УДК 004.021

ПРИМЕНЕНИЕ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ ДЛЯ АНАЛИЗА УСПЕВАЕМОСТИ СТУДЕНТОВ

Власова В.Д., Бужинская Н.В.

Нижнетагильский государственный социально-педагогический институт (филиал) ФГАОУ ВО «Российский государственный профессионально-педагогический университет», Нижний Тагил, e-mail: Neina1@yandex.ru

В статье рассматриваются возможности шаблонов проектирования для решения различных задач, перечислены их типы. Паттерн представляет собой готовый фрагмент кода (шаблон), который применяется для решения задач определенных типов. В зависимости от типа задачи определяется тип паттерна. В области программирования существуют разные мнения о целесообразности применения паттернов. Паттерн имеет следующую структуру: задачу, которая требует решения, описание способа решения, совокупность применяемых классов, примеры решения. Паттерны реализуются на основе классов и методов. Следовательно, применение паттернов требует хороших знаний в области парадигмы объектно-ориентированного программирования, умений работать с классами и объектами. С одной стороны, готовые конструкции для решения задач программирования приводят к стандартизации кода и упрощают его понимание другими разработчиками, а с другой – теряется творческая составляющая процесса решения задачи. В данной статье рассмотрена возможность использования высокоуровневого языка программирования C# для разработки паттерна «Декоратор». Данный паттерн реализуется за счет большого числа классов и поэтому достаточно простой в понимании. Данный паттерн был реализован для анализа успеваемости студентов вуза.

Ключевые слова: проектирование, программирование, программа, паттерн, код, C#

APPLICATION DESIGN PATTERNS FOR THE ANALYSIS OF STUDENT PERFORMANCE

Vlasova V.D., Buzhinskaya N.V.

Nizhny Tagil State Socio-Pedagogical Institute (branch) FSAEI of HE «Russian State Vocational Pedagogical University», Nizhny Tagil, e-mail: Neina1@yandex.ru

The article discusses the possibilities of design patterns for solving various problems, lists their types. A pattern is a ready-made code fragment (template) that is used to solve certain types of problems. Depending on the task type, the pattern type is determined. In the field of programming, there are different opinions about the appropriateness of using patterns. The pattern has the following structure: a problem that requires a solution, a description of the solution method, a set of classes used, examples of solutions. Patterns are implemented based on classes and methods. Therefore, the use of patterns requires a good knowledge of the paradigm of object-oriented programming, skills to work with classes and objects. On the one hand, ready – made designs for solving programming problems lead to the standardization of code and simplifies its understanding by other developers, and on the other hand, the creative component of the problem solving process is lost. This article discusses the possibility of using a high-level C# programming language for the development of the «Decorator»pattern. This pattern is implemented due to a large number of classes and is therefore quite easy to understand. This pattern was implemented to analyze the performance of University students.

Keywords: design, programming, program, pattern, code, C#

Высокая конкуренция в IT-сфере диктует свои правила поведения – современный программист должен хорошо ориентироваться в нескольких парадигмах программирования, грамотно составить техническое задание, уметь презентовать результаты своей работы. В свою очередь, усложнение технологий программирования порождает тенденцию к унификации программного кода, без потери его качества. На реализацию данной цели направлены паттерны проектирования, которые являются сравнительно новыми в программировании. В настоящее время заговорили о становлении «паттернового подхода», так как число паттернов постоянно увеличивается [1].

Рассмотрим особенности применения паттернов для обработки данных образовательного процесса. Нашей задачей является разработка программы для выставления оценки студентам по трем предметам: «Высокоуровневые методы информатики и программирования» (далее ВУИ), «Проектирование» и «Программная инженерия». В качестве входных параметров будем использовать следующие критерии оценок:

– выполнил все задания;

– допустил одну ошибку;

Типы паттернов проектирования

Название группы паттернов	Основная функция	Пример
Порождающие паттерны	Функция создания объектов, позволяют системе быть независимой от типов этих объектов и от процесса порождения	Фабричный метод, абстрактная фабрика, строитель, прототип, одиночка
Структурные паттерны	Функция установления связей между объектами	Адаптер, компоновщик, мост, декоратор, фасад, легковес, заместитель
Поведенческие паттерны	Функция коммуникации между объектами	Цепочка обязанностей, команда, состояние, снимок, итератор, посредник, наблюдатель, стратегия, шаблонный метод, посетитель

– не выполнил одно задание;

– не выполнил все задания.

Например, данный список можно оформить в следующем виде:

– экзамен по ВУИ (все верно) – оценка «5»;

– экзамен по проектированию (все верно) – оценка «5»;

– экзамен по инженерии (все верно) – оценка «5»;

– экзамен по ВУИ (допустил одну ошибку) – оценка «4» и т.д.

Для реализации данной цели нам необходимо определить тип паттерна, выбрать язык программирования для его реализации и протестировать разработанный продукт.

Материалы и методы исследования

Выберем тип паттерна. Паттерн (шаблон) проектирования – это уже встречавшаяся архитектура программы, которая служит для решения определенной задачи, то есть это шаблон кода, который достаточно взять и переделать для решения своей задачи [2]. Подобная двусмысленность приводит к тому, что эти шаблоны называют как паттернами проектирования, так и паттернами программирования.

К достоинствам применения таких конструкций – шаблонов можно отнести возможность сэкономить время для решения однотипных задач, большая «прозрачность» кода и тем самым лучшее его понимание другими разработчиками, стандартизация кода.

Но паттерны не лишены и недостатков. Во-первых, это сложности с определением области применения того или иного паттерна. Количество паттернов и их сложность приводит к тому, что программисту приходится потратить большее количество времени на их изучение, чем решение задачи с нуля. Во-вторых, программирования – это область для творчества, поскольку многие задачи нельзя решить стандартными способами, поэтому паттерны могут усложнить данный процесс. В-третьих – применение шаблонов, стандартов, готовых планов приводит к некой зависимости от их возможностей. Программист начинает надеяться, что с их помощью он может решить любую задачу, что абсолютно неверно.

Как правило, паттерн состоит из нескольких формальных частей [3].

– задачи, которую требуется рассмотреть;

– способа решения проблемы;

– структуры классов для решения;

– несколько примеров программного кода;

– связи с другими паттернами.

Следовательно, существует несколько видов паттернов, которые представлены в таблице [4].

Таким образом, каждый паттерн применяется для решения задач в определенной области. Мы остановим свой выбор на паттерне «Декоратор». Этот паттерн применяется для динамического изменения объекта. Кроме того, он позволяет представить решение задачи в виде микроклассов, что облегчает работу с кодом.

Программная реализация

Для реализации паттерна будем использовать высокоуровневый язык программирования C# [5, 6].

Упростим взаимодействие между объектами с помощью паттерна «Декоратор». Он позволит динамически в ходе написания кода расширять функционал объекта, назначая дополнительные возможности.

Создадим базовый класс «Экзамен», в котором зададим начальные условия. Изначально объявим оценку «2» для каждого предмета.

Листинг 1

```
namespace Декоратор
{
   public abstract class Экзамен
   {
      public string Дисциплина { get; protected set; }
      public int Оценка { get; protected set; }
      public Экзамен(string дисциплина)
      {
         if (string.IsNullOrEmpty(дисциплина))
         {
            throw new ArgumentNullException(nameof(дисциплина));
         }
         Дисциплина = дисциплина;
         Оценка = 2;
      }
      public override string ToString()
      {
         return $ «Оценка по предмету для {Дисциплина}»;
      }
   }
}
```

Теперь создадим три микрокласса «ВУИ», «Программная инженерия» и «Проектирование». Зададим в них изменение оценки до пяти.

Листинг 2

```
namespace Декоратор
{
   public class ВУИ : Экзамен
   {
      public ВУИ(string дисциплина): base(дисциплина)
      {
         Оценка += 3;
      }
   }
}
```

Микроклассы «Программная игнженерия» и «Проектирование» создаются аналогично.

Далее объявим класс «Снижение_оценки». Следом за ним опять создадим микроклассы «_ВУИ», «_Программная инженерия» и «_Проектирование». При их использовании оценка будет снижаться на единицу.

Листинг 3

```
namespace Декоратор
{
   public abstract class Снижение_оценки : Экзамен
   {
      protected Экзамен Итог { get; set; }
      public Снижение_оценки(Экзамен итог): base(итог.Дисциплина)
      {
         Итог = итог ?? throw new ArgumentNullException(nameof(итог));
         Оценка = итог.Оценка;
      }
   }
```

Листинг 4

```
namespace Декоратор
{
   public class _ВУИ : Снижение_оценки
   {
      public _ВУИ(Экзамен итог) : base(итог)
      {
         Оценка -= 1;
      }
   }
}
```

Введем перечисления для объявления критериев оценки. Преобразуем список в нужный вид.

Листинг 5

```
using System.ComponentModel;
namespace Декоратор
{
   public enum Перечисления : int
   {
      [Description(«Экзамен по ВУИ»)]
      vse_vui =1,
      [Description(«Экзамен по проектированию»)]
      vse_proekt = 2,
      [Description(«Экзамен по программной инженерии»)]
      vse_ing = 3,
   }
}
```

Листинг 6

```
using System.ComponentModel;
namespace Декоратор
{
   public enum Перечисления2 : int
   {
      [Description(«Допустил ошибку»)]
      ne_vui=1,
      [Description(«Не сделал одно задание»)]
      ne_proekt = 2,
      [Description(«Ничего не сделал»)]
      ne_ing = 3,
      [Description(«Все верно»)]
      None = 0
   }
}
```

В листинге 7 представлен код класса Class1 для выравнивания строчек результата программы для придания читабельного вида.

Листинг 7

```
using System.ComponentModel;
using System.Reflection;
namespace Декоратор
{
   public static class Class1
   {
      public static string GetDescription(this Enum enumElement)
      {
         Type type = enumElement.GetType();

         MemberInfo[] memInfo = type.GetMember(enumElement.ToString());
         if (memInfo != null && memInfo.Length > 0)
         {
               object[] attrs = memInfo[0].GetCustomAttributes(typeof(DescriptionAttribute),
false);
            if (attrs != null && attrs.Length > 0)
               return ((DescriptionAttribute)attrs[0]).Description;
         }
         return enumElement.ToString();
      }
   }
}
```

В основном теле программы запишем алгоритм решения поставленной задачи. Зададим запрос имени студента и оформим классы в порядке их использования.

Листинг 8

```
namespace Декоратор
{
   class Program
   {
      static void Main(string[] args)
      {
          Console.WriteLine(«Здравствуйте. Для того, чтобы узнать вашу оценку по экза-
мену, вам нужно вписать имя.»);
         Console.WriteLine(«Ваше имя: «);
         var имя = Console.ReadLine();
         Console.WriteLine(«Смотрите ниже»);
         Экзамен итог = null;
         do
         {
            итог = GetПеречисления(имя);
         }
         while (итог == null);
         while (true)
         {
            итог = GetПеречисления2(итог, out bool finish);
            if (finish)
            {
               break;
            }
         }
         Console.WriteLine($»{итог}. Ваша оценка: {итог.Оценка}»);
         Console.ReadLine();
      }
      private static Экзамен GetПеречисления2(Экзамен итог, out bool finish)
      {
         finish = false;
         Console.WriteLine(«Проверим правильность заданий:»);
         foreach (Перечисления2 t in Enum.GetValues(typeof(Перечисления2)))
         {
            Console.WriteLine($»{(int)t} - {t.GetDescription()}»);
         }
         var adding = Console.ReadLine();
         if (int.TryParse(adding, out int перечисления2))
         {
            switch ((Перечисления2)перечисления2)
            {
               case Перечисления2.None:
                  finish = true;
                  return итог;
               case Перечисления2.ne_vui:
                  return new _ВУИ(итог);
               case Перечисления2.ne_proekt:
                  return new _Проектирование(итог);
               case Перечисления2.ne_ing:
                  return new _Инженерия(итог);
               default:
                  Console.WriteLine(«Вы ввели некорректное значение!»);
                  return итог;
            }
         }
         else
         {
            Console.WriteLine(«Вы ввели некорректное значение!»);
            return итог;
         }
      }
```

```
        private static Экзамен GetПеречисления(string имя)
        {
            Console.WriteLine(«Выберите, пожалуйста, предмет:»);
            foreach (Перечисления t in Enum.GetValues(typeof(Перечисления)))
            {
                Console.WriteLine($»{(int)t} - {t.GetDescription()}»);
            }
            var type = Console.ReadLine();
            if (int.TryParse(type, out int перечисления))
            {
                switch ((Перечисления)перечисления)
                {
                    case Перечисления.vse_vui:
                        return new ВУИ(имя);
                    case Перечисления.vse_proekt:
                        return new Проектирование(имя);
                    case Перечисления.vse_ing:
                        return new Инженерия(имя);
                    default:
                        Console.WriteLine(«Вы ввели некорректное значение!»);
                        return null;
                }
            }
            else
            {
                Console.WriteLine(«Вы ввели некорректное значение!»);
                return null;
            }
        }
    }
}
```

```
Здравствуйте. Для того, чтобы узнать вашу оценку по экзамену, вам нужно вписать
имя.
Ваше имя:
Виктория
Смотрите ниже
Выберите, пожалуйста, предмет:
1 - Экзамен по ВУИ
2 - Экзамен по проектированию
3 - Экзамен по инженерии
1
Проверим правильность заданий:
0 - Все верно
1 - Допустил ошибку
2 - Не сделал одно задание
3 - Ничего не сделал
0
Оценка по предмету для Виктория. Ваша оценка: 5
```

Результат выполнения программы

Результаты исследования и их обсуждение

Студент вводит свои данные, выбирает предмет и указывает результат. Программа выставляет итоговую оценку студенту.

Результат работы программы представлен на рисунке.

Таким образом, данный паттерн применим, если требуется изменить какую-либо часть программы динамически. Декоратор достаточно просто реализовать за счет большого количества микроклассов, относящихся к основному телу программы.

В дальнейшем решение нашей задачи может быть модернизировано в следующих направлениях:

– разработка интерфейса;

– разработка базы данных студентов и/или предметов;

– разграничение прав доступа администратора и пользователя;

– применение других паттернов.

Выводы

Таким образом, при использовании паттернов можно упростить решение многих задач, тем самым код становится понятным, структурированным и уменьшаются затраты времени на написание кода. Однако при их использовании стоит помнить, что в программировании большая часть задач не может быть решена посредством стандартных шаблонов и требует внесения дополнительных изменений.

Список литературы

1. История паттернов [Электронный ресурс]. URL: https://refactoring.guru/ru/design-patterns/history (дата обращения: 27.02.2019).

2. Рудаков А.В. Технология разработки программных продуктов: учебное пособие для студентов среднего профессионального образования. М.: Издательский центр «Академия», 2007. 208 с.

3. Что такое Паттерн? [Электронный ресурс]. URL: https://refactoring.guru/ru/design-patterns/what-is-pattern (дата обращения: 27.02.2019).

4. Шпаргалка по шаблонам проектирования [Электронный ресурс]. URL: https://habr.com/ru/post/210288/ (дата обращения: 27.02.2019).

5. Шарп Д. Microsoft Visual C#. СПб.: Питер,2017. 848 с.

6. Шилдт Г. C# 4.0: полное руководство. М.: ООО «И.Д. Вильямс», 2011. 1056 с.