Objektové programování I

Strukturované a objektové programování, objekt, třída, základní objektové vlastnosti, třídy abstraktní, zapouzdřené a rozhraní

Joseph Soukup – P4

Strukturované a objektové programování

- Strukturované
  - Skoky
  - Vše v jednom místě
- Objektové
  - Objekty, Třídy
  - Často více souborů

Objekt

- Je v podstatě blok paměti, který byl přidělen a nakonfigurován podle podrobného plánu
- Program může vytvořit mnoho objektů stejné třídy
- Objekty se také nazývají instance a mohou být uloženy buď v pojmenované proměnné, nebo v poli nebo kolekci
- Kód klienta je kód, který používá tyto proměnné pro volání metod a přístup k veřejným vlastnostem objektu
- **Instance struktury vs. Instance třídy**
  - Vzhledem k tomu, že třídy jsou odkazové typy, proměnná objektu třídy obsahuje odkaz na adresu objektu na spravované haldě
  - Pokud je druhý objekt stejného typu přiřazen k prvnímu objektu, pak obě proměnné odkazují na objekt na dané adrese
  - Instance třídy jsou vytvořeny pomocí operátoru new
  - 

```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }
    // Other properties, methods, events...
}

class Program
{
    static void Main()
    {
        Person person1 = new Person("Leopold", 6);
        Console.WriteLine("person1 Name = {0} Age = {1}", person1.Name, person1.Age);

        // Declare new person, assign person1 to it.
        Person person2 = person1;

        // Change the name of person2, and person1 also changes.
        person2.Name = "Molly";
        person2.Age = 16;

        Console.WriteLine("person2 Name = {0} Age = {1}", person2.Name, person2.Age);
        Console.WriteLine("person1 Name = {0} Age = {1}", person1.Name, person1.Age);

        // Keep the console open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
/*
    Output:
    person1 Name = Leopold Age = 6
    person2 Name = Molly Age = 16
    person1 Name = Molly Age = 16
*/
```

- Vzhledem k tomu, že struktury jsou typy hodnot, proměnná objektu struct obsahuje kopii celého objektu
- Instance struktur lze také vytvořit pomocí new operátoru, ale to není vyžadováno
-

```C#
public struct Person
{
    public string Name;
    public int Age;
    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }
}

public class Application
{
    static void Main()
    {
        // Create  struct instance and initialize by using "new".
        // Memory is allocated on thread stack.
        Person p1 = new Person("Alex", 9);
        Console.WriteLine("p1 Name = {0} Age = {1}", p1.Name, p1.Age);

        // Create  new struct object. Note that  struct can be initialized
        // without using "new".
        Person p2 = p1;

        // Assign values to p2 members.
        p2.Name = "Spencer";
        p2.Age = 7;
        Console.WriteLine("p2 Name = {0} Age = {1}", p2.Name, p2.Age);

        // p1 values remain unchanged because p2 is  copy.
        Console.WriteLine("p1 Name = {0} Age = {1}", p1.Name, p1.Age);

        // Keep the console open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
/*
  Output:
    p1 Name = Alex Age = 9
    p2 Name = Spencer Age = 7
    p1 Name = Alex Age = 9
*/
```

- **Identita objektu vs. Hodnota rovnosti**
  - Když porovnáváme dva objekty pro rovnost, musíme rozlišovat zda chcete zjistit, zda tyto dvě proměnné představují stejný objekt v paměti nebo zda jsou hodnoty jednoho nebo více jejich polí ekvivalent
  - Pokud hodláte porovnat hodnoty, je nutné vzít v úvahu, zda jsou objekty instance typů hodnot (struktury) nebo typy odkazů (třídy, delegáti, pole)
    - Stejné umístění v paměti:
      - Equals
    - Stejné hodnoty:
      - ValueType.Equals

```
// Person is defined in the previous example.

//public struct Person
//{
//    public string Name;
//    public int Age;
//    public Person(string name, int age)
//    {
//        Name = name;
//        Age = age;
//    }
//}

Person p1 = new Person("Wallace", 75);
Person p2;
p2.Name = "Wallace";
p2.Age = 75;

if (p2.Equals(p1))
    Console.WriteLine("p2 and p1 have the same values.");

// Output: p2 and p1 have the same values.
```

Třída

- Odkazový typ
- V době běhu, pokud deklaruje proměnnou typu odkazu, proměnná obsahuje hodnota null, dokud explicitně nevytvoříte instanci třídy pomocí operátoru New, nebo přiřadíte objekt kompatibilního typu, který mohl být vytvořen jinde

```
//Declaring an object of type MyClass.
MyClass mc = new MyClass();

//Declaring another object of the same type, assigning it the value of the first object.
```

- Když je objekt vytvořen, je k dispozici dostatek paměti na spravované haldě pro daný objekt a proměnná obsahuje pouze odkaz na umístění daného objektu
- **Deklarace:**
  - Třídy jsou deklarovány pomocí klíčového slova Class následovaný jedinečným identifikátorem

```
//[access modifier] - [class] - [identifier]
public class Customer
{
   // Fields, properties, methods and events go here...
}
```

  - Public – kdokoliv může vytvořit třídu
- **Vytváření objektů**
  - Pomocí klíčového slova New následovaný názvem třídy

```
Customer object1 = new Customer();
```

- **Internal**
  - Přístupné pouze ve stejném souboru
  - Třída, která je uvnitř jiné třídy (spojové struktury)
- **Sealed**
  - Nedá se z nich dědit
  - Mohou být instancovány

Základní objektové vlastnosti

- Objekt si pamatuje svůj stav (v podobě dat/atributů)
- Sdílí přistup ke svým položkám
- Vlastnostmi objektů jsou:
    - Proměnné
    - Metody
- Vlastnosti objektů – proměnné i metody – je třeba deklarovat
- **Proměnné:**
    - Jsou nositeli „pasivních“ vlastností, jakýchsi atributů, charakteristik objektů
    - De facto jde o datové hodnoty svázané (zapouzdřené) v objektu
- **Metody:**
    - Jsou nositeli „výkonných“ vlastností, „dovedností“ objektů
    - De facto jde o funkce (procedury) pracující (převážně) nad proměnnými objektu

Statické třídy

- Stejná jako nestatická třída -> pouze nejde vytvořit (nelze použít operátor New)
- Přístup ke členům statické třídy pomocí samotného názvu třídy
- ```
  UtilityClass.MethodA();
  ```
- Lze použít jako pohodlný kontejner pro sady metod, které fungují pouze na vstupních parametrech a nemusejí získávat ani nastavovat žádná interní pole instance
- Příklady:
    - Console
    - Math
- Hlavní funkce statické třídy:
    - Obsahuje pouze statické členy
    - Nelze vytvořit instance
    - Je zapečetěný
    - Nemůže obsahovat konstruktory instancí
- Statické atributy / metody:
    - Nejsou závislé na instanci třídy
    - Pro všechny stejný

Abstraktní třída

- Abstract umožňuje vytvářet třídy a členy třídy, které jsou neúplné a které musí být implementovány v odvozené třídě
- Třídy lze deklarovat jako abstraktní vložením klíčového slova abstract před definici třídy:
    - ```
      public abstract class A
      {
          // Class members here.
      }
      ```
- Nelze vytvořit instanci
- Poskytnutí společné definice základní třídy, kterou může sdílet více odvozených tříd

- Knihovna tříd může například definovat abstraktní třídu, která se používá jako parametr pro mnoho jeho funkcí, a vyžadovat programátory, kteří používají tuto knihovnu k poskytnutí vlastní implementace třídy vytvořením odvozené třídy
- Abstraktní metody
  - ```
    public abstract class A
    {
        public abstract void DoWork(int i);
    }
    ```
- Abstraktní metody nemají implementaci
- Pokud abstraktní třída dědí virtuální metoda ze základní třídy, abstraktní třída může přepsat virtuální metody abstraktní metodou
  - ```
    // compile with: -target:library
    public class D
    {
        public virtual void DoWork(int i)
        {
            // Original implementation.
        }
    }

    public abstract class E : D
    {
        public abstract override void DoWork(int i);
    }

    public class F : E
    {
        public override void DoWork(int i)
        {
            // New implementation.
        }
    }
    ```

Zapouzdření

- Využíváno jako mechanismus ukrývání hodnot či stavů strukturovaných dat objektu uvnitř třídy
- Zamezuje neautorizovaným subjektům k přímému přístupu k daným atributům
  - **Public**
    - Modifikátor přístupu pro typy a členy typů
    - Veřejný přístup je nejpřísnější úroveň přístupu
    - Pro přístup k veřejným členům neexistují žádná omezení
      - ```
        class PointTest
        {
            public int x;
            public int y;
        }

        class Program
        {
            static void Main()
            {
                var p = new PointTest();
                // Direct access to public members.
                p.x = 10;
                p.y = 15;
                Console.WriteLine($"x = {p.x}, y = {p.y}");
            }
        }
        // Output: x = 10, y = 15
        ```
  - **Private**
    - Modifikátor přístupu pro typy a členy typů
        ```

        - Nejnižší úroveň přístupu
        - Soukromé členy jsou přístupné pouze v těle třídy nebo struktury, ve které jsou deklarovány
        - Mohou přistupovat i vnořené typy ve stejném těle
        -
```
class Employee2
{
    private string name = "FirstName, LastName";
    private double salary = 100.0;

    public string GetName()
    {
        return name;
    }

    public double Salary
    {
        get { return salary; }
    }
}

class PrivateTest
{
    static void Main()
    {
        var e = new Employee2();

        // The data members are inaccessible (private), so
        // they can't be accessed like this:
        //    string n = e.name;
        //    double s = e.salary;

        // 'name' is indirectly accessed via method:
        string n = e.GetName();

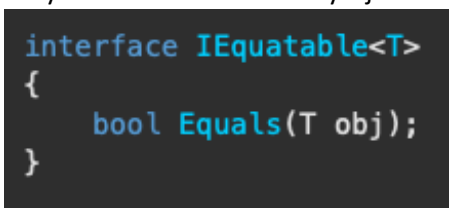
        // 'salary' is indirectly accessed via property
        double s = e.Salary;
    }
}
```
    - **Protected**
        - Modifikátor přístupu pro typy a členy typů
        - Chráněný člen základní třídy je přístupný v odvozené třídě pouze v případě, že k přístupu dojde prostřednictvím odvozené třídy typu
        -
```
class Point
{
    protected int x;
    protected int y;
}

class DerivedPoint: Point
{
    static void Main()
    {
        var dpoint = new DerivedPoint();

        // Direct access to protected members.
        dpoint.x = 10;
        dpoint.y = 15;
        Console.WriteLine($"x = {dpoint.x}, y = {dpoint.y}");
    }
}
// Output: x = 10, y = 15
```

Rozhraní

- Obsahuje definice pro skupinu souvisejících funkcí, které musí implementovat neabstraktní Třída nebo Struktura
- Rozhraní může definovat static metody, které musí mít implementaci

- Pomocí rozhraní můžeme například zahrnout chování z více zdrojů ve třídě
- Kromě toho je nutné použít rozhraní, pokud chceme simulovat dědičnost pro struktury, protože nemohou být ve skutečnosti děděny z jiné struktury nebo třídy
    - 
```
interface IEquatable<T>
{
    bool Equals(T obj);
}
```
- Rozhraní mohou obsahovat metody, instance, vlastnosti, události, indexery nebo libovolnou kombinaci těchto čtyř typů členů