Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

Высшего профессионального обучения

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

Отчет по лабораторной работе №3 по дисциплине

«Технология разработки программного обеспечения»

«Анализ на соответствие GRASP»

Выполнил:

Студент гр. 588-М1

\_\_\_\_\_\_Д.А. Домаскин

. «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

Проверил:

Доцент каф. КСУП

\_\_\_\_\_\_\_\_\_ А.А. Калентьев

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

Томск 2019

# **Оглавление**

[**Введение** 3](#_Toc7038730)

[**1 Information Expert (Информационный эксперт)** 4](#_Toc7038731)

[**2 Creator (Создатель)** 5](#_Toc7038732)

[**3 Controller** 7](#_Toc7038733)

[**4 Polymorphism** 8](#_Toc7038734)

[**5 Pure Fabrication (Чистая выдумка)** 9](#_Toc7038735)

[**6 Indirection (Посредник)** 10](#_Toc7038736)

[**7 Сокрытие реализации** 11](#_Toc7038737)

[**Заключение** 12](#_Toc7038738)

# **Введение**

GRASP - (англ. General Responsibility Assignment SoftwareиPatterns — общие образцы распределения обязанностей)— паттерны, используемые в ООП для решения общих задач по назначению обязанностей классам и объектам.

**GRASP**

1. Information Expert (Информационный эксперт)
2. Creator (Создатель)
3. Low Coupling (Слабое зацепление)
4. High Cohesion (Сильная связность)
5. Controller (Контроллер)
6. Polymorphism (Полиморфизм)
7. Pure Fabrication (Чистая выдумка)
8. Indirection (Посредник)
9. Protected Variations (Сокрытие реализации)

В данной лабораторной работе будет проведен анализ программного кода на соответствие принципам GRASP (кроме связности и зацепления).

## **1 Information Expert (Информационный эксперт)**

В качестве **положительного примера** информационного эксперта можно привести класс *DrawerFacade*, который обладает всей достаточной информацией для отрисовки любого объекта. На рисунке 1.1 приведена диаграмма класса *DrawerFacade*.

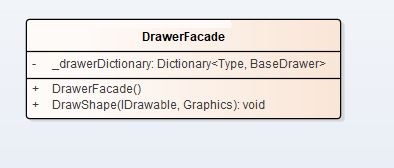


Рисунок 1.1 – Диаграмма класса *DrawerFacade*

**Негативным примером** является класс *StorageManager*, его диаграмма приведена на рисунке 1.2.

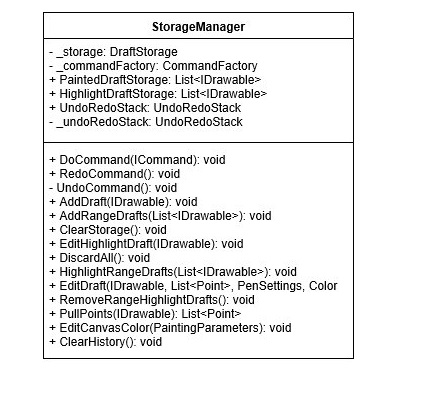
****

Рисунок 1.2 – Диаграмма класса *StorageManager*

Данный класс явно перегружен обязанностями. Необходимо декомпозировать его на более мелкие классы с более конкретными ответственностями.

## **2 Creator (Создатель)**

**Позитивны примером** является класс *DrawerFacade*, его диаграмма представлена на рисунке 2.1.

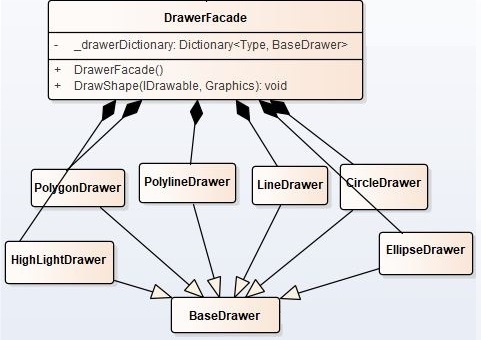


Рисунок 2.1 – Класс *DrawerFacade*

Данный класс композирует на себе объекты, которые являются инструментами рисования, предоставляя общий интерфейс для работы с ними. Композируемые классы создаются одновременно с композирующим и никуда не передаются. По этой причине был использован фасад, а не фабрика.

Негативным примером является класс *CommandFactory,* его диаграмма представлена на рисунке 2.2. Данный класс делегирует создание объектов типа *ICommand*. Но по причине того, что все команды имеют разные сигнатуры, не имеет полиморфного универсального метода для создания объектов.

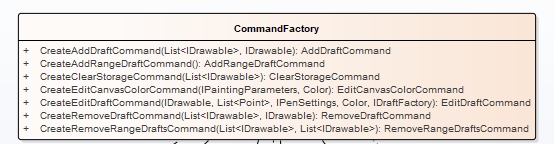


Рисунок 2.2 – Диаграмма класса *CommandFactory*

## **3 Controller**

В качестве **положительного примера** контроллера можно привести ранее описанный класс *DrawerFacade*, диаграмма которого приведена на рисунке 2.1. Ниже приведена реализация метода *DrawShape*, который обеспечивает универсальный доступ к любому инструменту рисования, делегируя им функции.

public void DrawShape(IDrawable shape, Graphics graphics)

{

if (shape == null)

return;

\_drawerDictionary[shape.GetType()]?.DrawShape(shape, graphics);

}

**Негативный пример –** класс *DrawManager* представленный на рисунке 3.1.

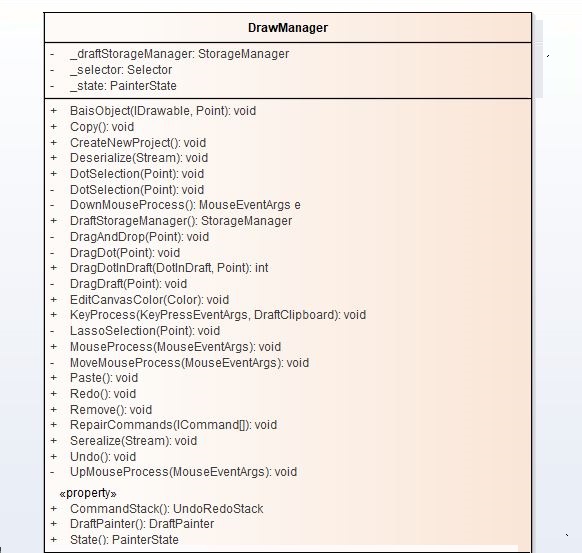


Рисунок 3.1 – Класс *DrawManager*

Класс делегирует различные функции другим сущностям. Но класс получился через мерно раздутым.

## **4 Polymorphism**

В качестве положительного примера обратимся к уже ранее описанному классу *DrawerFacade*, его диаграмма представлена на рисунке 4.1.

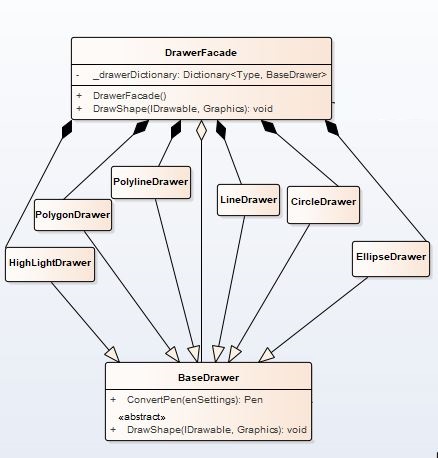


Рисунок 4.1 – Класс *DrawerFacade*

Класс *DrawerFacade* в методе *DrawShape* вызывает метод *DrawShape* базового класса *BaseDrawer.*

## **5 Pure Fabrication (Чистая выдумка)**

В качестве примера чистой выдумки можно привести класс *UndoRedoStack,* его диаграмма приведена на рисунке 5.1.

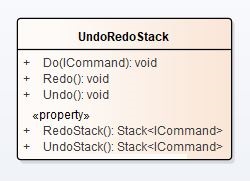


Рисунок 5.1 – Класс *UndoRedoStack*

Данный класс является искусственной сущностью, не привязанной к предметной области, которая обязана за реализацию функционала “назад-вперед”.

## **6 Indirection (Посредник)**

До IoC рефакторинга пример данного паттерна найти не удалось. Но после применения IoC практически все сущности в приложении связанны не на прямую, а через посредника. Пример связи сущностей через посредника приведен на рисунке 6.1.

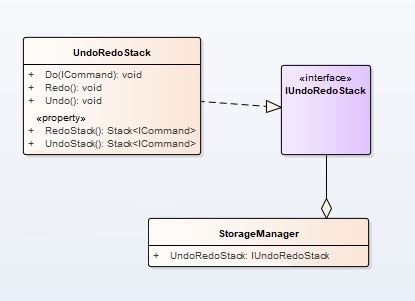


Рисунок 6.1 – Диаграмма классов связанных через посредника

## **7 Сокрытие реализации**

На данном этапе сокрытие реализации применялось только в виде использования инкапсуляции, интерфейсов и полиморфизма.

К примеру функция *SoloDraw(IDrawable draft)* класса *DraftPainter,* использует интерфейс, для того чтобы отрисовывать фигуры*.*

## **Заключение**

В ходе выполнения лабораторной работы был проанализирован программный код на соответствие принципам GRASP. Результаты анализа помогли выявить участки в коде, на которые нужно обратить особое внимание.