2. Základy OOP

Všechny programy, které vytváříme, jsou simulací našeho okolního světa. Okolní svět je světem objektů. Budeme-li chtít, aby naše programy modelovaly tento svět co nejvěrněji, měly by být schopny modelovat obecné objekty spolu s jejich specifickými vlastnostmi a schopnostmi.

Kvalita programu a rychlost jeho tvorby je velice úzce svázána s hladinou abstrakce, kterou při jejich tvorbě používáme. Budeme-li např. programovat ovládání robota, bude pro nás výhodnější jazyk, v němž můžeme zadat příkazy typu „zvedni pravou ruku", než jazyk, v němž musíme vše vyjadřovat pomocí strojových instrukcí. Máme-li proto být schopni rychle vytvářet kvalitní programy, měli bychom mít k dispozici jazyk, jehož jazykové konstrukce nám umožní co nejvyšší míru abstrakce, při níž se můžeme vyjadřovat tak, abychom mohli přirozeně popsat modelovanou skutečnost.

Všechny moderní programovací jazyky se honosí přídomkem „objektově orientované". Tím se nám snaží naznačit, že nabízejí konstrukce, které umožňují rozumně modelovat náš okolní, objekty tvořený svět.

Nejprve trocha teorie

Než se vrhneme na vlastní programování, povíme si nejprve trochu o nejdůležitějších termínech, s nimiž se budeme v dalším výkladu setkávat. Abych vás tou teorií příliš neunavoval, tak tyto termíny opravdu jen zavedu a nebudu je nijak rozpitvávat. Postupně je pak podrobněji vysvětlím na příkladech v dalším textu. Důležité je nyní pouze to, abyste získali základní, rámcovou představu, co daný termín znamená.

Objektově orientovaný přístup

Programovací jazyk Java je plně objektově orientovaný jazyk, tedy v tomto jakyce pracujeme pouze s objekty a jejich rodiči - třídami. Dále si specifikujeme objekt a jeho vlastnosti.

Objekt je základní, jedinečnou a jednoznačně identifikovatelnou entitou, která je dána identitou, neboli parametry, které odlišují objekt od ostatních objektů a dále chováním, neboli službami, které objekt poskytuje vzhledem ke svému okolí (ostatním objektům).

Následující výpis shrnuje pět vlastlostí objektově orientovaného programování:

1. **Všechno je objekt.** Považujte objekt za nějakou fiktivní proměnnou. I když uchovává **data**, můžete na něj mít své **požadavky**. Můžete po něm chtít, aby na sobě vykonával nějaké operace. Teoreticky můžete použít libovolný pojem řešeného úkolu (psy, domy, služby, ...) a v programu jej vyjádřit jako objekt.
2. **Program je tvořen skupinou objektů, které si posíláním zpráv navzájem říkají, co je třeba udělat.** Chcete-li objekt o něco požádat, **odešlete mu zprávu**. Zprávy můžete považovat za požadavky na zavolání funkce, která patří k určitému objektu.
3. **Každý objekt má vlastní paměť, vytvořenou na základě jiných objektů.** Znamená to, že nový typ objektů vytváříte seskupením několika dalších objektů. Tak můžete celou složitost programu ukrývat za jednoduchostí objektů.
4. **Každý objekt má určitý typ.** V programátorském jazyce by se řeklo, že **každý objekt je instancí třídy**, kde **třída je synonymem slova typ**. Asi nejdůležitější vlastností, jež od sebe odlišuje jednotlivé objekty, je: "Jaké zprávy mohu tomuto

objektu posílat?"

5. **Všechny objekty určitého tyu mohou přijímat tytéž zprávy.** Vzhledem k tomu, že je objekt typu **KRUŽNICE** také objektem typu **TVAR**, můžeme mít jistotu, že objekt **KRUŽNICE** bude přijímat všechny zprávy objektu *TVAR*. Znamená to, že můžeme psát kód, jenž komunikuje s jednotlivými tvary, ale zároveň automaticky popisuje všechno, co reprezentuje tvary obecně. Tato nahraditelknost neboli zastupitelnost je jednou z nevýkonnějších koncepcí OOP.

Třídy a jejich instance

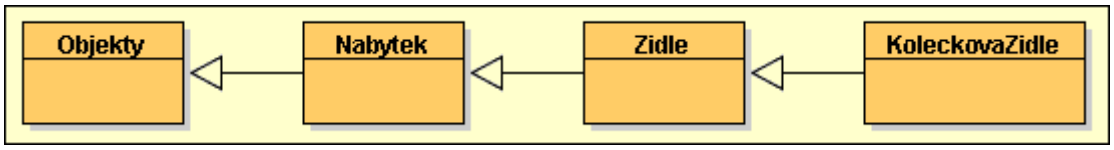
Objekty, s nimiž se ve svém okolí setkáváme, můžeme rozdělit do kategorií, které v programování označujeme jako třídy. Vlastní objekty pak označujeme jako instance jejich třídy – např. židle, na které sedíte, je instancí třídy židlí. Mohli bychom tedy říci, že instance (objekt) je nějakou konkrétní realizací své třídy (židle, na které sedím, je konkrétní realizací obecné židle).

Třídy popisují společné vlastnosti svých instancí. Zdá-li se nám, že uvnitř třídy existuje skupina objektů, které mají nějaké zvláštní vlastnosti, můžeme pro ně definovat nějakou speciální třídu, která bude podtřídou obecnější třídy, tzv. nadtřídy. (Když jsem se již zmínil o židlích, můžeme mezi nimi vyčlenit např. třídu kolečkových židlí, která bude podtřídou třídy židlí.) Podtřída bývá často označována jako dceřinná třída nebo odvozená třída a její nadtřída pak jako rodičovská třída. (V našem příkladu je třída kolečkových židlí dceřinnou třídou třídy židlí a naopak třída židlí je rodičovskou třídou třídy kolečkových židlí.)

Třída může mít většinou několik instancí (třídy s jedinou povolenou instancí – jedináčkem – jsou spíše výjimkou). Na druhou stranu každý objekt je vlastní instancí právě jedné třídy – té, která jej „porodila". Současně jej lze považovat za instanci kteréhokoli z jejích rodičů (prarodiče se v OOP počítají mezi rodiče).

Židle, na které sedím, je tedy podle předchozího výkladu vlastní instancí třídy kolečkových židlí, ale v případě potřeby ji mohu považovat i za instanci obecné třídy židlí a kdybych šel dál i třídy nábytku (rodičovská třída třídy židlí) a obecných objektů (rodičovská třída nábytku).

Pro znázornění hierarchie tříd (tj. pro znázornění, kdo je čím rodičem a potomkem) se používá grafický jazyk UML (Unified Modeling Language – můžete se s ním seznámit např. v publikaci J. Schmuller: Myslíme v jazyku UML.Grada, 2001.) V něm se třídy znázorňují vodorovně rozdělenými obdélníky, v jejichž horní části je uvedené jméno třídy. Rodičovskou posloupnost pak znázorňujeme šipkami zakončenými nevyplněným trojúhelníkem, které vedou od dceřinných tříd k jejich rodičům. Náš předchozí příklad s židlemi bychom v jazyku UML znázornili např. tak, jako na následujícím obrázku.



Metody

Zde se objevuje drobný terminologický guláš. Java převzala terminologii jazyka C++ a nehovoří o posílání zpráv, s nímž přišli zakladatelé objektově orientovaného programování, ale o volání metod, které je bližší programátorům pracujícím v klasických jazycích. Pokud jste již někdy programovali, tak pro ty z vás, kteří programovali v

Baltíkovi, budou metody něco podobného jako Baltíkovi pomocníci, ti, kteří programovali v jiných jazycích, je mohou považovat ze ekvivalenty procedur a funkcí.

Metoda je část programu, kterou instance spustí v reakci na obdržení zprávy. Každá zpráva má v programu přiřazenu svoji vlastní metodu. Programátor v metodě definuje, jak bude objekt na příslušnou zprávu reagovat. Svým způsobem je jedno, jestli řeknete, že instanci pošlete zprávu nebo že zavoláte její metodu. Termín zpráva budu proto používat spíš v souvislosti s popisem chování programu (tj. prakticky celou tuto kapitolu), termínu metoda pak budu dávat přednost ve chvíli, kdy budu hovořit o konkrétní části programu realizující odpověď na zaslání příslušné zprávy. Jak jsem ale řekl, oba termíny jsou si svým způsobem ekvivalentní.

Zprávy

Objekty si mohou navzájem posílat zprávy, ve kterých se žádají o různé služby, přičemž onou službou může být často jen informace. (Můžeme se např. židle zeptat, je-li čalouněná. V praxi bychom to realizovali nejspíše tak, že bychom k ní poslali upřený pohled, v programu k ní vyšleme zprávu.) Objektový program je pak množinou objektů, které si navzájem posílají zprávy.

V objektovém programování proto neplatí občas slýchaná definice, že program je posloupnost příkazů. Chcete-li se naučit programovat objektově, musíte nejprve tuto definici zapomenout.

Historická odbočka:

Definice, že program je zapsaná posloupnost příkazů, platila pro první tři zkonstruované počítače: Babageův Analytical Engine z poloviny devatenáctého století (to není překlep, ten počítač byl opravdu konstruován okolo roku 1850), Zuseovy počítače z konce třicátých let minulého století a Aikenův Mark 1 z počátku let čtyřicátých.

Avšak programy pro známý ENIAC z poloviny čtyřicátých let minulého století by bylo lze označit za posloupnost příkazů jen s velkým sebezapřením, protože obsahoval řadu paralelně pracujících jednotek, které si navzájem předávaly informace. Jeho programování však bylo natolik časově náročné, že jej později převedli na von Neumannovu koncepci, takže v druhé polovině jeho života již pro něj definice programu jako posloupnosti příkazů platila.

V průběhu padesátých let došlo k obrovskému rozmachu nejen počítačů, ale také teorie programování a programátorských technik. Na počátku šedesátých let se tak začalo hromadně používat paralelní, událostmi řízené programování, a takovéto programy bychom za posloupnost příkazů považovat neměli.

Posloupností příkazů nebyly ani programy psané v objektově orientovaných jazycích let sedmdesátých (např. Smalltalk) a už vůbec ne v logických programovacích jazycích let osmdesátých (např. Prolog).

Je samozřejmé, že většina programů se na konec transformuje na posloupnost příkazů ve strojovém kódu, jenže my bychom měli při návrhu a realizaci svých programů zůstávat na optimální hladině abstrakce, a tam pravidlo o posloupnosti příkazů již v současné době většinou neplatí.

Na dotaz, co je to program, lze odpovědět tak, že je to předpis, jak splnit zadanou úlohu, zapsaný v nějakém programovacím jazyce (dokud není zapsaný v programovacím jazyce, není to program, ale algoritmus). Tato definice je možná přesná (a bude nejspíš platit stále), nicméně je tak obecná, že je pro naše účely prakticky nepoužitelná.

Definujme si proto, že objektově orientovaný program je v nějakém programovacím jazyce zapsaný popis použitých tříd, objektů a zpráv, které si tyto objekty posílají, doplněný u složitějších programů ještě o popis umístění programů na jednotlivých počítačích a jejich svěření do správy příslušných služebních programů (např. operačních systémů či aplikačních serverů).

Terminologická odbočka:

Budete-li slyšet programátory velkých aplikací hovořit o „deploymentu“, tak vězte, že hovoří o výše zmíněném rozmísťovacím aspektu svých programů.

Jak je z uvedené definice patrné, přechodem na objektové programování vstupujete do jiného programátorského vesmíru. Do vesmíru, v němž sice použijete mnohé z toho, co jste se naučili při studiu klasického programování, ale v němž se základní úvahy o koncepci a konstrukci programu ubírají naprosto jinými cestami, než na které jste byli doposud zvyklí.