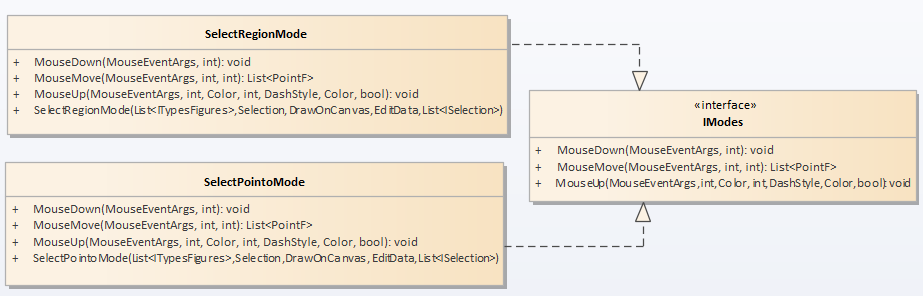


Рисунок 2.10 – Диаграмма классов SelectRegionMode, SelectPointoMode и IModes

# 3 Заключение

В ходе лабораторной работы были изучены девять основных принципов GRASP, а также на основании данных принципов были исправлены ошибки, допущенные при проектировании графического редактора.

# Список источников

1. Принципы GRASP в C#, [Электронный ресурс] – Режим доступа: https://habr.com/post/92570/, свободный (дата обращения 10.04.2018)
2. Шаблон Creator, [Электронный ресурс] – Режим доступа: https://studfiles.net/preview/5053690/page:3/, свободный (дата обращения 11.04.2018)
3. Принципы GRASP, [Электронный ресурс] – Режим доступа: http://regfordev.blogspot.ru/2015/08/grasp.html#.WwQGwiArnv0, свободный (дата обращения 12.04.2018)
4. Шаблоны GRASP: Pure Fabrication, [Электронный ресурс] – Режим доступа: http://nvoynov.blogspot.ru/2007/06/grasp-pure-fabrication.html, свободный (дата обращения 12.04.2018)
5. Шаблоны GRASP: Protected Variations, [Электронный ресурс] – Режим доступа: http://nvoynov.blogspot.ru/2007/06/grasp-protected-variations.html, свободный (дата обращения 13.04.2018)
6. Принципы GRASP [Электронный ресурс] – Режим доступа: http://www.izebit.ru/2015/12/grasp.html, свободный (дата обращения 13.04.2018)