36. OOP

OOP
Objektově orientované programování (OOP) je způsob abstrakce, kdy algoritmus implementujeme pomocí množiny zapouzdřených vzájemně komunikujících entit.

Objekt
Objekt je entita zapouzdřující stavové informace (data reprezentovaná dalšími objekty) a poskytující sadu operací nad tímto objektem nebo jeho částmi. Objekty vzájemně komunikují prostřednictvím zasílání zpráv.

Rozhraní
Množina všech zpráv, kterým objekt rozumí (protokol)

Základní koncepty OOP

1. objekty - spojují data a funkcionalitu
2. abstrakce - pohled na vybraný problém reálného světa a jeho počítačové řešení. Detaily jsou skryty do černé skříňky (s okolím komunikuje přes své rozhraní, vnitřní implementace je skryta)
3. zapouzdření - uživatel nemůže měnit interní stav objektu libovolným způsobem, využívá poskytované operace nad objektem (rozhraní)
4. polymorfizmus - mnohotvárnost - různé objekty mohou na stejnou zprávu reagovat různě
5. dědičnost - implementace sdíleného chování, nové objekty mohou využívat chování již existujících objektů

Identita x shodnost - identita porovnává, zda jsou objekty totožné, shodnost porovnává podle obsahu

Třídně orientované jazyky

Třída
Šablona, podle níž mohou být vytvářeny objekty (instance třídy). Vytvoření instance probíhá pomocí *konstruktoru* (speciální metoda)

Kopírování objektů
hluboká kopie - kromě objektu jako takového (soubor atributů) jsou kopírovány i objekty, které ony instanční proměnné referencují.

mělká kopie - je vytvořen nový objekt, ale všechny instanční proměnné obsahují odkazy na totožné objekty jako kopírovaná předloha

Rušení objektů v paměti
automaticky - provádí jej garbage collector, většinou pouze u virtuálních strojů. GC jednou za čas vyhledá objekty, které již nejsou odkazovány a odstraní je
manuálně - speciální metoda *destruktor* - uvolňuje objekt z paměti

Dědičnost
a) "kolik rodičů může mít potomek"
jednoduchá dědičnost - potomek má nejvýše jednoho rodiče (Java, C#)
vícenásobná dědičnost - třída dědí od více přímých předků, problémy s konflikty mezi jmény členů různých předků
b) "co se dědí"
dědičnost implementace - kromě atributů se dědí i metody včetně implementace
dědičnost rozhraní - dědí se seznam metod, které je nutno v potomkovi implementovat

Typy, podtypy, nadtypy
Třídu můžeme brát jako datový typ, potom je rodič *nadtypem*, potomek *podtypem*.
vyžadovaná dědičnost - pokud je někde vyžadována instance třídy nebo třídy z ní zděděné, dědičnost zajišťuje existenci potřebných metod a atributů (potomek musí mít stejné metody jako rodič, tudíž když ho někam pošlem místo rodiče, tak je jasný, že to bude fungovat)
skutečný podtyp - kontroluje se existence potřebných metod a atributů

Přístupy k typům
čistě objektový - vše je objekt (i základní datové typy), existuje třída, od které jsou všechny objekty odvozeny (Smalltalk)
hybridně objektový - základní datové typy lze skládat do struktur, třída je struktura, která kromě atributů obsahuje i metody (kořenová třída - Java, C# | není žádná kořenová třída - C++)

Staticky typovaný jazyk - množina operací, které objekt podporuje, je známa již v době překladu
Dynamicky typovaný jazyk - kontrola typů probíhá až za běhu, pokouší se o konverzi objektu na jiný typ, pokud nejde, vygeneruje se chyba (*slabě typované jazyky*) nebo "je na tuto skutečnost upozorněn přímo volaný objekt" (*silně typované jazyky*, dafuq)

Redefinice metody
Možnost definovat pro metodu podtřídy novou, specifičtější implementaci (nahrazení implementace z předka)

Časná vazba - překladač v době překladu ví, jaký podprogram (metoda) bude vyvolán. To může

být problém, pokud chceme volat specifické metody pro potomka (mohla by se volat metoda předka)
Pozdní vazba - výběr metody probíhá až za běhu programu, využití tabulky virtuálních metod

http://www.builder.cz/rubriky/c/c--/casna-versus-pozdni-vazba-uvod-do-polymorfismu-v-c--155697cz

Prototypově orientované jazyky

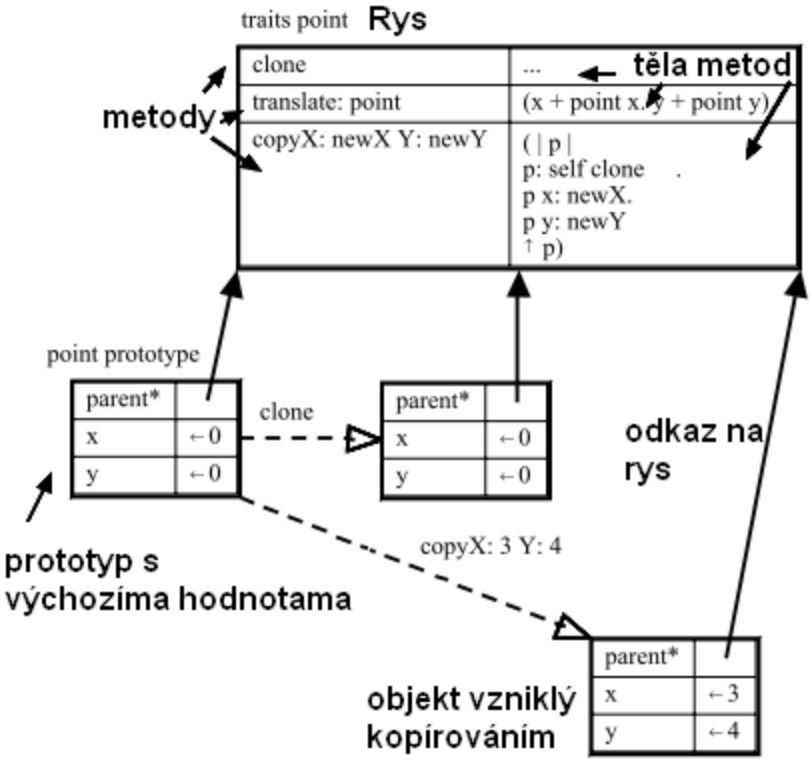
Tyto jazyky znají pouze jediný typ objektu. Jednotlivé složky objektu se jmenují sloty a mohou obsahovat jak proměnné, tak metody. Pokud objekt obdrží zprávu, prohléda množinu svých slotů a buď použije hodnotu proměnné nebo zavolá metodu. Nové objekty se vytváří klonováním (implicitně mělké).

Delegace
Procos podobný zaslání zprávy, při volání metody se nemění kontekt. Co to je - když je zpráva delegována na jiný objekt a v tom objektu se volá metoda, tak se v té zprávy nemění příjemce zprávy, ale ukazuje furt na ten původní objekt (normálně by se to asi změnilo na ten delegovaný objekt)

Rodičovské sloty
Odkazují na objekty, na které se deleguje zpráva, pokud pro daný objekt není nalezen vhodný slot.

Rys
Objekt, který obsahuje sdílené chování - nahrazuje dědičnost (potomek implementuje nějaké metody a v rodičovských slotech odkazuje na rodiče (rysy), kteří mají obecnější metody, takže to je jako dědičnost, navíc může být vícenásobná)

Prototyp
Šablona instance - obsahuje instanční proměnné. Protože rysy obsahují pouze metody, je potřeba k nim ještě nějak přidat proměnné. To se dělá tak, že se vytvoří prototyp s proměnnými inicializovanými na výchozí hodnotu, který obsahuje ještě odkaz na rys (objekt s metodami). Tenhle prototyp se pak klonuje a v těch klonech můžou být různé hodnoty proměnných.



Objektově orientovaný přístup k progamování

Objektově orientovaný přístup k programování je založen na intuitivní korespondenci mezi softwarovou simulací reálného systému a reálným systémem samotným. Analogie je především mezi vytvářením algoritmického modelu skutečného systému ze softwarových komponent výstavbou mechanického modelu pomocí skutečných objektů. Podle této analogie i ony softwarové komponety nazýváme objekty. Objektově orientované programování pak zahrnuje analýzu, návrh a implementaci aspektů, kde jsou reálné objekty nahrazeny těmi softwarovými (virtuálními).

(to sem někde zkopíroval)