Занятие 5. Генеративные паттерны

Цель занятия

1. Получить практический опыт в использовании генеративных паттернов.

Описание занятия

Генеративные паттерны позволяют создавать объекты в зависимости от контекста. В предыдущих занятиях мы уже использовали некоторые генеративные паттерны, например, `Factory Method` при создании автомобилей.

В данном занятии мы рассмотрим два паттерна:

- `Singleton`;
- `Builder`.

В качестве примера паттерна `Singleton` мы упростим код `Program.cs`, вынеся создание IoC контейнера в отдельный класс за счет использования данного паттерна.

В качестве примера паттерна `Builder` мы рассмотрим задачу построения отчета о работе нашей системы.

Вынесение IoC контейнера в отдельный класс

Для начала заведем класс `CompositionRoot`, который будет создавать IoC контейнер и заполнять его сервисами.

```
public static class CompositionRoot
{
}
```

В данном классе нам понадобится метод, который будет создавать IoC контейнер и заполнять его сервисами.

```
public static class CompositionRoot
{
    private static IServiceProvider CreateServiceProvider()
    {
        var services = new ServiceCollection();

        services.AddSingleton<CarService>();
        services.AddSingleton<CustomersStorage>();
        services.AddSingleton<PedalCarFactory>();
        services.AddSingleton<HandCarFactory>();

        services.AddSingleton<ICarProvider>(sp =>
```

```
sp.GetRequiredService<CarService>());
        services.AddSingleton<ICustomersProvider>(sp =>
sp.GetRequiredService<CustomersStorage>());
        services.AddSingleton<HseCarService>();


        return services.BuildServiceProvider();
    }
}
```

Как вы можете видеть, в данный метод мы поместили код, который был в `Program.cs`.

Теперь заведем свойство `Services` в классе `CompositionRoot`, которое будет возвращать IoC контейнер.

```
public static class CompositionRoot
{
    public static IServiceProvider Services { get; } = CreateServiceProvider();

    // остальной код
}
```

Как мы видим, свойство `Services` автоматически инициализируется при старте программы. Но это не всегда требуется. Иногда мы можем захотеть инициализировать IoC контейнер при первом обращении к нему. Для этого мы можем использовать так называемую ленивую инициализацию:

```
public static class CompositionRoot
{
    private static IServiceProvider? _services;

    public static IServiceProvider Services => _services ??=
CreateServiceProvider();

    // остальной код
}
```

Данный код довольно комплексный, поэтому мы рассмотрим его поподробнее.

Как мы видим, свойство `Services` всегда возвращает значение поля `_services`. При этом, если поле `_services` равно `null`, то оно будет инициализировано с помощью метода `CreateServiceProvider`. Это происходит за счет использования оператора `??=`, так называемого "coalescing assignment". Это обеспечивает ленивую инициализацию поля `_services`.

Теперь добавим использование нового класса `CompositionRoot` в `Program.cs`:

```
var services = CompositionRoot.Services;
```

```
var customers = services.GetRequiredService<CustomersStorage>();
var cars = services.GetRequiredService<CarService>();
var pedalCarFactory = services.GetRequiredService<PedalCarFactory>();
var handCarFactory = services.GetRequiredService<HandCarFactory>();

var hse = services.GetRequiredService<HseCarService>();
```

Как вы можете видеть, мы получили IoC контейнер и с его помощью получили все необходимые нам сервисы, при этом не загромождая класс `Program.cs` лишним кодом.

Построение отчета о работе нашей системы

Для начала заведем класс `Report`, который будет представлять собой отчет о работе нашей системы.

```
public record Report(string Title, string Content)
{
    public override string ToString()
    {
        return $"{Title}\n\n{Content}";
    }
}
```

В качестве заголовка отчета мы будем использовать слово "Отчет" и текущую дату, а в качестве содержания - содержимое хранилища покупателей и список проделанных операций.

Также перегрузим метод `ToString` для вывода отчета в консоль.

Для построения отчета мы будем использовать паттерн `Builder`. Для его реализации создадим класс `ReportBuilder`.

```
public class ReportBuilder
{
}
```

Добавим в класс `ReportBuilder` поле для хранения содержимого отчета. Для этого заведем поле `_content` типа `StringBuilder`.

```
public class ReportBuilder
{
    private readonly StringBuilder _content = new();
}
```

Класс `StringBuilder` представляет собой реализацию паттерна `Builder` для строк.

Добавим в наш класс метод `AddCustomers`, который будет добавлять в отчет содержимое хранилища покупателей.

```
public class ReportBuilder
{
    private readonly StringBuilder _content = new();

    public ReportBuilder AddCustomers(IEnumerable<Customer> customers)
    {
        _content.AppendLine("Покупатели:");

        foreach (var customer in customers)
        {
            _content.AppendLine($" - {customer}");
        }

        _content.AppendLine();

        return this;
    }
}
```

Обратите внимание, что метод `AddCustomers` возвращает `this`, что позволяет использовать цепочечные вызовы методов, что характерно для паттерна `Builder`.

Теперь добавим метод `AddOperation`, который будет добавлять в отчет проделанную операцию.

```
public class ReportBuilder
{
    // остальной код

    public ReportBuilder AddOperation(string operation)
    {
        _content
            .AppendLine($"Операция: {operation}")
            .AppendLine();

        return this;
    }
}
```

Теперь добавим метод `Build`, который будет возвращать построенный отчет.

```
public class ReportBuilder
{
    // остальной код

    public Report Build()
```

```
    {
        return new Report($"Отчет за {DateTime.Now:yyyy-MM-dd}",
_content.ToString());
    }
}
```

Теперь мы можем использовать класс `ReportBuilder` для построения отчета.

```
var report = new ReportBuilder()
    .AddOperation("Инициализация системы")
    .AddCustomers(customers);

hse.SellCars();

report
    .AddOperation("Продажа автомобиля")
    .AddCustomers(customers);

Console.WriteLine(report.Build());
```

Как мы видим, код стал проще и понятнее.

Самостоятельная работа

Добавьте в `ReportBuilder` возможность вывода текущего содержимого склада автомобилей.