ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра ПИ

Факультет КНТ

Лабораторная работа №6

Тема: «Формально е оценивание качества проектирования программных систем»

Выполнил

ст. гр. ПИ-18Б

Моргунов А.Г.

Проверил

асс. каф. ПИ

Артеменко О.Г.

асс. каф. ПИ

Ищенко А.П.

Донецк – 2020

**Задание к лабораторной работе**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Название (имя) паттерна проектирования | Условия применения паттерна  (назначение паттерна) | Пример программы, на которую может настраиваться данный паттерн |
| 16 | Состояние (State)  Паттерн поведения | Паттерн позволяет объекту изменять своё состояние в зависимости от внутреннего состояния. Извне создаётся впечатление, что объект изменил свой класс.  Паттерн State является объектно-ориентированной реализацией конечного автомата.  Паттерн Интерпретатор может использовать паттерн Состояние для определения контекстов при синтаксическом разборе.  Паттерн применяется, когда в коде операции встречаются состоящие из многих ветвей условные операторы, в которых выбор ветви зависит от состояния. Обычно в таком случае состояние5 представлено перечислимыми константами. Часто одна и та же структура условного оператора повторяется в нескольких операциях. Паттерн Состояние предлагает поместить каждую ветвь в отдельный класс. Это позволяет трактовать состояние объекта как самостоятельный объект, который может изменяться независимо от других. | Моделирование работы торгового автомата. Автоматы могут иметь различные состояния в зависимости от наличия товаров, суммы полученных монет, возможности размена денег и т.д. После того, как покупатель выбрал и оплатил товар, возможны следующие ситуации (состояния):   * выдать покупателю товар, выдавать сдачу не требуется; * выдать покупателю товар и сдачу; * покупатель товар не получит из-за отсутствия достаточной суммы денег; * покупатель товар не получит из-за его отсутствия.   Другой пример, есть класс ООП, реализующий сетевое соединение по протоколу TCP. Объект этого класса может находиться в одном из нескольких состояний: установлено, прослушивание, закрыто. Когда объект получает запросы от других объектов, то в зависимости от текущего состояния он должен отвечать по-разному.  В графическом редакторе для реализации его поведения в зависимости от выбранного текущего инструмента можно воспользоваться паттерном Состояние. |

**Состояние** — это поведенческий паттерн проектирования, который позволяет объектам менять поведение в зависимости от своего состояния. Извне создаётся впечатление, что изменился класс объекта.

Основная идея в том, что программа может находиться в одном из нескольких состояний, которые всё время сменяют друг друга. Набор этих состояний, а также переходов между ними, предопределён и конечен. Находясь в разных состояниях, программа может по-разному реагировать на одни и те же события, которые происходят с ней.

**Предметная ориентация приложения.** В исследовательском центре физических явлений проводятся эксперименты по изучению агрегатных состояний води и переходов между ними с помощью симуляции. В симуляции есть две команды, которые изменяют температуру воды. В зависимости от агрегатного состояния, в котором находится вода при вызове изменения температуры, изменяется и результат выполненной операции.

Структура паттерна в UML (рис. 1)

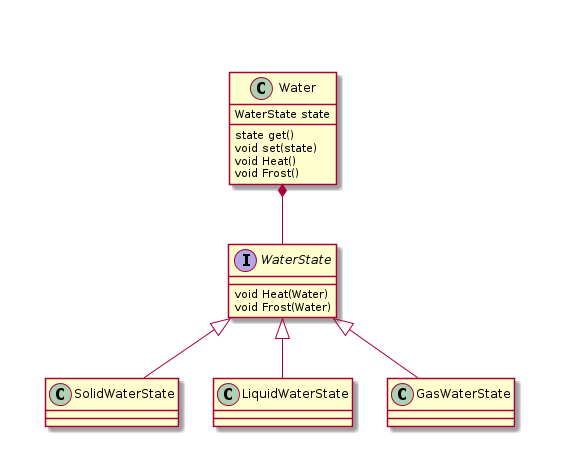


Рисунок 1 – Структура паттерна Состояние

Результаты выполнения программы (рис. 2).

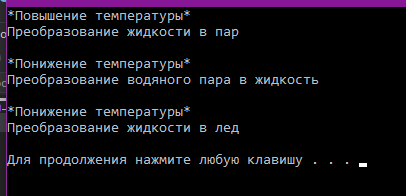


Рисунок 2 –Результат программы

Реализация паттерна по шагам:

1. Создается объект класса Вода с произвольным состоянием виде.
2. Пользователь инициирует изменение температуры в симуляции.
3. В зависимости от текущего состояния выполняется изменение агрегатного состояния воды.

Листинг программы:

using System;

namespace AiPPO\_lab\_6

{

class Program

{

static void Main(string[] args)

{

Water water = new Water(new LiquidWaterState());

water.Heat();

water.Frost();

water.Frost();

}

}

class Water

{

public IWaterState State { get; set; }

public Water(IWaterState ws)

{

State = ws;

}

public void Heat()

{

Console.WriteLine("\*Повышение температуры\*");

State.Heat(this);

Console.WriteLine();

}

public void Frost()

{

Console.WriteLine("\*Понижение температуры\*");

State.Frost(this);

Console.WriteLine();

}

}

interface IWaterState

{

void Heat(Water water);

void Frost(Water water);

}

class SolidWaterState : IWaterState

{

public void Heat(Water water)

{

Console.WriteLine("Преобразование льда в жидкость");

water.State = new LiquidWaterState();

}

public void Frost(Water water)

{

Console.WriteLine("Лед охладился");

}

}

class LiquidWaterState : IWaterState

{

public void Heat(Water water)

{

Console.WriteLine("Преобразование жидкости в пар");

water.State = new GasWaterState();

}

public void Frost(Water water)

{

Console.WriteLine("Преобразование жидкости в лед");

water.State = new SolidWaterState();

}

}

class GasWaterState : IWaterState

{

public void Heat(Water water)

{

Console.WriteLine("Пар нагревается");

}

public void Frost(Water water)

{

Console.WriteLine("Преобразование водяного пара в жидкость");

water.State = new LiquidWaterState();

}

}

}