Úvod do předmětu

- Literatura, osnova předmětu – viz. cvičení

OOP

Objekt

- Základní prvek OOP sw inženýrství
- = model reálných objektů (věcí) – člověk, auto, okno (ve windows), slovník,
- = model abstraktní věci: objednávka, účet u telefonní společnosti, atd.

Proč OOP?

- Procedurální programování:
  - Globální data (v `Main()`) a funkce s nimi manipulující – nepřehledné

```
// Příklad: dva slovníky
static void Main(string[] args)
{
        string[] czDe_Cz;
        string[] czDe_De;
        string[] czEn_Cz;
        string[] czEn_En;
        int czEnPocetSlov;
        int czDePocetSlov;
}
static void PridejSlovicko(string[] kamCz, string kamJazyk2[],
        string cz, string j2, ref int pocetSlov)
{ // kod metody }
static void SetrigSlovnik(bool jazyk, string[] cz, string j2[],
        int pocetSlov)
{ // kod metody }
```

- OOP
  - Sjednocení dat a funkcí pod jednu střechu – objekt
  - Objekt = black box:
    - Pro práci není nutné vidět vnitřní funkčnost – komunikace prostřednictvím definovaného rozhraní (sw = Metody)
    - Změna vnitř implementace nezpůsobí kolaps programu

Třída

- = šablona pro vytvoření objektu (popisuje jej)
- Příklad:
  - Formička na cukroví = třída
  - Jednotlivé kousky cukroví z formičky = objekty

Atributy

- = Charakteristiky (stav) objektu
- Objekt „člověk“: Váha, výška, věk, pohlaví …

- Objekt „účet u telefonní společnosti“: tarif, provolané minuty, splaceno / nesplaceno, zákazník (= také objekt)…

Metody

- Určují chování objektů

Třídy a objekty v C#

1. Definice třídy
2. Vytvoření proměnné od třídy = objekt = instance třídy

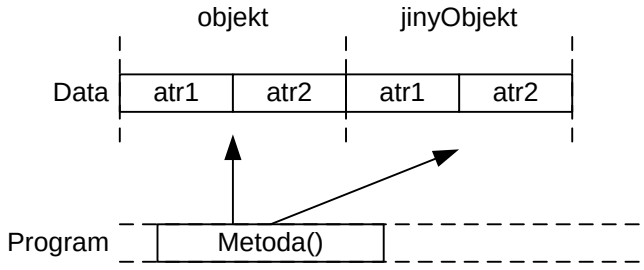
- Atributy = proměnné (členské proměnné)

(příklad není funkční)

```
class Trida                 // definice tridy
{
        int atribut;
        int atribut2;
        void Metoda()
        {
                Console.WriteLine("Ahoj");
        }
}

static void Main(string[] args)
{
        Trida objekt = new Trida();      // vytvoreni objektu
        Trida jinyObjekt = new Trida();
        objekt.atribut = 10;             // manipulace s atributem
        jinyObjekt.atribut = 55;
        objekt.Metoda();                 // objekt se projevuje
}
```

- Dále člen třídy = vše, co je uvnitř (definice) třídy (atributy, metody, …)
- Jak to vypadá v paměti?
    - Každý objekt má svou verzi atributů, metody jsou společné:



Zapouzdření (encapsulation)

1. Integrace atributů a metod pod jednu střechu (objekt) – širší smysl
2. = ochrana konzistence objektu před nechtěnou manipulací zvenčí – užší smysl

- Provedení: modifikátory přístupu + kódová zeď (metody, properties – viz. později)

Modifikátory přístupu

- Určuje, jaká část programu má ke členům třídy přístup
    - `public` = veřejný, člen je vidět zvenčí třídy
    - `private` = soukromý, člen není vidět zvenčí

- `protected` – viz dědičnost později

- není uveden = `private`

```
class Trida
{
	private int atribut;
	public void Metoda()
	{
		Console.WriteLine("Ahoj");
	}
}

static void Main(string[] args)
{
	Trida objekt = new Trida();
	objekt.atribut = 10;		// CHYBA: přístup k private položce
	objekt.Metoda();
}
```

```
error CS0122: Trida.atribut' is inaccessible due to its protection level
```

Kódová zeď

- přímá manipulace s atributy je nebezpečná – nelze zaručit konzistenci objektu

```
class Osoba
{
	public int vek;
}
franta.vek = -10238;	// ???
```

- zásada: žádný atribut přímý přístup (vše `private`), jen pomocí metod
  - nastavení atributu – set metoda, setter, např. `SetVek()`
  - čtení hodnoty atributu – get metoda, getter, např. `GetVek()`
- dle metod pro přístup k atributům:
  - atribut pro čtení i zápis – setter + getter
  - atribut jen pro čtení – jen getter
  - atribut jen pro zápis – jen setter, nedělat, svědčí o špatném návrhu!!!

```
class Osoba
{
	private int vek;
	public void SetVek(int v)
	{
		vek = v;
	}
	public int GetVek()
	{
		return vek;
	}
}
// v Main()
Osoba franta = new Osoba();
franta.SetVek(50);
Console.WriteLine(franta.GetVek());
```

Vlastnosti (properties)

- náhrada Set a Get metod

- filosofie: Vlastnost = metoda uvnitř, přístup k atributu „přímo“ (jako k proměnné)
- Vlastnost = metoda, uvnitř dva (jeden) spec. blok
- Zásada: jméno Vlastnosti = jméno atributu, jen s velkým písmenem

```
class Osoba
{
	private int vek;

	public int Vek
	{
		get
		{
			return vek;
		}
		set
		{
			vek = value;
		}
}
// v Main()
Osoba franta = new Osoba();
franta.Vek = 50;
Console.WriteLine(franta.Vek);
```

- Pouze `get` = Vlastnost jen pro čtení

```
	// vlastnost jen pro cteni
	private double suma;
	public double Suma
	{
		get { return suma; }
	}
```

- Dopředu: Vlastnost = metoda, lze `static`, `virtual` / `override`, `abstract`
- Již používáme – `Length` u polí
- V BCL .NET kombinace Vlastnosti + Set a Get metod
- Vlastnost lze vytvořit od jakéhokoli datového typu – i polí, tříd, výčtů…

```
class Trida
{
	private int[] pole = new int[10];	// inicializace viz. pozdeji
	public int[] Pole
	{
		get { return pole; }
		set { pole = value; }
	}
}
// v Main()
Trida objekt = new Trida();
objekt.Pole[5] = 20;
Console.WriteLine(objekt.Pole[0]);
```

Vedlejší efekty při manipulaci s atributy

- Náhrada přímého přístupu k atributu metodou – lze před nastavením (čtením) provést další operace, např. kontrola hodnoty, převod na jiné jednotky…

```
class Trida
{
	// Vlastnost s 2x vedlejsim efektem
	private double castka;
```

```
        public double Castka
        {
                get { return castka; }
                set
                {
                        if (castka > 0.0)
                        {
                                castka = value;
                                suma += castka;
                        }
                }
        }
}
```

Reference `this`

- = odkaz na aktuální instanci třídy („sebe samu“)
- Výhodné použití:
  - zamezení konfliktu identifikátorů:

```
class Osoba
{
        private int vek;
        public void SetVek(int vek)
        {
                this.vek = vek;     // this musí být
        }
        …
}
```

  - Zpřehlednění programu (odlišení členských proměnných od lokálních proměnných v dlouhých metodách)

```
class Trida
{
        int a, b, s, x;
        public void Metoda(double c, int u)
        {
                int k;
                this.a = c + 2 * this.s;  // this nemusí být
                k = this.x + u;
        }
}
```

  - Generování událostí (viz později – předává se odkaz na původce události)

```
Udalost(this, new EventArgs());
```

- Lze použít jen uvnitř třídy (při její definici – logické)
- `this` = odkaz na instanci – nelze použít na statické členy (viz později)

Vznik objektů

- Při vzniku objekt nutné (obvykle) provést jeho počáteční nastavení
- Způsoby:
  - Konstruktory
  - Přímá inicializace atributů

Konstruktor

- = `public` metoda volaná automaticky při deklaraci objektu (vznik objektu), účel = nastavení objektu do výchozího stavu
- Stejný název jako třída, nesmí nic vracet (ani `void`), může mít parametry
  - Bezparametrický konstruktor

```
class Trida
{
	int atr1;
	string atr2;

	public Trida()
	{
		atr1 = 10;
		atr2 ="text";
	}
}
// v Main()
Trida objekt = new Trida();
```

Volání konstruktoru

  - Konstruktor s parametry

```
class Trida
{
	int atr1;
	string atr2;

	public Trida(int atr1, string atr2)
	{
		this.atr1 = atr1;
		this.atr2 = atr2;
	}
}
// v Main()
Trida objekt = new Trida(10, "text");
```

- není nutné jej definovat – potom překladač vytvoří implicitně bezparametrický konstruktor, který nic nedělá
- lze jej přetížit

```
class Trida
{
	double[] pole;
	int atr1;
	string atr2;

	public Trida()	// prideli objektu vychozi funkcnost
	{
		atr1 = 10;
		atr2 = "text";
		pole = new double[100];
	}

	public Trida(int atr1, string atr2)	// nebo si ji urci uzivatel
	{
		this.atr1 = atr1;
		this.atr2 = atr2;
		pole = new double[100];
	}
}
```

```
// v Main()
Trida objekt = new Trida(
                ▲2 of 2▼  Trida.Trida (int atr1, string atr2)
```

- důvody využití – možnost nabídnout uživateli více možností využití třídy
- příklad v .NET:

```
StreamWriter(Stream)
StreamWriter(String)
StreamWriter(String, Boolean)
StreamWriter(String, Boolean, Encoding)
StreamWriter(String, Boolean, Encoding, Int32)
// a další
```

Přímá inicializace atributů

- = inicializace proměnných v deklaraci
- výhodné při několika přetížených konstruktorech – nemusím psát několikrát
- zásady:
  - do konstruktoru jenom to, co je známo až při vzniku objektu
  - vše ostatní přímo v deklaracích atributů
- min. příklad lépe:

```
class Trida
{
        double[] pole = new double[100];
        int atr1;
        string atr2;

        public Trida(int atr1, string atr2)
        {
                this.atr1 = atr1;
                this.atr2 = atr2;
        }
}
```

- např. v C++ nelze