УДК 004.41/.42.

ПРАКТИКО-ОРИЕНТИРОВАННЫЕ ЗАДАНИЯ ПРИ ИЗУЧЕНИИ ООП НА ЯЗЫКЕ C# В СРЕДЕ MS VISUAL STUDIO

Д. С. Гаврилович
УО «Белорусский государственный педагогический университет имени Максима Танка»
Минск (Республика Беларусь))
Науч. рук. – Г. А. Заборовский, к.ф.-м.. наук, доцент

PRACTICE-ORIENTED TASKS WHEN STUDYING OOP IN C# LANGUAGE IN MS VISUAL STUDIO ENVIRONMENT

C. S. Gavrilovich
Belarusian State Pedagogical University named after Maxim Tank
Minsk (Republic of Belarus)
Scientific advisor – G. A. Zaborovsky – Dr.PhD, Associate Professor

В статье рассмотрены некоторые особенности объектно-ориентированного программирования: создания программы с реализацией классов, экземпляров, методов перегрузки, наследования, конструкторов, работой с публичными и приватными полями.

The article discusses some features of object-oriented programming: creating a program with the implementation of classes, instances, overload methods, inheritance, constructors, working with public and private fields.

Ключевые слова: ООП; Класс; перегрузка; наследование
Key words: OOP; Class; overload; inheritance

Язык C# отлично подходит для реализации объектно-ориентированного программирования (ООП), в нем доступно большинство инструментов и имеется обширное сообщество разработчиков, которые постоянно вносят свой вклад в его развитие.

C# – это объектно-ориентированный язык программирования, который позволяет разработчикам создавать надежные и масштабируемые приложения. Он предоставляет такие возможности, как наследование, инкапсуляция и полиморфизм, которые являются фундаментальными понятиями в объектно-ориентированном программировании. В дополнение к сильной поддержке ООП C# также предлагает широкий спектр инструментов и фреймворков для разработки различных типов приложений. В данной статье мы описываем класс, который будучи реализованным, позволит создавать экземпляры класса, в которые можно вводить информацию о студенте (имя, фамилия, отчество, и дата рождения). Данная задача показывает перегрузку методов, наследование и работу объектно-ориентированного программирования.

Рассмотрим основные понятия ООП:

Класс – модель для создания объектов определённого типа, описывающая их структуру (набор полей и их начальное состояние) и определяющая алгоритмы (функции или методы) для работы с этими объектами.

Объектэто – сущность, способная сохранять свое состояние (информацию) и обеспечивающая набор операций (поведение) для проверки и изменения этого состояния.

Создание объекта – это некий процесс обращения к конкретному экземпляру описанного объекта. После описания объекта он имеет некую абстрактную форму и когда мы обращаемся к какому-то конкретному работнику, к какому-то конкретному экземпляру этого описания: работник 1, работник 2, работник 3.

Поля класса – это объекты любого типа; набор полей и их значения определяют состояние объекта.

Методы — это функции, связанные с классом.

Цели: демонстрация работы с классами и наследованием в C#; пример использования перегрузки методов и конструкторов.

1. Создадим класс под названием “Student”, который содержит приватные переменные: имя **firstName**, фамилия **lastName**, отчество **middleName**, дата рождения **birthday** (рис.1).

2. Создаем методы для получения информации из приватных полей. Private: закрытый или приватный компонент класса или структуры. Приватный компонент доступен только в рамках своего класса или структуры (рис. 2).

```
class Student
{
    //Создаем поля с информацией о студенте
    private string firstName;
    private string lastName;
    private string middleName;
    private DateTime birthday;
```

Рис. 6 – Класс «Student»

3. Использование перегрузки методов на примере конструктора класса. Перегрузка методов – это возможность создавать несколько методов с одинаковым названием, но разными параметрами (рис. 3).

```
Ссылок: 2
public string FirstName
{
    get { return this.firstName; }
}
Ссылок: 2
public string LastName
{
    get { return this.lastName; }
}
Ссылок: 2
public string MiddleName
{
    get { return this.middleName; }
}
Ссылок: 2
public DateTime Birthday
{
    get { return this.birthday; }
}
```

Рис. 7

```
public Student(string lastName)
{
    this.lastName = lastName;
}
Ссылок: 0
public Student(string lastName, DateTime birthday)
{
    this.lastName = lastName;
    this.birthday = birthday;
}

Ссылок: 3
public Student(string lastName, string firstName,
               string middleName, DateTime birthday)
{
    this.lastName = lastName;
    this.birthday = birthday;
    this.firstName = firstName;
    this.middleName = middleName;
}
```

Рис. 8

4. Переопределение метода ToString() для вывода информации о студенте. (рис. 4).

```
public override string ToString()
{
    return $"Имя: {firstName}\n" +
        $"Фамилия: {lastName}\n" +
        $"Отчество: {middleName}\n" +
        $"Дата рождения: {birthday.ToString("dd.MM.yyyy")} г.";
}
```

Рис. 9

5. Создание класса GraduatedStudent, который наследует от класса Student; Создание конструкторов и переопределение метода ToString() для вывода информации о выпускнике (рис. 5).

6. Создание экземпляров класса. Класс описывает поля и методы, которые будут доступны у объекта, построенного по описанию, заложенному в классе. Экземпляры используются для представления (моделирования) конкретных сущностей реального мира (рис. 6).

7. И при вызове программы получим (рис. 7).

```
Студент:
Имя: Иванов
Фамилия: Иван
Отчество: Иванович
Дата рождения: 02.04.1986 г.

Студент выпустился:
Имя: Иванов
Фамилия: Иван
Отчество: Иванович
Дата рождения: 02.04.1986 г.
Окончил: 07.05.2000 г.
```

Рис. 7

```
class GraduatedStudent : Student
{

    private DateTime graduated;

    // Конструктор для создания ВыпустившегосяСтудента
    ссылка: 1
    public GraduatedStudent(Student student, DateTime graduated) : base(student.LastName, student.FirstName,
                                                                         student.MiddleName, student.Birthday)
    {
        this.graduated = graduated;
    }
    Ссылок: 0
    public GraduatedStudent(Student student) : base(student.LastName, student.FirstName,
                                                    student.MiddleName, student.Birthday)
    {
        this.graduated = DateTime.Now;
    }
    //Переопределяем стандартное преобразование в строку с использованием преобразования в строку предка
    Ссылок: 2
    public override string ToString()
    {
        return base.ToString() + $"\nОкончил: {graduated.ToString("dd.MM.yyyy")} г.";
    }
}
```

Рис. 10

```
class Program
{
    Ссылок: 0
    static void Main(string[] args)
    {
        //Создадим студента
        Console.WriteLine("Студент:");
        Student student1 = new Student("Иван","Иванов","Иванович", new DateTime(1986,04,02));
        Console.WriteLine(student1);

        Console.WriteLine();

        // Наш дорогой студент выпустился
        Console.WriteLine("Студент выпустился:");
        GraduatedStudent graduatedStudent1 = new GraduatedStudent(student1, new DateTime(2000,
        Console.WriteLine(graduatedStudent1);

        Console.ReadKey();
    }
}
```

Рис. 6

В данном коде создается класс **Student** с полями **firstName**, **lastName**, **middleName** и **birthday**, и методами для получения информации из приватных полей. Далее используется перегрузка методов на примере конструктора класса и переопределение метода **ToString()** для вывода информации о студенте. Также создается класс **GraduatedStudent**, который наследует от класса **Student** и переопределяет метод **ToString()** для вывода информации о выпускнике.

Пример демонстрирует использование классов и наследования в C#, а также перегрузки методов, конструкторов и переопределения метода **ToString()**. Классы могут быть полезны при разработке приложений, где необходимо хранить информацию о студентах и выпускниках, что важно для дальнейших практических разработок.