**Лабораторная работа 5. Часть 2**

Состояние – поведенческий шаблон проектирования. Используется в тех случаях, когда во время выполнения программы объект должен менять своё поведение в зависимости от своего состояния. Классическая реализация предполагает создание базового абстрактного класса или интерфейса, содержащего все методы и по одному классу на каждое возможно состояние (рис. 1). Шаблон представляет собой частный случай рекомендации «заменяйте условные операторы полиморфизмом».

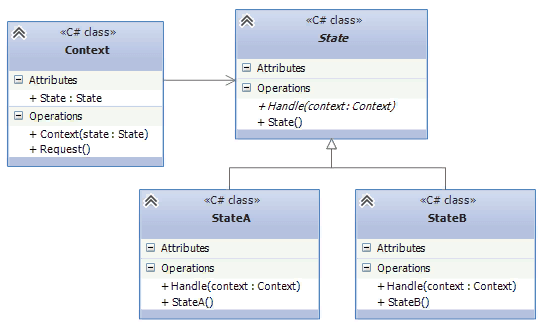


Рисунок 1. Шаблон проектирования Состояние

**Описание кода программы**

Сейчас вся логика управления героем сосредоточена в классе **Cockroach**. Основные методы передвижения Step и поворота ChangeTrend содержат «неприятные» операторы выбора. Шаблон проектирования Состояние позволит избавиться от них. Состояния объектов класса Cockroach очевидны: таракан смотрит налево, направо, вверх и вниз. В зависимости от этого состояния таракан передвигается по-разному, увеличивая или уменьшая координату X или Y, а так же по-разному переходит в другое свое состояние. Значит, нужно описать интерфейс и четыре класса состояний.

Создайте папку State и в ней интерфейс IDirection со следующим набором методов:

void Step(ref int X, ref int Y);

IDirection ChangeTrend(string command);

Direction Trend { get; }

Метод Step будет изменять одну из координат X или Y, в зависимости от направления. Метод ChangeTrend будет изменять состояние и направление таракана, в зависимости от команды направления. В метод передается не символ, а строка, то есть название всей команды. Свойство Trend будет возвращать текущее значение направления.

Подробно рассмотрим описание только одно состояние. Остальные состояния нужно описать самостоятельно. Добавьте в папку State класс DirectionUp. Он должен реализовать интерфейс IDirection. Приступим к перегрузке методов. Свойство Trend возвращает направление вверх:

public Direction Trend => Direction.Up;

Метод Step должен изменять только одну координату

Y -= step;

Очевидно, что в классе не хватает описания поля step. Перенесите его из класса Cockroach.

Метод ChangeTrend должен взять строку и преобразовать ее в направление. Для этого можно использовать следующую структуру:

protected Dictionary<string, Direction> Trends = new Dictionary<string, Direction>()

{

{"Up", Direction.Up },

{"Down", Direction.Down },

{"Left", Direction.Left },

{"Right", Direction.Right }

};

Получим направление:

Direction newtrend = Trends[command];

А затем, в зависимости от направления, заданного командой, поверните изображение и создайте объект нового состояния

switch (newtrend)

{

case Direction.Right:

image.RotateFlip(RotateFlipType.Rotate90FlipNone);

return new DirectionRight(image);

case Direction.Down:

image.RotateFlip(RotateFlipType.Rotate180FlipNone);

return new DirectionDown(image);

case Direction.Left:

image.RotateFlip(RotateFlipType.Rotate270FlipNone);

return new DirectionLeft(image);

default: return new DirectionUp(image);

}

В этом классе не хватает описания поля image и конструктора:

public DirectionUp(Bitmap image)

{

this.image = image;

}

Опишите остальные классы: DirectionDown, DirectionLeft, DirectionRight.

Все классы будут иметь одинаковый набор описания полей. Можно вынести его в отдельный класс. Создайте класс DirectionState и перенесите в него все поля классов направлений:

protected Dictionary<string, Direction> Trends = new Dictionary<string, Direction>()

{

{"Up", Direction.Up },

{"Down", Direction.Down },

{"Left", Direction.Left },

{"Right", Direction.Right }

};

protected const int step = 30;

protected Bitmap image;

Сделайте данный класс родительским для всех классов состояний.

Перейдите в класс Cockroach.

Опишите в нем поле, отвечающее за состояние:

IDirection direction;

Задайте ему начальное значение в конструкторе класса:

public Executor(Bitmap \_image)

{

image = \_image;

direction = new DirectionUp(image);

}

Сейчас в методе Step достаточно только одной строки:

direction.Step(ref x, ref y);

А в методе ChangeTrend:

direction = direction.ChangeTrend(command);

Таким образом, смена состояний «легла на плечи» самих состояний.

**Запуск**. Запустите приложение и убедитесь в его работоспособности. В данном решении используется интерфейс и родительский класс. Как можно их объединить? Можете заменить в своем решении эти две конструкции одной.