**Цель работы:** научиться применять порождающие шаблоны для создания программной системы.

**Вариант 25**

**Выполнение заданий:**

1. Нарисовать в диаграмму классов реализуемой программы (в зависимости от варианта либо выбрав проект и подходящий шаблон самостоятельно).

2. Реализовать программу на любом языке программирования с использованием порождающего шаблона проектирования (в зависимости от варианта либо выбрав проект и подходящий шаблон самостоятельно).

**Вариант №5, 10, 15, 20, 25, 30**

Шаблон Factory Method. Фигуры игры «тетрис». Реализовать процесс случайного выбора фигуры из конечного набора фигур. Предусмотреть появление супер-фигур с большим число клеток, чем обычные.

using System;

namespace MultySidedSolution

{

abstract class Figure

{

public abstract String Name { get; }

}

class O : Figure

{

String name = "O";

public override String Name { get { return name; } }

}

class I : Figure

{

String name = "T";

public override String Name { get { return name; } }

}

class S : Figure

{

String name = "S";

public override String Name { get { return name; } }

}

class Z : Figure

{

String name = "Z";

public override String Name { get { return name; } }

}

class L : Figure

{

String name = "L";

public override String Name { get { return name; } }

}

class J : Figure

{

String name = "J";

public override String Name { get { return name; } }

}

class T : Figure

{

String name = "T";

public override String Name { get { return name; } }

}

abstract class FigureCreator

{

public abstract Figure FactoryMethod();

}

class OCreator : FigureCreator

{

public override Figure FactoryMethod()

{

return new O();

}

}

class ICreator : FigureCreator

{

public override Figure FactoryMethod()

{

return new I();

}

}

class SCreator : FigureCreator

{

public override Figure FactoryMethod()

{

return new S();

}

}

class ZCreator : FigureCreator

{

public override Figure FactoryMethod()

{

return new Z();

}

}

class LCreator : FigureCreator

{

public override Figure FactoryMethod()

{

return new L();

}

}

class JCreator : FigureCreator

{

public override Figure FactoryMethod()

{

return new J();

}

}

class TCreator : FigureCreator

{

public override Figure FactoryMethod()

{

return new T();

}

}

class Program

{

static void Main(string[] args)

{

var rand = new Random();

FigureCreator[] creators = {

new OCreator(),

new ICreator(),

new SCreator(),

new ZCreator(),

new LCreator(),

new JCreator(),

new TCreator()

};

Console.WriteLine("[10 Рандомных фигур]");

for (int i = 0; i < 9; i++)

{

var figure = creators[rand.Next(0, creators.Length)].FactoryMethod();

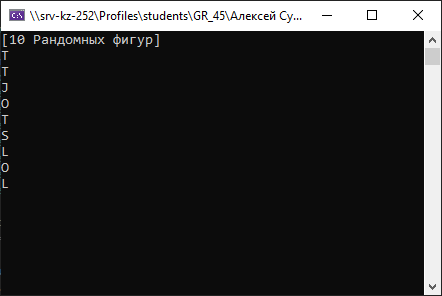
Console.WriteLine(figure.Name);

}

}

}

}



[Ссылка на файл](MultySidedSolution/bin/Debug/netcoreapp3.1/MultySidedSolution.exe)

*Ответы на контрольные вопросы:*

1. Каково назначение шаблона Строитель?

Ответ: Реализовать общий интерфейс для разделения создания объекта на различные этапы.

1. Какая проблема решается с помощью шаблона Строитель?

Ответ: Он решает проблему декомпозиции создания объектов на отдельные этапы объединенные в общем интерфейсе.

1. В чём заключается решение, предлагаемое в шаблоне Строитель?

Ответ: Создать общий интерфейс с этапами создания объектов, создать класс строитель, который вызывает методы класса.

1. Для чего предназначен шаблон Абстрактная фабрика?

Ответ: Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов.

1. Какое решение предлагается в шаблоне Абстрактная фабрика?

Ответ: создаём абстрактную фабрику — общий интерфейс, который содержит методы создания всех продуктов семейства. Эти операции должны возвращать абстрактные типы продуктов, представленные интерфейсами

1. Какую проблему позволяет решить шаблон Фабричный метод?

Ответ: позволяет обеспечить общий интерфейс для общего семейства объектов.