**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра математического обеспечения и применения ЭВМ**

**Отчет**

По лабораторной работе № 2

На тему: «Шаблоны поведения»

По курсу: «Архитектура, технологии и ИС разработки ПО»

Вариант № 10

Выполнили:

студенты гр. КТбо3-6

Гришак В. С.

Проверил:

доцент кафедры МОП ЭВМ

Лутай В. Н.

Таганрог 2022

Цель работы: использовать необходимый паттерн проектирования при написании программы.

1. Вариант задания 10

В некоторой видеоигре участвуют 2 типа персонажей, которые по ходу сценария часто появляются и исчезают. Используйте Пул объектов для уменьшения количества генерации персонажей во время игры.

1. Спецификация классов с комментариями

Был создан интерфейс ICar предназначенный для обозначения автомобилей. Было написано две реализации – классы HotRod и SuperCar. Объекты данного класса способны двигаться с помощью метода Ride. У каждого автомобиля есть свой номер.

Для печати в консоль был создан класс Printer и интерфейс IPrinterUser для его использования.

Для хранения объектов автомобилей был разработан шаблонный класс Pool. Данный класс хранит ограниченное количество объектов заданного типа и выдаёт их по мере необходимости. При окончании использования использованный объект необходимо вернуть в пул. При необходимости создаются объекты заданного типа с помощью интерфейса ICreator. Объект реализующий данный интерфейс передают при создании пула.

1. Используемые математические зависимости и алгоритмы, например, вычисления дня недели по дате или алгоритма пересечения фигур.

Никакие математические зависимости не были применены.

1. Диаграмма классов предоставлена на рисунке 1.

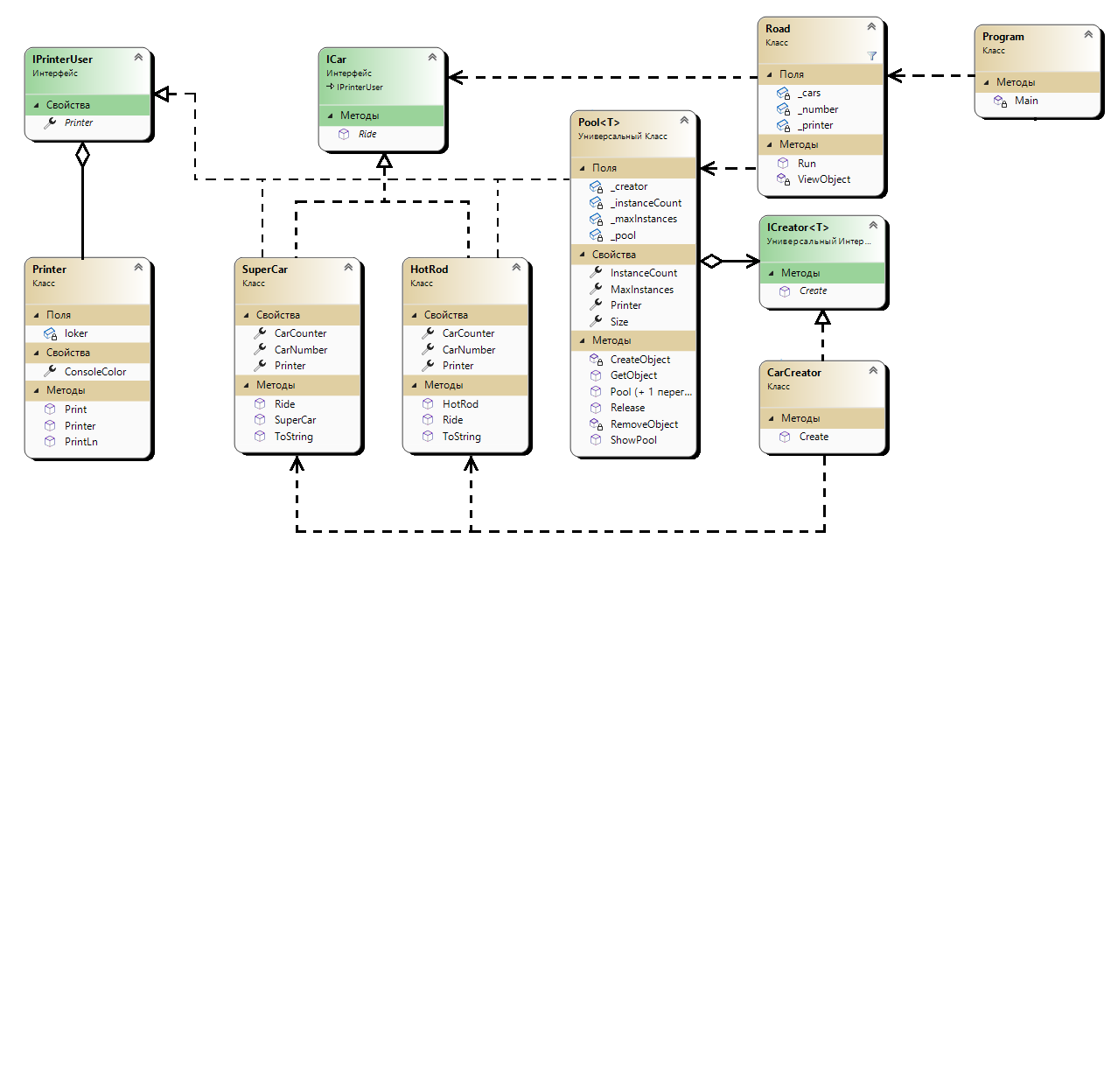


Рисунок 1 – Диаграмма классов

Вывод: в ходе выполнения лабораторной работы был успешно применён паттерн проектирования Пул объектов.

**ПРИЛОЖЕНИЕ**

Program.cs:

using patterns\_laba\_2.Cars;

using patterns\_laba\_2.Creation;

using patterns\_laba\_2.Pools;

using patterns\_laba\_2.Printers;

partial class Program

{

static void Main(string[] args)

{

var reusablePool = new Pool<ICar>(new CarCreator(), 20);

var thrd1 = new Thread(new Road(reusablePool, new Printer(ConsoleColor.Red)).Run);

var thrd2 = new Thread(new Road(reusablePool, new Printer(ConsoleColor.Blue)).Run);

thrd1.Start();

Thread.Sleep(2000);

thrd2.Start();

Console.ReadKey();

reusablePool.ShowPool();

}

}

Road.cs:

using patterns\_laba\_2.Cars;

using patterns\_laba\_2.Pools;

using patterns\_laba\_2.Printers;

public class Road

{

private Pool<ICar> \_cars;

private Printer \_printer;

private int \_number;

private static Random random = new Random();

public Road(Pool<ICar> cars, Printer printer)

{

\_cars = cars;

\_printer = printer;

\_number = ++Number;

}

public static int Number { get; private set; } = 0;

public static readonly int MAX\_CAR\_NAMBER = 10;

public void Run()

{

\_printer.Print($"Встречная полоса № {\_number}");

Queue<ICar?> cars = new Queue<ICar?>(MAX\_CAR\_NAMBER);

Queue<ICar> usedCars = new Queue<ICar>(MAX\_CAR\_NAMBER);

for (int i = 0; i < 5; i++)

{

int neededCarNum = random.Next(1, MAX\_CAR\_NAMBER);

for (int j = 0; j < neededCarNum; j++)

{

cars.Enqueue(\_cars.GetObject());

}

for (int j = 0; j < neededCarNum; j++)

{

ICar? car = cars.Dequeue();

if (car != null)

{

Thread.Sleep(500);

ViewObject(car);

usedCars.Enqueue(car);

}

}

while (usedCars.Count > 0)

{

\_cars.Release(usedCars.Dequeue());

Thread.Sleep(random.Next(900, 1500));

}

}

}

private void ViewObject(ICar car)

{

car.Printer = \_printer;

car.Ride();

}

}

HotRod.cs:

using patterns\_laba\_2.Printers;

namespace patterns\_laba\_2.Cars;

/// <summary>

/// Класс автомобиля Хот род, реализует интерфейс <see cref="ICar"/>

/// </summary>

public class HotRod : ICar, IPrinterUser

{

/// <summary>

/// Номер машины

/// </summary>

public int CarNumber { get; private set; }

/// <summary>

/// Количество машин данного класса

/// </summary>

public static int CarCounter { get; private set; } = 0;

/// <summary>

/// Объект класса <see cref="Printer"/> предназначен для вывода информации на консоль

/// </summary>

public Printer? Printer { get; set; }

public HotRod()

{

CarNumber = ++CarCounter;

}

/// <summary>

/// Метод для демонстрации программы

/// </summary>

public void Ride()

{

Printer?.PrintLn($"Дрын-дын-дын... проехал Хот род {CarNumber}");

}

public override string ToString() => $"Хот род - {CarNumber}";

}

ICar.cs:

using patterns\_laba\_2.Printers;

namespace patterns\_laba\_2.Cars;

/// <summary>

/// Интерфейс для обобщения видов автомобилей

/// <list type="bullet">

/// <item><see cref="HotRod"/></item>

/// <item><see cref="SuperCar"/></item>

/// </list>

/// </summary>

public interface ICar : IPrinterUser

{

/// <summary>

/// Метод для запуска

/// </summary>

public void Ride();

}

SuperCar.cs:

using patterns\_laba\_2.Printers;

namespace patterns\_laba\_2.Cars;

/// <summary>

/// Класс автомобиля Хот род, реализует интерфейс <see cref="ICar"/>

/// </summary>

public class SuperCar : ICar, IPrinterUser

{

/// <summary>

/// Номер машины

/// </summary>

public int CarNumber { get; private set; }

/// <summary>

/// Количество машин данного класса

/// </summary>

public static int CarCounter { get; private set; } = 0;

/// <summary>

/// Объект класса <see cref="Printer"/> предназначен для вывода информации на консоль

/// </summary>

public Printer? Printer { get; set; }

public SuperCar()

{

CarNumber = ++CarCounter;

}

/// <summary>

/// Метод для демонстрации программы

/// </summary>

public void Ride()

{

Printer?.PrintLn($"вжух... пролетел Суперкар {CarNumber}");

}

public override string ToString() => $"Суперкар - {CarNumber}";

}

CarCreator.cs:

using patterns\_laba\_2.Cars;

namespace patterns\_laba\_2.Creation;

/// <summary>

/// Класс предназначенный для создания объектов, реализующих интерфейс <see cref="ICar"/>

/// </summary>

public class CarCreator : ICreator<ICar>

{

/// <summary>

/// Возвращает объект, реализующих интерфейс <see cref="ICar"/>

/// </summary>

public ICar Create()

{

if (HotRod.CarCounter < SuperCar.CarCounter)

{

return new HotRod();

}

else

{

return new SuperCar();

}

}

}

ICreator.cs:

namespace patterns\_laba\_2.Creation;

/// <summary>

/// Интерфейс для использования шаблона "Object Pool" <see cref="ObjectPool"/>

/// </summary>

/// <typeparam name="T"></typeparam>

public interface ICreator<T>

{

/// <summary>

/// Возвращает вновь созданный объект

/// </summary>

T Create();

}

Pool.cs:

using patterns\_laba\_2.Creation;

using patterns\_laba\_2.Printers;

namespace patterns\_laba\_2.Pools;

/// <summary>

/// Реализация пула объектов

/// </summary>

/// <typeparam name="T"></typeparam>

public class Pool<T> : IPrinterUser where T : class

{

/// <summary>

/// Коллекция содержит управляемые объекты

/// </summary>

private List<T> \_pool;

/// <summary>

/// Ссылка на объект, которому делегируется ответственность

/// за создание объектов пула

/// </summary>

private ICreator<T> \_creator;

/// <summary>

/// Количество объектов, существующих в данный момент

/// </summary>

private int \_instanceCount;

/// <summary>

/// Максимальное количество управляемых пулом объектов

/// </summary>

private int \_maxInstances;

/// <summary>

/// Объект класса <see cref="Printer"/> предназначен для вывода информации на консоль

/// </summary>

public Printer? Printer { set; get; }

/// <summary>

/// Создание пула объектов

/// </summary>

/// <param name="creator">Объект, которому пул делегирует ответственность

/// за создание управляемых им объектов</param>

public Pool(ICreator<T> creator) : this(creator, int.MaxValue) { }

/// <summary>

/// Создание пула объектов

/// </summary>

/// <param name="creator">Объект, которому пул делегирует ответственность

/// за создание управляемых им объектов</param>

/// <param name="maxInstances">Максимальное количество экземпляров классов,

/// которым пул разрешает существовать одновременно

/// </param>

public Pool(ICreator<T> creator, int maxInstances)

{

\_creator = creator;

\_instanceCount = 0;

\_maxInstances = maxInstances;

\_pool = new List<T>();

Printer = new Printer(ConsoleColor.White);

}

/// <summary>

/// Возвращает количество объектов в пуле, ожидающих повторного

/// использования.

/// </summary>

public int Size

{

get

{

lock (\_pool)

{

return \_pool.Count;

}

}

}

/// <summary>

/// Возвращает количество управляемых пулом объектов,

/// существующих в данный момент

/// </summary>

public int InstanceCount => \_instanceCount;

/// <summary>

/// Получить или задать максимальное количество управляемых пулом

/// объектов, которым пул разрешает существовать одновременно.

/// </summary>

public int MaxInstances

{

get => \_maxInstances;

set

{

if (\_instanceCount <= value)

{

\_maxInstances = value;

}

}

}

/// <summary>

/// Возвращает из пула объект. При пустом пуле будет создан

/// объект, если количество управляемых пулом объектов не

/// больше или равно значению, возвращаемому методом

/// <see cref="ObjectPool{T}.MaxInstances"/>. Если количество управляемых пулом

/// объектов превышает это значение, то данный метод возварщает null

/// </summary>

/// <returns></returns>

public T? GetObject()

{

lock (\_pool)

{

// пытаемся получить объект из пула

T? thisObject = RemoveObject();

if (thisObject != null)

return thisObject;

// если возможно, то создаём объект

if (InstanceCount < MaxInstances)

return CreateObject();

// в противном случае

return null;

}

}

/// <summary>

/// Удаляет объект из коллекции пула и возвращает его

/// </summary>

private T? RemoveObject()

{

while (\_pool.Count > 0)

{

// берём последний с конца объект

T thisObject = \_pool[^1];

\_pool.RemoveAt(\_pool.Count - 1);

// уменьшаем количство доступных объектов

\_instanceCount--;

// возвращаем если это объект

if (thisObject != null)

return thisObject;

} // до тех пор пока не найдём объект или не закончиться пул

return null;

}

/// <summary>

/// Создать объект, управляемый этим пулом

/// </summary>

private T CreateObject()

{

T newObject = \_creator.Create();

Printer?.Print($"создан объект {newObject}");

\_instanceCount++;

return newObject;

}

/// <summary>

/// Освобождает объект, помещая его в пул для

/// повторного использования

/// </summary>

/// <param name="obj"></param>

/// <exception cref="NullReferenceException"></exception>

public void Release(T obj)

{

if (obj == null)

throw new NullReferenceException();

lock (\_pool)

{

// увеличиваем счётчик объектов

\_instanceCount++;

Printer?.Print($"объект {obj} вернулся в пул");

// возвращаем объект в пул

\_pool.Add(obj);

}

}

/// <summary>

/// Вывод всего содержимого пула на данный момент

/// </summary>

public void ShowPool()

{

Printer?.PrintLn("");

foreach (var obj in \_pool)

{

Printer?.Print($"объект {obj} в пуле");

}

}

}

IPrinterUser.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace patterns\_laba\_2.Printers;

/// <summary>

/// Интерфейс предназначен для пометки классов,

/// которым необходимо выводить информацию на консоль

/// </summary>

public interface IPrinterUser

{

/// <summary>

/// Объект класса <see cref="Printer"/> предназначен для вывода информации на консоль

/// </summary>

public Printer? Printer { set; get; }

}

Printer.cs:

namespace patterns\_laba\_2.Printers;

/// <summary>

/// Объекты данного класса предназначены для цветного вывода информации в консоль

/// </summary>

public class Printer

{

/// <summary>

/// Цвет которым будет печататься информация

/// </summary>

public ConsoleColor ConsoleColor { get; set; }

/// <summary>

/// переменная заглушка для захвата контроля потока

/// </summary>

private static readonly object loker = new object();

/// <summary>

/// Создание обекта принтера

/// </summary>

/// <param name="consoleColor">Цвет которым будет печататься информация</param>

public Printer(ConsoleColor consoleColor)

{

ConsoleColor = consoleColor;

}

/// <summary>

/// Печать строки в консоль установленным в переменную <see cref="ConsoleColor"/> цветом

/// </summary>

/// <param name="str">строка для вывода на консоль</param>

public void Print(string str)

{

lock (loker)

{

Console.ForegroundColor = ConsoleColor;

Console.WriteLine(str);

}

}

/// <summary>

/// Печать строки в консоль установленным в переменную <see cref="ConsoleColor"/>

/// цветом и постановка пустой строки после

/// </summary>

/// <param name="str">строка для вывода на консоль</param>

public void PrintLn(string str)

{

lock (loker)

{

Console.ForegroundColor = ConsoleColor;

Console.WriteLine(str);

Console.WriteLine();

}

}

}