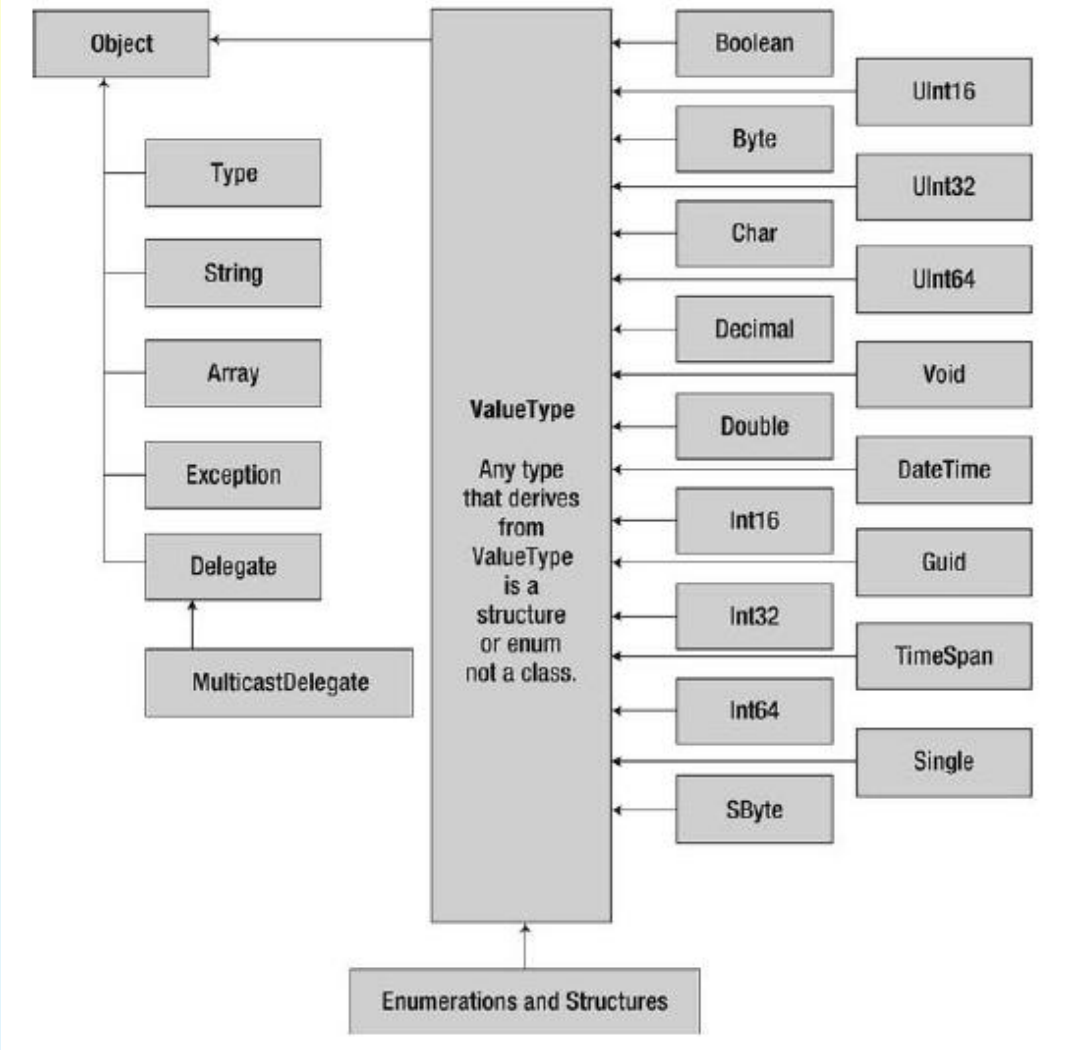
Розробка функціональності класу

Клакович Л.М.

- **Перевизначення методів `Object`**
- **Перевантаження операторів**
- **Особливості конструювання класів-агрегатів**
- **Похідні типи - наслідування реалізації**

Клас object. Ієрархія типів



Class Object

```
public class Object
{
  // Virtual members.
  public virtual bool Equals(object obj);
  protected virtual void Finalize();
  public virtual int GetHashCode();
  public virtual string ToString();

  // Instance-level, nonvirtual members.
  public Type GetType();
  protected object MemberwiseClone();

  // Static members.
  public static bool Equals(object objA, object objB);
  public static bool ReferenceEquals(object objA, object objB);
}
```

System.Object

Public Methods of the *System.Object*

bool Equals()	Для типу-посилання - чи вказують точно на один і той же об'єкт Для типу-значення – чи типи об'єктів ідентичні та 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 of the *System.Object*

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

Object.Equals

```
class Object {
    public virtual Boolean Equals(Object obj) {
        // If both references point to the same
        // object, they must be equal.
        if (this == obj) return true;
        // Assume that the objects are not equal.
        return false;
    }
    public static Boolean Equals (Object objA, Object objB)
    {
        // If objA and objB refer to the same object, return true.
        if (objA == objB) return true;
        // If objA or objB is null, they can't be equal, so return false.
        if ((objA == null) || (objB == null)) return false;
        // Ask objA if objB is equal to it, and return the result.
        return objA.Equals(objB);
    }
    ....
}
```

Point_s- порівняння на рівність

```
class Point_s {
      ...
   public override bool Equals (Object obj) {
      if(obj==null) return false;
      if(this.GetType()!=obj.GetType() ) return false;
      Point_s p=(Point_s) obj;
      if(x==p.x&&y==p.y)return true;
      return false;
   }
   public static bool operator ==(Point_s p1, Point_s p2) {
      return Object.Equals(p1, p2);
   }
   public static bool operator !=(Point_s p1, Point_s p2) {
      return !(p1== p2);
   }
}
```

Point_s - використання

```
Point_s pt = new Point_s();
Console.WriteLine("pt : {0}", pt);
Point_s pt1 = new Point_s(6,16);
Console.WriteLine("pt1: {0}", pt1);

Point_s ptReal = new Point_s(10);
Console.WriteLine("ptReal: {0}", ptReal);

Point_s pt2 = new Point_s(10, 20);
Point_s pt3 = new Point_s(10, 20);

Console.WriteLine("pt2==pt3:{0}", pt2 == pt3);
Console.WriteLine("pt2==pt3:{0}", pt3.Equals(pt2));
Console.WriteLine("pt2==pt3:{0}", Equals(pt3, pt2));
```

Перевантаження операторів

Шаблон оператора

```
public static retval operator op(object1 [,object2]){…}
```

Вимоги до перевантажуваних операторів
(перевантажується для **типу T**):

- повинен бути членом **class** або **struct**
- завжди `public static`
- для унарного оператора – тип аргумента є типом **T**
- для бінарного оператора – тип одного з аргументів **повинен** бути типом **T**

binary + →

int+Point →

Point+int →

unary + →

error →

```
class Point
{
  private int x;
  private int y;

  public static Point operator+(Point p, Point q){
    return new Point(p.x + q.x, p.y + q.y);
  }
  public static Point operator +(int a, Point p){
    return new Point(a + p.x, a + p.y);
  }
  public static Point operator+(Point p, int a){
    return new Point(p.x + a, p.y + a);
  }
  public static Point operator+(Point p){
    return new Point(p.x, p.y);
  }
  public static Point operator+(int x, int y){
    return new Point(x, y);
  }

...
}
```

Обмеження

- **Лише деякі оператори можуть бути перевантажені:**
  - унарні: `+ - ! ~ ++ -- true false`
  - бінарні: `+ - * / % & | ^ << >> == != > < >= <=`
- **Не можна**
  - перевантажувати оператор **new**
  - змінювати фіксований пріоритет
  - змінювати фіксовану кількість аргументів
  - перевантажувати префіксні/постфіксні версії окремо
  - Передавати параметри оператора через `ref` або `out`

Пари операторів

- Вимагається перевантаження деяких операторів парами:
  1) `==` `!=`
  2) `>` `<`
  3) `>=` `<=`
  4) `true` `false`

рівність →

```
struct Point
{
  public static bool operator==(Point p, Point q)
  {
    return p.x == q.x && p.y == q.y;
  }
  ...
}
```

error, повинен бути перевантажений оператор нерівності →

Комбіноване присвоєння

- Оператор комбінованого присвоєння **генерується автоматично**, коли відповідний бінарний оператор перевантажується

define binary+ →

```
struct Point
{
  public static Point operator+(Point p, Point q)
  {
    return new Point(p.x + q.x, p.y + q.y);
  }
  ...
}
```

```
Point a = new Point(1, 2);
Point b = new Point(3, 4);
Point c;

c = a + b;

c += b;
```

operator+ →

operator+= →

Оператори та їх реалізація

C# Operator	Special Method Name	Suggested CLS-Compliant Method Name
+	op_UnaryPlus	Plus
-	op_UnaryNegation	Negate
~	op_OnesComplement	OnesComplement
++	op_Increment	Increment
—	op_Decrement	Decrement
+	op_Addition	Add
+=	op_AdditionAssignment	Add
-	op_Subtraction	Subtract
-=	op_SubtractionAssignment	Subtract
*	op_Multiply	Multiply
*=	op_MultiplicationAssignment	Multiply
/	op_Division	Divide
/=	op_DivisionAssignment	Divide
%	op_Modulus	Mod
%=	op_ModulusAssignment	Mod
^	op_ExclusiveOr	Xor
^=	op_ExclusiveOrAssignment	Xor

Оператори перетворення типу

Шаблон оператора перетворення типу

```
public static implicit operator conv-type-out (conv-type-in operand) - неявне перетворення
public static explicit operator conv-type-out (conv-type-in operand) - явне перетворення
```

- назва оператора є типом до якого здійснюється конвертування
- параметр оператора визначає тип з якого конвертується

```
class Point
{
   ...
   public static explicit operator Point(int x) { ... }
}                  implicit
```

```
class Point {
        public static implicit operator Point(int x)
        {
                return new Point(x);
        }
        public static explicit operator int( Point p)
        {
                return p.x;
        }
…
}
```

- **Конвертування відбувається при:**
  - присвоєнні
  - передачі параметрів
  - поверненні значення методом

Неявне конвертування →

Явне конвертування →

```
Point p;
p=3;

int x;
Point q=new Point(5,0);
x=(int) q;
```

- **Обмеження щодо операторів конвертування:**
  - повинні бути **`public static`**
  - можуть мати тільки один параметр
  - параметри не можна передавати як **`ref`** чи **`out`**
  - Параметр або тип що повертається повинен бути типом, в якому визначений цей оператор

Point_cl – утворення копій

Види копіювання об’єктів:

- Поверхневе (shallow)
- Глибоке (deep) – утворення нового об’єкта з новими підоб’єктами
- MemberwiseClone() – метод з класу object для поверхневого копіювання
- ICloneable інтерфейс для здійснення глибокого копіювання

```
class Point: ICloneable
{
      public object Clone()
      {
        return new Point(x, y);
      }
}
```

```
      Point pt = new Point(10,10);
      Console.WriteLine("pt : {0}", pt);

      Point pt1 = (Point)pt.Clone();
      Console.WriteLine("pt1: {0}", pt1);
```

class Line

- Рекомендується не використовувати Clone(), оскільки він не несе інформацію про глибину копіювання, а реалізовувати власне копіювання, наприклад через конструктор

```
class Line:ICloneable {
        Point beg;
        Point end;
        public Line() {
            beg = new Point(); end = new Point();
        }
        public Line(Point b, Point e){
            beg = (Point)b.Clone();
            end = (Point)e.Clone();
        }
        public override string ToString(){
             return "Line["+beg+", "+end+ "]";
        }
        public object Clone() {
          return new Line(beg, end);
       }
}
```

Line - використання

```
Line l = new Line();
Console.WriteLine("l : {0}", l);

Line l1 = new Line(new Point(10,20),new Point(40,50));
Console.WriteLine("l1 : {0}", l1);

Line lcl =(Line)l.Clone();
Console.WriteLine("lcl : {0}", lcl);
```

```
l : Line [(0, 0), (0, 0)]
l1 : Line [(10, 20), (40, 50)]
lcl : Line[(0, 0), (0, 0)]
```