Číslo a název šablony	III/2 Inovace a zkvalitnění výuky prostřednictvím ICT
Číslo didaktického materiálu	EU-OPVK-VT-III/2-ŠR-318
Druh didaktického materiálu	DUM
Autor	RNDr. Václava Šrůtková
Jazyk	čeština
Téma sady didaktických materiálů	Programování v C# v příkladech III
Téma didaktického materiálu	Polymorfismus
Vyučovací předmět	Seminář z informatiky
Cílová skupina (ročník)	Žáci ve věku 17–18 let
Úroveň žáků	Středně pokročilí
Časový rozsah	1–2 vyučovací hodiny
Klíčová slova	Třída, dědičnost, polymorfismus, typová konverze, přetypování
Anotace	Studenti pracují s dědičností tříd, programují hromadné zpracování objektů prostřednictvím přetypování a polymorfismu.
Použité zdroje	ELLER, Frank. *C# - začínáme programovat: podrobný průvodce začínajícího uživatele*. 1. vyd. Praha: Grada, 2002, 240 s. ISBN 80-247-0324-6. OCHRANOVÁ, Renata a Michal KOZUBEK. *Objektově orientované programování v Turbo Pascalu*. 1. vyd. Brno: Masarykova univerzita, 1993, 117 s. ISBN 80-210-0659-5. TÖPFEROVÁ, Dana a Pavel TÖPFER. *Sbírka úloh z programování*. Vyd. 1. Praha: Grada, 1992, 98 s. Educa '99. ISBN 80-854-2499-1. VYSTAVĚL, Radek. *Moderní programování: sbírka úloh k učebnici pro středně pokročilé*. 1. vyd. Ondřejov: moderníProgramování, 2008-2009, 2 sv. ISBN 978-80-903951-3-8. VYSTAVĚL, Radek. *Moderní programování: učebnice pro pokročilé*. Ondřejov: moderníProgramování, 2011, 149 s. ISBN 978-80-903951-7-6. VYSTAVĚL, Radek. *Moderní programování: učebnice pro středně pokročilé*. Ondřejov: moderníProgramování s.r.o, 2008. ISBN 978-80-903951-2-1.
Typy k metodickému postupu učitele, doporučené výukové metody, způsob hodnocení, typy k individualizované výuce apod.	Text je možno využít ke společné práci, samostatné přípravě studentů, domácímu studiu apod. Při společné práci je vhodné nejprve obtížnější

	úlohy rozebrat, potom společně se studenty implementovat na počítači. (Rozbor nejlépe na tabuli, synchronní řešení s promítáním) Prezentace obsahuje stručné shrnutí poznatků potřebných pro řešení příkladů. V pracovním listu je zadání cvičení – většinou se jedná o úlohy, které by měli studenti naprogramovat samostatně. Není nutné, aby všichni zpracovali všechno, vhodné je diferencovat podle jejich zájmu a schopností. Obtížnější úlohy jsou označeny hvězdičkou. Součástí materiálu je zdrojový kód těchto příkladů. Návrh způsobu hodnocení: ohodnocení samostatné práce během hodiny např. podle volby a počtu úloh a elaborace řešení (efektivnost, komentáře…).

Metodický list k didaktickému materiálu

Prohlášení autora

Tento materiál je originálním autorským dílem. K vytvoření tohoto didaktického materiálu nebyly použity žádné externí zdroje s výjimkou zdrojů citovaných v metodickém listu.

Obrázky (schémata a snímky obrazovek) pocházejí od autora.

318. Polymorfismus

Hromadné zpracování dat

Důležitou vlastností dědičnosti je, že kdekoli je očekávána instance předka, lze použít instance potomka. Do proměnné typu předek lze tedy uložit objekt typu potomek.

Pro větší názornost se vrátíme k jednoduchým objektům – čtverci a jeho potomku obdélníku.

Připomeňme si definici těchto tříd:

```
class Ctverec
    {
        double a;
        public double A
        {
            get
            {
                return a;
            }
            set
```

```
            {
                a = value;
            }
        }
            public Ctverec()
            {
            }
            public Ctverec(double a)
            {
                this.A=a;
            }

            public double ObsahCtverce()
            {
                return A * A;
            }
            public string InfoCtverce()
            {
                return "Ctverec: a = " + a.ToString("F2") + " S = " +
ObsahCtverce().ToString();
            }
          }


class Obdelnik:Ctverec
    {
        double b;
        public double B
        {
            get
            {
                return b;
            }
            set
            {
                b = value;
            }
        }
            public Obdelnik():base()
            {
            }
            public Obdelnik(double a, double b):base(a)
```

```
            {
                this.B=b;
            }

            public double ObsahObdelnika()
            {
                return A * B;
            }
            public string InfoObdelnika()
            {
                return "Obdelnik: a = " + A.ToString("F2")+" b = " + B.ToString("F2")+
" S = " + ObsahObdelnika().ToString();
            }
    }
```

Vyzkoušejme si vytvořit novou instanci obdélníka a dosadit ho do předka – čtverce a pak si nechat udělat výpisy při stisknutí tlačítka Pokus1, Pokus2.

```
private void buttonPokus_Click(object sender, EventArgs e)
        {
            Obdelnik o = new Obdelnik(3, 4);
            Ctverec predek = o;
            textBoxVypis.Text += predek.ToString() + Environment.NewLine;
            textBoxVypis.Text += predek.A.ToString() + Environment.NewLine;
        }
```

Metoda ToString, tak jak ji definuje C# (ukážeme se později její předefinování) vypíše správně obdélník a správně jeho první stranu A.

NaPOlymorf.Obdelnik
3

Horší to bude s naší metodou InfoObdelnika – pro předka ji našeptávač nenabízí a samozřejmě, vnutit mu ji nedokážeme.

```
textBoxVypis.Text += o.InfoObdelnika() + Environment.NewLine;
textBoxVypis.Text += predek.InfoObdelnika() + Environment.NewLine;
```

Nelze tedy přímo provádět nic, co třída Ctverec neumí nebo používat vlastnosti, které v ní nejdou nadefinované.

Jde to obejít **přetypováním,** jak ukazuje následující výpis.

```
private void buttonPokus2_Click(object sender, EventArgs e)
        {
            Obdelnik o = new Obdelnik(3, 4);
            Ctverec predek = o;
            textBoxVypis.Text += o.InfoObdelnika() + Environment.NewLine;
            if (predek is Obdelnik)
            textBoxVypis.Text += (predek as Obdelnik).InfoObdelnika() +
Environment.NewLine;
```

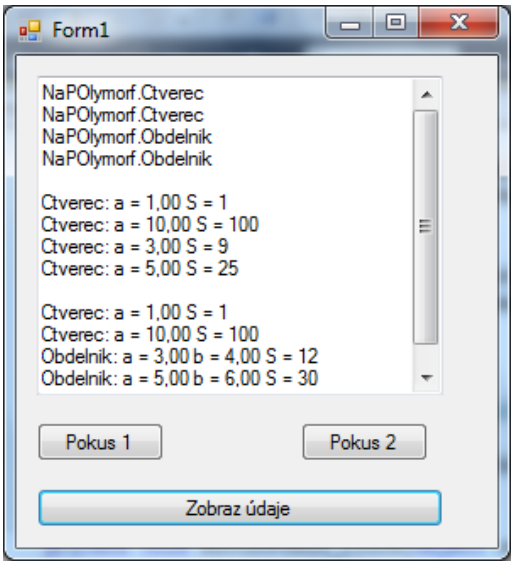
```
        }
```

Typová konverze se provádí buď tak, že uvedeme nový typ v závorkách před názvem proměnné nebo pomocí operátoru **as**.

Aby nedocházelo k zbytečným běhovým chybám, můžeme typ objektu otestovat operátorem **is**.

Jakou nám to přináší výhodu? Můžeme všechny objekty zpracovat najednou.

```
private void button1_Click(object sender, EventArgs e)
        {
            Ctverec c1 = new Ctverec(1);
            Ctverec c2 = new Ctverec(10);
            Obdelnik o1 = new Obdelnik(3,4);
            Obdelnik o2 = new Obdelnik(5, 6);
            //Inicializované objekty přidáme do seznamu
            List<Ctverec> utvary = new List<Ctverec>();
            utvary.Add(c1);
            utvary.Add(c2);
            utvary.Add(o1);
            utvary.Add(o2);
            //Seznam pohodlně vypíšeme cyklem foreach – nejprve metodou ToString
            foreach (Ctverec c in utvary)
                textBoxVypis.Text += c.ToString()+Environment.NewLine;
            textBoxVypis.Text += Environment.NewLine;
            //- potom vlastní metodou Info
            foreach (Ctverec c in utvary)
                textBoxVypis.Text += c.InfoCtverce() + Environment.NewLine;
            textBoxVypis.Text += Environment.NewLine;
            //S přetypováním už to funguje, jak si představujeme
            foreach (Ctverec c in utvary)
                if (c is Obdelnik)
                textBoxVypis.Text += (c as Obdelnik).InfoObdelnika() +
Environment.NewLine;
                else
                    textBoxVypis.Text += c.InfoCtverce() + Environment.NewLine;
        }
Poznámka: Kdybychom prohodili podmínku  if (c is Ctverec)
           textBoxVypis.Text += (c as Ctverec).InfoCtverce()  + Environment.NewLine;
           else
               textBoxVypis.Text += (c as Obdelnik).InfoObdelnika() +
Environment.NewLine;
Objevili by se výpisy pouze pro čtverec – každý náš obdélník je přece původem čtverec.
```



Pokud bychom ovšem měli více dědiců, bylo by větvení s přetypováním zbytečně složité.

Polymorfismus

Nejpřirozenější řešení předchozího problému by vypadalo tak, že by se „tytéž“ metody (Info, Objem) jmenovaly stejně a systém by nějak poznal, kterou má použít.

```
private void button1_Click(object sender, EventArgs e)
        {
            Ctverec c1 = new Ctverec(1);
            Ctverec c2 = new Ctverec(10);
            Obdelnik o1 = new Obdelnik(3, 4);
            Obdelnik o2 = new Obdelnik(5, 6);
            List<Ctverec> utvary = new List<Ctverec>();
            utvary.Add(c1);
            utvary.Add(c2);
            utvary.Add(o1);
            utvary.Add(o2);

            //Info
            foreach (Ctverec c in utvary)
             textBoxVypis.Text += c.Info() + Environment.NewLine;
           }
```

Jak tedy vysvětlit systému, že má použít metodu potomka a ne předka? Je třeba metody vhodně deklarovat: u předka použijeme klíčové slovo **virtual** a u potomka **override**.

```
class Ctverec
    {
            …
            public virtual double Obsah()
```

```
            {
                return A * A;
            }
            public virtual string Info()
            {
                return "Ctverec: a = " + a.ToString("F2") + " S = " +
Obsah().ToString();
            }
    }
class Obdelnik:Ctverec
    {
           …
            public override double Obsah()
            {
                return A * B;
            }
            public override string Info()
            {
                return "Obdelnik: a = " + A.ToString("F2")+" b = " + B.ToString("F2")+
" S = " + Obsah().ToString();
            }
    }
```

Shrňme si tedy jak je to s dědičností a polymorfismem. Odvozené třídy mohou změnit (předefinovat) složky bázové třídy. Týká se to především metod a vlastností. (proměnné nemají kód)

Při předefinování metod předka většinou potřebujeme polymorfní chování, tedy aby se používala metoda instance potomka, i když s ní pracujeme v typu předka. Metodu bázové třídy, u které požadujeme toto chování označujeme slovem **virtual,** metodu odvozené třídy, která ji předefinovává pak slovem **override**.

Důležité

Do proměnné typu předek lze tedy uložit objekt typu potomek. Proto je možné objekty odvozených tříd zpracovávat v cyklech – typicky: **foreach**

Protože ovšem předek nemusí mít (a obvykle nemá) všechny vlastnosti a metody potomka, používáme přetypování pomocí operátorů **is** a **as**. (is zjistí typ, as provede přetypování)

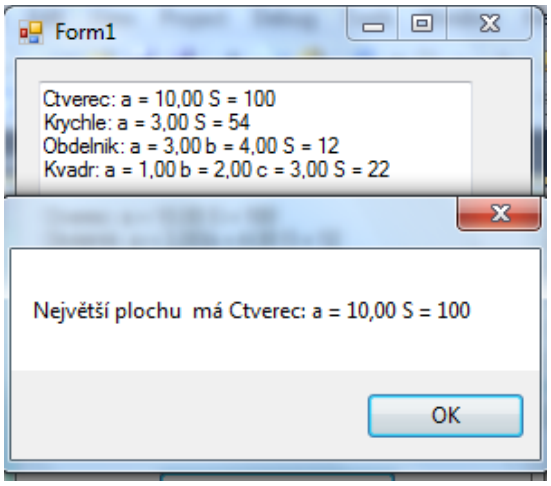
Metodu bázové třídy, u které požadujeme polymorfní chování označujeme slovem **virtual,** metodu odvozené třídy, která ji mění pak slovem **override**.

Pracovní list

Cvičení

1. Definujte třídu Krychle a Kvádr – tentokrát jako potomky čtverce a obdélníku, Obsah por ně bude mít význam povrchu. Můžete přidat další metody. Vyzkoušejte výpisy.

2. Zopakujte si práci s polem, do kterého vložíte jednotlivé instance vašich tříd a které hromadně zpracujete, jednak proveďte výpis, jednak vyberte objekt s největším obsahem. (Plochou)



Řešení

1.

```
class Krychle:Ctverec
    {
         public Krychle():base()
            {
            }
            public Krychle(double a):base(a)
            {
            }

        public override double Obsah()
        {
            return A * A * 6;
        }
        public override string Info()
        {
            return "Krychle: a = " + A.ToString("F2") + " S = " + Obsah().ToString();
        }
    }
class Kvadr:Obdelnik
    {
        double c;
        public double C
        {
            get
            {
                return c;
```

```
        }
        set
        {
            c = value;
        }
    }
        public Kvadr():base()
        {
        }
        public Kvadr(double a, double b, double c)
            : base(a,b)
        {
            this.C=c;
        }

        public override double Obsah()
        {
            return 2 * (A * B + B *  C+  A* C);
        }
        public override string Info()
        {
            return "Kvadr: a = " + A.ToString("F2")+" b = " + B.ToString("F2")
                + " c = " + C.ToString("F2") + " S = " + Obsah().ToString();
        }
}
```

```
         private void button1_Click(object sender, EventArgs e)
        {
        Ctverec c = new Ctverec(10);
        Obdelnik o = new Obdelnik(3, 4);
        Krychle k = new Krychle(3);
        Kvadr q = new Kvadr(1, 2, 3);
        textBoxVypis.Text += c.Info() + Environment.NewLine;
        textBoxVypis.Text += k.Info() + Environment.NewLine;
        textBoxVypis.Text += o.Info() + Environment.NewLine;
        textBoxVypis.Text += q.Info() + Environment.NewLine;
        textBoxVypis.Text += Environment.NewLine;
        }
```

2.

```
        …
        Ctverec[] pole = new Ctverec[4];
        pole[0] = c;
```

```
 pole[1] = o;
 pole[2] = k;
 pole[3] = q;
 Ctverec max = new Ctverec(0);
 for (int i = 0; i < 4; i++)
 {
     textBoxVypis.Text += pole[i].Info() + Environment.NewLine;
     if (max.Obsah() < pole[i].Obsah()) max = pole[i];
 }
 MessageBox.Show("Největší plochu  má " + max.Info());
}
```