ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

C#. Абстрактні класи та інтерфейси

Лекція 11

План

1. Абстрактні класи
2. Інтерфейси
3. Відмінності між абстрактними класами та інтерфейсами
4. Приклади

Абстрактні класи

В ООП **абстрактний клас** – це базовий клас, що призначений для успадкування, але не для створення екземплярів класів.

Абстрактні класи використовуються для визначення загальної структури та функціональності, яку спадкоємці будуть реалізовувати у своїх конкретних реалізаціях.

Абстрактний клас втілює принцип поліморфізму, дозволяючи оголошувати методи та властивості без їхньої реалізації. На відміну від інтерфейсів, абстрактні класи можуть містити як абстрактні, так і реалізовані методи та властивості.

Абстрактні класи

Абстрактний клас у C# позначається ключовим словом *abstract*. Такі класи служать основою для інших класів і часто використовуються під час створення складних ієрархій класів або розробки фреймворків.

Синтаксис :

```
[модифікатор] abstract class <Ім'яАбстрактногоКласу>
 {
        // Поля, властивості, методи
}
```

Абстрактні класи

Абстрактні класи:

- можуть містити поля;
- можуть містити властивості;
- Не можна створити екземпляр абстрактного класу. Він існує тільки для того, щоб слугувати базою для інших класів.
- Однією з ключових особливостей абстрактних класів є те, що вони можуть містити абстрактні методи , які є методами без реалізації. Абстрактні методи, оголошені в абстрактному класі, повинні бути реалізовані в похідних класах.
- Абстрактні класи можуть містити також і звичайні методи, які можуть бути успадковані похідними класами

Абстрактні класи

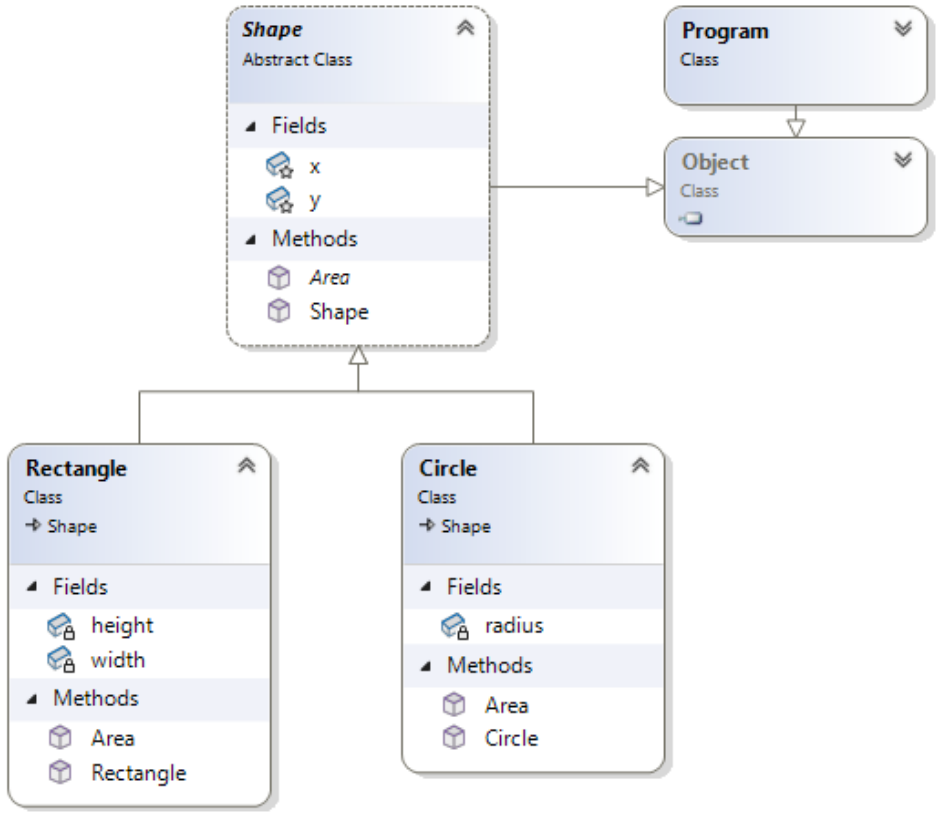
```
abstract class Shape {
    protected int x;
    protected int y;

    public Shape(int x, int y){
        this.x = x;
        this.y = y;
    }
    public abstract double Area();
}

class Circle : Shape{
    private int radius;

    public Circle(int x, int y, int radius) : base(x, y){
        this.radius = radius;
    }

    public override double Area(){
        return Math.PI * radius * radius;
    }
}
```

Абстрактні класи можуть мати конструктори, але їх не можна використовувати для створення об'єктів. Натомість для створення об'єктів абстрактного класу використовуються конструктори похідних класів.

Конструктори в абстрактних класах використовуються для ініціалізації полів та виконання загальних операцій, які потрібні для всіх похідних класів.

Абстрактні класи

```
class Rectangle : Shape{
    private int width;
    private int height;

    public Rectangle(int x, int y, int width, int height) : base(x, y){
        this.width = width;
        this.height = height;
    }

    public override double Area() {
        return width * height;
    }
 }
```




```
Circle circle = new Circle(0, 0, 5);
Rectangle rectangle = new Rectangle(1, 2, 4, 6);

Console.WriteLine($"Площа кола: {circle.Area()}");
Console.WriteLine($"Площа прям: {rectangle.Area()}");
```

Абстрактні класи

Абстрактні класи можуть мати конструктори, але їх не можна використовувати для створення об'єктів. Натомість для створення об'єктів абстрактного класу використовуються конструктори похідних класів.

Абстрактні класи

```
public abstract class Animal
{
    public string Name { get; set; }
    public int Age { get; set; }

    public abstract void Sound();

    public virtual void Move(){
        Console.WriteLine("Тварина рухається.");
    }

    public void DisplayInfo(){
        Console.WriteLine($"Ім'я: {Name}, Вік: {Age}");
    }
}
```

Абстрактний клас Animal оголошує абстрактний метод Sound, який повинен бути реалізований будь-яким похідним класом.

Віртуальний метод Move має реалізацію в базовому класі, але може бути перевизначений у похідних класах.

Повністю реалізований метод DisplayInfo, який можуть використовувати всі дочірні класи.

Приклад

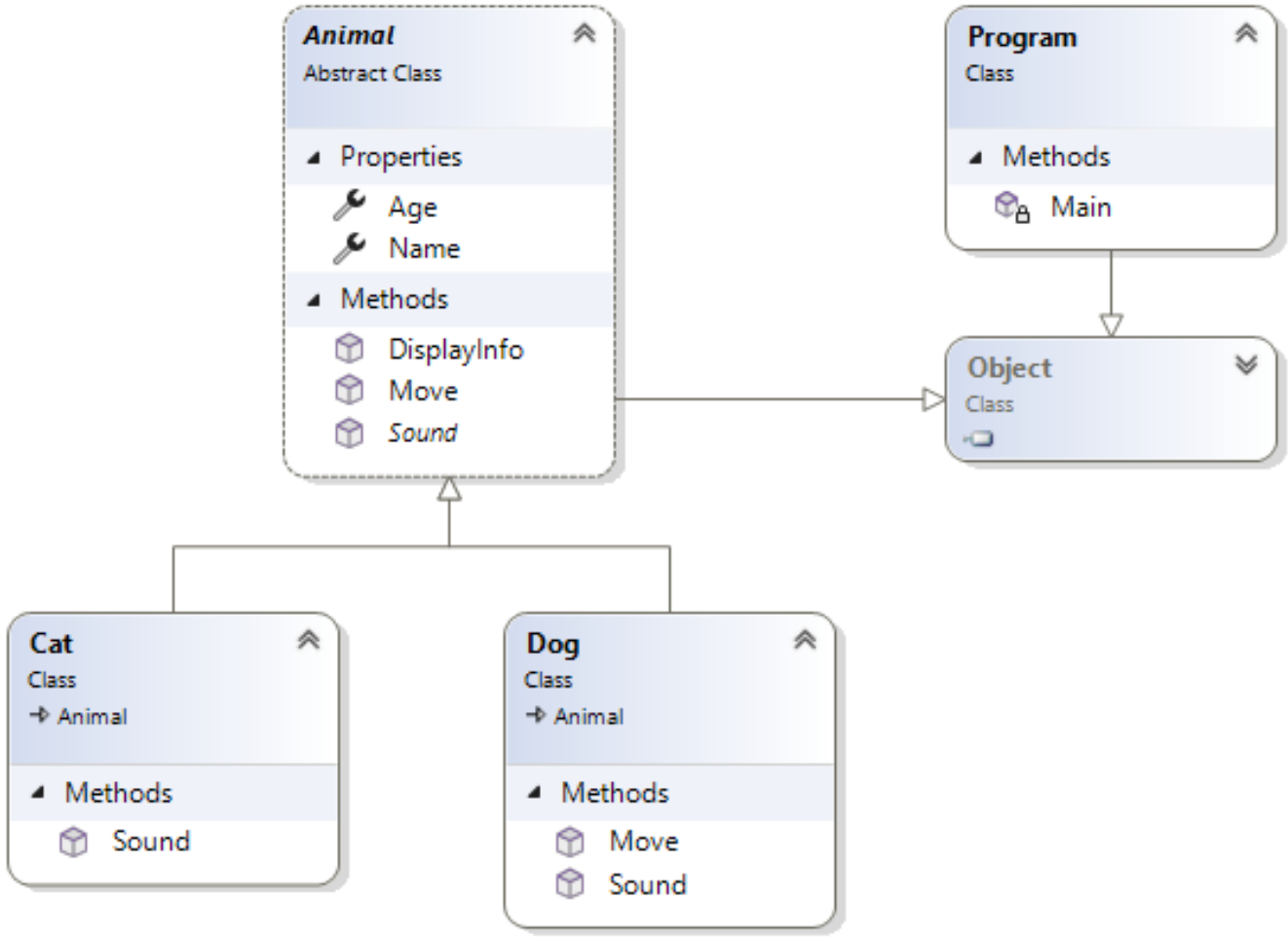
```
public class Dog : Animal
{
    public override void Sound(){
        Console.WriteLine("Гав!");
    }

    public override void Move()
    {
        Console.WriteLine("Собака біжить.");
    }
}

public class Cat : Animal
{
    public override void Sound(){
        Console.WriteLine("Мяу!");
    }
}
```

Клас Dog і Cat є нащадками класу Animal.

Ці класи реалізують абстрактний метод Sound з класу Animal, а також перевизначають віртуальний метод Move.

Приклад

```
Dog dog = new Dog { Name = "Бобік", Age = 13 };
Cat cat = new Cat { Name = "Мурка", Age = 2 };

dog.DisplayInfo();
dog.Sound();
dog.Move();

cat.DisplayInfo();
cat.Sound();
cat.Move();
```



Абстрактні класи

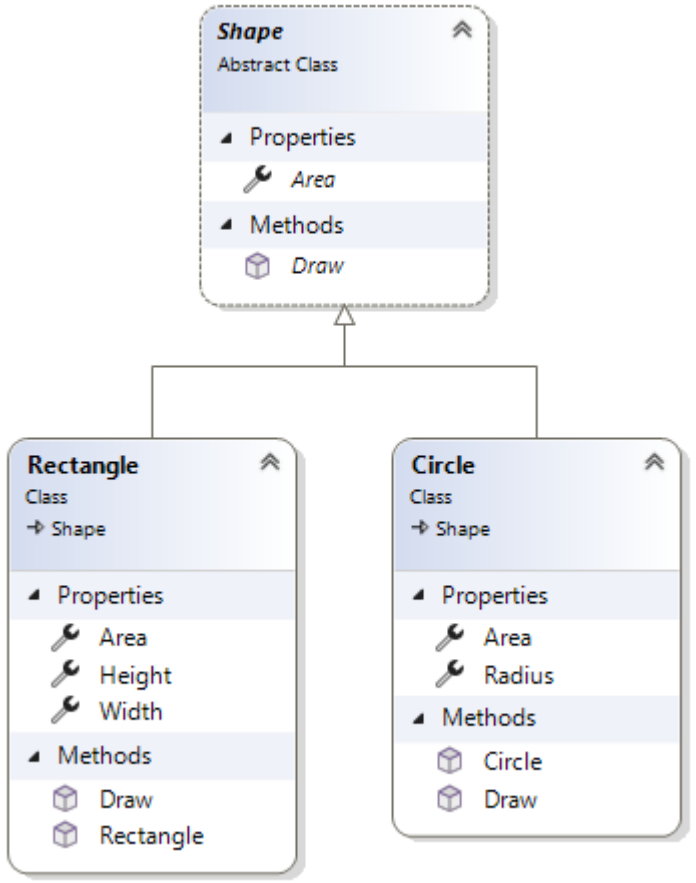
Похідний клас повинен перевизначити і реалізувати всі абстрактні методи і властивості, які є у базовому абстрактному класі.

При перевизначенні у похідному класі такий метод або властивість також оголошуються з модифікатором override (як і за звичайного перевизначення віртуальних методів та властивостей). Також слід врахувати, що якщо клас має хоча б один абстрактний метод (або абстрактні властивості), цей клас має бути визначений як абстрактний.

Абстрактні класи

Абстрактні члени, як і віртуальні, є частиною поліморфного інтерфейсу.

Але якщо у випадку з віртуальними методами говоримо, що клас-нащадок успадковує реалізацію, то у випадку з абстрактними методами успадковується інтерфейс, представлений цими абстрактними методами.



Абстрактні класи

```
public abstract class Shape
  {
     public abstract double Area { get; }
     public abstract void Draw();
  }

  public class Circle : Shape
  {
     public double Radius { get; set; }

     public Circle(double radius){
        Radius = radius;
     }

     public override double Area => Math.PI * Radius * Radius;

     public override void Draw(){
        Console.WriteLine($"Circle with radius {Radius} and area {Area}");
     }
  }
```

Абстрактні члени Area і Draw() реалізуються за допомогою ключового слова override.

Абстрактні класи

Відмова від реалізації абстрактних членів

!!! Похідний клас повинен реалізувати всі абстрактні члени базового класу.

Однак є можливість відмовитися від реалізації, але в цьому випадку похідний клас також має бути визначений як абстрактний

```
abstract class Transport
{
    public abstract void Move();
}

abstract class Car :Transport
{
}

class Auto: Car
{
    public override void Move()
    {
        Console.WriteLine("…");
    }
}
```

Інтерфейси

Інтерфейси в C# є контрактом, що визначає набір методів, властивостей, подій або індексаторів, які клас повинен реалізувати. Вони забезпечують абстракцію та стандартизацію взаємодії між класами.

Синтаксис:

```
public interface IShape
{
        double CalculateArea () ;
        double CalculatePerimeter () ;
}
```

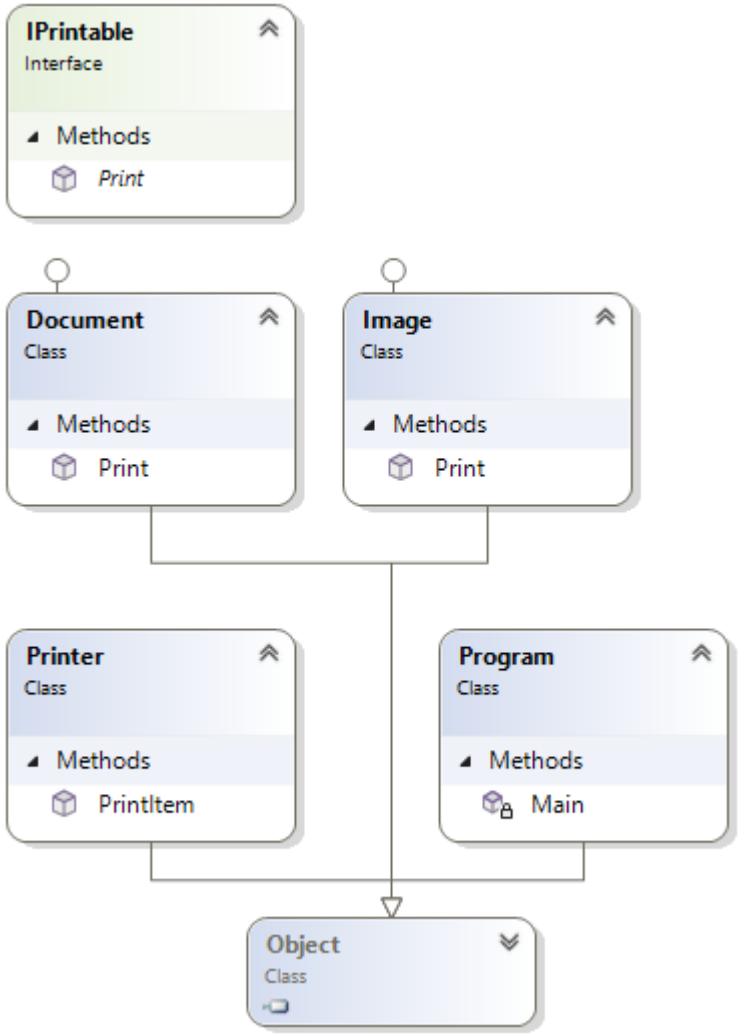
Інтерфейси

Основні характеристики інтерфейсів:

Інтерфейси *оголошують лише методи та властивості*, не забезпечують жодної реалізації.
Клас може реалізувати кілька інтерфейсів , що забезпечує більшу гнучкість і розширюваність.
Інтерфейси не можуть містити нестатичні поля. Вони описують поведінку лише через сигнатури методів і властивостей.

Інтерфейси

```
public interface IPrintable{
   void Print();
}

public class Document : IPrintable{
   public void Print(){
      Console.WriteLine("Друк документу...");
   }
}

public class Image : IPrintable{
   public void Print(){
      Console.WriteLine("Друк зображення...");
   }
}

public class Printer{
   public void PrintItem(IPrintable item)
   {
      item.Print();
   }
}
```



Комбінування інтерфейсів і абстрактних класів

Часто поєднують як абстрактні класи, так і інтерфейси, щоб отримати переваги обох.

Можна визначити абстрактний клас, який реалізує один або більше інтерфейсів, а потім конкретні класи можуть походити від абстрактного класу, дотримуючись інтерфейсного контракту.

Основні відмінності між абстрактними класами та інтерфейсами

Ключові відмінності між абстрактним класом та інтерфейсом

1. Призначення та використання:

Інтерфейс визначає контракт, який можуть реалізувати кілька класів. Він використовується, коли ви хочете вказати, які методи або властивості має мати клас, але не те, як вони повинні бути реалізовані.

Абстрактний клас служить базовим класом, який може надавати спільну функціональність і стан для похідних класів. Він використовується, коли ви хочете поділитися кодом між спорідненими класами.

Основні відмінності між абстрактними класами та інтерфейсами

2. Реалізація методів:

Інтерфейс починаючи з C# 8.0, можуть мати реалізацію методу за замовчуванням та статичні поля. Однак інтерфейси все ще не можуть мати нестатичні поля чи конструктори, тобто вони не можуть підтримувати стан.

Абстрактний клас може мати як абстрактні методи (без реалізації), так і звичайні методи (з реалізацією). Абстрактні класи можуть мати поля, властивості та конструктори, що дозволяє їм підтримувати стан.

Основні відмінності між абстрактними класами та інтерфейсами

3. Множинне успадкування:

Interface дозволяє класу або структурі реалізувати кілька інтерфейсів (множинне успадкування).

Клас може успадковувати лише **один абстрактний клас**, що обмежує успадкування однією ієрархією.

4. Поля та властивості:

Інтерфейс не може містити нестатичні поля

Абстрактний клас може містити поля та повністю реалізовані властивості, що дозволяє класу керувати станом.

Основні відмінності між абстрактними класами та інтерфейсами

5. Модифікатори доступу:

Інтерфейс - усі члени є неявно public, а інші модифікатори доступу не дозволені, за винятком певних випадків (наприклад, private члени в реалізаціях за замовчуванням).

Абстрактний клас - учасники можуть мати будь-який модифікатор доступу (public, protected, private тощо), що забезпечує більше контролю над рівнями доступу.

Основні відмінності між абстрактними класами та інтерфейсами

Коли використовувати інтерфейс

Коли потрібно визначити загальний набір методів або властивостей, які мають реалізувати кілька, можливо, непов'язаних класів.

Коли очікуєте, що класи реалізовуватимуть контракт різними способами та потенційно через незв'язані ієрархії.

Коли потрібно реалізувати кілька контрактів в одному класі.

Основні відмінності між абстрактними класами та інтерфейсами

Коли використовувати абстрактний клас

Коли є спільні функції або стан, які мають бути спільними для кількох похідних класів.

Якщо хочете застосувати певну базову функціональність, дозволяючи похідним класам розширювати або замінювати певні методи.

Коли між базовим класом і нащадком існує чіткий зв'язок.

Висновки

- Абстрактні класи можуть містити реалізовані методи, тоді як інтерфейси містять лише сигнатури методів.
- Класи можуть реалізовувати декілька інтерфейсів, але вони можуть успадковувати лише один абстрактний клас.
- Абстрактні класи можуть мати конструктори, а інтерфейси – ні.
- Абстрактні класи можуть мати поля та властивості, тоді як інтерфейси можуть мати лише властивості.
- Абстрактні класи зазвичай використовуються для створення базового класу для успадкування інших класів, тоді як інтерфейси використовуються для визначення контракту, який повинні реалізувати класи.

Інтерфейс IDisposable

Інтерфейс IDisposable у C# - це важливий інструмент для управління ресурсами, особливо тими, які не контролюються безпосередньо .NET Framework.

IDisposable використовується для звільнення ресурсів, які не керуються збирачем сміття .NET, таких як файлові дескриптори, мережеві з'єднання, або пам'ять, виділена з неуправляємих бібліотек.

Дозволяє явно звільнити ресурси, коли вони більше не потрібні, замість того, щоб покладатися на збирач сміття, який може звільнити їх пізніше.

Інтерфейс IDisposable містить єдиний метод Dispose(), який потрібно реалізувати в класі. У цьому методі слід розмістити код для звільнення всіх ресурсів, що використовуються класом.

Інтерфейс IDisposable

Правило перше: не застосовувати (доки це дійсно не знадобиться).

Існує лише дві ситуації, коли необхідно реалізовувати IDisposable. Подивіться на клас і визначте, чи потрібен вам цей інтерфейс:

- У класі є некеровані ресурси
- У класі є керовані (IDisposable) ресурси

Правило друге: для класу, який володіє *керованими* ресурсами, реалізуйте IDisposable (але не фіналізатор)

Правило третє: для класу, який володіє *некерованими* ресурсами, реалізуйте IDisposable та фіналізатор

Інтерфейс IDisposable

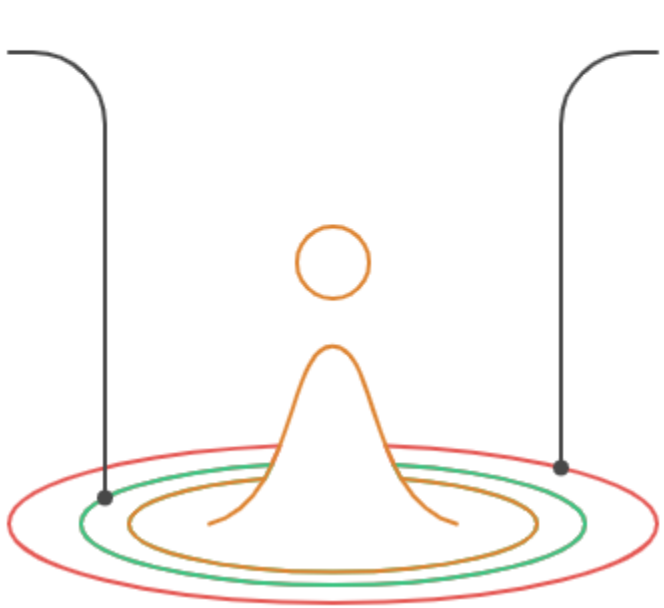
```
using System;

public sealed class Foo : IDisposable
{
    private readonly IDisposable _bar;

    public Foo()
    {
        _bar = new Bar();
    }

    public void Dispose() => _bar.Dispose();
}
```

Інтерфейс IDisposable

Керовані та некеровані ресурси в C#

Керовані ресурси

Об'єкти, які управляються CLR і звільняються автоматично збирачем сміття. Приклади: Об'єкти класів, рядки, масиви.



Некеровані ресурси

Ресурси, які потребують явного звільнення, щоб уникнути витоків пам'яті. Приклади: Дескриптори файлів, з'єднання з базами даних, графічні дескриптори, COM-об'єкти.