Výčtový typ

Motivační příklad

- řízení semaforu na křižovatce = přepínání červená/oranžová/zelená

```
const int CERVENA = 0;
const int ORANZOVA = 1;
const int ZELENA = 2;
int pristiStav = CERVENA;
while (true)
{
        switch (pristiStav)
        {
                case CERVENA:
                        Console.WriteLine("cervena");
                        pristiStav = ORANZOVA;
                        Cekej(20);
                        break;
                case 1:
                        Console.WriteLine("oranzova");
                        pristiStav = ZELENA;      // zelena
                        Cekej(0.5);
                        break;
                // atd.
        }
}
```

Co zabrání programátorovi uložit do `pristiStav` -38?

Principy a příklady

- Výčtový typ = seznam konstant, do proměnné nelze uložit jiné

```
enum Semafor
{ cervena, oranzova, zelena }
static void Main(string[] args)
{
        Semafor pristiStav = Semafor.cervena;
        while (true)
        {
                switch (pristiStav)
                {
                        case Semafor.cervena:
                                Console.WriteLine("cervena");
                                pristiStav = Semafor.oranzova;
                                Cekej(20);
                                break;
                        case Semafor.oranzova:
                                Console.WriteLine("oranzova");
                                pristiStav = Semafor.zelena;
                                break;
                        // atd.
                }
        }
}
```

Nový uživatelský datový typ

Metody – volání hodnotou a odkazem

- volání = způsob předávání parametrů metodám

Volání hodnotou

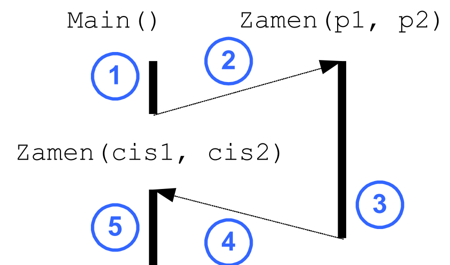
- Min. přednáška: „při volání metody jsou hodnoty skutečných parametrů přiřazeny do proměnných formálních parametrů“
- Volání metody:
  1) vypočet výrazů představujících skutečné parametry
  2) vytvoření lokálních proměnných a proměnných formálních parametrů
  3) kopie hodnot skutečných parametrů do proměnných lokálních parametrů
  4) běh těla metody
  5) konec metody – návratová hodnota použita ve výrazu jako výsledek metody
- volání hodnotou – metoda pracuje s hodnotami (kopiemi) skutečných parametrů
  - → ve funkci nelze změnit hodnoty skutečných parametrů

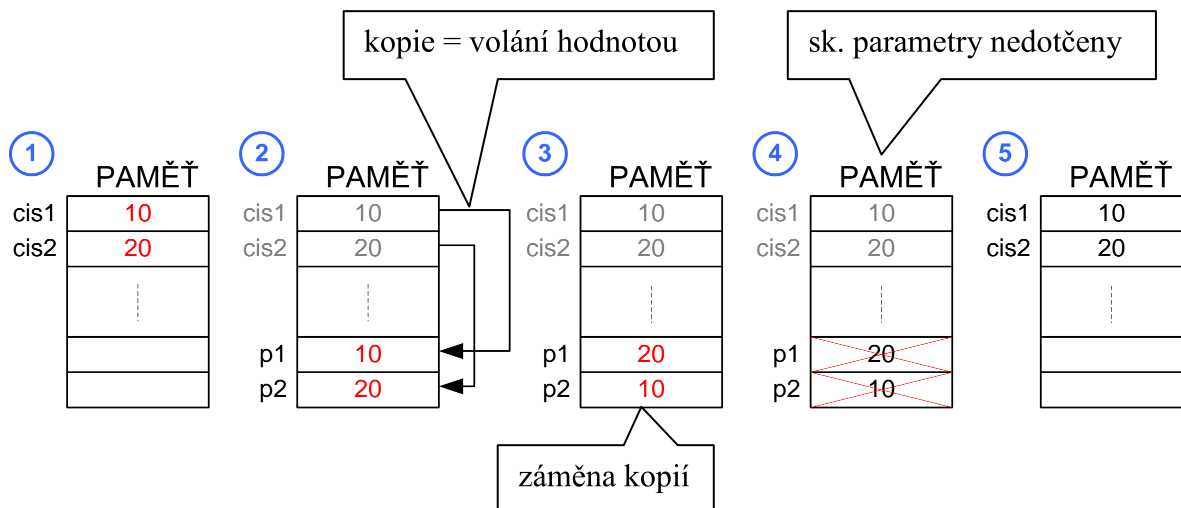
Příklad

- Napište metodu, která zamění obsah (hodnoty) dvou proměnných typu `int`
- Nefunkční řešení

```
using System;
class Program
{
        static void Main(string[] args)
        {
                int cis1 = 10, cis2 = 20;
                Zamen(cis1, cis2);
        }
        static void Zamen(int p1, int p2)
        {
                int pomoc;
                pomoc = p1;
                p1 = p2;
                p2 = pomoc;
        }
}
```

po skončení metody:
cis1 = 10, cis2 = 20





Volání odkazem

- metoda pracuje se skutečnými parametry, nikoli s kopiemi

```
using System;
class Program
{
	static void Main(string[] args)
	{
		int cis1 = 10, cis2 = 20;
		Zamen(ref cis1, ref cis2);
	}
	static void Zamen(ref int p1, ref int p2)
	{
		int pomoc;
		pomoc = p1;
		p1 = p2;
		p2 = pomoc;
	}
}
```

po skončení metody: `cis1 = 20, cis2 = 10`

- před voláním `ref` parametru musí být do proměnné přiřazena hodnota (přiřazením, inicializací v deklaraci)

Výstup více hodnot z metod

- Příklad: Napište metodu, která najednou vypočte obsah i obvod kruhu

```
using System;
class Program
{
	static void Main(string[] args)
	{
		double polomer = 10;
		double obsah, obvod;
		ParamKruhu(polomer, out obsah, out obvod);
	}
	static void ParamKruhu(double R, out double S, out double O)
	{
		S = Math.PI * R * R;
		O = 2 * Math.PI * R;
	}
}
```

  - variantní řešení → nedělat, nelogické

```
static double ParamKruhu(double R, out double S)
{
	S = Math.PI * R * R;
	return 2 * Math.PI * R;
}
```

`O` vracen normálně, `S` odkazem

- před voláním `out` parametru nemusí být do proměnné přiřazena hodnota

Metody a dekompozice problému

- použití podprogramů = přirozená snaha při řešení problémů postupovat hierarchicky:
    1) „shora dolů“ – problém rozkládáme na menší celky, které separátně řešíme
    2) „zdola nahoru“ – dílčí části problému skládáme do větších celků
    3) výhodné v týmu → každý řeší pouze část úkolu
    4) program se lépe ladí
- „větší program“ = 3 vrstvy:
    1) Datová – proměnné
    2) Aplikační – metody, upravují data
    3) Prezentační – plní a prezentuje datovou vrstvu

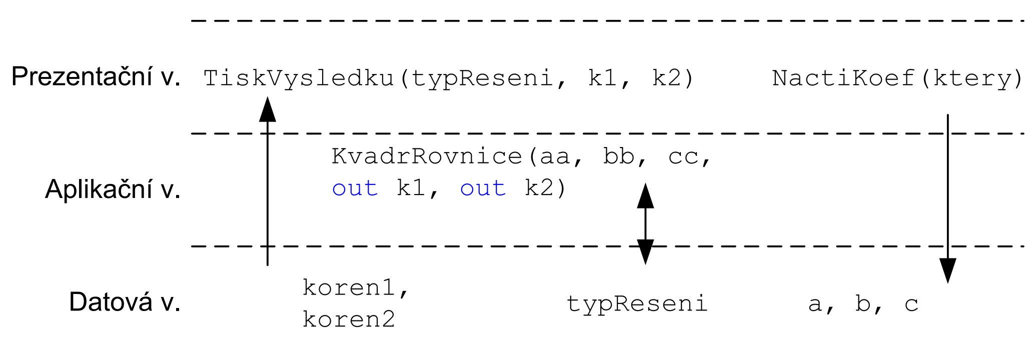
Příklad – program na řešení kvadratické rovnice

$ax^2 + bx + c = 0$

$D = b^2 - 4ac$

$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$

- požadavky:
    - vstupy: z klávesnice
    - řešení i v oboru komplexních čísel
    - výstupy: obrazovka
- analýza:
    - data: `a`, `b`, `c`, `D`, `x1`, `x2`
    - vstupy, prezentace výsledků: `Console`
- dále možnosti řešení:
- Varianta č. 1 – „vše v `Main()`“ (žádné další metody)
    - Nikdy!!!
    - použití „jádra řešení“ KvR v jiných programech obtížné
- Varianta č. 2 – Vstupy v `Main()`, zbytek metoda (počítá KvR, tiskne výsledky na obrazovku)
    - metodu nelze použít v programech s jiným uživatelským rozhraním
- Varianta č. 3 – 3 vrstvy



```
static void Main(string[] args)
{
	double a, b, c, koren1, koren2;
	a = NactiKoef("a");
	b = NactiKoef("b");
	c = NactiKoef("c");
	TypKorenu typReseni = KvR(a, b, c, out koren1, out koren2);
	TiskVysledku(koren1, koren2, typReseni);
}
static TypKorenu KvR (double a, double b, double c,
	out double k1, out double k2)
{
	// z koef. a, b, c spocita k1, k2, dle D vraci typ reseni
}
static double NactiKoef(string ktery)
{
	// nacte koef. z klavesnice
	return koef;
}
static void TiskVysledku(double k1, double k2, TypKorenu typReseni)
{
	// dle typReseni vytiskne k1, k2
}
enum TypKorenu
{
	Realne,
	Nasobne,
	Komplexni
}
```

Přetěžování metod

- (methods overloading)
- v .NET masivně používáno
- motivace – jazyk C, výpočet absolutní hodnoty

```
int abs(int x);
long labs(long x);
double fabs(double j)
```

jazyk C: jedinečné identifikátory

- přetížené metody = v jednom programu definice více metod se stejným jménem, ale různým
    - počtem parametrů
    - typem parametrů
    - pořadím parametrů – nedělat!!!
    - kombinace

- typické příklady
  - stejná činnost na různých typech parametrů

```
int abs(int x);
long abs(long x);
double abs(double j)
```

  - různý počet parametrů – upřesnění činnosti metody

```
static void TiskVyplaty(int koruny)
{
        Console.WriteLine("Vyplata je: {0:C0}", koruny);
}
static void TiskVyplaty(int koruny, int halere)
{
        Console.WriteLine("Vyplata je: {0:C0}, {1} haleru",
              koruny, halere);
}
```

- nelze přetěžovat (pouze) typem návratové hodnoty

```
double f(int a)
int f(int a)
```

- např.: `Console.WriteLine()` → 19 přetížení

```
Console.WriteLine ()
Console.WriteLine (Boolean)
Console.WriteLine (Char)
Console.WriteLine (Char[])
Console.WriteLine (Decimal)
Console.WriteLine (Double)
Console.WriteLine (Int32)
Console.WriteLine (Int64)
Console.WriteLine (Object)
Console.WriteLine (Single)
Console.WriteLine (String)
Console.WriteLine (UInt32)
Console.WriteLine (UInt64)
Console.WriteLine (String, Object)
Console.WriteLine (String, Object[])
Console.WriteLine (Char[], Int32, Int32)
Console.WriteLine (String, Object, Object)
Console.WriteLine (String, Object, Object, Object)
```

- překladač vždy zvolí dle parametrů nejlepší přetížení (pozor např. na přetypování!!!)

Třída System.Math

- základní matematické funkce
- konstanty: `Math.PI`, `Math.E`

Goniometrické

- Všechny typu `double Funkce(double x)`
- `Math.Sin(), Math.Cos(), Math.Tan()`
- `Math.Asin(), Math.Acos(), Math.Atan()`
- Příklad:

```
double sinUhlu = 0.5;
double uhelVeStupnich = 180 * (Math.Asin(sinUhlu)) / Math.PI;
// uhelVeStupnich = 30
```

Hyperbolické

- Všechny typu `double Funkce(double x)`
- `Math.Sinh(), Math.Cosh(), Math.Tanh()`

Logaritmy

- Všechny typu `double Funkce(double x)`
- Desítkový: `Math.Log10()`
- Přirozený: `Math.Log()`

Mocninné funkce

- druhá odmocnina: `double Math.Sqrt(double x)`
- e^x: `double Math.Exp(double x)`
- x^y: `double Math.Pow(double x, double y)`
- Příklad:

```
double cislo = 3.245;
double vysledek = Math.Pow(cislo, 10);
```

Práce se znaménkem

- Absolutní hodnota: `typ Math.Abs(typ)`
  - `typ` → jakýkoli jednoduchý datový typ
- Test znaménka: `int Math.Sign(typ)`
  - Vrací celé číslo dle argumentu:
    - $> 0 \rightarrow 1$
    - $= 0 \rightarrow 0$
    - $< 0 \rightarrow -1$

Generování náhodných čísel

- Princip: generátor pseudonáhodných čísel s rovnoměrným rozložením
- Třída `Random`

Postup

- Vytvoření generátoru

```
Random mujGenerator = new Random();
```

- Volání metod → náhodná celá čísla

  `int mujGenerator.Next()` → NČ v rozsahu ⟨0; `Int32.MaxValue`)

  `int mujGenerator.Next(int max)` → ⟨0; `max`)

  `int mujGenerator.Next(int min, int max)` → ⟨`min`; `max`)

- Volání metod → náhodná reálná čísla

  `double mujGenerator.NextDouble()` → ⟨0.0; 1.0)

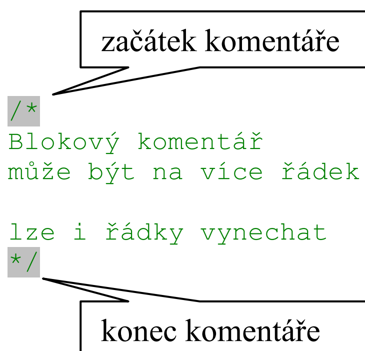
- Příklad: Vygenerujte 10 celých čísel v rozsahu 0 – 100

```
static void Main(string[] args)
{
        const int POCET_CISEL = 10;
        const int MAX_NC = 100;
        Random generator = new Random();
        for (int i = 0; i < POCET_CISEL; i++)
        {
                Console.WriteLine(generator.Next(MAX_NC));
        }
}
```

Komentáře

- pomocný text pro čtenáře ZK s nápovědnými informacemi (co daný úsek kódu dělá, proč, jak …)
- při překladu jsou ze ZK vypuštěny – překladač je „nevidí“
- kód se komentuje průběžně při jeho vytváření!!!

Blokové komentáře




```
/* komentář na jednom řádku */
```

Řádkové komentáře

```
příkaz;
// řádkový komentář končí na konci řádku
příkaz;
```

Poznámky

- jakákoli „vychytávka“ = komentář
- TODO komentáře = komentář, označující a popisující další postup programování (nedodělky)
- Část dobře komentovaného kódu

```
//******************************************************************
// eliminacni smycka
//******************************************************************
m = 1;                  // smycka bezi az od druheho radu
while {                 // eliminacni smycka
      spatneReseni = 0;
      // hledani m-teho maxima
      naselMax = 0;     // pokud najde lok. max. dam si to do 1
      // TODO: osetrit mozne preteceni naselMax
      maxPwrCe[m] = 0;
      for(i=BLIZKA_ZONA;i<N-1;i++) {
            iJeSpatne = 0;
            if (kkk>0) { // kkk = 0 -> nemam zatim zadne spatne id
                  for(ii=0;ii<kkk;ii++) {
                        // kkk je index pole spatnych id
                        if (i == zakazaneID[ii])
                              iJeSpatne = 1;
                        }
            }
      }
} // end of eliminacni smycka
```