Zkušenosti z praktické aplikace objektového přístupu

František Huňka

Katedra informatiky a počítačů, PřF. Ostravská univerzita, Bráfova 7, 701 03 Ostrava 1, Česká republika

Abstrakt
Příspěvek se zabývá zkušenostmi získanými při výuce objektového přístupu. Tento přístup je dokumentovaný pomocí objektových diagramů Coad & Yourdona. V článku jsou popsány příklady jednoduchých objektů, příklady struktur generalizace / specializace, celek / část a instanční propojení objektů.

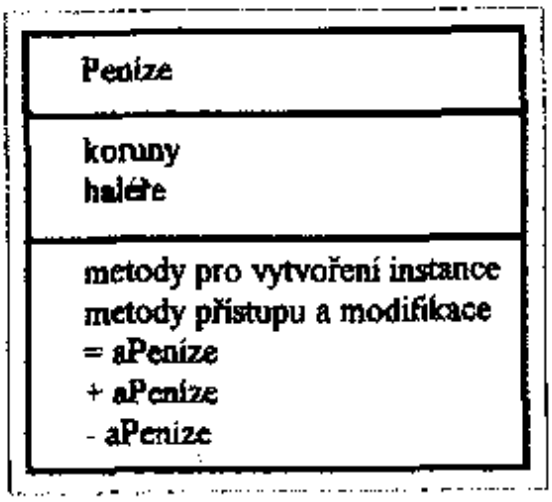
1. Úvod

Objektově orientovaný přístup patří bezpochyby mezi nejvýznamnější změny se kterými se setkáváme na poli software. Hlavní změna je v tom, že objektově orientovaný přístup (Objektově Orientované Programování OOP) umožňuje programátorovi vytvářet software který je orientovaný přímo na problémové domény. Tím myslíme, že přímo odpovídá pojmově i abstraktně modelované realitě daleko vhodněji než jiné způsoby.
Přínosy tohoto přístupu jsou ve znovupoužitelnosti, větší přizpůsobivosti, uniformitě, stabilitě a konečně také ve srozumitelnosti.

Naše zkušenosti říkají, že pro dobré pochopení principů objetkového přístupu je výhodné vše prakticky procvičovat na vhodném softwarovém systému, který umožňuje pouze objcktový přístup. Tím prostředkem je pro nás systém Smalltalk/V for Windows. Ve svém příspěvku zaměřím na obecnější principy, takže Vás nebudu zatěžovat se sémantikou tohoto jazyka.

2. Jednoduché - izolované objekty

Nejlépe je začít od jednoduchého a pak pokračovat ke složitějšímu. Tím jednodušším jsou v našem případě tzv. izolované objekty a jejich implementace. Při implementaci nemáme problémy s určováním, co bude objekt, resp. co budou jeho atributy, protože se jedná o elementární případy. Příkladem takového objektu může být objekt, který uchovává finanční hotovost zadávanou v korunách a haléřích. Jeho struktura (dle Coad - Yourdana) je uvedena na obr. 1.



Obr. 1 Objekt Peníze

Metody:
= *aPeníze* porovnává příjemce zprávy s *aPeníze*. Výsledkem je pravda při rovnosti resp. nepravda.
+ *aPeníze* sečte příjemce s *aPeníze* a vrátí nový objekt jako výsledek součtu.

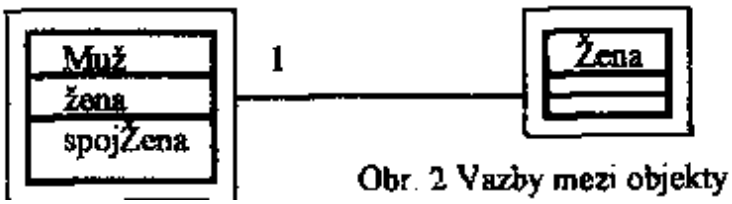
Při bližším experimentování přijdeme na to, že je výhodnější mít místo dvou proměnných *koruny haléře* pouze jednu proměnnou např. *celkemHaléřů*. Tato proměnná pak může obsahovat celkovou částku v haléřích. Při vstupu (přičtení, odečtení) se nejdříve převede vstupní hodnota na haléře a pak provede operace součtu. Při tisku se zase nejdříve převede *celkemHaléře* na koruny a haléře. Vnitřní operace se tím zjednoduší.

Obdobný příklad pak existuje s převáděním jednotek např. teplot, nebo délkových jednotek. Zde potřebujeme dvě proměnné. Jedna pro vlastní uchování počtu jednotek a druhá pro uložení typu jednotky. Dále je třeba doplnit všechny nutné metody, aby byl příklad funkční.

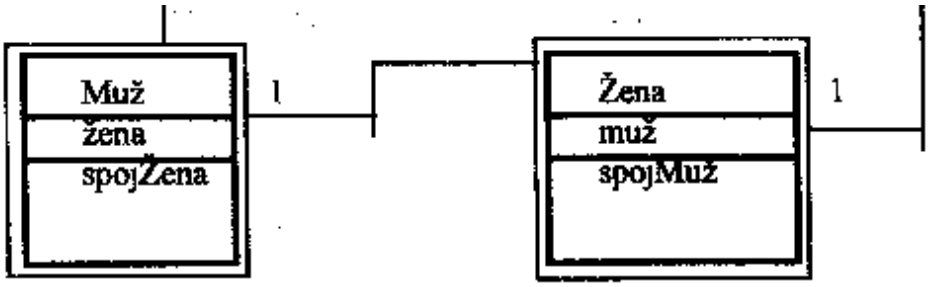
3. Vazby mezi jednoduchými objekty

Zde předpokládám vazbu mezi objekty, které spolu nemají vazbu generalizace / specializace (dědění). Jedná se tedy o dva nezávislé objekty. V tomto případě se vazba realizuje pomocí vnitřních proměnných objektu (instančních proměnných). Existují celkem čtyři možnosti, jak to realizovat

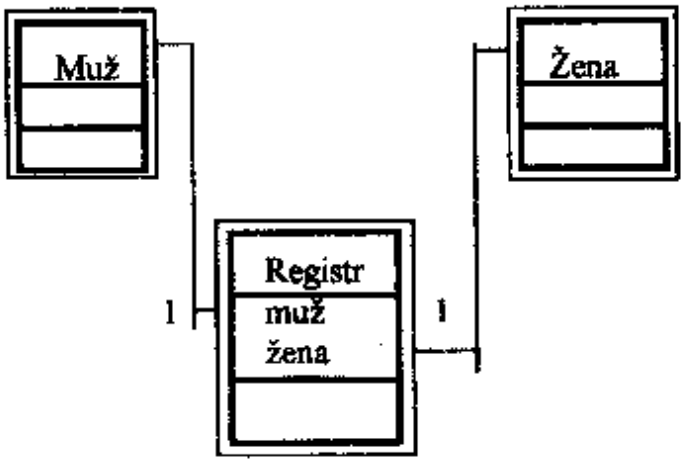
- pouze první objekt explicitně ví o druhém objektu (obr. 2)
- pouze druhý objekt explicitně ví o prvním objektu (obr. 2)
- oba objekty o sobě vědí explicitně (obr. 3)
- žádný z nich o sobě explicitně neví, ale existuje třetí objekt, který má vazbu na oba objekty (obr. 4)



Obr. 2 Vazby mezi objekty



Obr. 3 Vazby mezi objekty

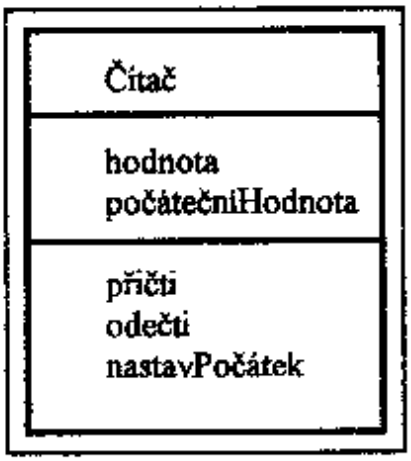


Obr. 4. Vazby mezi objekty

4. Vazba generalizace / specializace

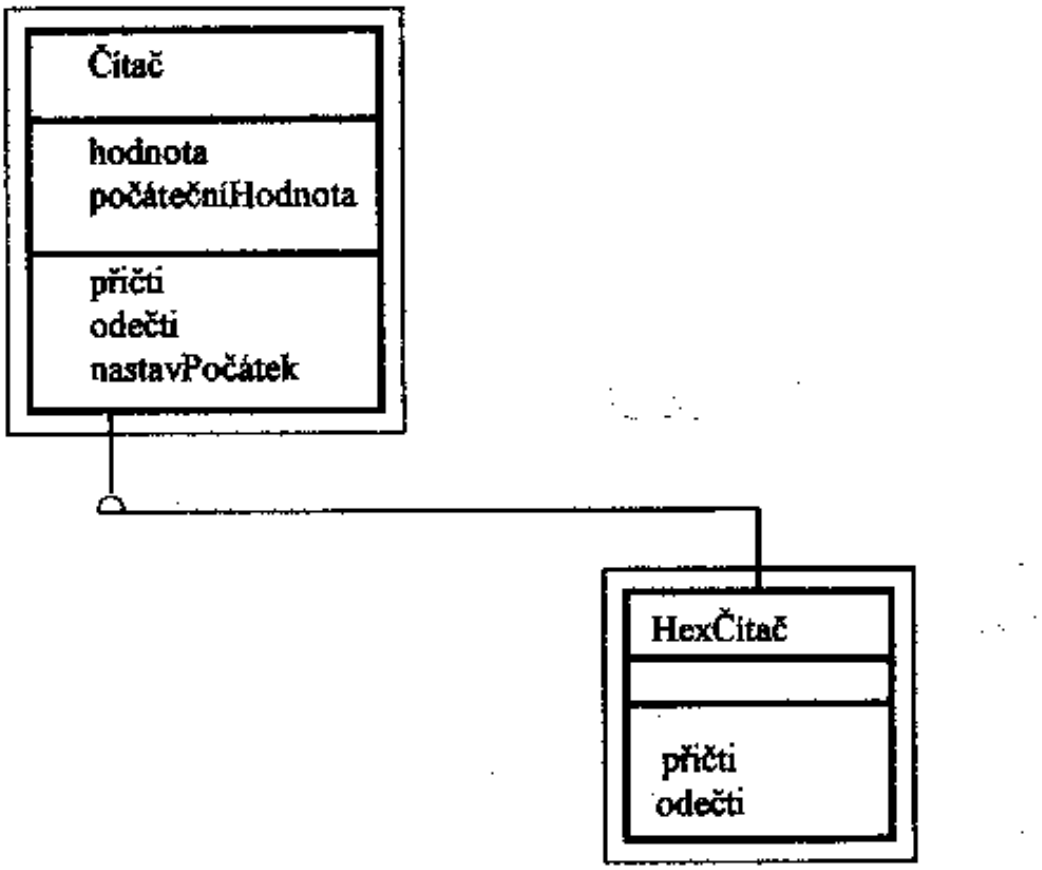
Tato vazba vyjadřuje dědičnost mezi objekty. Čím postupujeme v hierarchii objektů níže, tím konkretizujeme jednotlivé vlastnosti objektů. (Předpokládáme samozřejmě systém, kde již existuje hierarchie tříd). Při praktické aplikaci je důležité vytrvat co nejdéle u abstraktního vyjádření a konkretizovat až později.

Příkladem takové strukturu je realizace jednoduchého čítače, který má tři operace:
přičti (přičtení jedničky k obsahu vnitřní proměnné), **odečti** (odečtení jedničky od obsahu vnitřní proměnné), **nastavPočátek** (nastavení vnitřní proměnné na počáteční hodnotu). Struktura tohoto objektu je uvedena na obrázku 5.



Obr. 5 Jednoduchý čítač

Čítač realizuje své operace, jak je asi zřejmé, v desítkové soustavě. Jak by ale celá úloha vypadala při požadavku na realizaci počítání navíc v šestnáctkové soustavě? Jedno z možných řešení je na obr. 6.

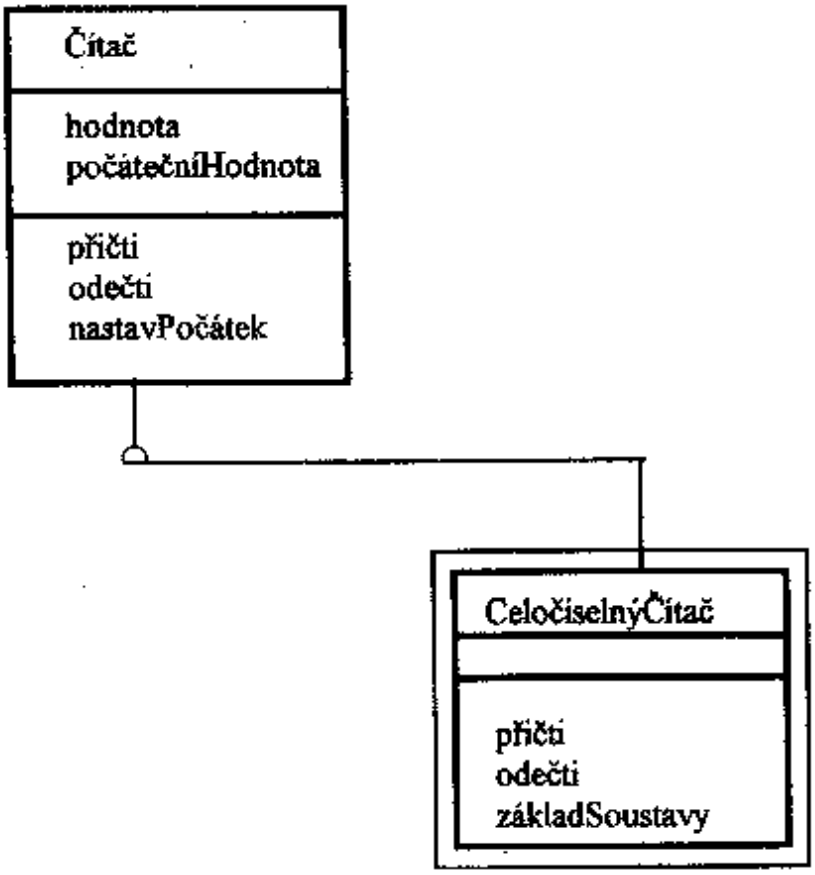


Obr. 6 Desítkový a šestnáctkový čítač

V našem případě je Čítač generalizační třída a HexČítač specializační třída. Tato struktura generalizace / specializace se používá v hierarchii tříd. V našem případě HexČítač dědí atributy a metody z Čítače a zároveň rozšiřuje to, co je definováno pro metody přičti a odečti.

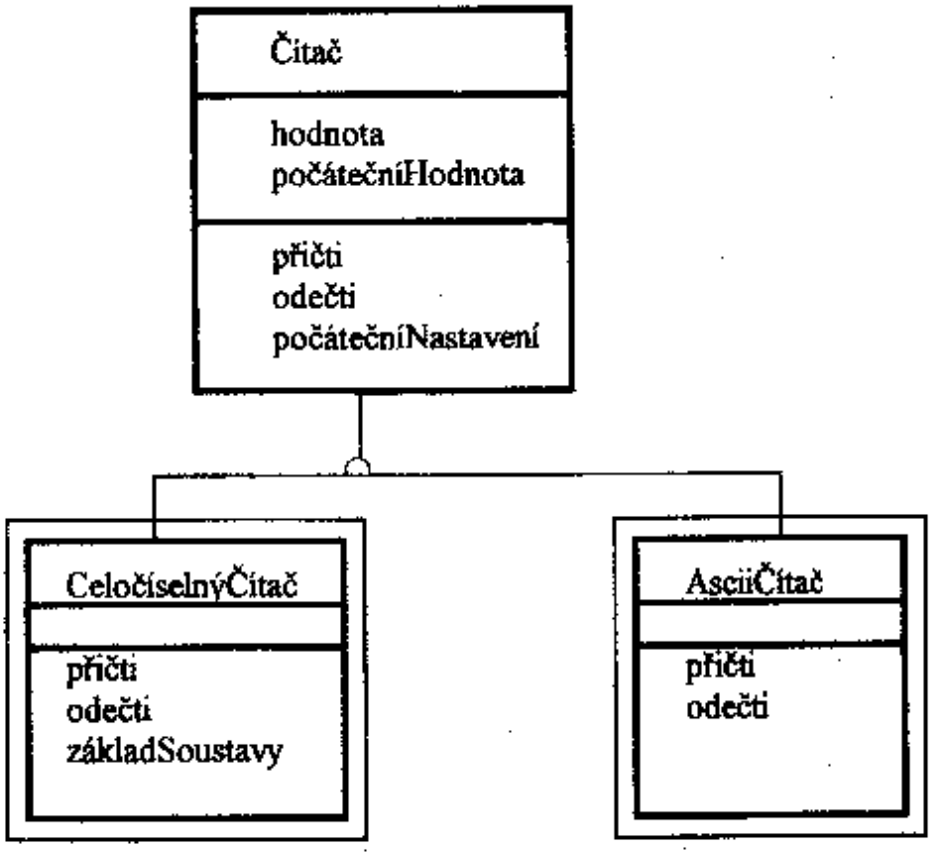
Tato aplikace funguje správně, ale problémy nastanou, pokud chceme ještě zároveň počítat se základem jiným než 10 nebo 16 (např. 5, 8 nebo 24). Nejjednodušší způsob, jak bychom to realizovali, je přidání dalších tříd na úroveň celočíselnéhoČítače. Tím by nám ale rostl počet závislých tříd. Jiný způsob řešení spočívá v další specializaci. Při této specializaci se vytvoří z původního Čítače tzv. abstraktní třída Čítač. Podtřída HexČítač se specializuje na CeločíselnýČítač a realizuje veškeré požadavky na celočíselné zvyšování a snižování Čítače v libovolné soustavě viz obr. 7.

Metoda základSoustavy pak provádí převod čísel na požadovaný základ číselné soustavy. Abstraktní třída reprezentuje třídu, která nemá objekty, má pouze specifikace. Jako výsledek se nyní vlastní HexČítač objevil ve specializované třídě, ne v generalizační třídě jako původně. K realizaci uvedených požadavků nám nyní stačí pouze dvě třídy.



Obr. 7 Celočíselný čítač

Další rozšíření uvedené aplikace může být např. o čítač ASCII znaků. Svým způsobem velmi koresponduje s CeločíselnýmČítačem.



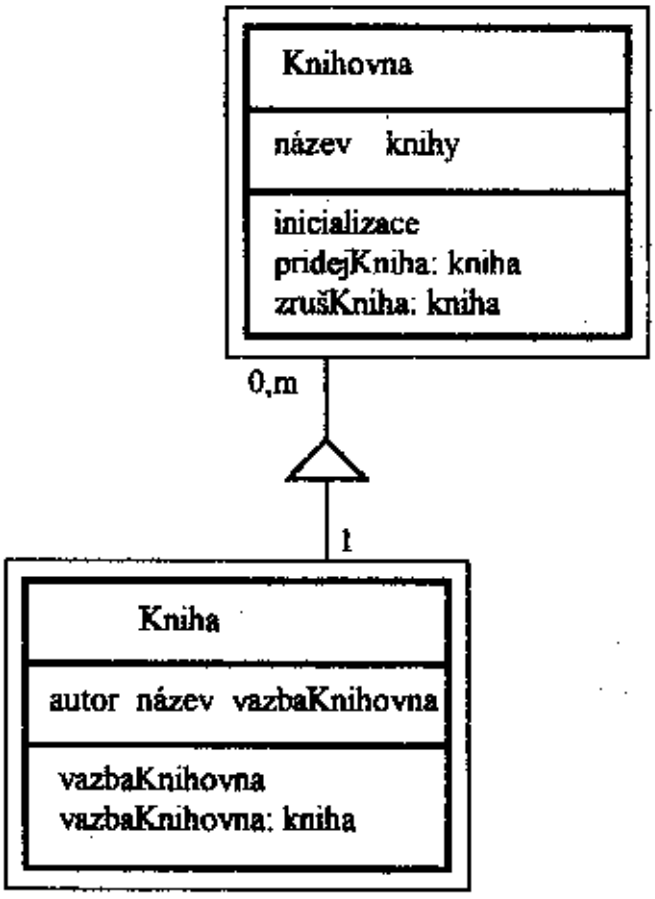
Obr. 8 Obecný čítač

Stejným způsobem na úrovni CeločíselnéhoČítače můžeme přidat ještě např. sekvenčníČítač, který vybírá sekvenčním způsobem zadané konstanty, nebo čítač pro hodnoty datum.

Jak je vidět, správným výběrem a postupnou specializací dosáhneme mnohdy lepšího řešení, než bylo původní. Při praktické aplikaci jsou tyto konstrukce pochopitelně složitější.

5. Vazba celek/část

Další strukturou, se kterou se setkáme při objektovém přístupu, je struktura celek / část (whole/part). Tuto strukturu si můžeme ukázat na jednoduchém příkladu vazby objektů Kniha - Knihovna. Zatímco *kniha* reprezentuje jeden objekt, *knihovna* je jakýsi kontejner objektů. Při realizaci takové struktury potřebujeme mít zachovány obě vazby, to je od *knihy* na *knihovnu* a obráceně. Co se týká *knihovny*, tak ta implicitně z podstaty věci v sobě obsahuje *knihy*, tedy má na ně spojení. Obráceně propojení *knihy* z *knihovnou* neexistuje a pokud to chceme, musíme ho explicitně doplnit přes vnitřní proměnnou. Struktura těchto objektů je pak následující:



Obr. 9 Struktura celek / část

Zde jsme uvedli pouze metody (operace), které jsou nutné k výkladu. U objektu *Kniha* proměnná *název* reprezentuje *název knihy*, u objektu *Knihovna* pak proměnná *název* reprezentuje název Knihovny.

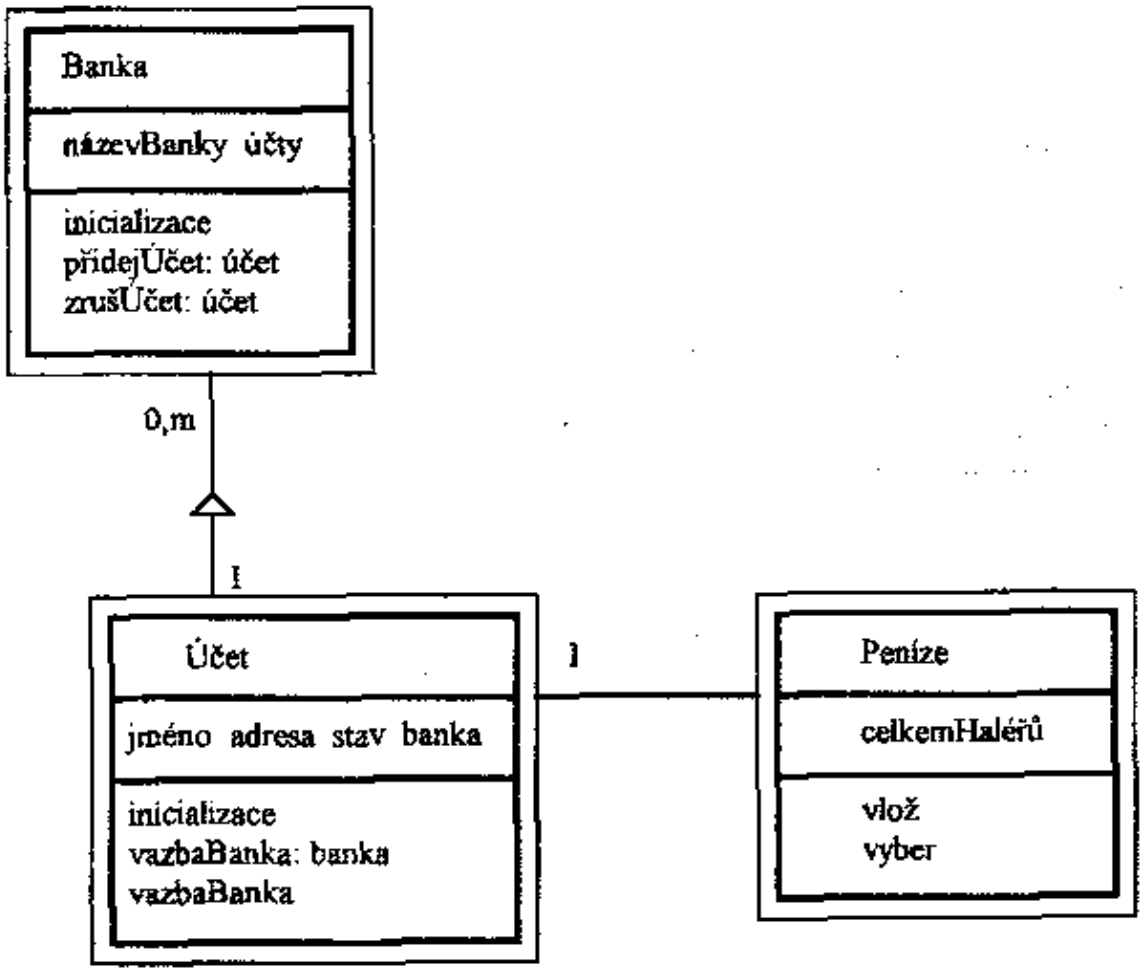
Vysvětlení metod:
vazbaKnihovna - vrací spojení na knihovnu

vazbaKnihovna: knihovna - nastavuje, modifikuje, nebo ruší spojení (v závislosti na parametru knihovna)

U objektu knihovna má zvláštní význam proměnná *knihy*, která vlastně reprezentuje vlastní kontejner pro ukládání. Zde využíváme další vlastnosti objektového přístupu a to je skládání objektů. Právě toto skládání objektů je někdy označováno za silnější vlastnost než dědění. V našem případě *knihy* reprezentuje objekt třídy *OrderedCollection*, což si zjednodušeně můžeme představit jako pole, jehož velikost se může dynamicky měnit.

Metody *přidejKniha: kniha* a *zrušKniha: kniha* realizují příslušné operace a implicitně také vazby. Díky vlastnosti skládání se pak nemusíme obávat, že by přímo rozumněly také jiným zprávám (např. třídy OrderedCollection).

Další příklad na uvedenou vazbu a na skládání objektů je komplexnější a realizuje spojení mezi bankou, účtem a penězi - viz I. příklad. Banka je jedna a eviduje spoustu účtů. Účet se zase prostřednictvím své proměnné odkazuje na objekt třídy Peníze. Struktura je následující:



Obr. 10 Struktura celek / část banka-účet-peníze

Objekt Banka:
účty - proměnná, která reprezentuje kontejner na objekty - skládání objektů
metoda *inicializace* nastavuje proměnnou *účty*
metody *přidejÚčet* a *zrušÚčet* realizují uvedené operace

Objekt Účet:

banka - proměnná pro spojení s bankou, propojení realizují metody *vazbaBanka: banka* a *vazbaBanka*
stav - proměnná, která je objektem třídy *Peníze* - skládání objektů

6. Interakce s uživatelem

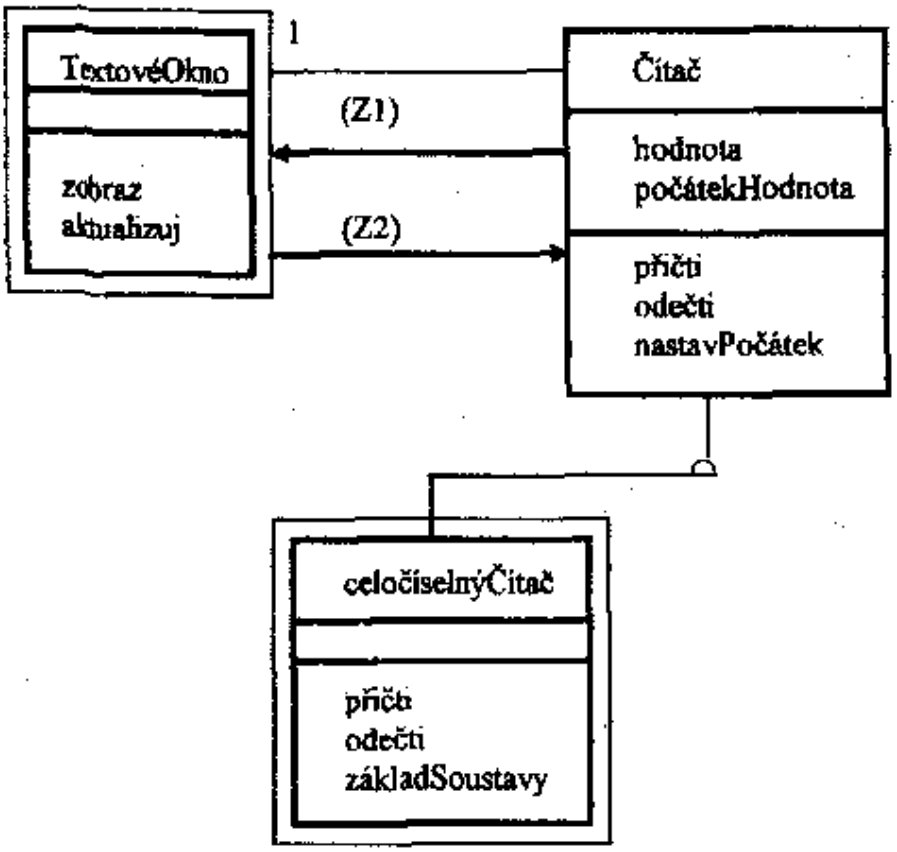
Dosud jsme se zabývali tzv. datovými doménami, což je základní vrstva. Další a trochu náročnější vrstva je tzv. interakce s uživatelem. Tato interakce je v objektových systémech vytvářena trochu odlišně než u klasických systémů. Při tvorbě je třeba oddělit od sebe objekty které tvoří model (aplikaci), pohledy a ovládání vstupů.

Uvažujme případ čítače. Při interakci s uživatelem potřebujeme doplnit okno, které má vykonávat následující funkce:

- nalézt informaci, kterou potřebuji zobrazit
- zobrazit informaci
- změnit svoji hodnotu při vnější změně

Zobrazení informace může znamenat různé formy zobrazení např. grafické nebo textové.

Jednoduché schema vypadá následovně:



Obr. 11 Celočíselný čítač - jednoduché okno

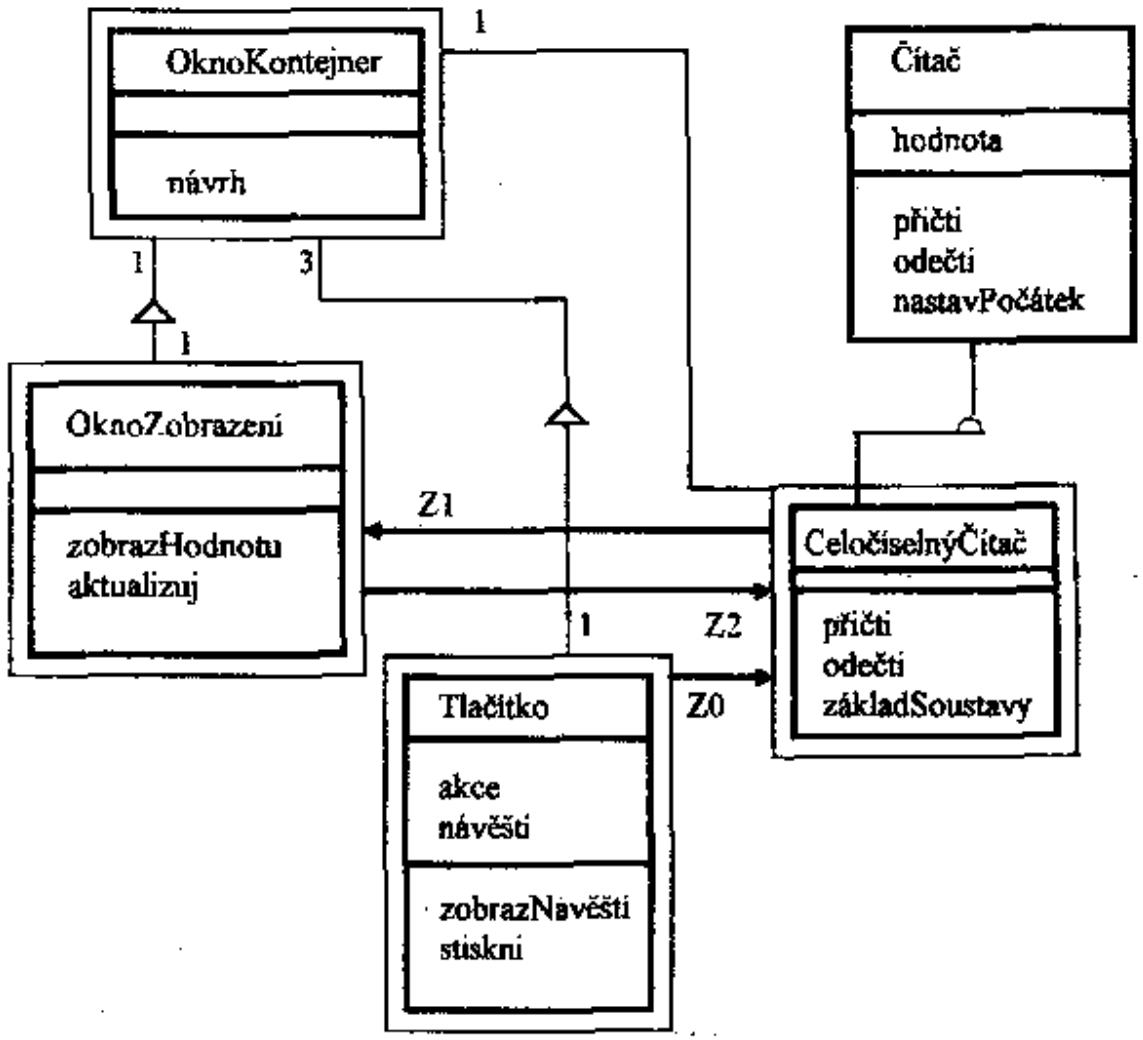
Celočíselný čítač reaguje podle následujícího scénáře:

dostal jsem zprávu přičti
zvýším svoji hodnotu
oznámím, že moje hodnota se změnila (changed) - zpráva Z1

Textové okno vykonává následující činnosti:

dostal jsem zprávu aktualizace (update) říkající, že se změnila hodnota čítače
požádám čítač o novou hodnotu
zobrazím novou hodnotu - zpráva Z2

V dalším kroku budeme specializovat TextovéOkno. To při bližším zkoumání nemusí být pouze jednoduchý objekt, ale může fungovat jako kontejner objektů, který sdružuje vlastní okno a objekt tlačítka, pomocí kterého můžeme ovládat požadované operace (přičti odečti nastavPočátek). Struktura je na následujícím obrázku.

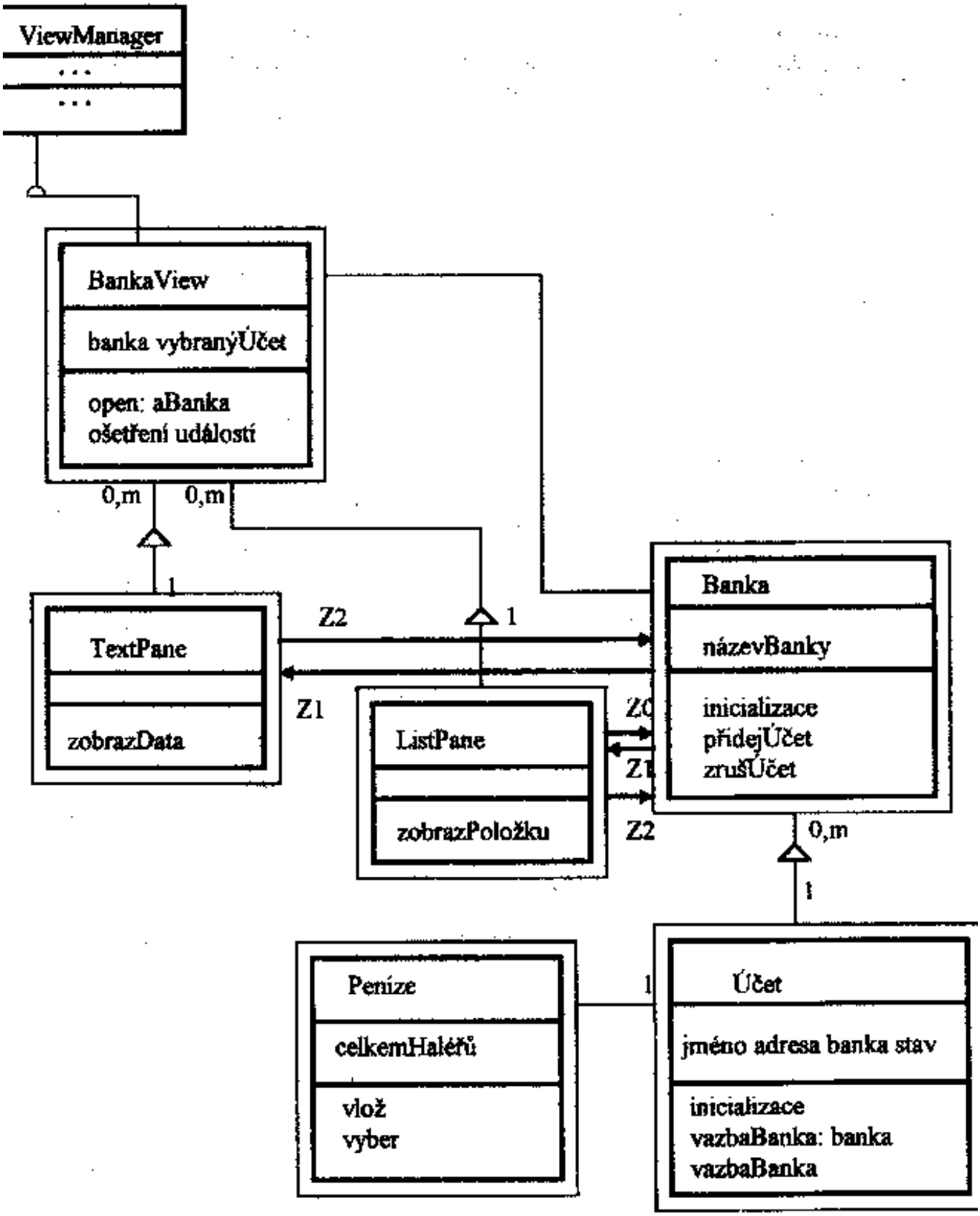


Obr. 12 Celočíselný čítač - interakce s uživatelem

Trochu složitější případ nastane, když potřebujeme do naší aplikace vložit okno, které slouží pouze pro výběr jedné položky. Potřebujeme například vytvořit telefonní seznam, kde v jednom okně se budou zobrazovat jména a ve druhém okně další doplňující informace jako např. telefonní číslo a adresa. V prvém okně vybereme pouze jméno a ve druhém se zobrazí dodatečné informace k vybranému jménu.

Nyní si ukážeme, jak by interakce s uživatelem a požadavek na vybírání jedné položky seznamu vypadal pro příklad z obr. 10. Pokud vyjdeme z konkrétní situace ve Smalltalku/V for windows, máme k dispozici třídu *ViewManager*, pod kterou si vytvoříme konkrétní vlastní třídu naší aplikace např. *BankaView*. Tato třída pak v rámci struktury *celek/část* bude

obsahovat ještě další dvě podtřídy a to textovou tabuli (TextPane) a tabuli pro výběr položky (ListPane). Mezi objekty *BankaView* a *Banka* pak existuje instanční vazba, vlastní zprávy které řídí výměnu informací jsou prováděný mezi *TextPane* resp. *ListPane* a objektem *Banka*.



Obr. 13 Struktura bankovního účtu

7. Závěr

Domníváme se a naše zkušenosti to potvrzují, že pro pochopení principů objektového přístupu je právě počáteční experimentální postup nejlepším řešením.

Literatura:

1. LaLonde, W. : Discovering Smalltalk. The Benjamin/Cumming Publishing Company, Inc Redwood City, California 1994
2. Coad,P.,Nicola,J. : Object-Oriented Programming. Yourdon Press Prentice-Hall 1993