Основным понятием в объектно-ориентированном программирование является понятие объекта и класса.

Объект

Объект, в самом общем смысле, это сущность, которая объединяет в себе данные и код, который может манипулировать этими данными [1]. Достаточно часто объект обладает следующими признаками:

1. Как правило, объект является **абстракцией** некоторой реальной сущности той предметной области, в рамках которой разрабатывается программный продукт.
2. Как правило, объект должен **поддерживать инкапсуляцию** - менять данные вне этого объекта можно только путём вызова определённого кода (публичных методов, свойств и т.д.).

Класс

Класс - это **схема построения объекта**, в которой указываются данные (члены-данные) и код, который будет оперировать этими данными (члены-функции), а также модификаторы их доступа. Если объект получается по схеме такого класса, то его можно назвать **экземпляром** или **объектом класса**.

Стоит также отметить, что существуют другие подходы к определению классов и объектов. Так, в моём случае, определения класса даётся после определения объекта, но можно действовать наоборот: сначала определить понятие класса, а затем дать определение объекту как экземпляр класса.

Модификаторы доступа

ООП поддерживает 3 вида модификаторов доступа для членов класса:

1. **Публичный** - каждый публичный член класса доступен из вне этого класса для чтения / изменений (если это данные класса) или выполнения (функции-члены).
2. **Приватные** - каждый приватный член класса не доступен за пределами этого класса.

3. **Защищённые** - защищённый член класса доступен всем дочерним классам и никому кроме дочерних классов.

Члены-данные

Данные класса можно разделить на два типа:

1. Переменные экземпляра, которые могут отличаться у разных объектов.
2. Статические переменные, которые являются общими для всех экземпляров данного класса.

Члены-функции

Функции члены бывают следующих типов:

1. **Методы** - функции, которые можно вызвать у экземпляра класса и, **в общем случае**, изменить состояния этого класса (метод может и не менять состояния класса).
2. **Конструкторы** - функции, которые вызываются один раз, после создания объекта класса.
3. **Деструктор** - функции, которые вызываются один раз, перед уничтожением объекта класса.
4. **Свойства** - пара функции для получения и записи значений, обращения к которым из вне класса ничем не отличается от обращения к открытым полям класса.

Помимо объектов и классов, методика программирования ООП активно использует механизмы **наследования** и **полиморфизма**.

Наследование

Механизм наследования позволяет **дополнить** схему уже существующего класса новыми данными и функциями, или **переопределить** уже имеющиеся функции, если, кончено, это позволяет сделать наследуемый класс (например: переопределить виртуальный метод основного класса)[2]. Когда класса B наследует класса A, то класс B называется **дочерним** по отношению к классу A, а класс A **родительским** по отношению к классу B. Основное достоинство механизма наследования неразрывно связан с полиморфизмом, который мы сейчас рассмотрим. (хотя наследования и само по

себе является достаточно хорошим инструментом, позволяющим избавиться от дублирования кода).

Полиморфизм

Полиморфизм, по отношению к классам, позволяет использовать дочерний класс в тех участках кода, которые требуют использования родительского класса. Таким образом, благодаря этому механизму, между потомком и родительским классом формируются отношения вида: **подмножество основного множества**.

Пример использования полиморфизма и наследования на ЯП C#:

```
class Animal
{
   public virtual string MakeNoise()
   {
      return string.Empty;
   }
}

class Dog : Animal
{
   public override string MakeNoise()
   {
      return "woof woof";
   }
}

class Cat : Animal
{
   public override string MakeNoise()
   {
      return "meow meow";
   }
}

class Program
{
   static void Main()
   {
      Dog dog = new Dog();
      Cat cat = new Cat();

      ExampleMethod(dog);
      ExampleMethod(cat);
   }
```

```
    static void ExampleMethod(Animal animal)
    {
        string animalNoise = animal.MakeNoise();
        Console.WriteLine($"This animal make noise like: {animalNoise}");
    }
}
```

Результат выполнения программы:

```
This animal make noise like: woof woof
This animal make noise like: meow meow
```

Также в объектно-ориентированном подходе существует полиморфизм на основа **интерфейсов**, который позволяет использовать в одном и том же коде разные классы, если они “скрыты”(реализуют) определённый интерфейс, необходимы для текущей работы.

Сами интерфейсы представляют собой перечисления определённого функционала (чаще всего перечисления сигнатур методов), который должны присутствовать с публичными модификаторами доступа в классе, реализующий данный интерфейс [3].

Пример использования интерфейсов и полиморфизма на ЯП C#:

```
interface NoisyObject
{
    string MakeNoise();
}

class Dog : NoisyObject
{
    public string MakeNoise()
    {
        return "woof woof";
    }
}

class Cat : NoisyObject
{
    public string MakeNoise()
    {
        return "meow meow";
    }
}
```

```
class Program
{
    static void Main()
    {
        Dog dog = new Dog();
        Cat cat = new Cat();

        ExampleMethod(dog);
        ExampleMethod(cat);
    }

    static void ExampleMethod(NoisyObject noisyObject)
    {
        string noise = noisyObject.MakeNoise();
        Console.WriteLine($"This object make noise like: {noise}");
    }
}
```

Результат выполнения программы:

```
This object make noise like: woof woof
This object make noise like: meow meow
```

Источники

1. C# OOP: https://www.w3schools.com/cs/cs_oop.php
2. C# Inheritance: https://www.w3schools.com/cs/cs_inheritance.php
3. Interfaces - define behavior for multiple types: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/interfaces