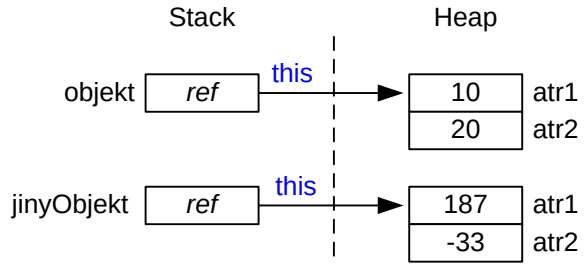
Objekt jako proměnná

Objekty a metody

- Objekt = proměnná referenčního typu – vznik pomocí `new`, chování viz „pole jako referenční proměnná“ minulý semestr



- Objekt – může být parametrem metod a návratovou hodnotou (samozřejmé)
- Referenční proměnné – volány odkazem (viz pole min. semestr)
- Příklad:

```
class T1
{
	private int cislo;
	public int Cislo
	{
		get { return cislo;}
		set { cislo = value;}
	}
	public T1(int cislo)
	{
		this.cislo = cislo;
	}
	public void Tisk()
	{
		Console.WriteLine(this.cislo);
	}
	public int Dvojnasobek()
	{
		return 2 * this.cislo;
	}
}
class Program
{
	// objekt jako parametr
	static int Metoda(T1 o)
	{
		o.Tisk();
		return o.Dvojnasobek();
	}
	// objekt jako navratova hodnota
	static T1 Vytvor()
	{
		T1 temp = new T1(10);
		return temp;
	}
	// vyuziti volani odkazem - objekt je menen v metode
	static void Zmen(ref T1 o)	// ref neni nezbytne nutne
	{
```

```
            o.Cislo = 10;
        }

        static void Main(string[] args)
        {
            T1 o1 = new T1(555);
            int x = Metoda(o1);
            T1 o2 = Vytvor();
            Zmen(ref o2);
        }
    }
```

Pole objektů

- Zcela bez problémů, pouze pozor na neinicializované proměnné

```
T1[] poleObjektu = new T1[3];
poleObjektu[0] = new T1(10);
T1 o3 = new T1(-444);
poleObjektu[1] = o3;
poleObjektu[1].Tisk();
poleObjektu[2].Cislo = 10;      // Error, poleObjektu[2] je zatím null
```

Zánik objektů

- opak konstruktoru = destruktor = metoda volaná při zániku objektu
- V C# objekty odstraňuje automatická správa paměti – okamžik není přesně determinován
- Garbage Collector – volá periodicky .NET (cca každých 10 ms)
    - Běh programu je zastaven
    - Veškerá data, na která nevede odkaz, jsou z paměti odstraněna

```
{
    int a;
    Trida objekt1 = new Trida();
    if (a > 10)
    {
        double b;
        Trida objekt2 = new Trida();
        // nejaky kod
    }   // končí: referenční proměnná „objekt2“, „b“
    // nejaky kod
    objekt1 = null;    // příští GC = odstranění „objekt1“
}   // končí: „a“
```

Statické členy

- = člen, který se pojí se třídou (ne s instancí) – objekt ke statickým členům nemá přístup
- Označeny `static`

Atributy, Metody, Vlastnosti

- Typický příklad: počítání počtu vytvořených instancí

```
class Zamestnanec
{
	string jmeno;
	public Zamestnanec(string jmeno)
	{
		this.jmeno = jmeno;
		pocetCelkem++;
	}
	public static int pocetCelkem;  // public jen pro tuto ukazku!
}
// v Main()
Zamestnanec zednik = new Zamestnanec("Franta Vomacka");   // pc = 1
Zamestnanec skladnik = new Zamestnanec("Alfonz Mucha");   // pc = 2
Console.WriteLine(zednik.pocetCelkem);        // CHYBA
Console.WriteLine(Zamestnanec.pocetCelkem);  // OK
```

- Příklad statické metody (doplnění min. příkladu)

```
class Zamestnanec
{
	string jmeno;
	public Zamestnanec(string jmeno)
	{
		this.jmeno = jmeno;
		pocetCelkem++;
	}
	private static int pocetCelkem;
	public static int GetPocet()
	{
		return pocetCelkem;
	}
}
// v Main()
Zamestnanec zednik = new Zamestnanec("Franta Vomacka");
Zamestnanec skladnik = new Zamestnanec("Alfonz Mucha");
Console.WriteLine(Zamestnanec.GetPocet());   // 2
```

  - Lepší řešení – statická Vlastnost

```
private static int pocetCelkem;
public static int PocetCelkem
{
	get { return pocetCelkem; }
}
// v Main()
Console.WriteLine(Zamestnanec.PocetCelkem);
```

- Nelze použít `this` (logické)

Konstruktory

- Účelem je provést prvotní nastavení, společné pro všechny instance, např.:

  - připojení ke sdílenému prostředku – databázi, komunikačnímu rozhraní (sériový port, síť, apod.)
  - pokud objekty zapisují např. informace do společného souboru, tak statický konstruktor jej otvírá
- omezení:
  - nesmí mít parametry ani přístupový modifikátor
  - nelze jej přímo zavolat – to proběhne automaticky „někdy těsně před vytvořením první instance“ nebo před použitím prvního statického členu – okamžik volání nemá uživatel pod kontrolou
- Výchozí konstruktor – má každá třída v případě, že nedefinujeme vlastní
- Příklad:

```
class Trida
{
        public Trida()
        {
                Console.WriteLine("Instancni ctor");
        }
        static Trida()
        {
                Console.WriteLine("Static ctor");
        }
}
// v Main()
Trida o = new Trida();
```

  - Vytiskne:

```
Instancni ctor
Static ctor
```

Čistě statické třídy

- Obsahují pouze statické členy – nelze vytvořit instanci (nemělo by jít)
- V .NET např. `Console`, `Math`
- možnosti zamezení vytvoření instance:
  - označit třídu jako `static` – doporučeno
  - privátní konstruktor
- více viz. MSDN

Skládání tříd

Třída jako atribut jiné třídy

- Vztah **„má“** – např. auto má motor

```
class Motor
{
        double vykon;
        public Motor(double vykon)
        {
                this.vykon = vykon;
        }
        public void Nastartuj()
        {
```

```
        }
        public double NastavSkrKlapku(double pozice) // 0-1 = 0-100%
        {
                return vykon * pozice;          // vraci aktualni vykon motoru
                                                dle nastaveni plynu
        }
}
class Auto
{
        int pocetMist;
        public Motor motor;// public aby motor byl vidět zvenčí Auta
        public Auto(int pocetMist, double vykon)
        {
                this.pocetMist = pocetMist;
                motor = new Motor(vykon);
        }
}
// v Main()
Auto mojeAuto = new Auto(5, 85);
mojeAuto.motor.Nastartuj();
mojeAuto.motor.NastavSkrKlapku(1);
```

- Problém: skutečné auto nemá přístupný motor zvenčí
    - motor skryjeme pod ovladače auta

```
// Trida Motor je stejna
class Auto
{
        int pocetMist;
        private Motor motor;      // s motorem primo nebudeme manipulovat
        public Auto(int pocetMist, double vykon)
        {
                this.pocetMist = pocetMist;
                motor = new Motor(vykon);
        }
        public void Nastartuj()
        {
                motor.Nastartuj();
        }
        public void NastavPlyn(double pozice)
        {
                motor.NastavSkrKlapku(pozice);
        }
}
// v Main()
Auto mojeAuto = new Auto(5, 85);
mojeAuto.Nastartuj();
mojeAuto.NastavPlyn(1);
```

Třída obsahuje definici jiné třídy

- definici třídy lze umísti dovnitř definice jiné třídy
- vnořené třídě lze přiřadit specifikátor přístupu:
    - `public class Trida` – vnořená třída bude přístupná mimo svoji mateřskou třídu
    - `private class Trida` – nebude …
- Příklad:

```
class Trida
{
        public void Metoda()
        {
        }

        public class VnorenaTrida
        {
                int atribut;
                public VnorenaTrida(int atribut)
                {
                        this.atribut = atribut;
                }
                public int GetAtribut()
                {
+
                        return atribut;
                }
        }
}
// v Main()
Trida.VnorenaTrida objekt = new Trida.VnorenaTrida(10);
Console.WriteLine(objekt.GetAtribut()); // přístup ke členům třídy je již
                                                normální
```

- Málokdy využívané, to samé platí ovšem i pro `enum`, `struct` (později)

```
class KvadratickaRovnice
{
        public enum TypKorenu     // mimo tuto třídu nemá jinak smysl
        {
                realne,
                nasobny,
                cmplx
        }
        public KvadratickaRovnice(double a, double b, double c) { }
        public TypKorenu Vypocti(out double k1, out double k2) { }
}
// v Main()
KvadratickaRovnice polynom = new KvadratickaRovnice(10, -3, 185);
KvadratickaRovnice.TypKorenu typVysledku = polynom.Vypocti(out x1, out x2);
```

  - (Třída je špatně navržena – lepší: `a`, `b`, `c` jako Vlastností)