

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра информационной безопасности**

**ОТЧЕТ**  
**по практической работе №2**  
**по дисциплине «Объектно-ориентированное программирование»**  
**Тема: Паттерн “Наблюдатель”**

Студенты гр. 3363

Преподаватель

Гончаренко О. Д.  
Овсейчик Н. И.,  
Минко Д. А.

Новакова Н. Е.

Санкт-Петербург  
2024

## **Цель работы**

Разработать программу, реализующую паттерн "Наблюдатель". В данной работе создаётся система, которая моделирует прогноз погоды. Классы взаимодействуют друг с другом согласно принципам паттерна: объект-субъект (WeatherStation) оповещает объекты-наблюдатели (WeatherObserver) об изменении состояния.

## ХОД РАБОТЫ

### 1. Изучение паттерна "Наблюдатель"

Паттерн "Наблюдатель" реализует связь "один-ко-многим", где изменение состояния одного объекта приводит к автоматическому уведомлению всех связанных с ним объектов.

Для реализации паттерна была разработана UML – диаграмма классов, отображающая связь между классами Program, WeatherStation, WeatherObserver и Simulator (рис1).

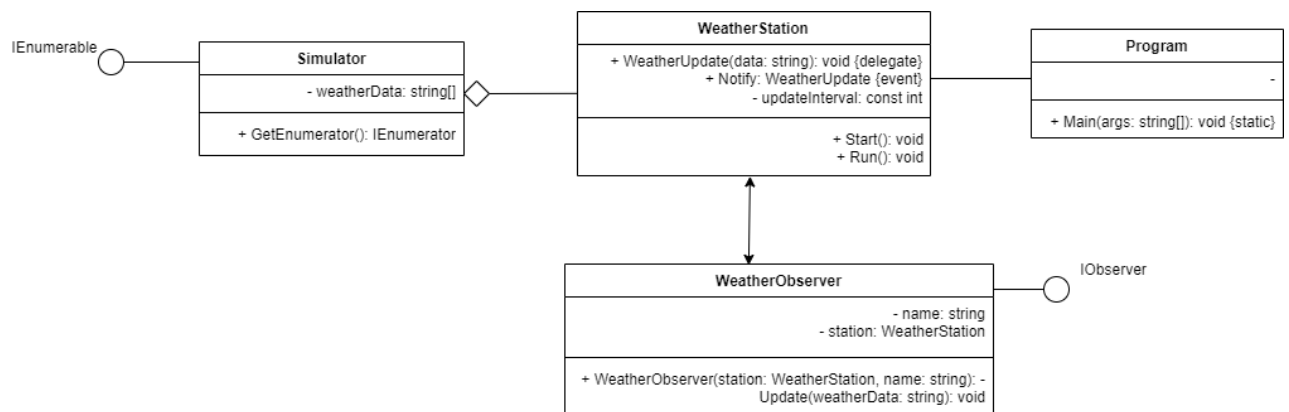


Рисунок 1 – UML диаграмма классов программы

На диаграмме представлены основные классы и их взаимодействие:

- Program управляет созданием и запуском программы;
- WeatherStation является субъектом, который уведомляет наблюдателей о состоянии погоды;
- WeatherObserver представляет наблюдателей, получающих данные от субъекта;
- Simulator предоставляет данные для субъекта

### 2. Реализация программы

- Создан класс WeatherStation, который генерирует данные о погоде с помощью класса Simulator и уведомляет наблюдателей через событие Notify.

- Создан класс WeatherObserver, который подписывается на изменения в WeatherStation и выводит полученные данные.
- Класс Simulator используется для предоставления данных о погоде.
- В классе Program создаются экземпляры субъектов и наблюдателей, иницируется работа станции.

## Спецификация программы

### 1. Класс WeatherStation

#### Описание методов класса WeatherStation

Таблица 1.1 Описание методов класса WeatherStation

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
Start	void	public	-	Запуск процесса уведомления наблюдателей
Run	void	private	-	Генерация данных о погоде и вызов события

#### Описание полей класса WeatherStation

Таблица 1.2. Описание полей класса WeatherStation

Имя	Тип	Модификатор доступа	Назначение
Notify	WeatherUpdate	public	Событие для оповещения наблюдателей
simulator	Simulator	private	Объект, предоставляющий данные о погоде
updateInterval	const int	private	Интервал времени между оповещениями

## Описание делегата

Таблица 1.3. Описание делегата WeatherUpdate

Имя	Тип	Входные параметры	Модификатор доступа	Назначение
WeatherUpdate	delegate void	string data	public	Делегат, используемый для уведомления подписанных методов

## 2. Класс WeatherObserver

### Описание методов класса WeatherObserver

Таблица 2.1. Описание методов класса WeatherObserver

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
WeatherObserver	-	public	WeatherStation station, string name	Конструктор класса, регистрирует наблюдателя в WeatherStation
Update	void	public	string weatherData	Получение обновления от WeatherStation и выводит данные

### Описание полей класса WeatherObserver

Таблица 2.2. Описание полей класса WeatherObserver

Имя	Тип	Модификатор доступа	Назначение
name	string	private	Имя наблюдателя
station	WeatherStation	private	Станция, на которую подписан наблюдатель

### 3. Класс Simulator

#### Описание методов класса Simulator

Таблица 3.1. Описание методов класса Simulator

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
GetEnumerator	IEnumerator	public	-	Перебор данных о погоде

#### Описание полей класса Simulator

Таблица 3.2. Описание полей класса Simulator

Имя	Тип	Модификатор доступа	Назначение
weatherData	string[]	private	Массив строк с данными о погоде

## 4. Класс Program

### Описание методов класса Program

Таблица 4.1. Описание методов класса Program

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
Main	void	public	string[] args	Создание объекта WeatherStation и регистрация наблюдателей

### Пример работы программы

- Программа создаёт объект WeatherStation (субъект) и три объекта WeatherObserver (наблюдатели);
- Наблюдатели подписываются на событие Notify в WeatherStation;
- WeatherStation начинает генерировать данные о погоде с помощью объекта Simulator и уведомляет подписанных наблюдателей;
- Наблюдатели выводят полученные данные на консоль (рис.2):

```
Станция сообщает: Температура: 25°C, Влажность: 60%, Давление: 1013 hPa
Наблюдатель 1 получил обновление: Температура: 25°C, Влажность: 60%, Давление: 1013 hPa
Наблюдатель 2 получил обновление: Температура: 25°C, Влажность: 60%, Давление: 1013 hPa
Наблюдатель 3 получил обновление: Температура: 25°C, Влажность: 60%, Давление: 1013 hPa
Станция сообщает: Температура: 27°C, Влажность: 55%, Давление: 1010 hPa
Наблюдатель 1 получил обновление: Температура: 27°C, Влажность: 55%, Давление: 1010 hPa
Наблюдатель 2 получил обновление: Температура: 27°C, Влажность: 55%, Давление: 1010 hPa
Наблюдатель 3 получил обновление: Температура: 27°C, Влажность: 55%, Давление: 1010 hPa
Станция сообщает: Температура: 22°C, Влажность: 70%, Давление: 1020 hPa
Наблюдатель 1 получил обновление: Температура: 22°C, Влажность: 70%, Давление: 1020 hPa
Наблюдатель 2 получил обновление: Температура: 22°C, Влажность: 70%, Давление: 1020 hPa
Наблюдатель 3 получил обновление: Температура: 22°C, Влажность: 70%, Давление: 1020 hPa
```

Рисунок 2 – Вывод результата программы

## **Вывод**

В рамках данной работы была реализована программа, демонстрирующая принцип работы паттерна "Наблюдатель". Основной задачей было создать систему, в которой субъект (WeatherStation) уведомляет подписанных на него наблюдателей (WeatherObserver) об изменениях своего состояния, используя событийный механизм.

Решение может быть адаптировано для реальных задач, таких как мониторинг систем или управление событиями в реальном времени. Работа с паттерном позволила глубже понять его принципы и применимость в разработке программного обеспечения.



## ПРИЛОЖЕНИЕ 1 ИСХОДНЫЙ КОД ПРОГРАММЫ

### 1. Интерфейс IObserver

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Pattern2
{
    // Интерфейс наблюдателя
    interface IObserver
    {
        void Update(string weatherData);
    }
}
```

### 2. Класс Simulator

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Pattern2
{
    class Simulator : IEnumerable
    {
        private string[] weatherData =
        {
            "Температура: 25°C, Влажность: 60%, Давление: 1013 hPa",
            "Температура: 27°C, Влажность: 55%, Давление: 1010 hPa",
            "Температура: 22°C, Влажность: 70%, Давление: 1020 hPa"
        };

        public IEnumerator GetEnumerator()
        {
            foreach (string data in weatherData)
                yield return data;
        }
    }
}
```

### 3. Класс WeatherObserver

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Pattern2
{
    // Класс Наблюдателя
    class WeatherObserver : IObservable
    {
        private string name;
        private WeatherStation station;

        public WeatherObserver(WeatherStation station, string
name)
        {
            this.station = station;
            this.name = name;
            station.Notify += Update;
        }

        public void Update(string weatherData)
        {
            Console.WriteLine($"{name} получил обновление:
{weatherData}");
        }
    }
}
```

### 4. Класс WeatherStation

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Pattern2
{
    // Класс субъекта
    class WeatherStation
    {
        public delegate void WeatherUpdate(string data);
        public event WeatherUpdate Notify;

        private Simulator simulator = new Simulator();

        private const int updateInterval = 2000; // Интервал
обновления в миллисекундах
    }
}
```

```

        public void Start()
        {
            new Thread(Run).Start();
        }

        private void Run()
        {
            foreach (var data in simulator)
            {
                Console.WriteLine($"Станция сообщает: {data}");
                Notify?.Invoke(data.ToString());
                Thread.Sleep(updateInterval);
            }
        }
    }
}

```

## 5. Класс Program

```

using Pattern2;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Threading;

// Главный метод
class Program
{
    public static void Main(string[] args)
    {
        WeatherStation station = new WeatherStation();

        WeatherObserver observer1 = new WeatherObserver(station,
"Наблюдатель 1");
        WeatherObserver observer2 = new WeatherObserver(station,
"Наблюдатель 2");
        WeatherObserver observer3 = new WeatherObserver(station,
"Наблюдатель 3");

        station.Start();

        Console.ReadLine();
    }
}

```