Визначення типу як класу

Типи даних як результат абстрагування

- Абстрагування – вибіркове "незнання"
- Вбудовані типи **int** і **double** – наближена модель математичних сутностей, близьких до дійсності в межах своїх діапазонів значень
- Клас – визначений тип
- Типи користувача – результат абстрагування користувачем щодо об'єктів конкретної предметної області
  - **Point**: характеристика стану об'єкта – координати точки на площині; вся робота з точкою – виклик пов'язаних з класом функцій - методів
  - **Pixel**: дані об'єкта **Point** доповнити даними про колір (вже не хочемо не знати про колір) і методами, які працюють з кольором об'єкта

Common Type System

- Основа ефективного коду
- Всі типи – об’єктні, задані ієрархією класів
  System.Object – базовий клас
- Дві категорії типів стосовно виділення пам’яті та функціонування :
  - типи-значення: *примітиви, структури, переліки* – об’єкти безпосередньо містять значення
  - типи-посилання: *класи, інтерфейси, делегати, масиви* –опосередкований доступ до об’єктів у купі(*heap*) або ***null***

Загальна схема визначення типів

атрибути $_{opt}$ *модифікатори*$_{opt}$
class *ідентифікатор* : *ідентифікатор базового класу*$_{opt}$, *список інтерфейсів* $_{opt}$
{
визначення членів класу
}

атрибути $_{opt}$ *модифікатори*$_{opt}$
struct *ідентифікатор*: *список інтерфейсів* $_{opt}$
{
визначення членів класу
}

атрибути $_{opt}$ *модифікатори*$_{opt}$
interface *ідентифікатор*: *список інтерфейсів* $_{opt}$
{
оголошення членів інтерфейсу
}

Модифікатор:

- **new, abstract, sealed**
- *модифікатори доступу:* **public, protected, internal, private**

- **Анонімні типи**

System.Object

Public Methods

`bool Equals()`	Посилання - чи вказують точно на один і той же об'єкт Значення – чи типи об'єктів ідентичні і значення рівні
`int GetHashCode()`	Повертає хеш-код об'єкта для розміщення в хеш-таблиці з метою підвищення продуктивності
`Type GetType()`	Для отримання інформації про тип
`string ToString()`	За замовчуванням повертає назву об'єкта, в похідних класах може перевизначатися для текстового представлення об'єкта

Protected Methods

`void Finalize()`	Може викликатися при звільненні пам'яті, яку займав об'єкт
`Object MemberwiseClone()`	Копіювання посилання (а *shallow copy)* без утворення дійсної копії *(* а *deep copy)* об'єкта в купі

Протокол класу

- **Протокол класу** – *логічне* об'єднання (інкапсуляція) даних, методів та інших засобів обробки даних в один логічний модуль (a logical unit)
- **Протокол є визначенням класу**
- **Протокол** задає scope класу, тому всі його члени доступні всередині методів
- **Визначення методу** – у протоколі
- **Об'єкт** – екземпляр (an object or instance) визначеного класом типу, містить лише поля (і деякі спеціальні "службові" дані)
- **Частковий протокол** – частковий (**partial**) опис, який може бути поповнений у іншому місці того ж або іншого файла

Члени класу

- **Поля** – змінні, які зберігають дані про стан об’єкта
- **Методи** – код, який задає дії над даними об’єктів
- **Властивості** – методи, які з точки зору клієнта класу виглядають як поля, а насправді надають варіанти опосередкованого доступу до даних об’єкта; реалізуються аксесорними методами **get** і **set**.
- **Константи** – поля, значення яких є однаковими в кожного об’єкта
- **Індексатори** – подібні до методів засоби доступу до полів об’єкта як до елементів деякого масиву за значенням порядкового індексу; реалізуються аксесорними методами **get** і **set**
- **Делегати** – типи, спеціалізовані **delegate**; їхні екземпляри інкапсулюють список виклику, елементи якого (один або кілька) є методами, доступними для виклику, можуть призначатися в процесі виконання програми
- **Події** – спеціалізовані **event** екземпляри-делегати; разом з оголошенням свого типу-делегата вони є **public-**специфіковані класом, об’єкти якого через делегати-події викликтимуть потрібний метод (обробітник події) іншого класу, який в своїх методах підписався на конкретну подію
- **Оператори** – перевантажені для класу стандартні оператори
- **Вкладені (nested) типи** – як правило, **private**-специфіковані для створення об’єктів лише для внутрішнього використання

Модифікатори доступу

- Визначають рівень доступу *(accessibility level)* до члена класу :
  - **public** – необмежений доступ; член класу доступний як в межах своєї збірки, так і в інших збірках за умови посилання на зазначену збірку
  - **protected** – член класу недоступний скрізь поза визначенням класу крім ієрархії похідних класів
  - **private** – член класу може використовуватися лише в методах свого класу
  - **internal** – поза визначенням класу член класу доступний лише в збірці, де знаходиться визначення класу
  - **protected internal** – член класу доступний у збірці, де знаходиться визначення класу, або в ієрархії похідних класів
  - **private protected** – член класу доступний лише у збірці, де знаходиться визначення класу, у коді самого класу або в ієрархії похідних класів
- Визначають область доступності *(accessibility domain)* визначень типів і їхніх членів
- Доступ для типів і їх членів задається явно або неявно за замовчуванням. За замовчуванням:
  - типи верхнього рівня в збірці – **public**
  - вкладені типи і члени типів – **private**

Поля класу. Ініціалізація

- Ініціалізація за замовчуванням :
  - 0 для числових типів
  - false для bool
  - '\x0000' для char
  - null для посилань
- Ініціалізація полів при оголошенні
- Ініціалізація в конструкторах

instance fields →

```
class Rational
{
    public int numerator;
    public int denominator = 1;
   ...
}
```

set to default values →

```
Rational a = new Rational();
```

a →

numerator	0
denominator	1

- Використання об’єктного ініціалізатора

```
var number = new Rational ( ){ numerator = 10, denominator = 17 };
```

Конструктори

- Визначення конструктора

модифікатори_доступу Id класу (*список аргументів*$_{opt}$) : ініціалізатори $_{opt}$ {....}

- Утворення об'єкта

Id класу Id об'єкта = new Id класу (*список аргументів*$_{opt}$);

- Ініціалізатори: this(...), base(...)
- Особливості C#
  - може бути кілька конструкторів
  - лише при повній відсутності компілятор генерує конструктор за замовчуванням
  - конструктор копіювання компілятор не генерує за замовчуванням
- Expression body definition: member => expression;
- Статичний к-тор (закритий за замовчуванням)

static Id класу (){...}

- Закриті к-тори
- Ініціалізація readonly полів

Поля readonly та const

- **readonly** – тільки для читання
  - ініціалізація – в конструкторі або ініцілізатором в оголошенні
  - може бути довільного типу
  - після виконання конструктора:
    поле-значення залишається незмінним
    поле-посилання завжди посилається на той самий об'єкт, сам цей об'єкт може змінюватися
- **const** за замовчуванням – статичні
  - ініціалізація в оголошенні **під час компілювання**
  - лише вбудованих типів, **string** та **enum**
  - можливі вирази з константами, визначеними в програмі

```
class MagicNumbers
{
    public const double pi = 3.1415;
    public readonly int numb;
    public readonly float infimum = 0;

    public MagicNumbers(int init) { numb = init; }
}
```

```
MagicNumbers mg = new MagicNumbers(45);
Console.WriteLine($"pi = {MagicNumbers.pi}, everything else = {mg.numb}");
```

Статичні члени класу: поля

- Спільні для всіх об’єктів одного класу дані в одному екземплярі
- Утворення і ініціалізація при завантаженні програми (CLR за замовчуванням обнулює)
- Обробка незалежно від екземплярів
- Доступ через екземпляр заборонений (в т.ч. до статичних методів)
- Константи по суті є статичними полями, але явно не оголошуються статичними
- Існують до кінця виконання програми

```
class Rectangle
{
    static Color defaultColor;  // once per class
    static readonly int scale;  // -- " --
    int x, y, width, height;    // once per object
  ...
}
```

Статичні члени класу: методи

- Виклик за назвою класу (без врахування існування об’єктів)
- Можуть використовувати лише статичні дані класу
- Статичний конструктор викликається до утворення першого об’єкта класу, першого виклику статичного методу чи доступу до статичного поля
- **Main()** – статичний метод

```
class Rectangle
{
    static Color defaultColor;
    public static void ResetColor()
    {
        defaultColor = Color.white;
    }
}
```

Доступ з Rectangle

```
...defaultColor
... scale
   ResetColor();
```

Доступ з інших класів

```
Rectangle.defaultColor
Rectangle.scale
Rectangle.ResetColor();
```

Визначення типу як struct

```
struct Point {
    int x, y;
    public Point(int xx, int yy) {x = xx;  y = yy; }
    public MoveTo (int x, int y) {...}
}
```

- objects are allocated on the stack
- efficient, low memory consumption, no burden for the garbage collector
  live only as long as their container (not suitable for dynamic data structures)
- Can be allocated with **new**
- Fields must not be initialized at their declaration
- Parameterless (default) construcor cannot be declared explicitly (one is provided automatically by the compiler)
- A constructor of a **struct** must initialize all fields
- Can neither inherit nor be inherited, but can implement interfaces