Kapitola 6

Delphi - objektově orientované programování

Objektově orientované programování (zkracováno na OOP, z anglického Object oriented programming) je metodika vývoje softwaru, založená na těchto myšlenkách, koncepci:

6.1 Koncepce

Objekty – jednotlivé prvky modelované reality (jak data, tak související funkčnost) jsou v programu seskupeny do entit, nazývaných objekty. Objekty si pamatují svůj stav a navenek poskytují operace (přístupné jako metody pro volání).

Abstrakce – programátor, potažmo program, který vytváří, může abstrahovat od některých detailů práce jednotlivých objektů. Každý objekt pracuje jako černá skříňka, která dokáže provádět určené činnosti a komunikovat s okolím, aniž by vyžadovala znalost způsobu, kterým vnitřně pracuje.

Zapouzdření – zaručuje, že objekt nemůže přímo přistupovat k vnitřním parametrům jiných objektů, což by mohlo vést k nekonzistenci. Každý objekt navenek zpřístupňuje rozhraní, pomocí kterého (a nijak jinak) se s objektem pracuje.

Skládání – objekt může využívat služeb jiných objektů tak, že je požádá o provedení operace.

Dědičnost – objekty jsou organizovány stromovým způsobem, kdy objekty nějakého druhu mohou dědit z jiného druhu objektů, čímž přebírají jejich schopnosti, ke kterým pouze přidávají svoje vlastní rozšíření. Tato myšlenka se obvykle implementuje pomocí rozdělení objektů do tříd, přičemž každý objekt je instancí nějaké třídy. Každá třída pak může dědit od jiné třídy (v některých programovacích jazycích i z několika jiných tříd).

Polymorfismus – odkazovaný objekt se chová podle toho, jaký je jeho skutečný typ. Pokud několik objektů poskytuje stejné rozhraní, pracuje se s nimi stejným způsobem, ale jejich konkrétní chování se liší. V praxi se tato vlastnost projevuje např. tak, že na místo, kde je očekávána instance nějaké třídy, můžeme dosadit i instanci libovolné její podtřídy (třídy, která přímo či nepřímo z této třídy dědí), která se může chovat jinak, než by se chovala instance rodičovské třídy, ovšem v rámci mantinelů, daných popisem rozhraní.

Pokud předchozí koncepci již rozumíme, bude pro nás pochopení objektově orientovaného programování jednoduché. Pokud však nerozumíme, což se předpokládá, nevadí. K dalšímu vysvětlení nám vřele pomůže ukázkový příklad.

6.2 Ukázkový příklad

Nejprve si představme, že celý svět se skládá z objektů. Objektem můžeme rozumět města, ulice, domy, stromy, auta, silnice, atd. Aby se nám tyto objekty nepletli, rozdělíme si je na různé druhy, resp. různé **třídy**. V programování bývá nejčastějším ukázkovým příkladem tříd třída aut.

Představme si auto (objekt). Každé auto, pokud neuvažujeme některé speciální případy, je přesně určeno státní poznávací značkou, respektive písmeny a číslicemi na SPZ. Můžeme tedy říci, že SPZ je charakteristická vlastnost auta, neboli vlastnost objektu. Pochopitelně má každé auto mnohem více vlastností, jako je například značka, barva, typ, objem motoru, atd.

6.3 Definice třídy

Nadefinujme si tedy třídu aut, tak aby obsahovala tyto vlastnosti: `spz`, `typ`, `objem Motoru`. Třídu aut, jak je v Delphi zvykem, nazveme `TAuto`. V této třídě budeme uvažovat `spz` a `typ` jako řetězce a `objemMotoru` jako celočíselnou hodnotu. Syntaxe této třídy je následující:

```
type
  TAuto = class
     spz, typ: String;
     objemMotoru: Integer;
  end;
```

Tuto definici nové třídy `TAuto` umístíme v Delphi do oblasti, kde definujeme nové typy, tedy za příkaz `type`. Každá nová třída se zadává stejným způsobem: `jmenotridy = class`, kde `class` nám udává, že se jedná o třídu. Po výčtu vlastností třídy, ukončíme definici příkazem `end;`, jak je vidět výše.

6.4 Vytvoření objektu

Nyní máme nadefinovanou třídu `TAuto`, ale zatím nemáme žádný objekt. Představme si, že vlastníme dva automobily (objekty): služební a soukromý. Aby jsme mohli tyto objekty vytvořit, musíme si je nejdříve deklarovat. Objekty budeme deklarovat stejně jako proměnné v části programu určené k deklaraci (tedy za klíčovým slovem `var`) a tyto objekty budou typu `TAuto`.

```
var
  sluzebniAuto, soukromeAuto: TAuto;
```

Abychom mohli s objekty pracovat, je zapotřebí je nejdříve vytvořit. Vytvoření se provede pomocí konstruktoru (konstruktor bude vysvětlen později) tímto způsobem:

```
sluzebniAuto := TAuto.Create;
soukromeAuto := TAuto.Create;
```

Takto byli vytvořeny dva objekty `sluzebniAuto` a `soukromeAuto`. Přesněji řečeno, jsme si uvolnily v paměti počítače místo pro tyto objekty a nyní s nimi můžeme pracovat. V objektově orientovaném programování je zapotřebí stále pamatovat na to, že objekty musíme nejdříve vytvořit a až pak s nimi můžeme pracovat. Jestliže už objekt k práci nepotřebujeme, je zapotřebí objekt zrušit, přesněji uvolnit zabrané místo v paměti počítače. To se provede příkazem `Free` následovně:

```
sluzebniAuto.Free;
soukromeAuto.Free;
```

Po tomto příkazu již naše objekty neexistují, nemůžeme tedy s nimi pracovat.

6.5 Vlastnosti objektu

Vraťme se nyní trochu zpět a předpokládejme, že máme vytvořeny objekty `sluzebni Auto` a `soukromeAuto`. Tyto objekty jsou typu `TAuto` a tudíž obsahují vlastnosti `spz`, `typ` a `objemMotoru`. Pokud chceme přistupovat k vlastnostem objektu, budeme k nim přistupovat pomocí tečky. Pokud tedy chceme zadat vlastnosti našich aut, uděláme to takto:

```
sluzebniAuto.spz := '1B0 25-69';
sluzebniAuto.typ := 'Škoda';
sluzebniAuto.objemMotoru := 1600;

soukromeAuto.spz := '2B3 85-31';
soukromeAuto.typ := 'Porshe';
soukromeAuto.objemMotoru := 3200;
```

Naopak, pokud budeme chtít vypsat do dialogového okna typ našeho služebního auta, napíšeme tento příkaz:

```
MessageDlg('Typ mého soukromého auta je:
' + soukromeAuto.typ, mtInformation, [mbOK], 0);
```

6.6 Metody objektu

Dosud jsme si vysvětlili, že každý objekt může obsahovat různé vlastnosti a my k nim pak jednoduše pomocí tečky přistupujeme. Nabízí se otázka: "K čemu je to dobré?", když stejně tak funguje v Delphi záznam, neboli `record`. Vysvětlení je jednoduché. Objekty nám nabízejí mnohem více možností a jednou z nich jsou metody, neboli funkce či procedury uvnitř objektu.

Vraťme se zpět k našim autům a chtějme sledovat stav benzínu v nádrži pomocí vlastnosti `palivo`. Dále budeme předpokládat, že množství paliva ubývá, pokud ujedeme nějakou vzdálenost v závislosti na spotřebě (`spotreba`) a přibývá, pokud tankujeme na čerpací stanici. Tyto dva poznatky nám zaručí procedury `Ujed` a `Tankuj`. Deklarace třídy `TAuto` bude vypadat následovně:

```
type
  TAuto = class
     spz, typ: String;
     objemMotoru: Integer;
     palivo, spotreba: Double;
     procedure Ujed(vzdalenost: Double);
     procedure Tankuj(mnozstviPaliva: Double);
  end;
```

Dále je zapotřebí vytvořit těla našich nových dvou procedur. To provedeme kdekoliv v části zdrojového textu, tedy za příkazem `implementation`, jak jsme zvyklí. Procedury budou vypadat následovně:

```
procedure TAuto.Ujed(vzdalenost: Double);
begin
  palivo := palivo - vzdalenost * (spotreba / 100);
end;

procedure TAuto.Tankuj(mnozstviPaliva: Double);
begin
  palivo := palivo + mnozstviPaliva;
end;
```

Nyní si všimněme, že při definici nových procedur jsme jako název uvedli `TAuto.Ujed`, resp. `TAuto.Tankuj` místo pouhého `Ujed`, resp. `Tankuj`. Důvod je velmi jednoduchý. Pokud bychom v programu měli více tříd, které by obsahovaly proceduru `Ujed` a přitom v každé třídě by procedura měla jinou definici, nastal by při překladu zdrojového textu zmatek.

Další důležitý poznatek je v proměnné `palivo`. Všimněme si, že neuvádíme např. `soukromeAuto.palivo`, ale pouze `palivo`. Jak tedy program pozná jaké `palivo` má změnit? Jednoduše. Změní `palivo` toho objektu, na kterém byla daná procedura volána. Tedy pokud voláme nad objektem soukromého auta (`soukromeAuto.Ujed(50)`), odečte požadovanou hodnotu z paliva tohoto objektu, tedy z `soukromeAuto.palivo`.

6.7 Konstruktor a destruktor objektu

Už dříve jsme se zmínili o konstruktoru. Používá se k tomu, aby nám vytvořil objekt, resp. aby alokoval místo v paměti pro tento náš objekt. Často je však velmi vhodné, když už při vytváření objektu specifikujeme některé vlastnosti, v našem případě `spz`, `typ`, `objemMotoru`, `palivo` a `spotreba`. Docílíme toho tak, že předefinujeme klasický konstruktor Delphi. Do definice třídy `TAuto` přidáme zmínku o tom, že budeme konstruktor vytvářet vlastní. Definice třídy pak bude vypadat:

```
type
  TAuto = class
     spz, typ: String;
     objemMotoru: Integer;
     palivo, spotreba: Double;
     constructor Create(sspz, ttyp: String;
     oobjemMotoru: Integer;
     sspotreba: Double);
     procedure Ujed(vzdalenost: Double);
     procedure Tankuj(mnozstviPaliva: Double);
  end;
```

Konstruktor je uveden klíčovým slovem `constructor` a název může být jakýkoliv, avšak je vhodné název ponechat tak, jak je zvykem, tedy `Create`.

Definice samotného konstruktoru se opět provádí v části implementace a může vypadat takto:

```
constructor TAuto.Create(sspz, ttyp: String;
 oobjemMotoru: Integer; sspotreba: Double);
begin
  inherited Create;
```

```
  spz := sspz;
  typ := ttyp;
  objemMotoru := oobjemMotoru;
  palivo := 0;
  spotreba := sspotreba;
end;
```

Příkaz `inheroted Create` nám slouží k vlastnímu vytvoření objektu, další příkazy jsou snad zřejmé a jednoduché. Pokud tedy budeme chtít vytvořit opět služební a soukromý vůz, budeme postupovat takto:

```
sluzebniAuto := TAuto.Create('1B0 25-69', 'Škoda', 1600, 7.8);
soukromeAuto := TAuto.Create('2B3 85-31', 'Porshe', 3200, 11.4);
```

Někdy je zapotřebí něco provést, než se objekt zruší. Třeba uložit důležitá data do souboru. K tomu nám hravě poslouží předefinování destruktoru. Je to stejné jako v případě konstruktoru, pouze klíčové slovo je `destructor` a je běžné používání názvu `Destroy`. Destruktor pak může vypadat takto:

```
destructor TAuto.Destroy;
begin
  UlozeniDat;
  inherited Destroy;
end;
```

Pozorný čtenář si všimne, že výše jsme k zrušení objektu použili příkazu `Free`, nikoliv `Destroy`. Bylo to z důvodu, že procedura `Free` nejdříve zjistí zda daný objekt existuje a pokud existuje, tak pak teprve volá destruktor třídy `Destroy`.

6.8 Dědičnost

Dědičnost je další důležitou vlastností při objektově orientovaném programování. Představme si, že máme již vytvořenou třídu `TAuto`, které splňuje požadavky pro veškerá auta. Nyní však budeme chtít vytvořit třídy pro auta osobní a nákladní, které budou mít stejné vlastnosti jako třída `TAuto`. Navíc u třídy `TOsobniAuto` budeme požadovat vlastnost `pocetMist` a u třídy `TNakladniAuto` budeme požadovat vlastnost `nosnost`.

Díky dědičnosti v OOP je definice těchto tříd velmi jednoduchá:

```
type
  TOsobniAuto = class(TAuto)
    pocetMist: Integer;
  end;
  TNakladniAuto = class(TAuto)
    nosnost: Integer;
  end;
```

Jak je vidět, syntaxe nových tříd je velmi jednoduchá, stačí pouze do závorky za příkaz `class` napsat jméno třídy, po které bude nová třída dědit vlastnosti a metody. Znamená to tedy, že např. nová třída `TOsobniAuto` obsahuje vlastnosti `spz`, `typ`, `objemMotoru`, `palivo`, `spotreba` a `pocetMist` a metody `Ujed` a `Tankuj`.

Pokud chceme aby vlastnost `pocetMist` byla zadávána již při vytváření objektu osobního auta, musíme změnit konstruktor třídy `TOsobniAuto`. To uděláme obdobně, jako jsme to dělali ve třídě `TAuto`.