

ФУНДАМЕНТАЛЬНІ КОНЦЕПЦІЇ ООП. АБСТРАГУВАННЯ ДАНИХ ТА ПАРАМЕТРИЧНИЙ ПОЛІМОРФІЗМ

Лекція 05
Об'єктно-орієнтоване програмування

План лекції

- Абстрактні класи
- Інтерфейси. Робота із вбудованими інтерфейсами
- Параметричний поліморфізм. Узагальнені типи даних
- Делегати та анонімні типи

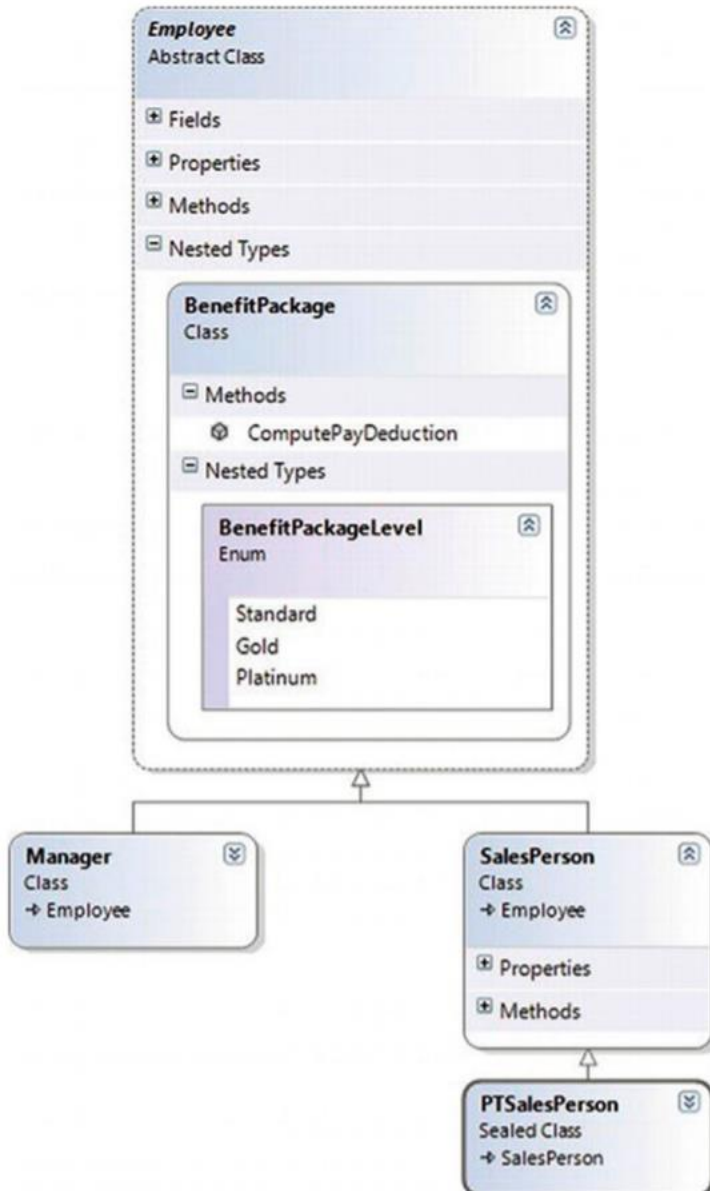


АБСТРАКТНІ КЛАСИ

Питання 5.1.

Абстрактні класи

- Зараз базовий клас Employee спроектований так, що поставляє різні дані-члени своїм нащадкам, а також пропонує два віртуальних методи (GiveBonus() і DisplayStatus()), які можуть заміщатись нащадками.
 - У прикладі базовий клас Employee має одне призначення — визначити спільні члени для всіх підкласів.
 - За всіма ознаками ви не маєте наміру дозволяти кому-небудь створювати безпосередні екземпляри цього класу, оскільки тип Employee є занадто загальним за своєю природою.
 - Багато базових класів схильні бути досить невизначеними сутностями, тут більш вдале проектне рішення – не дозволяти безпосереднє створення в коді нового об'єкта Employee.
 - Для цього в C# використовується ключове слово abstract у визначенні класу.

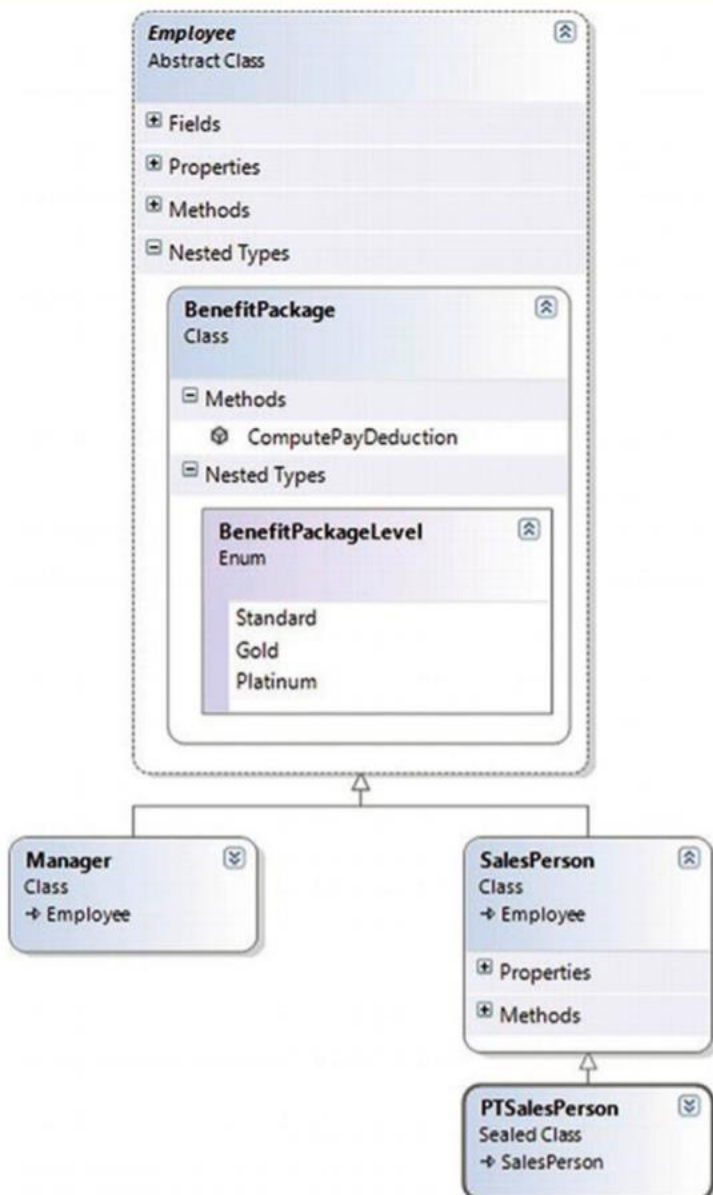


Абстрактні класи

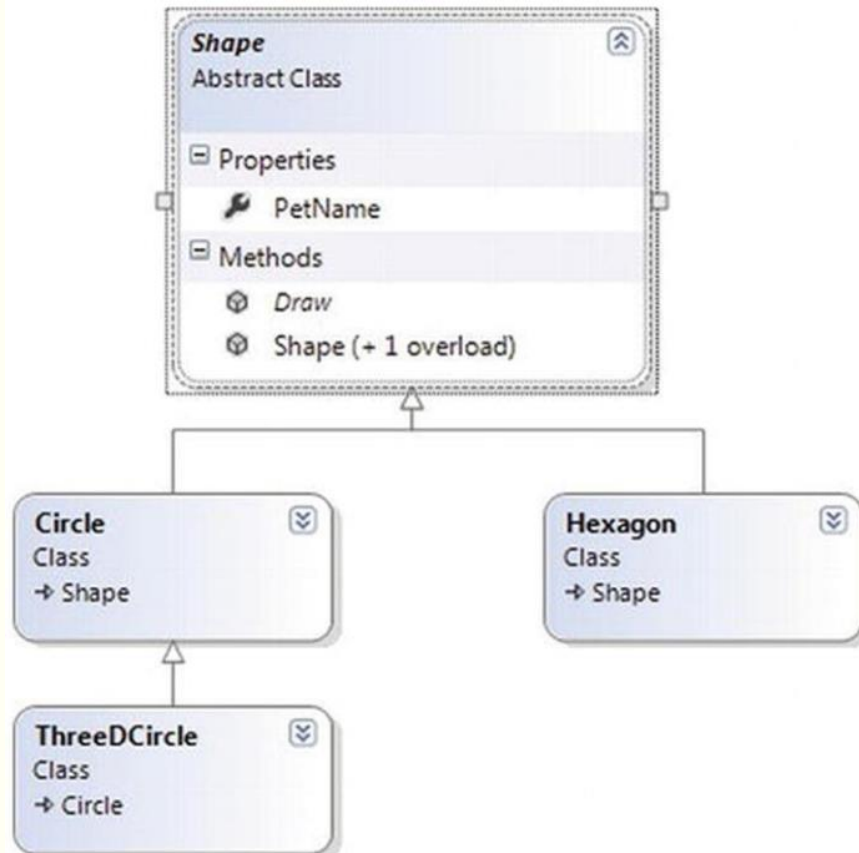
- Після цього спроба створити екземпляр класу Employee призведе до помилки на етапі компіляції:

```
// Ошибка! Нельзя создавать экземпляр  
// абстрактного класса!  
Employee X = new Employee();
```

- Хоч безпосередньо створити екземпляр абстрактного класу не можна, він все ж присутній у пам'яті, коли створено екземпляр його похідного класу.
 - Таким чином, абсолютно нормально (і прийнято) для абстрактних класів визначати довільну кількість конструкторів, що викликаються опосередковано при розміщенні в пам'яті екземплярів породжених класів.



Поняття поліморфного інтерфейсу



- Коли клас визначено як абстрактний базовий, у ньому може визначатись довільна кількість абстрактних членів.
 - Абстрактні члени можуть використовуватись повсюди, де необхідно визначити член, що не передбачає стандартної реалізації.
 - За рахунок цього нав'язується **поліморфний інтерфейс** кожному нащадку, покладаючи на них задачу реалізації конкретних деталей абстрактних методів.
- Поліморфний інтерфейс абстрактного базового класу просто посилається на його набір віртуальних і абстрактних методів.
 - Дана особливість ООП дозволяє будувати легко розширюване та гнучке програмне забезпечення.

Розглянемо реалізацію типів Circle і Hexagon

```
// Абстрактный базовый класс иерархии.
abstract class Shape
{
    public Shape(string name = "NoName")
    { PetName = name; }

    public string PetName { get; set; }

    // Единственный виртуальный метод.
    public virtual void Draw()
    {
        Console.WriteLine("Inside Shape.Draw()");
    }
}

// Circle не переопределяет Draw().
class Circle : Shape
{
    public Circle() {}
    public Circle(string name) : base(name) {}
}

// Hexagon переопределяет Draw().
class Hexagon : Shape
{
    public Hexagon() {}
    public Hexagon(string name) : base(name) {}
    public override void Draw()
    {
        Console.WriteLine("Drawing {0} the Hexagon", PetName);
    }
}
```

- Подібно до будь-якого базового класу, в Shape визначено набір членів (тут властивість PetName і метод Draw()), загальних для всіх нащадків.
 - Щоб запобігти прямому створенню екземплярів Shape, можна визначити його як абстрактний клас.
 - Також, з огляду на те, що похідні типи повинні унікальним чином реагувати на виклик методу Draw (), позначимо його як virtual і визначимо стандартну реалізацію.
 - Підкласи не є повинними заміщати віртуальні методи (як у випадку Circle).
 - Тому якщо створити екземпляр типу Hexagon і Circle, то виявиться, що Hexagon знає, як правильно "малювати" себе (або, принаймні, виводить на консоль відповідне повідомлення).

Це не особливо інтелектуальне проектне рішення для поточної ієрархії.

- Щоб змусити кожен клас заміщати метод Draw(), можна визначити Draw() як абстрактний метод класу Shape, а це означає відсутність будь-якої стандартної реалізації.

```
static void Main(string[] args)
{
    Console.WriteLine("***** Fun with Polymorphism *****\n");
    Hexagon hex = new Hexagon("Beth");
    hex.Draw();
    Circle cir = new Circle("Cindy");
    // Вызывает реализацию базового класса!
    cir.Draw();
    Console.ReadLine();
}
```

Вывод этого метода Main() выглядит следующим образом:

```
***** Fun with Polymorphism *****
Drawing Beth the Hexagon
Inside Shape.Draw()
```


Абстрактні методи

- Можуть визначатись тільки в абстрактних класах, інакше – помилка компіляції.
- Методи, відмічені як `abstract`, є чистим протоколом.
 - Вони просто визначають ім'я, вихідний тип (якщо є) і набір параметрів (за потреби).
 - Тут абстрактний клас `Shape` інформує типи-нащадки про те, що у нього є метод `Draw()`, який не приймає аргументів і нічого не повертає.
 - Про необхідні деталі повинен подбати нащадок.
- Враховуючи це, метод `Draw()` у класі `Circle` тепер повинен обов'язково заміщуватись.
 - Інакше `Circle` теж повинен бути абстрактним типом, що, очевидно, недоречно в даному прикладі.

```
class Circle : Shape
{
    public Circle() {}
    public Circle(string name) : base(name) {}
    public override void Draw()
    {
        Console.WriteLine("Drawing {0} the Circle", PetName);
    }
}
```

Демонстрація поліморфізму

```
static void Main(string[] args)
{
    Console.WriteLine("***** Fun with Polymorphism *****\n");
    // Создать массив совместимых с Shape объектов.
    Shape[] myShapes = {new Hexagon(), new Circle(), new Hexagon("Mick"),
        new Circle("Beth"), new Hexagon("Linda")};
    // Пройти в цикле по всем элементам и взаимодействовать
    // с полиморфным интерфейсом.
    foreach (Shape s in myShapes)
    {
        s.Draw();
    }
    Console.ReadLine();
}
```

```
***** Fun with Polymorphism *****
Drawing NoName the Hexagon
Drawing NoName the Circle
Drawing Mick the Hexagon
Drawing Beth the Circle
Drawing Linda the Hexagon
```

Хоча неможливо безпосередньо створювати екземпляри абстрактного базового класу (Shape), можна вільно зберігати посилання на об'єкти будь-якого підкласу в абстрактній базовій змінній.

- створений масив об'єктів Shape може зберігати об'єкти, успадковані від базового класу Shape (спроба помістити в масив об'єкти, несумісні з Shape, призводить до помилки на етапі компіляції).
 - Оскільки всі елементи в масиві myShapes дійсно породжені від Shape, всі вони підтримують один і той же поліморфний інтерфейс.
 - Здійснюючи ітерацію по масиву посилань Shape, виконавча система самостійно визначає, який конкретний тип має кожен його елемент – у цей момент викликається коректна версія методу Draw().
- Ця техніка також робить дуже простою і безпечною задачу розширення поточної ієрархії.
 - Нехай від абстрактного базового класу Shape успадковано ще п'ять класів (Triangle, Square і т.д.).
 - Завдяки поліморфному інтерфейсу код всередині циклу foreach не потребує змін, якщо компілятор побачить, що в масив myShapes поміщені тільки Shape-сумісні типи.

Приховування членів

- Мова C # надає засіб, логічно протилежний заміщенню методів, який називається **приховуванням**.
 - якщо похідний клас визначає член, який ідентичний члену, визначеному в базовому класі, то похідний клас *приховує* батьківську версію.
 - У реальному світі така ситуація найчастіше виникає при спадкуванні від класу, який створювали не ви (і не ваша команда), наприклад, у разі придбання пакета програмного забезпечення .NET у незалежного постачальника.
 - З метою ілюстрації припустимо, що ви отримали від колеги клас по імені ThreeDCircle, в якому визначено метод Draw(), який не приймає аргументів:

```
class ThreeDCircle
{
    public void Draw()
    {
        Console.WriteLine("Drawing a 3D Circle");
    }
}
```

Приховування членів

- Ви виявляєте, що ThreeDCircle "є" Circle, тому успадковуєте його від існуючого типу Circle:

```
class ThreeDCircle : Circle
{
    public void Draw()
    {
        Console.WriteLine("Drawing a 3D Circle");
    }
}
```

- Після компіляції отримуєте наступне попередження:

```
'Shapes.ThreeDCircle.Draw()' hides inherited member 'Shapes.Circle.Draw()'. To
make the current member override that implementation, add the override keyword.
Otherwise add the new keyword.
```

```
'Shapes.ThreeDCircle.Draw()' скрывает унаследованный член 'Shapes.Circle.Draw()'.
Чтобы заставить текущий член переопределить эту реализацию, добавьте ключевое
слово override. В противном случае добавьте ключевое слово new.
```

- Проблема в тому, що у вас є похідний клас (ThreeDCircle), що містить метод, який є ідентичним успадкованому методу.

Способи вирішення проблеми

- (1) оновити батьківську версію Draw(), використовуючи ключове слово override (як рекомендує компілятор).
 - Тоді тип ThreeDCircle може розширювати стандартну поведінку батьківського типу, що й було потрібно.
 - Однак якщо доступ до коду, який визначає базовий клас, відсутній (як зазвичай трапляється з бібліотеками від незалежних постачальників), то немає можливості модифікувати метод Draw(), зробивши його віртуальним.
- (2) додати ключове слово new у визначення члена Draw() породженого типу.
 - Роблячи це явно, ви встановлюєте, що реалізація похідного типу навмисно спроектована так, щоб ігнорувати батьківську версію (в реальному проекті це може допомогти, якщо зовнішнє програмне забезпечення .NET якимось чином конфліктує з вашим програмним забезпеченням).

```
class ThreeDCircle : Circle
{
    // Скрыть поле shapeName, определенное выше в иерархии.
    protected new string shapeName;

    // Скрыть любую реализацию Draw(), находящуюся выше в иерархии.
    public new void Draw()
    {
        Console.WriteLine("Drawing a 3D Circle");
    }
}
```



ДЯКУЮ ЗА УВАГУ!

Наступне питання: Інтерфейси. Принцип впровадження залежностей