УДК 004

ПРИНЦИПЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

Сиводедова М.В., студент гр. ПИб-202, I курс
Научный руководитель: Глебова Е.А., ассистент
Кузбасский государственный технический университет
имени Т.Ф. Горбачева
г. Кемерово

Объектно-ориентированные языки программирования пользуются популярностью в настоящее время. Многие высшие учебные заведения базируют учебную программу студентов первого курса на объектно-ориентированном программировании, из-за наглядности этой парадигмы.

Главные преимущества ООП: возможность визуализировать свои задумки сразу на этапе проектирования, наличие множества практических готовых решений, простота и доступность разработки. Все это привлекает крупные компании, банки и государственные организации, для которых в первую очередь важны функционал и стабильность программного обеспечения.

ООП самостоятельная и полноценная парадигма, которая имеет свои особенности и принципы. Принципы ООП будут рассмотрены на примере языка C Sharp. C# - один из самых популярных объектно-ориентированных языков программирования, для которого есть большое количество учебной литературы. Это и стало основным параметром выбора языка для исследования

Инкапсуляция

Этот принцип можно трактовать двояко:

Инкапсуляция – объединение данных и функций, управляющих этими данными, в один компонент [1,2].

Инкапсуляция – механизм, позволяющий ограничить доступ одних компонентов программы к другим компонентам.

```
1   class Number
2   {
3       private const char V = 42;
4       private string Number = V; // данные
5       public void Shownumber() // метод, управляющий данными
6       {
7           Console.WriteLine(this.Number);
```

Рисунок 1 – Пример инкапсуляции в C#

Пример из рисунка 1 соответствует обеим трактовкам. В действительности трактовки не противоречат друг другу, а расставляют

различные акценты во взгляде на этот принцип. Назначение инкапсуляции – обеспечение связи внутри объекта.

В некоторых источниках инкапсуляцию приравнивают к сокрытию. По определению: «Сокрытие – принцип проектирования, обеспечивающий разграничение доступа различных частей программы к внутренним компонентам друг друга.». На первый взгляд может показаться, что принципы действительно идентичны, однако фактически в некоторых языках программирования есть обширные возможности для инкапсуляции, а сокрытие отсутствует вовсе. Таким образом, инкапсуляция обеспечивает сокрытие, но не является им.

Наследование

Этот принцип предусматривает возможность определения базового класса для конкретных функций, а затем, на его основе, создавать производные классы.

C# поддерживает только одиночное наследование: каждый класс может наследовать только от одного класса. Но доступно транзитное наследование, которое позволяет определить порядок и иерархию наследования для группы членов [1,3,4]. Важно заметить, что наследование применяется только для интерфейсов и классов.

Ниже рассмотрен пример класса, который описывает отдельного человека (рис. 2).

```
class Person // Пусть есть класс Person, который описывает отдельного человека
{
    private string name1;
    public string name2;
    {
        get {return name1;}
        set {name1 = value;}
    }
    public void Display()
    {
        Console.WriteLine(name2)
```

Рисунок 2 – Создание базового класса

Затем потребовался класс, описывающий сотрудника некого предприятия реализующий тот же функционал, что и класс «Person». Так как сотрудник – это также и человек, то логично будет, называется базовым классом или родителем (рис. 3). Иначе можно оказаться в ситуации, что у одного родительского класса так много наследников, что любые изменения в нем неминуемо приведут к ошибкам в том или ином производном классе.

```
class Employee : Person;
{
    static void Main(string[]args)
    {
        Person p = new Person {Name = "Artem"};
        p.Display();
        p = new Employee { Name = "Anna"};
        p.Display();
        Console.Read();
```

Рисунок 3 – Пример наследования в C#

При необходимости любой класс можно защитить от наследования ключевым словом sealed. Запечатанный класс нельзя использовать в качестве базового класса.

Полиморфизм

Возможность функции обрабатывать данные различных типов – еще один из принципов ООП. Он в свою очередь подразделяется на еще 3 разновидности [1,2]:

Ad-hoc-полиморфизм:

Принудительный полиморфизм - возможность динамической подмены типов (объектов) во время выполнения программы, но без второй формы Инкапсуляции без сокрытия реализации (Когда вы в классическом полиморфизме делаете upcast то происходит сокрытие свойств и методов из производного типа). Возможность вызывать разные функции по единому идентификатору, а в слабо типизированных языках – посредством преобразования фактического типа аргумента к ожидаемому функцией.

Пример:

Пусть у некоторых классов нет базового класса, но у них одинаковый интерфейс взаимодействия (рис. 4).

```
class A{public void Method(){Console.WriteLine("A");}}

class B{public void Method(){Console.WriteLine("B");}}

class C{public void Method(){Console.WriteLine("C");}}
```

Рисунок 4 – Создание нескольких классов, несвязанных базовым классом, но имеющих одинаковый интерфейс

Ad-hoc полиморфизм позволяет обращаться схожим образом к объектам, не связанным классическим наследованием.

```
// данный интерфейс создан для создания своего рода базового типа для классов A, B, и C у которых нету таковой.

interface IInterface
    {
        void Method();
    }
//в данном случае мы создаем новые классы обертки с схожими интерфейсами и с базовым типом
    class MyA : A, IInterface { }
    class MyB : B, IInterface { }
    class MyC : C, IInterface { }
```

Рисунок 5 – Пример Ad-hoc полиморфизма в C#. Часть 1

```
    class Program
    {
        static void Main()
        {
//тем самым мы работаем с классами как будто у них был базовый тип но  без сокрытия реализации
            IInterface instance = new MyA();
            instance.Method();
            instance = new MyB();
            instance.Method();
            instance = new MyC();
            instance.Method();
```

Рисунок 6 – Пример Ad-hoc полиморфизма в C#. Часть 2

Параметрический полиморфизм:

Один метод работает с аргументами различных типов одинаково, вне зависимости от их точного типа. Тривиальный пример для языков с наследованием - функция, работающая с объектом некоторого класса, часто может без изменений работать с объектом порождённого от C класса (этот вид полиморфизма часто называют полиморфизм включения). Менее тривиальный пример — generic-методы, которые могут, в зависимости от generic-параметра, работать с разными типами объектов. Дженерики свойственны для Java.

Полиморфизм подтипов:

Разновидность, которую понимают под полиморфизмом в ООП. Подход заключается в том, что вызывающий код использует объект, опираясь только на его интерфейс. Такой подход позволяет без перекомпиляции изменять поведение программы (рис. 6).

```
public abstract class Animal
{
    public abstract String Talk();
}

public class Cat : Animal
{
    public override String Talk()
    {
        return "Meow!";
    }
}

public class Dog : Animal
{
    public override String Talk()
    {
        return "Woof!";
    }
}

public class Program
{
    private static void Write(Animal animal)
    {
        Console.WriteLine(animal.Talk());
    }

    public static void Main(String args[])
    {
        Write(new Cat());
        Write(new Dog());
```

Рисунок 7 – Реализация параметрического полиморфизма в C#

Здесь базовый тип - класс «Animal», объявляющий интерфейс. Интерфейс состоит всего из одного метода. Далее два дочерних класса – «Cat» и «Dog» переопределяют метод «Talk» своим собственным поведением.

Метод «Program.Write» является в данном случае клиентом - он принимает на вход объект типа «Animal» и вызывает метод «Talk». При этом он не имеет информации о фактическом типе объекта, а пользуется только объявленным интерфейсом. Используя разные экземпляры разных типов «Cat» и «Dog», мы получаем разное поведение (на консоль выводится разный текст). Для языков со статической типизацией полиморфизм подтипов является очень важным принципом, без которого сложно проектировать большие проекты.

Таким образом в ходе работы были изучены основные принципы программирования в ООП: инкапсуляция, наследование, полиморфизм.

Список литературы:

1. Абрамян М.Э. Visual C# на примерах (+ CD-ROM); БХВ-Петербург – М., 2008. – 685 с.
2. Подбельский В.В. Язык C#. Базовый курс; РГГУ – Москва, 2015. – 408 с.
3. Подбельский В.В. Язык C#. Базовый курс; Финансы и статистика, Инфра-М - М., 2011. – 384 с.
4. Макконнелл С. Совершенный код. Мастер-класс / Пер. с англ. – М.: Издательство «Русская редакция», 2010. –896 с.