Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 18.12. 2000
Url: http://www.builder.cz/art/cpp/cpp_zaklad_obp.html

Základní pojmy objektově orientovaného programování

Základní pojmy objektově orientovaného programování

V tomto článku bych se chtěl zmínit o základních pojmech objektově orientovaného programování (Dále jenom OOP). Možná se bude tento článek někomu zdát příliš teoretický, protože zde nebude ani řádka zdrojového textu. Mě se ale zdá vhodné zmínit se o významu jednotlivých pojmů jako je třída, instance třídy a podobně, než začnu popisovat jejich implementace v jazyce C++. V dalších kapitolách svého seriálu článků se budu odkazovat na zde definované pojmy. U jednotlivých pojmů úmyslně pro vaší lepší představu vybírám příklady nejen z oblasti programování, ale i z reálného světa.

Objekt

Jak již asi každého napadne, základem OOP je objekt. Co je to objekt? - Objekt je entita, která má svou identitu (Každý objekt lze jednoznačně odlišit od objektu jiného.) a své vlastnosti (Každý objekt má nějaký vnitřní stav a nějak se chová vůči svému okolí). Každý asi i bez této definice podvědomě tuší, co to objekt je. Objekty jsou například: strom, auto, člověk, ale také datový soubor, okno v grafickém operačním systému, tlačítko v grafickém operačním systému, různé datové struktury jako je pole, fronta, zásobník atd... Je zřejmé, že všechny tyto objekty splňují podmínky uvedené v definici. Každý objekt, který má být užitečný, musí poskytovat svému okolí nějaké služby. Například auto se umí rozjet, zastavit atd. Okno v grafickém OS se umí minimalizovat, zavřít, aktivovat, atd. Datový soubor se umí otevřít, zavřít, lze do něj zapsat, lze z něj číst. Uživatel objektu k jednotlivým objektům přistupuje jako k tak zvaným "černým skříňkám". Nezajímá se, jakým způsobem objekty služby poskytují, ale zajímá se jen jaké služby poskytují. Například každý ví, co způsobí sešlápnutí brzdového pedálu v autě, ale málokdo by byl schopen přesně popsat, co se v motoru v tom okamžiku přesně děje. Pro zabrždění auta to prostě nemusí vědět. Stejně tak mnoho programátorů (já taky) ve svých programech běžně otevírá i zavírá soubor, ale příliš netuší, co všechno program a operační systém při otevírání souborů provádí. Právě tento přístup "černých skříněk" k objektům v OOP je dost podstatný. Programátor, který používá objekt, se nestará o to, jak byl naprogramován. Vytvořím-li objekt v OOP, vždy se jedná o jakýsi model objektu reálného světa.

Zpráva

Objekt nikdy neexistuje sám v nějakém vakuu. Objekt je vždy obklopen jinými objekty, kterým poskytuje svoje služby a po kterých služby požaduje. Požadavku na provedení služby se říká zpráva. "Zašlu-li" objektu zprávu, nastane jeden ze dvou případů:

- Objekt zprávě rozumí (Je schopen ji přijmout.) a zareaguje na ni nějakou svou metodou. Například sešlápnutím pedálu brzdy v autě vyvolám metodu motoru - brždění.

- Objekt zprávě nerozumí (Není schopen zprávu přijmout.) a nijak nezareaguje, případně jako svou reakci oznámí uživateli, že není schopen zprávu přijmout. Například u okna v grafickém OS nelze sešlápnout brzdový pedál.

Množina zpráv, které objekt může přijmout, se nazývá rozhraní (Anglicky interface). Popisu, jakým způsobem lze objektům zasílat zprávy (Pořadí a podobně), se nazývá komunikační protokol objektu. Například datový soubor lze nejprve otevřít pro zápis, až poté do něj zapisovat, poté jej zase zavřít. Pojmy rozhraní objektu a komunikační protokol objektu se v OOP rovněž velmi často používají. Jak jsem se zmiňoval v předchozím odstavci o "černých skříňkách", uživatel objektu zná jen rozhraní objektu, a komunikační protokol objektu. K datům (atributům) v objektu lze přistupovat jen pomocí rozhraní. Tedy vlastně vyvoláním metod. Tento jev se nazývá zapouzdření. Více o zapouzdření se budu věnovat v následujícím článku, kde zapouzdření ukážu přímo v jazyce C++.

Dalsím důležitým pojmem v OOP je polymorfismus (Mnohotvarost). Mohou existovat různé objekty, které na stejnou zprávu zareagují různě - polymorfismus. O polymorfismu se zmíním podrobněji v následujících kapitolách, kde ukážu, jak implementovat polymorfismus v C++.

Program napsaný v OOP je vlastně množina mezi sebou navzájem komunikujících objektů.

Třída objektů

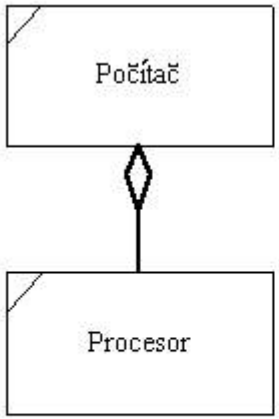
Opět definice: "Třída objektů je abstrakce množiny podobných objektů". Je zřejmé, že některé objekty mají mnoho společných vlastností - jsou stejné třídy. Například objekt, na který se dívám při psaní tohoto článku, má mnoho společných vlastností s objektem, který sledujete vy při čtení tohoto článku - oba jsou třídy monitor. Nebo máte-li nějaký grafický OS, máte patrně na monitoru několik otevřených oken. Každé okno je objekt. Každý objekt vypadá jinak - každé okno zobrazuje něco jiného, přesto jsou si nějak podobné. Všechny objekty jsou stejné třídy. Každý objekt je nějaké třídy. Je-li objekt nějaké třídy, nazývá se instancí této třídy. Každá třída může mít libovolný počet instancí. Dále ve svých článcích budu místo pojmu objekt používat spíše pojem instance. Budu tím myslet objekt třídy, která je zřejmá, nebo kterou uvedu.

Vztahy mezi třídami

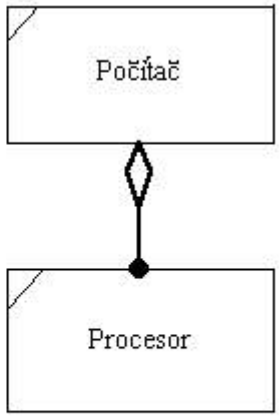
Pro popisy vztahů mezi třídami existuje mnoho druhů třídních diagramů. Mně se nejvíce zamlouvá syntaxe diagramů, které zavedl pan Rumbaugh, proto je zde budu používat. V těchto třídních diagramech jsou třídy znázorněny obdélníky, a vazby mezi třídami jsou nějak označené úsečky, které spojují obdélníky. Pro matematiky: Třídy jsou uzly grafu, vztahy jsou hranami grafu. Mezi třídami existují 3 druhy vztahů (vazeb):

- Agregace - "obsažení" (Anglicky aggregation) Jestliže třída A agreguje třídu B, potom instance třídy A v sobě obsahují instance třídy B. Například v

každé instanci třídy počítač existuje instance třídy procesor. Znázorněno v třídním diagramu:

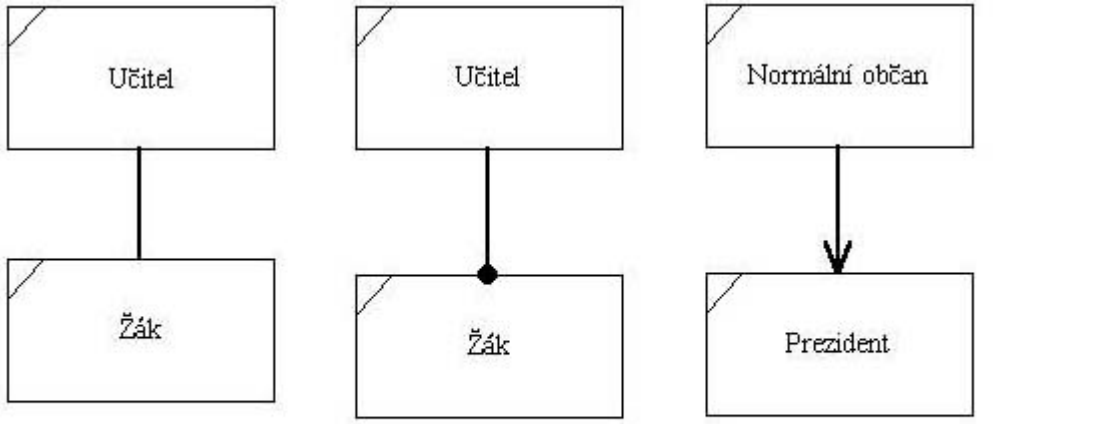


Počítač obsahuje právě jeden procesor - PC



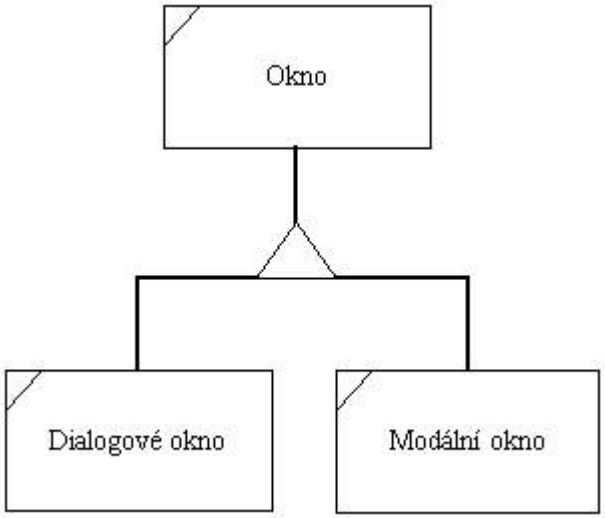
Počítač obsahuje více procesorů.

- Asociace - "Link" (Anglicky association) Jestliže Třída A asociuje třídu B, potom instance třídy A nějakým způsobem musejí "vědět" o instancích třídy B. Zná-li objekt nějaký jiný objekt, znamená to, že jej může používat, zasílat mu zprávy a podobně. Například instance třídy učitel zná instance třídy žák. Není zde vhodná vazba agregace, protože učitel neobsahuje žáky uvnitř sebe. Asociace je obecnější vazba než agregace. Znázorněno v třídním diagramu:



Učitel zná právě jednoho žáka. Učitel zná více žáků. Občan zná jednoho prezidenta.
Žák zná právě jednoho učitele. Žák zná jen jednoho učitele. "Jednostranná asociace"
Prezident asi nezná všechny občany

- Dědičnost - "Specializace" (Anglicky inheritance) Dědičnost je doslovný překlad slova inheritance. Inheritance však někdy bývá přeložena jako specializace. Pojem specializace se mi líbí více, i když není přesným překladem. Zdá se mi, že lépe vystihuje daný vztah. Dále ve svých článcích budu používat zavedený pojem dědičnost. Třída A dědí ze třídy B, jestliže třída A je podmnožinou třídy B. Třída A se nazývá podtřídou třídy B, a třída B se nazývá nadtřídou třídy A. V tomto případě instance třídy A jsou zároveň instancemi třídy B, naopak to však nemusí platit. Například třída okna v grafickém OS je nadtřídou třídy dialogových oken, nadtřídou třídy modálních oken a podobně. Znázorněno v třídním diagramu:



Pro programátora je nutné, aby rozeznával mezi těmito vztahy. Rozdíl mezi asociaci a agregaci je zřejmý. Při agregaci objekty vždy obsahují jiné objekty v sobě. Při vztahu agregace (A agreguje B) by měly cizí objekty požadovat služby instancí agregované třídy (Třídy B) jen pomocí služeb instancí třídy A. Často se chybně zaměňuje mezi agregací, a dědičností. Je-li vztah mezi objekty "má", nebo "vlastní", potom se jedná agregaci. Například řetězec má (vlastní) znaky. Naopak je-li mezi objekty vztah "je", potom se jedná o dědičnost. Například dialogové okno je okno.

Upřímně děkuji všem, kteří tuto teorii dočetli až do konce. V příštích několika článcích naopak bude mnoho zdrojových textů, ve kterých budu používat jednotlivé pojmy, ale nebudu je již vysvětlovat. V příštím článku se dozvíte, jak vytvářet v C++ třídy a jejich instance.

------ http://www.builder.cz ------