Лабораторная работа №19
Использование шаблонов проектирования

1 Цель работы

1.1 Научиться применять паттерны проектирования в разработке программ.

2 Литература

2.1 https://metanit.com/sharp/patterns/

3 Задание

Требуется перевести код программ, реализующих паттерны проектирования, с языка Java на язык C# с учетом требований к наименованиям элементов, типов данных, объектов и подходов, использующихся при разработке на C#.

На схемах, приведенных на рисунках 1-5, аналог класса Program – это класс, в котором указан метод main(). Класс Program переименовывать не требуется.

Отличия кода на C# от кода на Java:

- для вывода данных вместо System.out.println() используется Console.WriteLine()
- строковый тип данных принято писать с маленькой буквы: string
- названия методов нужно писать с заглавной буквы: ИмяМетода
- вместо методов get и set использовать свойства (при допустимости: автосвойства)
- названия интерфейсов должны начинаться с буквы I: IИнтерфейс
- не нужно писать public в интерфейсах
- @Override при реализации интерфейса указывать не нужно
- при реализации интерфейсов вместо implements пишется двоеточие
- при наследовании класса вместо extends пишется двоеточие
- при наследовании класса для обращения к родительскому классу вместо super пишется base

3.1 Реализовать и объяснить поведенческий паттерн «Стратегия» согласно UML-диаграмме на рисунке 1.

Добавить еще одну стратегию к описанному интерфейсу.

https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm

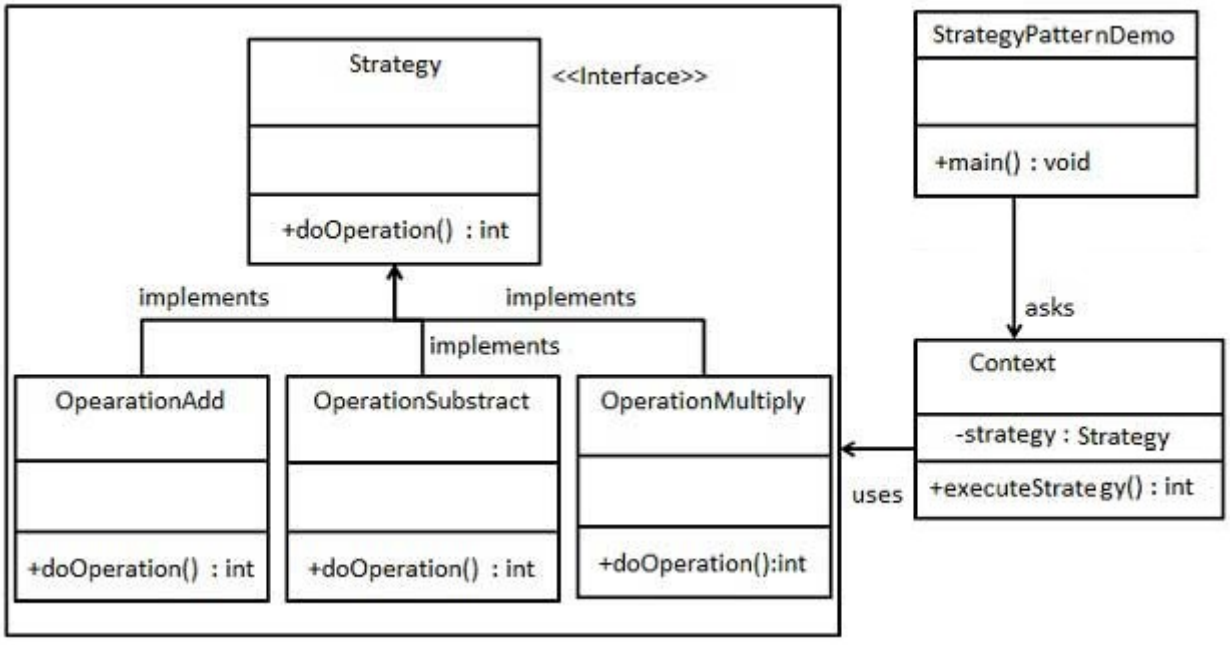

Рисунок 1

3.2 Реализовать и объяснить порождающий паттерн «Фабричный метод» согласно UML-диаграмме на рисунке 2.

Добавить еще одну фигуру к описанному интерфейсу.

https://www.tutorialspoint.com/design_pattern/factory_pattern.htm

Для сравнения строк без учета регистра вместо equalsIgnoreCase используется:
строка1.Equals(строка2, StringComparison.OrdinalIgnoreCase)

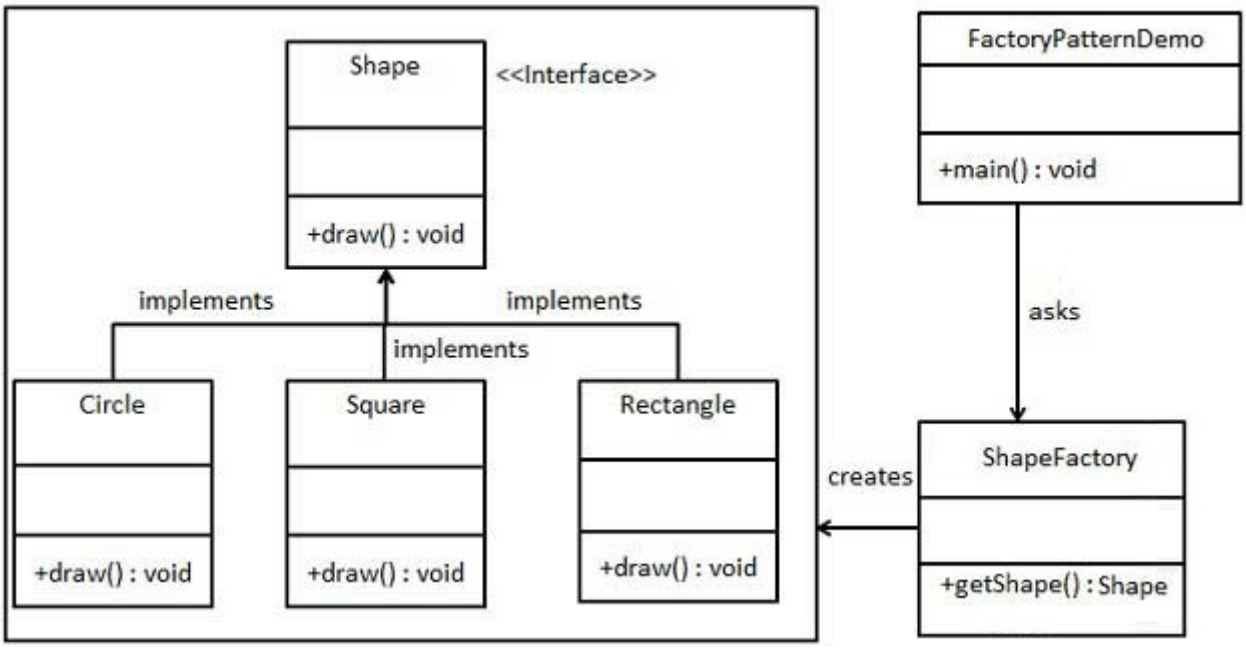

Рисунок 2

3.3 Реализовать и объяснить структурный паттерн «Декоратор» согласно UML-диаграмме на рисунке 3.

Добавить еще один декоратор к описанному интерфейсу.

https://www.tutorialspoint.com/design_pattern/decorator_pattern.htm

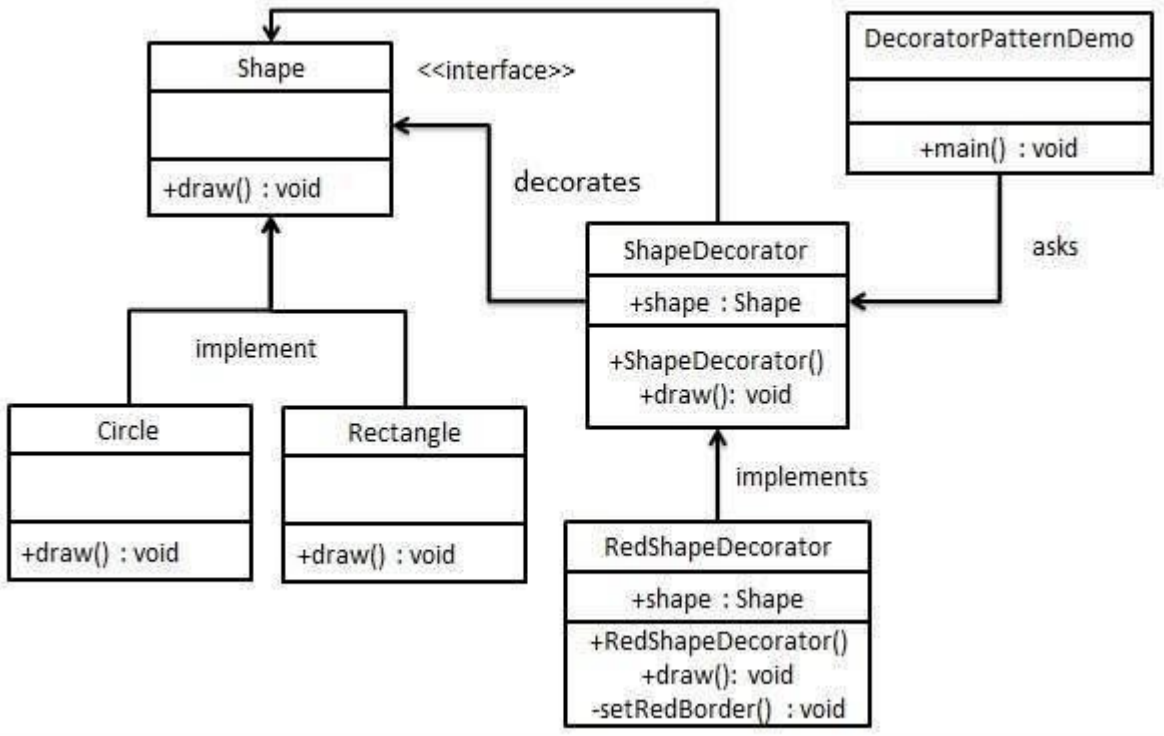

Рисунок 3

3.4 Реализовать и объяснить поведенческий паттерн «Наблюдатель» согласно UML-диаграмме на рисунке 4. Вместо списка наблюдателей использовать модель событий.

Добавить еще одного наблюдателя, отображающего число в десятичной системе.

https://www.tutorialspoint.com/design_pattern/observer_pattern.htm

Для перевода в систему счисления по основанию 2/8/10/16 вместо Integer.toОснованиеString используется:

Convert.ToString(число, основание)

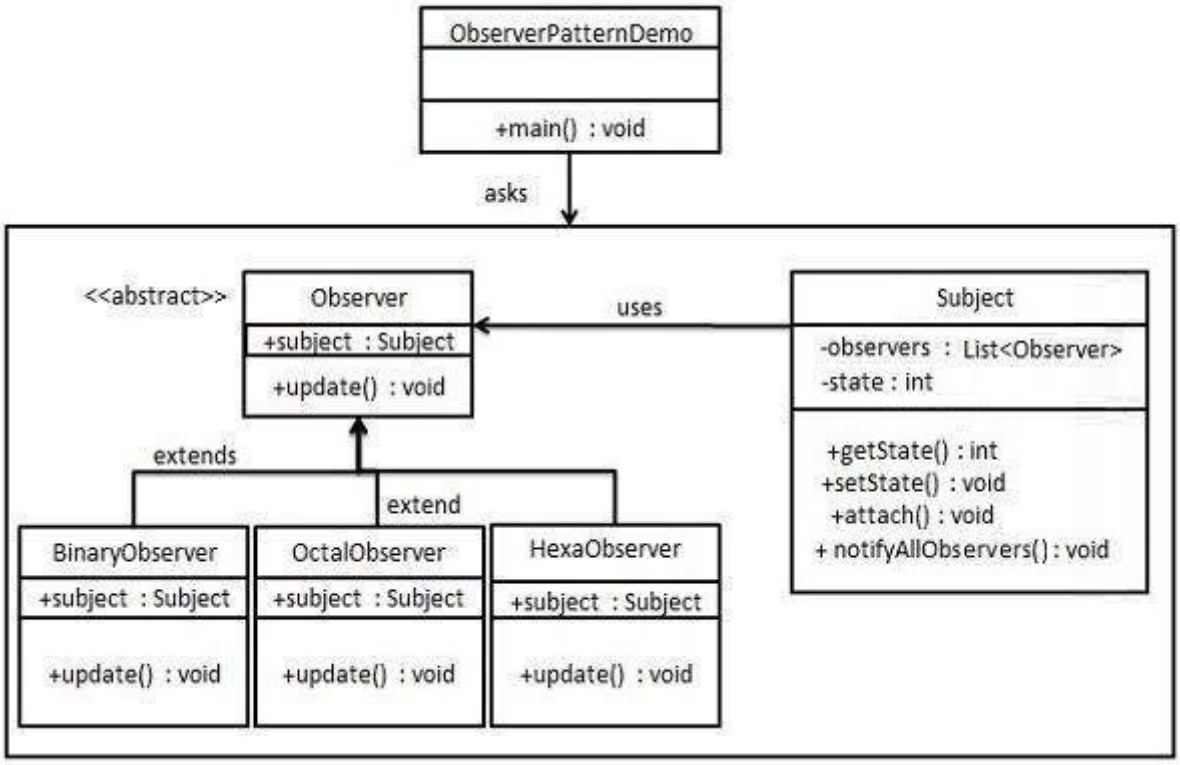

Рисунок 4

3.5 Реализовать и объяснить порождающий паттерн «Одиночка» для сохранения конфигурации приложения.

Создать класс по шаблону «Одиночка» и добавить в него:

- словарь для хранения названий настроек и их значений (например: цвет текста: зеленый, размер текста: 14),

- два отдельных открытых метода или индексатор для добавления настройки (если такая уже есть, заменять ее значение) и удаления настройки по названию,

- переопределенный метод ToString для возврата данных в следующем формате (названия настроек и строковые данные должны быть в кавычках):

```
{
  "настройка1": "значение1",
  "настройка2": значение2,
  …
}
```

4 Порядок выполнения работы

4.1 Выполнить все задания из п.3 в одном решении LabWork19, каждое – в своем консольном проекте. Выполнить форматирование и рефакторинг кода.

4.2 Ответить на контрольные вопросы.

5 Содержание отчета

5.1 Титульный лист
5.2 Цель работы
5.3 Ответы на контрольные вопросы
5.4 Вывод

6 Контрольные вопросы

6.1 Для чего используются порождающие паттерны?
6.2 Какие паттерны относятся к порождающим?
6.3 Для чего используются структурные паттерны?
6.4 Какие паттерны относятся к структурным?
6.5 Для чего используются поведенческие паттерны?
6.6 Какие паттерны относятся к поведенческим?