**Лабораторная работа №4**

**«Реализация одного из структурных паттернов проектирования»**

**Цель работы:** применить паттерн Легковес (Flyweight)

**Продолжительность работы:** 4 часа.

**Паттерн Легковес (Flyweight)**

Легковес – структурный паттерн уровня объектов, который позволяет вместить большее количество объектов в отведенную память. Легковес экономит память, разделяя общее состояние объектов между собой, вместо хранения одинаковых данных в каждом объекте.

**Пример из жизни**

Давайте представим, что вы делаете компьютерную игру и вам необходимо отрисовать один миллион деревьев. Какие поля определяют дерево в игре? Самыми важными являются его спрайт (изображение) и координаты. Посчитаем, сколько потребуется памяти для хранения всех деревьев. Спрайт весит около 20 килобайт, а координаты 12 байт. Итого для хранения одного миллиона объектов потребуется:

Довольно много для одних только деревьев. Подумаем, как это оптимизировать. Возможно у этих деревьев есть что-то общее? Крайне маловероятно, что в игре используется миллион различных видов деревьев. Скорее всего существует всего лишь несколько видов деревьев, каждый из которых используется для огромного количества деревьев. Тогда зачем нам хранить спрайты для каждого дерева, если они одинаковые для большого количества объектов? Тут то нам и поможет паттерн Легковес.

Давайте посчитаем примерную экономию памяти. Допустим, у нас есть 10 различных спрайтов. На это тратится килобайт. Координаты у каждого дерева уникальны, следовательно потребуется ещё байт. Итого для миллиона объектов нужно:

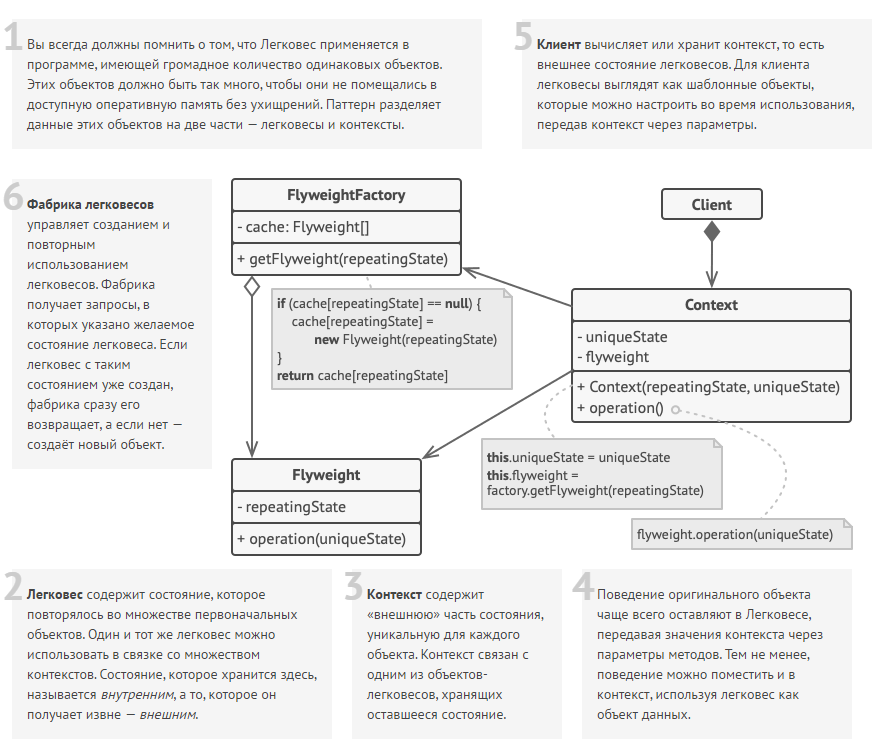
Получается довольно серьёзная экономия памяти.

**Структура:**

Для начала давайте введем пару новых определений.

Внутреннее состояние – неизменяемые данные объекта. Все остальное будем называть «внешним состоянием».

Паттерн Легковес предлагает не хранить в классе внешнее состояние, а передавать его в те или иные методы через параметры. Таким образом, одни и те же объекты можно будет повторно использовать в разных контекстах. Но главное – понадобится гораздо меньше объектов, ведь теперь они будут отличаться только внутренним состоянием, а оно имеет не так много вариаций.



**Неизменяемость легковесов**

Так как объекты легковесов будут использованы в различных контекстах, вы должны быть уверены, что их состояние не меняется после создания. Все параметры должны быть получены через конструктор. А также не должно быть сеттеров и публичных полей.

**Фабрика легковесов**

Для удобства работы с легковесами и контекстами можно создать фабричный метод или фабрику, принимающие в параметрах всё внутреннее (а иногда и внешнее) состояние желаемого объекта.

Главная польза этого подхода в том, чтобы искать уже созданные легковесы с таким же внутренним состоянием, что и требуемое. Если легко находится, его можно повторно использовать. Если нет – создаем новый.

**Пример реализации**

using System**;**using System**.**Collections**.**Generic**;**namespace FlyweightPattern  
**{** *// Вспомогательный класс.  
 // Представим, что он хранит спрайт дерева.* public class Sprite **{ }** *// Это класс-легковес, который содержит поля,  
 // общие для нескольких деревьев.* public class TreeType  
 **{** *// Так как множество деревьев имеет  
 // одинаковые спрайты, то это поле  
 // выносим в легковес.* private Sprite \_sprite**;** *// Конструктор, принимающий внутреннее состояние.* public TreeType**(**Sprite sprite**)  
 {** \_sprite = sprite**;  
 }** public void Draw**(**int x**,** int y**)  
 {** *// Тут будет отрисовано дерево  
 // со спрайтом, сохраненным в \_sprite  
 // и в координатах x и y* **}  
 }**

*// Статический класс, создающий объекты-легковесы.* public static class TreeFactory  
 **{** *// Коллекция для хранения уже созданных  
 // объектов-легковесов.* private static Dictionary<Sprite**,** TreeType> \_treeTypes  
 = new Dictionary<Sprite**,** TreeType>**();** *// Метод получения нужного легковеса.* public static TreeType GetTreeType**(**Sprite sprite**)  
 {** *// Если такой легковес уже создавали - вернуть его* if **(**\_treeTypes**.**ContainsKey**(**sprite**))  
 {** Console**.**WriteLine**(**"Get from cache"**);** return \_treeTypes**[**sprite**];  
 }** else  
 **{** *// Если нужного легковеса нет - создать его  
 // и добавить в коллекцию* Console**.**WriteLine**(**"Created"**);** var treeType = new TreeType**(**sprite**);** \_treeTypes**[**sprite**]** = treeType**;** return treeType**;  
 }  
 }  
 }** *// Объект, который хранит изменяющееся состояние.  
 // Таких объектов может быть очень много, так как весят они мало:  
 // 2 числа и ссылка на легковес.* public class Tree  
 **{** *// Изменяющееся состояние* private int \_x**;** private int \_y**;** *// Ссылка на объект легковес* private TreeType \_treeType**;** *// Конструктор, который находит нужный объект легковес.* public Tree**(**int x**,** int y**,** Sprite sprite**)  
 {** \_x = x**;** \_y = y**;** \_treeType = TreeFactory**.**GetTreeType**(**sprite**);  
 }** *// Объект адрессует эту операцию легковесу* public void Draw**()  
 {** \_treeType**.**Draw**(**\_x**,** \_y**);  
 }  
 }**

internal static class Program  
 **{** public static void Main**(**string**[]** args**)  
 {** *// Три возможных спрайта для дерева* Sprite sprite1 = new Sprite**();** Sprite sprite2 = new Sprite**();** Sprite sprite3 = new Sprite**();** *// Создадим несколько деревьев с одним спрайтом* Tree tree1 = new Tree**(**1**,** 1**,** sprite1**);** Tree tree2 = new Tree**(**2**,** 2**,** sprite1**);  
 }  
 }  
}**

В результате запуска программы вывод будет следующий:

*Created*

*Get from cache*

**Применимость**

Используйте паттерн Компоновщик, если у вас есть большое количество объектов с повторяющимся состоянием, причем бо́льшую часть состояния объектов можно вынести за пределы их классов.

**Преимущества и недостатки**

**Преимущества:**

* Экономия памяти.

**Недостатки:**

* Усложнение кода программы из-за введения множества дополнительных классов.

**Отношения с другими паттернами**

* Компоновщик часто совмещают с Легковесом, чтобы реализовать общие ветки дерева и сэкономит при этом память;
* Легковес показывает, как создавать много мелких объектов, а Фасад показывает, как создать один объект, который отображает целую подсистему;
* Паттерн Легковес может напоминать Одиночку. Но между двумя этими паттернами есть кардинальные отличие:
  1. В отличие от Одиночки, вы можете иметь множество объектов-легковесов;
  2. Объекты-легковесы должны быть неизменяемые, тогда как объект-одиночка допускает изменение своего состояния.

**Задание**

**Требования к отчету**

Отчет к лабораторной работе должен содержать код работающей программы и результат выполнения.

**Вопросы**

1. Объясните целесообразность применения паттерна для решения задачи лабораторной работы.
2. Какие паттерны являются родственными паттерну *Легковес*?