6 Objektově-orientovaný vývoj programového vybavení

6.1 Co značí „objektově-orientovaný“

- organizace SW jako kolekce diskrétních *objektů*, které zahrnují jak data tak chování

- objekt:

  OMG: Objekt je „věc“ (thing). Je vytvořen jako instance *objektového typu*. Každý objekt má jednoznačnou *identitu*, která je odlišná a nezávislá na jiných jeho vlastnostech. Každý objekt nabízí jednu nebo více *operací*.

  Firesmith: Objekt je definován jako *SW abstrakce*, která modeluje všechny relevantní aspekty skutečné nebo konceptuální entity nebo věci z aplikační domény nebo prostoru řešení. Objekt je jedna z hlavních SW entit v OO aplikaci, typicky odpovídá SW modulu, obsahuje množinu *typů* atributů, *atributů*, *zpráv*, *výjimek*, *operací* a nepovinných *vložených objektů*.

Code/Yourdon: Objekt je abstrakce něčeho v doméně problému, odrážející schopnosti systému uchovávat informaci o něm, interagovat s ním nebo obojí, *zapouzdření* hodnot *atributů* a výlučných *služeb*.

Martin/Odell: Objekt je libovolná „věc“, reálná nebo abstraktní, o níž ukládáme *data* a *metody*, které s nimi manipulují.

Jacobson: Objekt je charakterizován řadou *operací* a *stavem*, který pamatuje efekt těchto operací.

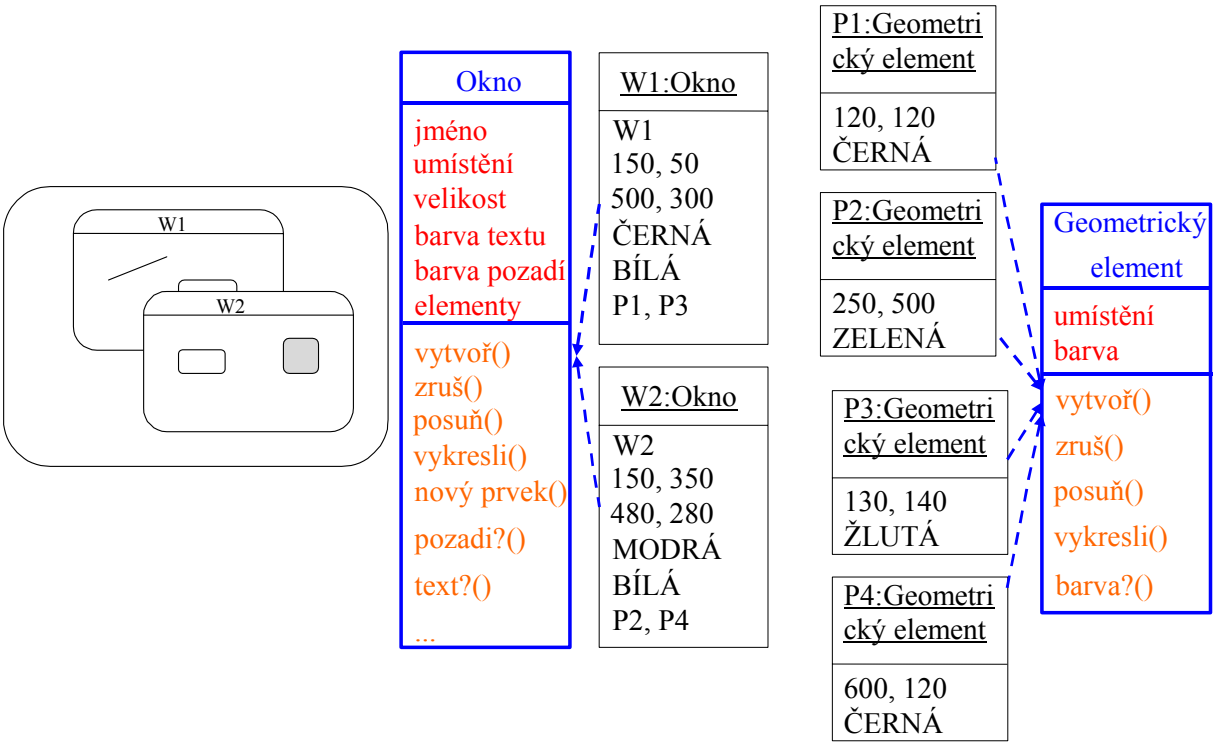
Rumbaugh: Koncept, abstrakce nebo „věc“ s jasným *rozhraním* a významem pro daný problém.

- společné rysy:
  - ♦ reálná „věc“ (koncový uživatel) nebo abstrakce (reprezentace - programátor)
  - ♦ objekt reálného světa může být konkrétní nebo abstraktní (konceptuální)
  - ♦ každý objekt je jednoznačně identifikovatelný
  - ♦ zahrnuje data a operace s nimi

6.2 Základní vlastnosti objektové orientace

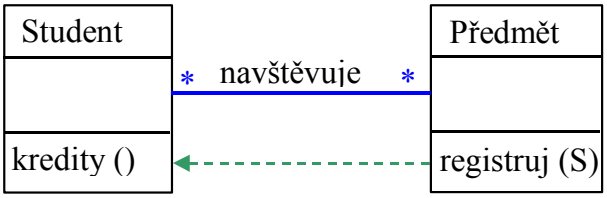
- ***abstrakce*** - zaměření se na podstatné rysy, klasifikace objektů do *tříd* (objektových typů).
  Třída - abstrakce popisující vlastnosti důležité pro aplikaci a ignorující zbytek. *Objekt* je instance třídy.
  Př) Osoba, Okno, Grafický element

- ***zapouzdření*** (encapsulation)- ukrývání informace (information hiding), oddělení externích aspektů od interních detailů (implementace). Praktický důsledek: data objektu a funkce, které s nimi operují, tvoří celek s dobře definovaným rozhraním.
  Vlastnosti třídy: atributy, operace (implementace se nazývá *metoda*).
  - chceme-li objekt použít, musíme pouze znát jeho rozhraní
  - třídu lze chápat jako „šablonu“ pro tvorbu objektů daného typu (objekty téže třídy mají stejnou strukturu a definici operací)
  - atributy objektů, vlastní atributy třídy (jako celku)
  - operace objektů, vlastní operace třídy (jako celku)

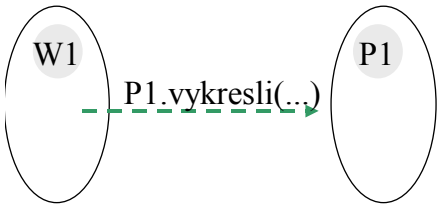


- struktura objektu je dána třídou, stav provedenými operacemi

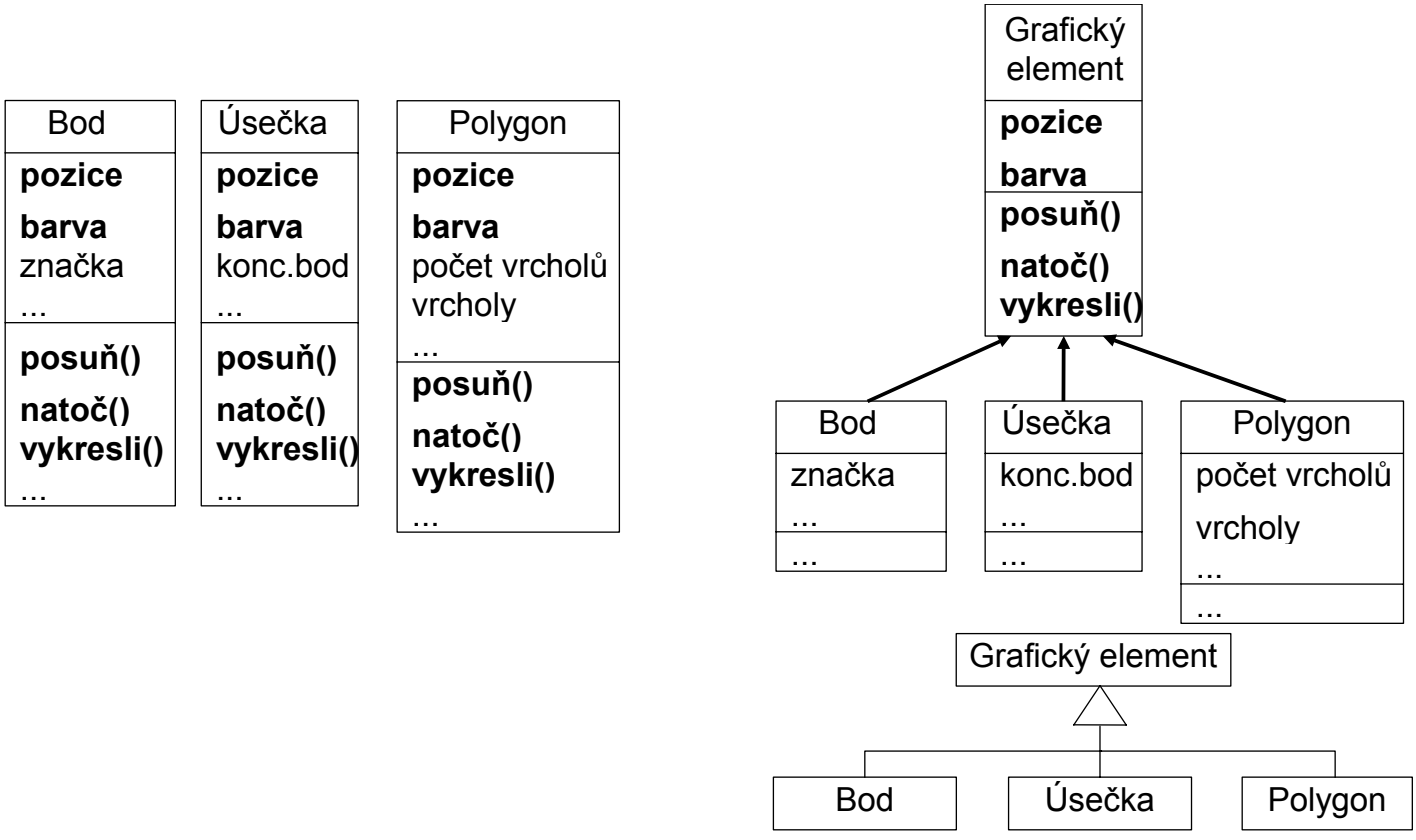
- **mezi objekty tříd existují vztahy:**
  - **a) statické (asociace, generalizace/specializace, agregace)**

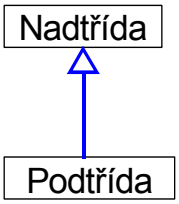


  - **b) dynamické (zprávy)**



- ***dědičnost*** **- sdílení atributů (definic ne hodnot) a operací třídami**





Podtřída *dědí* všechny vlastnosti nadtřídy.

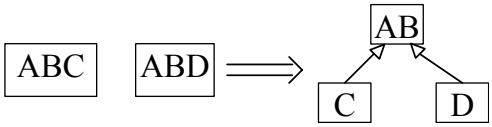
- různé úrovně abstrakce → hierarchie dědičnosti

- třídy abstraktní a konkrétní

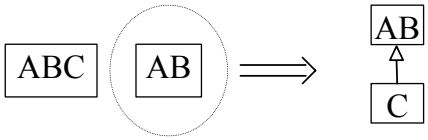
- důvody použití dědičnosti:

♦ opakované použití (úspora, údržba, testování)

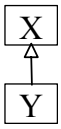
a) vytváření abstraktních tříd



b) využití knihoven tříd

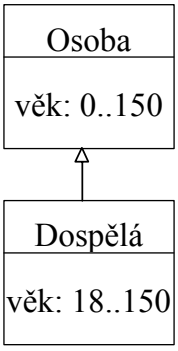


♦ vytváření podtypů (specializace)



X a Y mají *kompatibilní chování*, lze-li použít všude Y místo X , tj. Y jen přidává vlastnosti.

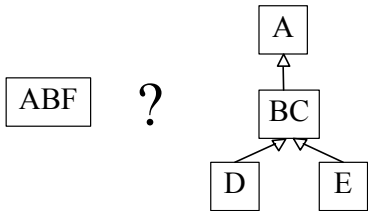
♦ zpřesnění



♦ konceptuální

ve fázi konceptuálního modelování během analýzy - vztahy generalizace/specializace (ISA)

- vytvoření dobré hierarchie dědičnosti je jedním z klíčových problémů OO návrhu



a) vytvoření třídy bez dědění

b) nalezení vhodného předchůdce

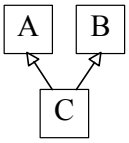
c) restrukturalizace hierarchie

d) redefinice vlastností

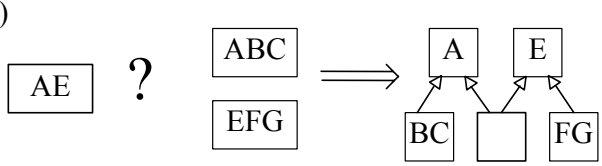
- pojmy redefinice a přetěžování

- Která varianta nejlepší?
  - další modifikace
  - co chceme modelovat
  - srozumitelnost, jasné pochopení
  - cena restrukturalizace

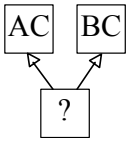
- dědičnost jednoduchá, vícenásobná



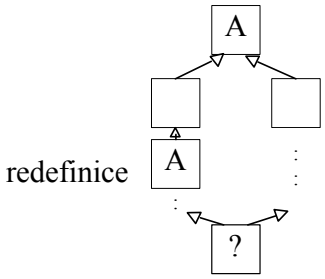
Př)



- problémy:



C++: obě vlastnosti

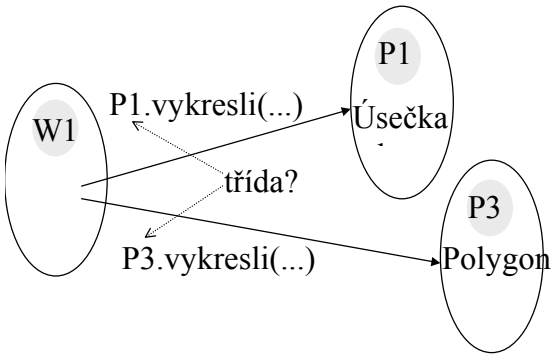


C++: vlastnost A dvakrát nebo použití tzv. virtuální bázové třídy

- ***polymorfismus*** - různé chování téže operace pro různé třídy (projev).

  Př) operace vykresli()

  - jiný pohled (komunikace objektů): vysílající objekt nemusí znát třídu adresáta (podstata).



- polymorfismus neomezený/omezený (hierarchií dědičnosti)
- virtuální metody, pozdní (dynamická) vazba

abstrakce + zapouzdření → „object-based“ (ADT),
+ dědičnost, polymorfismus → „object-oriented“

- další důležité pojmy:
  - *agregace* - významný statický vztah mezi objekty (PART OF)
  - *souběžnost* (concurrency) - důležitým aspektem OO vývoje je určit, které objekty jsou/mají být současně aktivní.
  - *perzistence* - objekt může být přechodný nebo perzistentní
  - *viditelnost* - vlastnosti veřejné nebo privátní, důležitý aspekt návrhu a implementace.
- důvody použití OO technologie při vývoji programů:
  - růst produktivity - až 10x (aplikace s GUI), OO technologie dává předpoklady, nutnost připravenosti organizace
  - rychlejší vývoj aplikací
  - kvalita, udržovatelnost - jednotný konceptuální rámec (třídy, objekty, ...), méně chyb díky opakovanému použití, zapouzdření, lepší podmínky pro údržbu

6.3 Objektově-orientované techniky modelování

- představují reprezentaci určitého pohledu na systém

a) model objektové struktury
- zahrnuje zpravidla:
  - ♦ třídy domény problému a třídy pro implementaci,
  - ♦ atributy a operace tříd,
  - ♦ statické vztahy mezi třídami a objekty,
- základní model - používán ve fázi analýzy i návrhu
- důvody pro používání dalších modelů:
  - ♦ zjišťování a pochopení požadavků, včetně dynamického chování,
  - ♦ dokumentace v podobě srozumitelné zákazníkovi,
  - ♦ dostatečně podrobná informace pro návrháře a programátora,
  - ♦ zvládnutí vývoje rozsáhlých systémů - rozdělení na menší jednotky.

c) modely chování systému

- modely požadované funkčnosti a použití systému (Use Case) a toků aktivit,
- modely stavů systému a přechodů mezi stavy,

c) modely interakce objektů

- s okolím systému,
- vnitřních objektů

d) modely stavů objektů a přechodů mezi stavy

e) modely rozčlenění systému na podsystémy

6.4 Objektově orientované metody a metodologie

a) metody navazující na techniky strukturované analýzy a návrhu (Rumbaugh - OMT, Coad/Yourdon - OOA/OOD, Martin/Odell - OOIE, Booch)

b) metody založené na nových technikách modelování (Jacobson - OOSE (Use Case), Wirfs-Brock - RDD (Responsibility Driven Approach))

c) metody druhé generace - využití perspektivních rysů obou předchozích (Fusion, Mainstream Objects)

d) metody třetí generace - snaha o standardizaci (modelovací jazyk UML (Unified Modeling Language))