Inheritance in C#. Abstract class. Polymorphism

By Ira Zavushchak

softserve

AGENDA

- Implementation inheritance
- Abstract class
- Virtual methods
- Sealed classes and methods

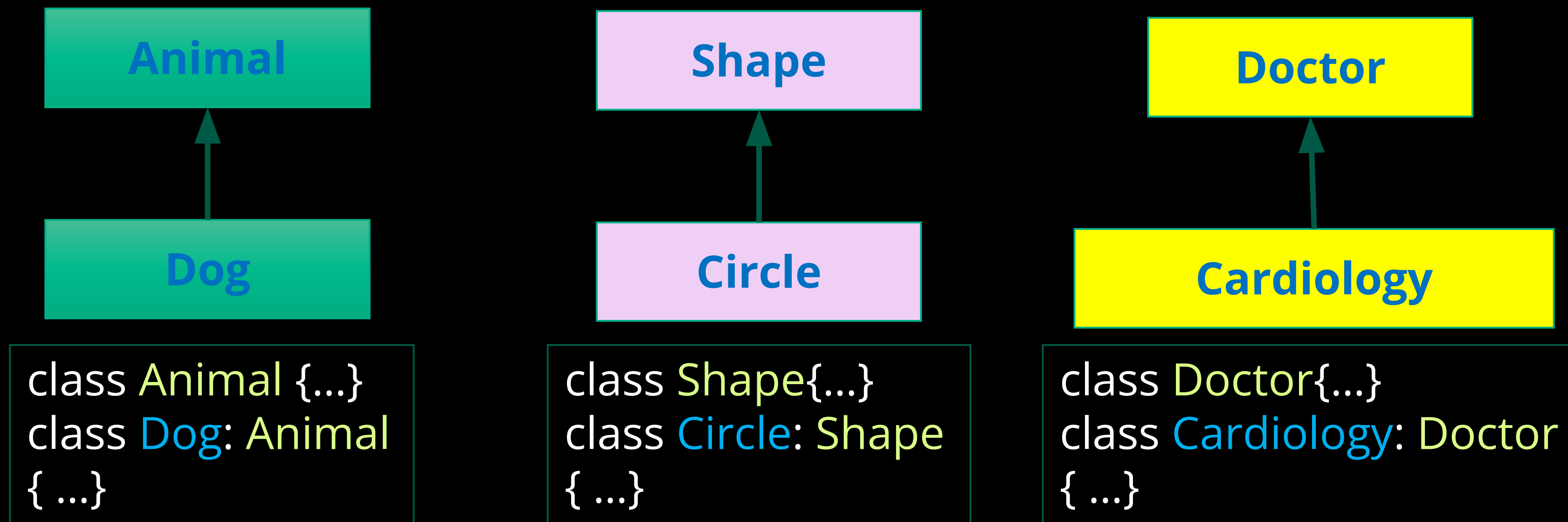
TYPES OF INHERITANCE

- **Implementation inheritance** means that a type derives from a base type, taking all the base type's member fields and functions.

- **Interface inheritance** means that a type inherits only the signatures of the functions and does not inherit any implementations.

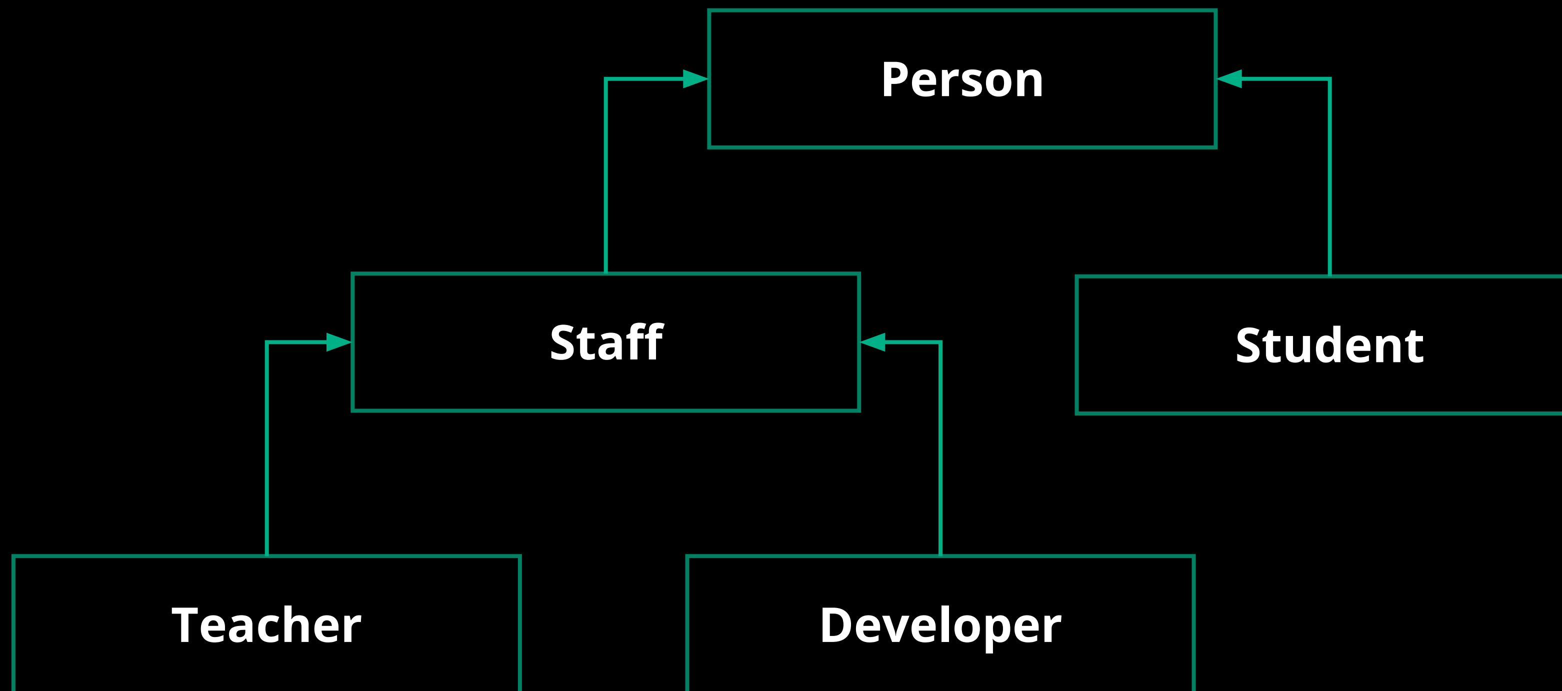
IMPLEMENTATION INHERITANCE

- Inheritance enables us to create new classes that reuse, extend, and modify the behavior that is defined in other classes.
- The class whose members are inherited is called the base class, and the class that inherits those members is called the derived class.
- The idea of inheritance implements the **IS-A** relationship.



IMPLEMENTATION INHERITANCE

- A derived class can have only one direct base class.
- Inheritance is transitive. If ClassC is derived from ClassB, and ClassB is derived from ClassA, ClassC inherits the members declared in ClassB and ClassA.



SoftServe Confidential

EXAMPLE

```
public class Person
{
    private string name;
    2 references
    public Person(string name)
    { this.name = name; }
    1 reference
    public string Name { get { return name; } }
    2 references
    public void Print()
    {
        Console.WriteLine("Name: {0}", this.name);
    }
}
4 references
public class Staff : Person
{
    private int salary;
    2 references
    public Staff(string name, int salary) : base(name)
    { this.salary = salary; }

    1 reference
    public void Print()
    {
        Console.WriteLine("Person {0} has salary: ${1}",
                          Name, this.salary);
    }
}
```

```
static void Main(string[] args)
{

    Person person1 = new Person("Oleg");
    person1.Print();

    Staff staff1 = new Staff("Igor", 200);
    staff1.Print();

    person1 = new Staff("Ira", 300);
    person1.Print();

    Console.ReadKey();
}
```

```
Name: Oleg
Person Igor has salary: $200
Name: Ira
```

THE base keyword IS USED TO ACCESS MEMBERS OF THE BASE CLASS FROM WITH IN A DERIVED CLASS

```
class Student : Person
{
    private string groupName;

    0 references
    public Student(string name, string groupName) : base(name)
    { this.groupName = groupName; }

    0 references
    public void Print()
    {
        base.Print();
        Console.WriteLine("Student of group: {0}", this.groupName);
    }
}
```

```
static void Main(string[] args)
{

    Person person1 = new Person("Oleg");
    person1.Print();

    Staff staff1 = new Staff("Igor", 200);
    staff1.Print();

    person1 = new Staff("Ira", 300);
    person1.Print();

    Student olga = new Student("Olga", ".Net Core");
    olga.Print();

    Console.ReadKey();
}
```

```
Name: Oleg
Person Igor has salary: $200
Name: Ira
Name: Olga
Student of group: .Net Core
```

ABSTRACT CLASS

- An abstract class cannot be instantiated.
- The purpose of an abstract class is to provide a common definition of a base class that multiple derived classes can share.
- Abstract classes may define abstract methods.
- Derived classes of the abstract class must implement all abstract methods.

```
public abstract class A
{
        // Class members here.
}
```

```
public abstract class A
{
  public abstract void DoWork(int i);
  // Class members here.
}
```

EXAMPLE

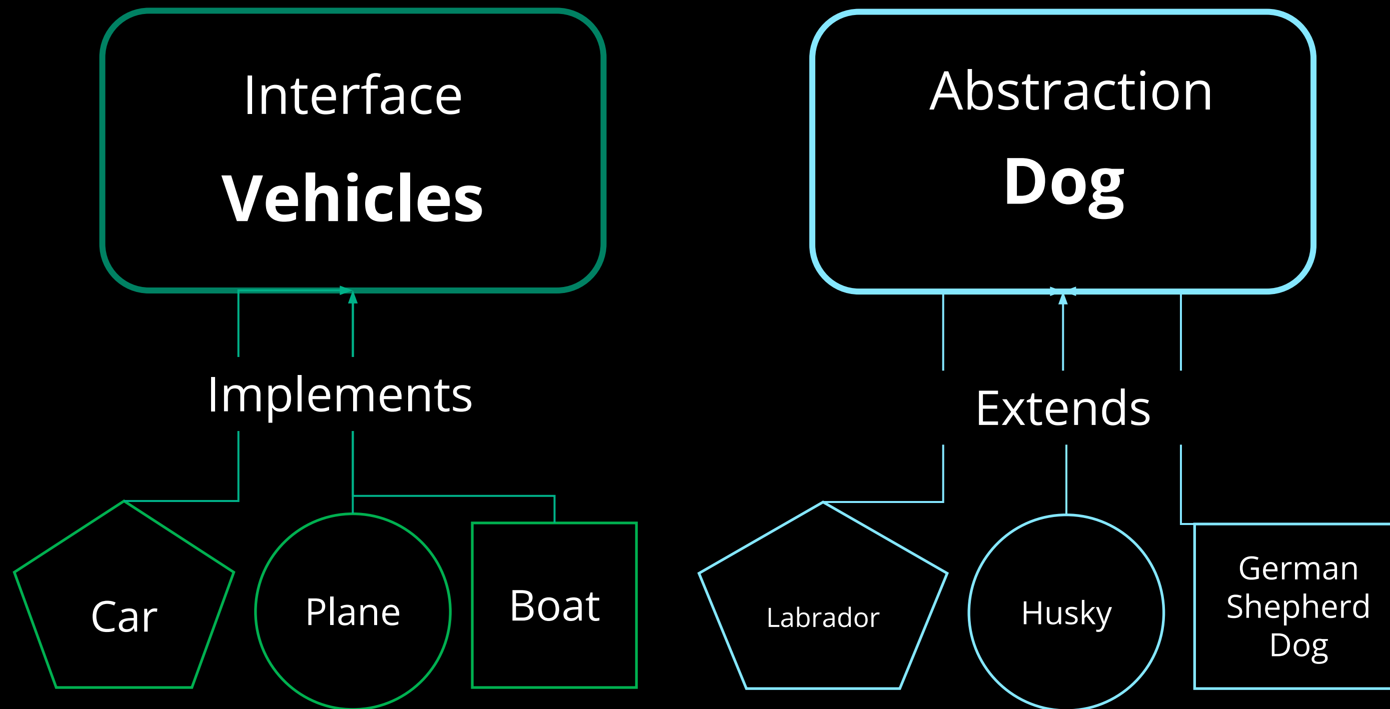
```
abstract class Person
{
    2 references
    public string Name { get; set; }
    2 references
    public Person(string name)
    {
        Name = name;
    }
    2 references
    public void Display()
```

```
class Client : Person
{
    1 reference
    public int Sum { get; set; }    // сума на рахунку
    2 references
    public Client(string name, int sum) : base(name)
    {
        Sum = sum;
    }
}
```

```
:; set; } // посада
, string position) : base(name)
```

```
static void Main(string[] args)
{

    Client client = new Client("Tom", 500);
    Employee employee = new Employee("Bob", "Employee1");
    client.Display();
    employee.Display();

    //Або так
    Person client2 = new Client("Tom", 500);
    Person employee2 = new Employee("Bob", "Employee2");

    //Но ми НЕ можемо створити об'єкт Person, використовуючи конструктор класу Person
    //Person person = new Person("Bill");

    Console.ReadKey();
}
```

INTERFACE vs ABSTRACT CLASS



- What is the difference between an **abstract class** and an **interface**?
  - ✔ An abstract class can have **fields** and **implementation** of **methods**.
  - ✔ An abstract class is essentially the same thing as an interface except it is an **actual class**, not just a **contract**.
  - ✔ abstract classes with virtual methods have **better performance** than interface implementation

INTERFACE vs ABSTRACT CLASS

Abstract Class	Interface
An Abstract class doesn't provide full abstraction	Interface does provide full abstraction
Using Abstract we can not achieve multiple inheritance	using an Interface we can achieve multiple inheritance.
We can declare a member field	We can not declare a member field in an Interface
An abstract class can contain access modifiers for the subs, functions, properties	We can not use any access modifier i.e. public , private , protected , internal etc. because within an interface by default everything is public
An abstract class can be defined	An Interface member cannot be defined using the keyword static, virtual, abstract or sealed
A class may inherit only one abstract class.	A class may inherit several interfaces.
An abstract class can provide complete, default code and/or just the details that have to be overridden.	An interface cannot provide any code, just the signature.

VIRTUAL METHODS

- **Virtual method** - a method that can be overridden in a derived class.
- **Overriding method** - a change of its implementation in derived classes.
- Static method can not be virtual

```
[модифікатор доступу] virtual [тип] [ім'я методу] ([аргументи])
{
  // Тіло методу
}
```

```
[модифікатор доступу] override [тип] [ім'я методу] ([аргументи])
{
  // Нове тіло методу
}
```

VIRTUAL AND ABSTRACT METHODS

SoftServe Confidential

- **Abstract method** is a method that does not have its implementation in the base class, and it should be implemented in the derived class. **Abstract method** can be declared only in abstract class.
- What is the difference between the virtual and the abstract method?
  - ✔ The virtual method can have its implementation in the base class, abstract - no (body is empty);
  - ✔ An abstract method must be implemented in the derived class, the virtual method is not necessary to override.
- Announcement of the abstract method:

```
[модифікатор доступу] abstract [тип] [ім'я методу] ([аргументи]);
```

- The implementation of the abstract method in the derived class occurs in the same way as the override of the method - using the keyword override:

```
[модифікатор доступу] override [тип] [ім'я методу] ([аргументи])
{
  // Реалізація методу
}
```

softserve

ABSTRACT PROPERTIES

- Creating abstract properties is not very different from the methods:

```
protected [тип] [поле, яким управляє властивість];
[модифікатор доступу] abstract [тип] [ім'я властивості]{get; set;}
```

- Realization in the derived class:

```
[модифікатор доступу] override [тип] [ім'я властивості]
{
  get {тіло аксессор get}
  set {тіло аксессор set}
}
```

EXAMPLE

SoftServe Confidential

```
abstract class Person
{
    4 references
    public string Name { get; set; }
    2 references
    public Person(string name)
    {
        Name = name;
    }
    3 references
    public virtual void Display()
    {
        Console.WriteLine(Name);
    }
}
```

```
class Client : Person
{
    2 references
    public int Sum { get; set; }    // сума на рахунку
    1 reference
    public Client(string name, int sum) : base(name)
    {   Sum = sum;    }
    3 references
    public override void Display() // перевизначення методу
    {
        Console.WriteLine("Клієнт \nІмя:" + Name + "\n" + "Сума на рахунку:" + Sum + "\n");
    }
}
2 references
class Employee : Person
{
```

```
g Position { get; set; } // посада

yee(string name, string position) : base(name)
= position;    }

ide void Display() // перевизначення методу

івник \nІмя:" + Name + "\n" + "Посада:" + Position + "\n");
```

```
static void Main(string[] args)
{
    List<Person> persons = new List<Person>();
    persons.Add(new Client("Tom", 500));
    persons.Add(new Employee("Bob", "Employee1"));

    foreach (Person p in persons)

    p.Display();
    Console.ReadKey();
}
```

```
Клієнт
Імя:Tom
Сума на рахунку:500

Працівник
Імя:Bob
Посада:Employee1
```

softserve

EXAMPLE


```
namespace demo9
{
    public class Person
    {
        private string name;
        public Person(string name)
        { this.name = name; }
        virtual public string Name { get { return name; } }
        virtual public void Print()
        {
            Console.WriteLine("Name: {0}", this.name);
        }

        public class Staff : Person
        {
            private int salary;
            public Staff(string name, int salary) : base(name)
            { this.salary = salary; }
            override public string Name { get { return base.Name + " Staff"; } }//перевизначення методу
            override public void Print()
            {
                Console.WriteLine("Person {0} has salary: ${1}", Name, this.salary); //перевизначення методу з додаванням salary
            }
        }

        class Student : Person
        {
            private string groupName;

            public Student(string name, string groupName) : base(name)
            { this.groupName = groupName; }

            override public void Print()
            {
                base.Print();
            Console.WriteLine("Student of group: {0}", this.groupName);
            }
    }
```

```
    static void Main(string[] args)
    {
        List<Person> people = new List<Person>();
        people.Add(new Person("Yura"));
        people.Add(new Staff("Ira", 300));
        people.Add(new Person("Ivan"));
        people.Add(new Staff("Petro", 500));
        people.Add(new Student("Vasyl", "C# OOP"));
        foreach (var p in people)
            p.Print();
        Console.ReadLine();
    }
}
```

softserve

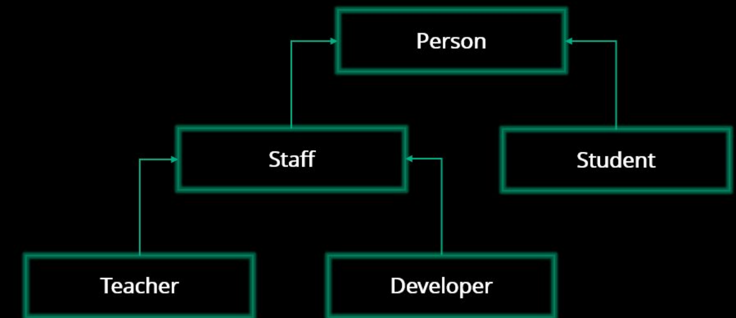
TASK 8

1. Add two classes Persons and Staff (use the presentation code)
2. Create two classes Teacher and Developer, derived from Staff.

✔Add field ***subject*** for class Teacher;

✔Add field ***level*** for class Developer;

✔override method Print for both classes.



3. In Main, specify a list of Person type and add objects of each type to it. Call for each item in the list method Print ().
4. Enter the person's name. If this name present in list - print information about this person
5. Sort list by name, output to file
6. Create a list of Employees and move only workers there. Sort them by salary.

SoftServe Confidential

HOMEWORK 8

1) Create abstract class Shape with field name and property Name.

Add constructor with 1 parameter and abstract methods Area() and Perimeter(), which can return area and perimeter of shape;

Create classes Circle, Square derived from Shape with field radius (for Circle) and side (for Square). Add necessary constructors, properties to these classes, override methods from abstract class Shape.

a) In Main() create list of Shape, then ask user to enter data of 10 different shapes. Write name, area and perimeter of all shapes.

b) Find shape with the largest perimeter and print its name.

3) Sort shapes by area and print obtained list (Remember about IComparable)