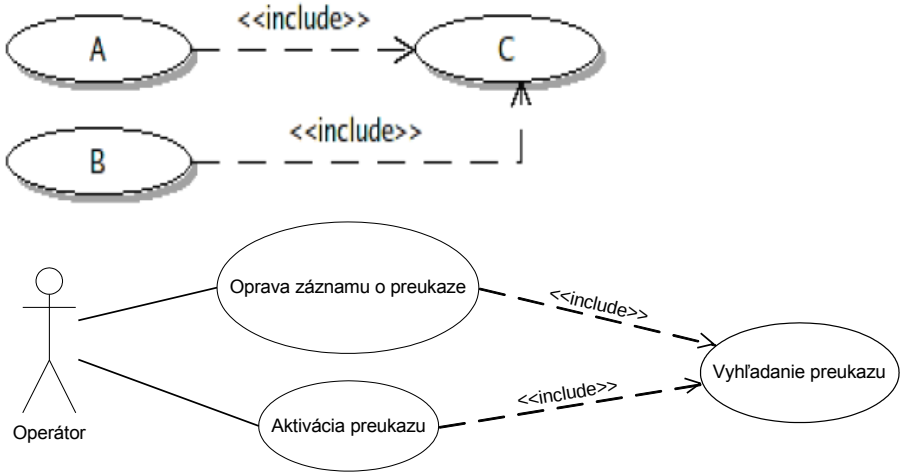
1. vztah <<include>> pri use case

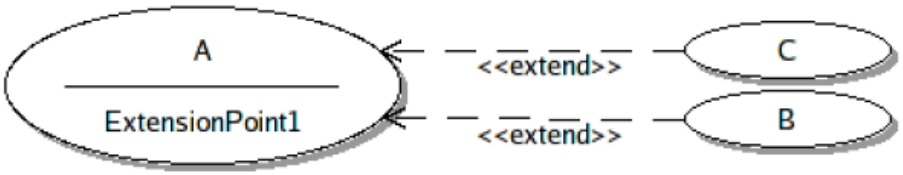
Další názvy: *užívá*
• Vztah mezi dvěma případy užití
• Společné chování dvou nebo více p.u. může být vyčleněno do zvláštního případu užití (vyhýbá se opakování a kopírování společných kroků)
• A a B nejsou kompletní bez C
• C je odkazováno alespoň jednou v A i B
• C může být (a často je) použito ve více případech užití (jinak <<include>> postrádá smysl)
• Pokud je C úplný p.u., může být vyvolán i přímo aktérem
• C je popsáno stejně, jako jiné případy užití



- include – ide o vylúčenie opakovaného popisu
  - základný UC nie je bez vloženého UC spravidla „funkčný“
  - popis základného UC obsahuje miesto, kde treba „vložiť“ vkladaný UC

2. vzťah <<extend>> medzi use cases (vernostni program)

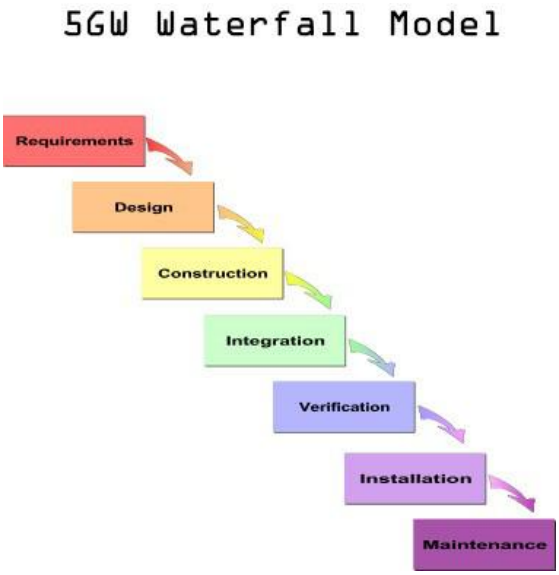
- Další názvy: *rozšiřuje*
- Vztah mezi dvěma případy užití
- Pro vložení **nového chování**, volitelných částí a pro zpracování chyb
- A je úplný p.u., který v ideálním případě neví nic o existenci svých rozšíření B a C
  - ⇨ základnímu scénáři je jedno, kdo ho rozšiřuje
- Umístění rozšiřujícího p.u. je označeno *bodem rozšíření* (extension point)
  - ⇨ v diagramu a/nebo v popisu rozšiřovaného p.u. *“ve vrstvě nad tokem událostí”*
  - ⇨ rozšiřující p.u. ví, jak se přidat do základního scénáře
- Rozšiřující p.u. B a C jsou často *neúplné* p.u. a mohou mít více fragmentů
- *Podmínka pro rozšíření* je uvedena v popisu jednotlivých rozšiřujících p.u. nebo v
- podmínce rozšiřujícího vztahu (od UML 2.0)
- Rozšíření se často používají v následných verzích systému



- Pokud vztah <<extend>> nespecifikuje body rozšíření, aplikuje se na všechny
- Rozšiřující p.u. musí mít přesně tolik segmentů, kolik bodů rozšíření používá
- Je dovoleno aby dva p.u. rozšiřovali základní p.u. ve stejném bodě, pořadí je pak ale náhodné

- extend – ide o sprehľadnenie alternatív v rámci UC
- základný UC je „funkčný“ aj bez rozšírenia
- nepokrýva však všetky alternatívne scenáre
- popis základného UC obsahuje miesto/miesta slúžiace ako body rozšírenia

3. inkrementalny vyvoj, iterativny vyvoj, vodopad - porovnanie



Vodopád:
Rozděluje celý projekt na základě prováděných aktivit:

- Analýza požadavků, návrh, kódování, testování apod.
- Příklad rozdělení ročního projektu:
  - 2 měsíce analýzy
  - 4 měsíce návrhu
  - 3 měsíce kódování
  - 3 měsíce testování

Nikdy to není tak jednoduché, jak to na první pohled vypadá:

- Chyby způsobují návrat do předchozí etapy/etap
- Nejasná hranice mezi etapami
- Překrývaní etap (např. Se implementuje ještě během návrhu)
- Pozdní odhalení chyb
- Odhad ceny

Iteratívny vývoj

- Celý projekt se vyvíjí v několika iteracích
- Iterace směřují k postupnému vylepšení, zpřesnění, dodělání nebo opravení části systému
- Každá iterace obsahuje analýzu, návrh, testování apod. (tj. miniaturní vodopád), ale s různou intenzitou, např.:
  - V první iterace provést celkovou analýzu požadavků a obrysový plán vývoje, více rozpracovat jádro systému, implementovat základní testovací třídy
  - V druhé iteraci podrobně rozpracovat důležité části systému, rozmodelovat je a částečně implementovat
  - Ve třetí iteraci podrobně rozpracovat méně podstatné části systému, doimplementovat věci z předchozí iterace
- Vývoj typu vodopád je tedy iterativní vývoj s jedinou iterací
- Analýza požadavků, analýza, návrh, kování i testovaní se provádí pouze jednou, výsledkem je hotový systém
- rychlejší (dílčí) výsledky – rodina dříve bydlí, byť provizorně
- rychlejší odhalení chyb – rodina zjistí mnohem dříve, že krb je špatný

Inkrementálny vývoj

- Inkrementálně = „přidávat k“
- Uplatňuje se zejména u větších projektů a/nebo v agilním vývoji
- Jednotlivé části systému (přírůstky, inkrementy) vytváříme „nezávisle“ na zbytku a pak integrujeme
- Vývoj jednotlivých přírůstků může probíhat iterativně, vodopádem, XP, …
- Nejčastěji se používá iterativní vývoj přírůstků
- Vývoj typu vodopád je tedy inkrementální vývoj s jediným přírůstkem představujícím celý systém

4. stav, chování a identita objektu

Každý objekt má

- Jednoznačnou **identitu** – odlišení objektu od ostatních (stejných) objektů.

  Každý objekt má unikátní výskyt v prostoru a čase
  - Každý objekt je jednoznačně identifikovatelný
  - Identita je určena hodnotami atributů a adresou v paměti

- **Stav** – hodnoty atributů uchovávají data objektu. Data objektu definují jeho stav.

  Stav je určen *hodnotami atributů* a *vazbami/vztahy na ostatní objekty* v daném časovém okamžiku
  Př. pro tiskárnu:

Stav objektu	Název atributu	Hodnota atributu	Vazba
Zapnuto	power	on	N/A
Vypnuto	power	off	N/A
DošlaNáplň	inkCartridge	empty	N/A
Připojeno	N/A	N/A	Propojeno s objektem počítače
Odpojeno	N/A	N/A	Neprojeno s objektem počítače

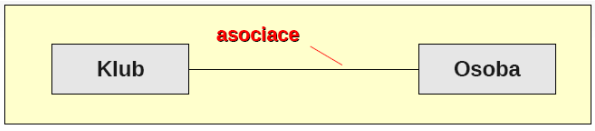
- **Chování** – operace, které je možné s objektem provádět.
  - Chování vyjadřuje, jak objekt koná a reaguje
  - Chování je specifikováno operacemi. Každá operace tedy definuje část chování objektu.
  - Chování se může měnit v závislosti na stavu
  - Př: chování tiskárny při stavu „DošlaNáplň“
  - Vyvolání operace může vést ke změně stavu
  - Implementace operace se nazývá *metoda*
  - Operace často mění stav a často závisí na aktuálním stavu (na aktuálních datech).

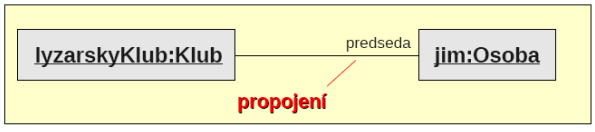
5. Napište rozdíl mezi objektem a třídou, co mají objekty jedné třídy společné a v čem se mužou lišit.

- Každý objekt je instancí právě jedné třídy
- Třída popisuje vlastnosti množiny objektů
  - šablona objektu, která předepisuje všem instancím atributy,
  - operace a vztahy k ostatním objektům/třídám
- instance jedné třídy se ale mohou lišit stavem (aktuálními hodnotami atributů) a chováním předepsaných operací.

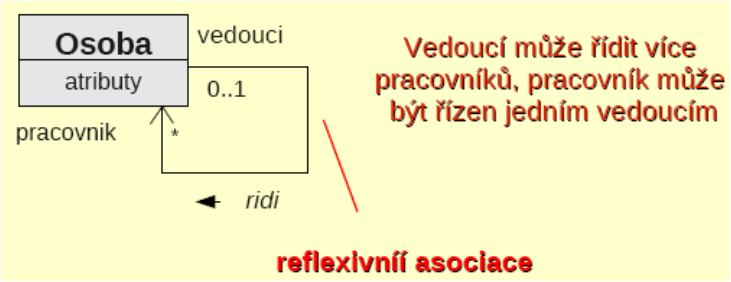
6. Co je to reflexivní asociativita, uvést příklad, nakreslit odpovídající

Asociace je vztah mezi třídami (obdoba propojení u objektů)
- Pokud existuje asociace mezi třídami, lze vytvářet propojení mezi instancemi tříd na diagramu tříd
- Propojení mezi objekty nelze vytvořit (namodelovat), pokud neexistuje asociace na příslušném diagramu tříd





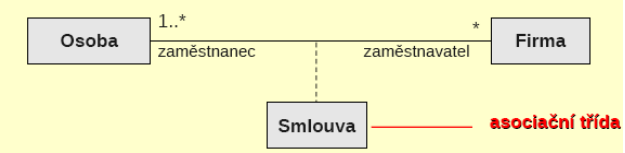
Odkazovanie sa na na inštancie v rámci jednej triedy
RA - Měly by mít vždy role, jinak vzniká chaos



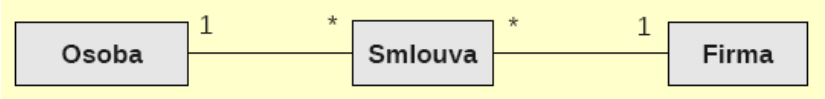
<- v podstate strom

Další “ozdoby” asociací:
» uspořádání: {ordered}
» měnitelnost: {addOnly}
» viditelnost: +, #, -
» navigace: směrové šipky
» omezení: {union}, {subset}, {unique}

7. Co je to asociační třída, příklad + dekompozice



- Veľmi dôležitá konštrukcia pre analytickú fázu
- Trieda, ktorá nemá väzbu na inú triedu, ale na asociáciu
- V návrhu sa potom rozbíjajú na obyčajné triedy
- Obmedzenie – konkrétna inštancia je viazaná práve na 1 asociáciu medzi triedami
- Jedna osoba může mít s jednou firmou pouze jedinou smlouvu
- může existovat pouze jediné propojení mezi objekty třídy Osoba a Firma v daném čase
- ak chceme viac zmlúv medzi jednou konkrétnou osobou a firmou, tak AC dekomponujeme:



8. Dedicnost, polymorfizmus, ich vztah, priklad

Dedičnosť:

- „druh“ hierarchie abstrakcí
- definuje vztah mezi třídami, kde jedna třída (podtřída) sdílí strukturu nebo chování definované v jedné nebo více třídách (nadtřídách):
  - jednoduchá - jedna nadtřída
  - násobná - více než jedna nadtřída
- podtřída může rozšířit nebo změnit existující strukturu/chování nadtříd(y)
- Mechanismus, který vyjadřuje podobnost mezi třídami, zjednodušuje definici třídy pomocí dříve definované(ných) třídy(tříd). Vyjadřuje generalizaci a specializaci tím, že v hierarchii tříd explicitně určuje společné atributy a služby.

Polymorfizmus:

- koncept z teorie typů, kdy jedno jméno může označovat různé věci:
  - „+“ znamená „stejnou věc“ pro real a integer
  - „+“ je implementováno odlišně pro real a integer
- **polymorfismus je důsledkem interakce mezi dědičností a dynamickou vazbou**
  - (pod)třída dědí jméno operace
  - vazba implementované metody na toto jméno nastává až při provádění

9. Model-Driven Architecture, vysvetlit, ake modely zahrnuje.

Vývoj riadený modelmi

- „Seriózní“ přístup, kdy implementujeme striktně na základě UML modelů
- Průběžně vniká dokumentace
- Podpora pro fázi nasazení a údržby
- UML je základním nástrojem MDE
- Metodiky Iconix, Select Perspective, ...

- OMG standard definující rozsah softwarových modelů, způsob jejich vytvoření a použití
- Rozvíjí myšlenku vývoje pomocí modelů (MDE) o *formalizaci* modelů a jejich *automatickou transformaci* (automatizované zpřesňování modelů)
- MDA definuje 4 úrovně modelů:
  - **CIM** – Computation Independent Model - Model nezávislý na počítačovém zpracování (business analýza)
  - **PIM** – Platform Independent Model - Platformově nezávislý model řešení
  - **PSM** – Platform Specific Model - Platformově specifický model řešení
  - **Code** – Kód aplikace, tj. výsledná realizace řešení
- MDA definuje způsob (automatické) transformace modelů

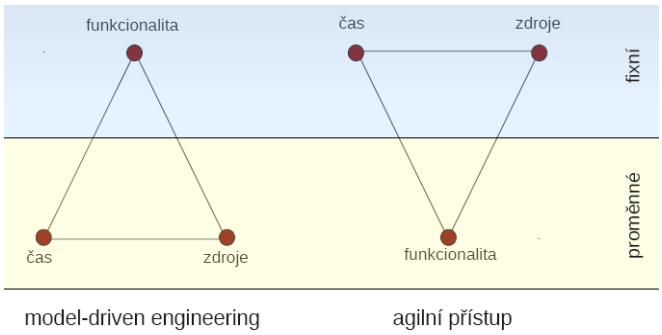
10. Charakterizuj agilné metódy programovania, porovnaj s Model Driven Engineering

Metody zaměřené více na lidi – určujícím faktorem v úspěchu projektu je kvalita lidí pracujících na projektu a jejich spolupráce

- Velmi krátké iterace (jeden měsíc a méně)
- UML se používá pouze jako doplněk
- Malé ale výkonné týmy (dvojice)
- Zapojení zákazníka do vývoje (zákazník se zúčastní sestavování návrhu a testů, ideálně je součástí vývojového týmu)

- Agilní metodiky Extreme Programming (XP), Scrum, Feature Driven Development (FDD), Crystal, Dynamic Systems Development Method (DSDM), ...

- Manifesto of Agile Software Development http://agileManifesto.org
- Individuality a interakce mají přednost před nástroji a procesy
- Fungující software má přednost před obsáhlou dokumentací
- Spolupráce se zákazníkem má přednost před sjednáváním smluv
- Reakce na změnu má přednost před plněním plánu



11. modely UML pro statickou strukturu systemu, dynamicke chovani systemu a dynamicke chovani tridy

Modely ukazující **statickou strukturu** systému:
- diagramy tříd, balíků a objektové diagramy
- implementační diagramy: diagramy komponent, diagramy rozmístění

Modely ukazující **dynamické chování systému**:
- model případů užití => externí pohled na systém
- diagramy aktivit => externí/interní pohled na systém
- interakční diagramy: diagramy sekvencí, komunikace a časování
- diagramy spolupráce => interní pohled na systém

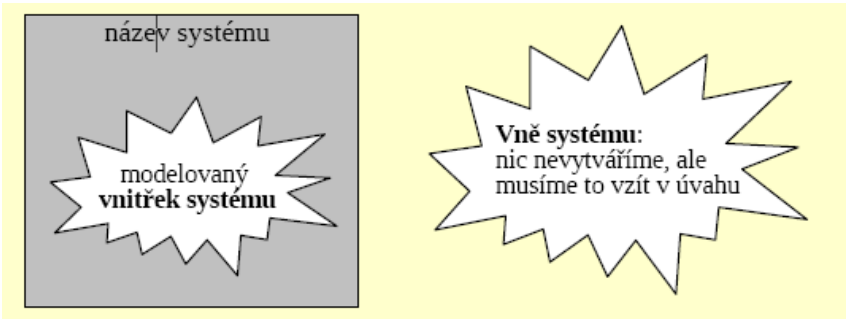
Modely ukazující **dynamické chování jediné třídy**:
- stavové diagramy, diagramy aktivit

12. Jake prostredky poskytuje UML pro ziskani pozadavku na system a urceni hranice systemu.

Use Case Diagram = prostředek k vyjádření konkrétních cílů, kterých chce uživatel dosáhnout prostřednictvím sekvence interakcí se systémem
Use Case Diagram = grafické znázornění + detailní dokumentace jednotlivých případů užití

Subjekt (hranice systému)

Další názvy: *System Boundary*, *Subject* je nový název od UML 2.0
- Definice hranic systému probíhá interaktivně.
- Hranice systému mohou být definovány na několika úrovních abstrakce, účastníci interakcí se mění podle úrovně abstrakce!
- vysoká úroveň bez žádných reprezentantů
- střední úroveň, která bere v úvahu kdo/co skutečně vkládá za informaci
- nižší úroveň, která bere v úvahu, jaká dat opravdu vstupují
- Př: Zálohování systému



13. Čo je to scenár, kde sa používa a aký má vzťah k toku udalostí

Scénář (scenario) : jedna konkrétní cesta případem užití, používa sa na špecifikáciu prípad užitia

- Jiný pohled na případ užití, vhodné pro složité případy užití
- Jeden scénář neobsahuje větvení! – narozdiel od flow of events

Primární scénář : normální průchod, kdy vše probíhá normálně, nebo nejpravděpodobnější cesta. Pro daný p.u. je **jediný**!
Sekundární scénáře: jiné cesty popsané méně podrobně nebo jen vyjmenované

- **alternativní** scénáře: jiné povolené cesty (větvení)
- **výjimečné** scénáře: pro zpracování chyb

Otázky pro nalezení sekundárních scénářů:

- jaké jiné akce mohou být prováděny v tomto bodě?
- je něco, co by se mohlo pokazit v tomto bodě?
- nějaké chování, které může nastat kdykoli během primárního scénáře, např. zrušení nebo znovuspuštění?

14. Toky udalosti, kde sa vyuzivaju, aky je rozdiel medzi nimi a scenarom.

Používajú sa na špecifikáciu činností v prípadoch užitia. Dokumentace pomocí toků nejčastěji obsahuje:
• **vstupní podmínky** (preconditions) – kritéria, která musí být splněna před spuštěním p.u.
• **výstupní podmínky** (postconditions) – kritéria, která musí být splněna na konci p.u.
• **normální tok** událostí nebo interakcí – jednotlivé kroky v případu užití
• **alternativní a výjimečné toky** událostí

Jak popsat tok událostí:

- Volný text
- Číslované kroky
- Pseudo kód (obvykle příliš detailní)
- Diagramy aktivit
- Diagramy interakcí s pseudokódem nebo textem na levé straně

1. p.u. začíná volbou “zobraz obsah košíku”
2. KDYŽ je košík prázdný
   2.1 systém zobrazí uživateli, že košík neobsahuje žádné položky
   2.2 p.u. končí
3. systém zobrazí seznam všech položek v košíku

…

15. Co je to akce, událost? K čemu jsou Entry a Exit události? Pak uvést rozdíl mezi dvěma stavy (je to ve skriptech)

Událost

- vnější stimul, který může vést ke změně stavu
- pri std libovolné jméno, kromě ***entry***, ***exit*** a ***do***
- Př: *stisknuté tlačítko (n)*

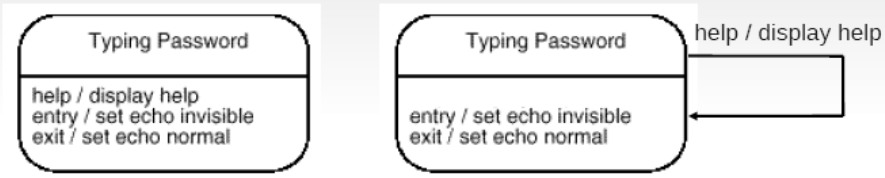
Akce

- reakce na událost
- atomická nepřerušitelná operace, operace jsou prováděny sekvenčně
- mohou přiřazovat hodnoty atributům a spojením

Vstupní a výstupní akce (událost ***entry*** a ***exit***):

- alternativa k zobrazování akcí v přechodech
- používá se pokud všechny přechody z/do stavu vykonávají stejnou akci
- nepoužívat na úrovni analýzy, ale až v návrhu

3) Ostatní vnitřní události:
- události, které nastanou v daném stavu a *ponechávají původní stav*
- alternativa k externí akci na lokálním přechodu
- Př: když nastane událost 'help', je provedena akce 'display help':



rozdíl mezi těmito diagramy?

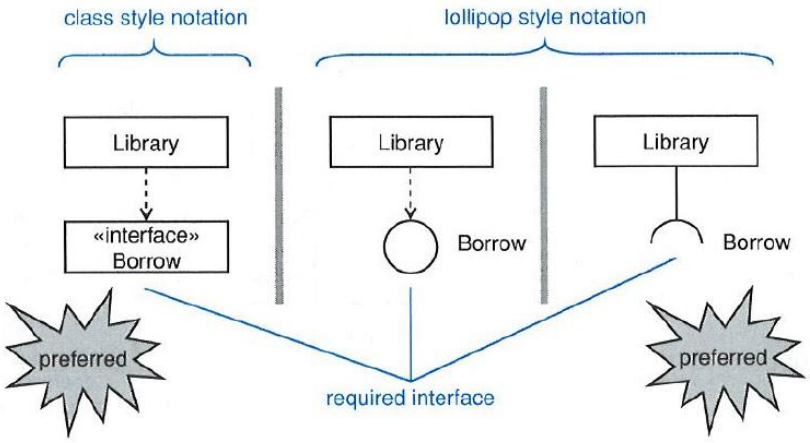
- ***externí akce*** opouští stav a znovu do něj vstupuje
  => spustí se ***enty*** a ***exit*** akce
- ***entry***, ***exit*** a ***interní akce*** často vedou na privátní operace dané třídy,
  externí akce často vedou na veřejné operace

Lisia sa v tom, ze nalavo je to vonkajsia udalost. Lisia sa mierne aj v chovani – pri vstupe nalavo ak zadam help, tak na terminale nic neuvidim

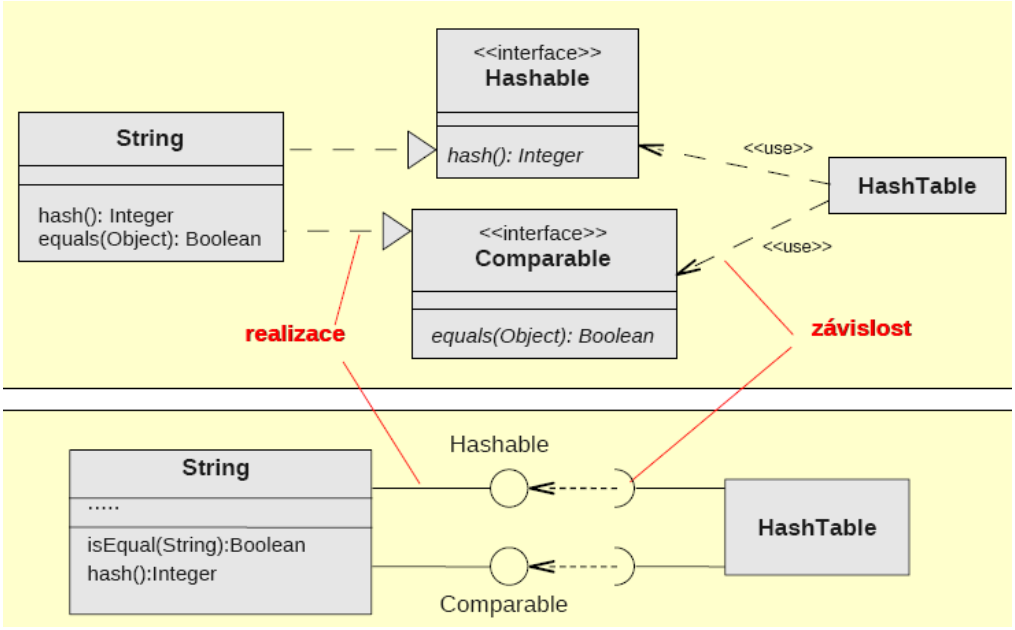
16. Co je rozhrani? Rozdil tridy a rozhrani. Uvest vlastni priklad.

Angl. *Interface*

- Speciální třídy, které definují *externě viditelné služby* (tzv. kontrakt) nějaké třídy nebo komponenty, bez *specifikace interní struktury* (atributy, stavy, implementace metod).
- Často popisují pouze část, nikoliv celé chování příslušné třídy.
- Používají stejné vztahy jako třídy, navíc ještě vztah *realizace* (angl. *realization*).
- Rozhraní = nabídka služeb realizovaných třídami
- 2 typy notácie – class style a lolipop

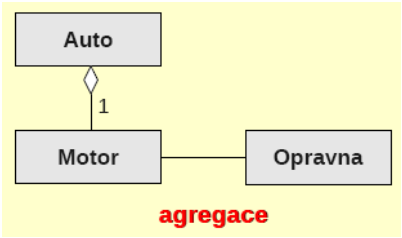


17. Rozhraní, vč. druhů používaných vazeb a nákres s použitím obou možných notací



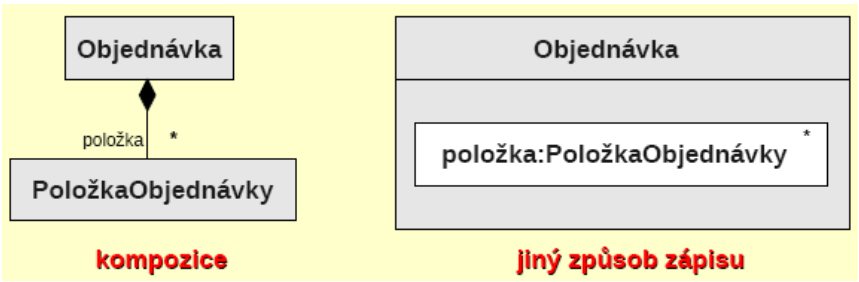
18. Agregace

- Speciální případ asociace pro **vztah celek-část**
- Symbol prázdného diamantu
- Tranzitivní - pokud je píst součást motoru a motor součástí auta, pak je i píst součástí auta
- Asymetrická - pokud je motor součástí auta, pak auto není součástí motoru, diagram objektů musí být acyklický
- Sémantika: totéž co asociace, pouze zdůrazňujeme vztah celek-část, součásti mohou existovat nezávisle na celku, součást může být sdílena více celky

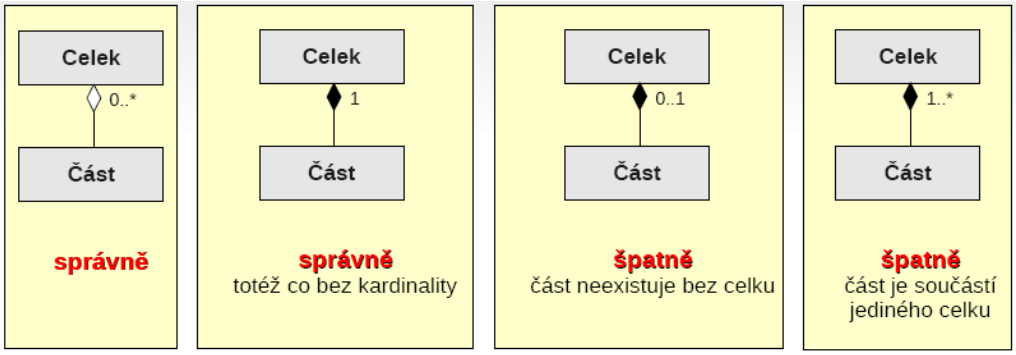


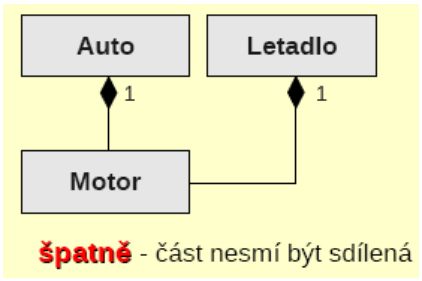
19. Kompozice

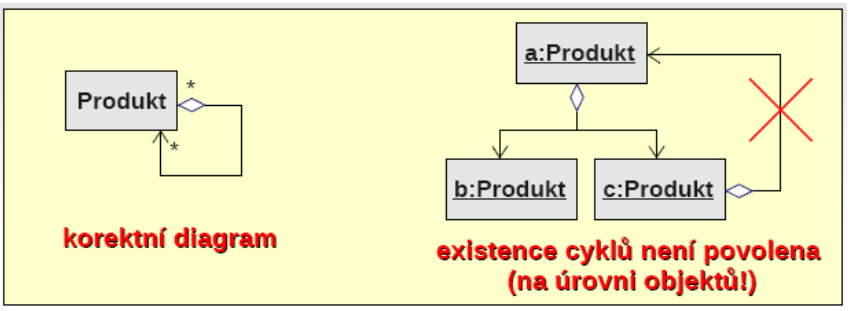
- Silnější forma agregace
- Symbol plného diamantu
- Tranzitivní a asymetrická
- Sémantika:
  - na úrovní instancí (objektů) části neexistují vně celku
  - každá část patří pouze do jediného celku (části nelze sdílet)
  - je-li celek zničen, musí zničit i svoje součásti, nebo převést zodpovědnost za ně na jiný objekt

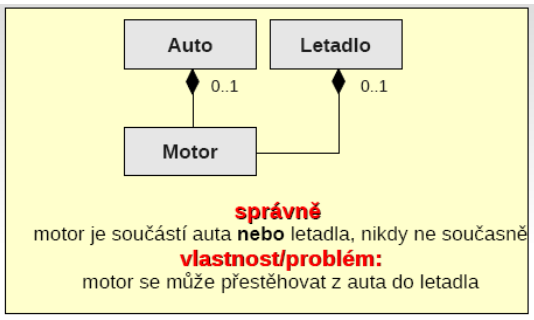


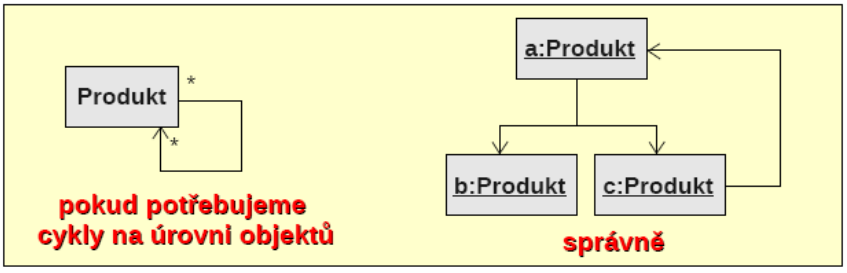
20. Byla dana kompozice mezi A a B. Byly dany 4 sekvencni diagramy a melo se urcit a zduvodnit, ktere diagramy jsou spravne a spatne. Byly to presne priklady ze slidu.

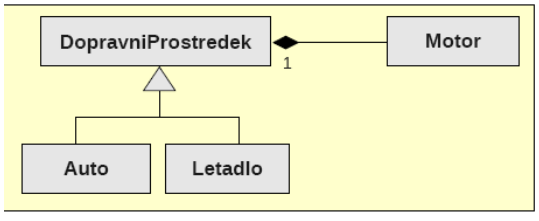






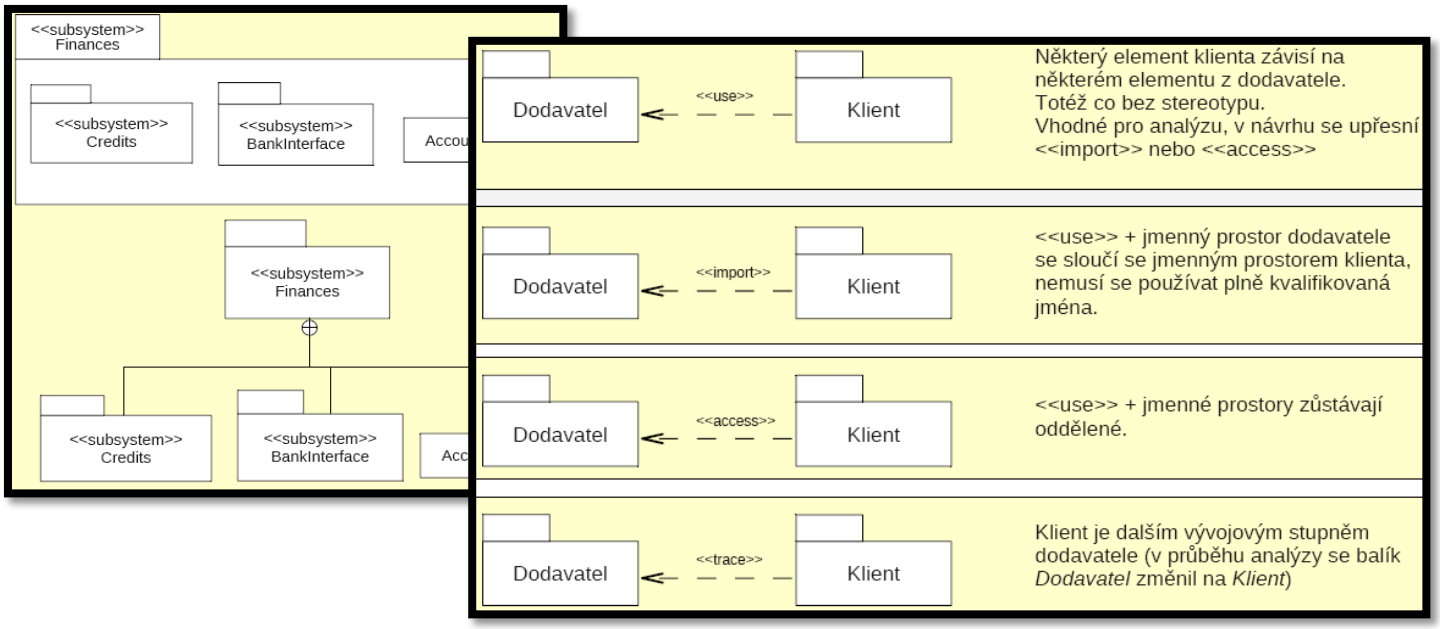






21. Co je to zavislost mezi baliky? Popiste jake stereotypy se uplatnuji.

- Balíky umožňují seskupit třídy a další modelové prvky (compile-time grouping)
  - do systémů (stereotyp <<system>>)
    - obsahuje všechny třídy (celý modelovaný systém)
  - do subsystémů (stereotyp <<subsystem>>)
  - do soustavy (stereotyp <<framework>>)
- Modelované samostatně v diagramu balíků nebo v rámci diagramu tříd
  - diagram balíků nemá třídy, soustředí se opravdu jen na balíky
- Základní vazby: generalizace/realizace, „containment“ a závislost
- Balík definuje jmenný prostor pro svoje elementy. Každý element lze jednoznačně identifikovat kvalifikovaným jménem (posloupnost jmen balíků a podbalíků od kořene až po daný element, např. ProdejceAut.Servis.Zakazka)
- Balík definuje viditelnost svých elementů (private nebo public), viz. balíky v Javě.



22. Aktivity XP, vyhody paroveho programovani, zhodnotit testovaci aktivity u XP

Extrémne programovanie - agilná metóda programovania.
Zrejmou výhodou tohto prístupu je to, že čím častejšie vývojový tím vydáva nové verzie, tým má lepšiu spätnú väzbu od zákazníka. Dokáže rýchlejšie reagovať na vznikajúce požiadavky a problémy. Čím skôr sa teda zavedie do systému nová funkcionalita, tým viac času má vývojový tím na jej prípadné opravy.

Akékoľvek komplikácie sú odstraňované hneď ako sa spozorujú a do systému sa zo zásady pridáva nová funkcionalita až vtedy, keď je naozaj potrebná. Tento princíp je známy pod skratkou YAGNI (*You Arent Gonna Need It*) alebo KISS (*Keep It Simple, Stupid*)

Rozdiel oproti klasickému modelu je v tom, že jednotlivé fázy sa nevykonávajú iba raz, ale mnohokrát – v iteráciách. XP predpokladá, že výsledkom veľkého počtu takýchto iterácií začne postupne vznikať (emergovať) aj celková architektúra systému.

Základom je to, že zdrojový kód tvoria dvaja ľudia naraz. Jeden monitor, jedna klávesnica, jeden počítač a dvaja ľudia. Človek, ktorý práve sedí pri klávesnici a píše kód, premýšľa o tom, ako najlepšie implementovať konkrétnu metódu. Ten druhý, čo mu hľadí cez rameno, rozmýšľa globálnejšie. Rozmýšľa o tom, či by sa systém nedal nejako zjednodušiť, či sa neporuší nejaká iná funkcionalita systému. Ak vidí, že partnerovi niečo nejde tak, ako by si predstavoval, bez váhania zoberie klávesnicu a roly sa vymenia

V XP sa kladie mimoriadny dôraz na testovanie, pričom existujú dva typy testov:

- Testy jednotiek, ktoré slúžia na testovanie jednotlivých častí kódu, väčšinou na úrovni tried.
- Akceptačné testy, ktoré vychádzajú z takzvaných užívateľských scenárov, pričom testujú očakávanú funkcionalitu produktu ako celku.

Oba typy testov sú však plne automatizované, úplne pokrývajú potrebnú funkcionalitu a je možné ich kedykoľvek spustiť. Ak sú k dispozícii takéto testy, tak otestovanie kompletnej funkčnosti systému trvá väčšinou len zopár minút, na otestovanie triedy stačí maximálne niekoľko sekúnd. Dôsledky tohto prístupu sú však obrovské. Nikto z programátorov sa teda nemusí báť zmeniť akýkoľvek kód

23. SCRUM

- metodika řízení projektů označovaná též jako agilní metodika
- Projekty řízené SCRUMem dokáží rychle reagovat na změny požadavků a okolí. Výsledky (inkrementy) přinášejí projekty řízené SCRUMem často a v pravidelných intervalech, *sprintech,* a tím se stávají pro management a zákazníka průhlednými. V rámci metodiky rozlišujeme role *SCRUM tým*, *SCRUM Master* a *Product Owner*.
- **Sprint -** fáze *SCRUMu*, časové období, jehož cílem je implementace vybraných položek *product backlogu*. Tak jako čas ubíhá v sekundách, *SCRUM* ubíhá ve sprintech. Obvyklá délka jednoho sprintu je týden až
- měsíc, méně ani více se nedoporučuje. Sprint je naplánován při *sprint planningu* dokumentovat ve *sprint*
- *backlogu*. Sprint je zakončen *sprint review* a *sprint retrospective.*
- **Sprint Goals -** cíle jednotlivého *sprintu. SCRUM tým* by měl po celou dobu běhu *sprintu* jasně vědět, k jakému úkolu směřuje. Vždy by se mělo jednat o určitý inkrement systému, tedy jasně doručené a prezentovatelné funkcionality viditelné i pro zákazníka (souvisí s agilností SCRUMu).
- **User Story -** způsob definice požadavků. Na základě dlouhodobých zkušeností bylo zjištěno, že nejlepší způsob, jak zaznamenat požadavek je použít vlastních slov zákazníka. Správně definivaná User story nám říká CO očekávám, Z JAKÉHO POHLEDU. Příkladem může být: „Jako operátor telefonické podpory a uživatel systémů A potřebuji mít možnost zobrazit historii všech kontaktů zákazníka s telefonickou podporou jedním kliknutím z karty zákazníka.“ Jiným příkladem potom může být: „Jako vývojář aplikace potřebuji mít možnost zapnout z příkazové řádky log všech přístupů do databáze.“

24. Syntaxe zpráv na komunikačním diagramu

Komunikační diagram ukazuje **interakci** organizovanou podle interagujících objektů, a jejich **propojení** mezi sebou.

- ukazuje **vztahy** mezi rolemi objektů
- **časová dimenze** není ukázána

Bez časové osy -> **sekvenční čísla** jsou použita pro uspořádání zpráv.

- **procedurální tok řízení**: vnořené číslování podle vnořeného volání
- **neprocedurální tok**: nevnořené číslování, které indikuje pořadí zpráv a synchronizaci mezi vlákny

2.1b: *[i:=1..akt_rok] vsechny_predmety := vyhledej(i) // poznamka

Číslo sekvence

- lexikografické číslování
- písmena označují paralelní zprávy
- * indikuje iteraci

Podmínka, cykly

- uvnitř []

Jména návratových hodnot

- jsou uvedena před :=

Jméno zprávy

- může být událost nebo operace

Argumenty

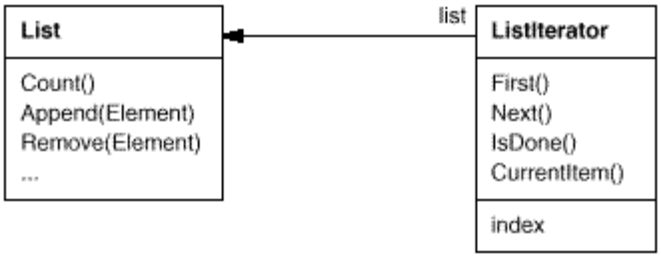
- v kulatých závorkách

Poznámky

- za dvěma lomítky

25. Návrhový vzor Iterator

sekvenční přístup k elementům agregovaného objektu bez odkrytí jejich reprezentace



Aplikovatelnost

- přístup k obsahu agregovaného objektu bez vystavení jeho vnitřní reprezentace
- podpora násobných průchodů agregovaným objektem
- poskytnutí jednotného rozhraní pro procházení odlišných agregovaných struktur (podpora polymorfní iterace)

Důsledky

- Podporuje variace průchodů agregátem.
- Zjednodušuje rozhraní agregátu.
- Může být spuštěno více souběžných průchodů agregátem (stav procházení je uložen v iterátoru).

26. Návrhový vzor Adapter

Aplikovatelnost

- chceme použít existující třídu a její rozhraní neodpovídá tomu, které potřebujeme, nebo
- chceme vytvořit znovupoužitelnou třídu, která spolupracuje s nepředvídanými nebo nevztaženými třídami, které nemusí mít kompatibilní rozhraní, nebo
- ***(pouze objektový adapter)*** potřebujeme použít několik existujících podtříd, ale není praktické je modifikovat mechanismem podtříd (adaptujeme rozhraní jejich rodičovské třídy)

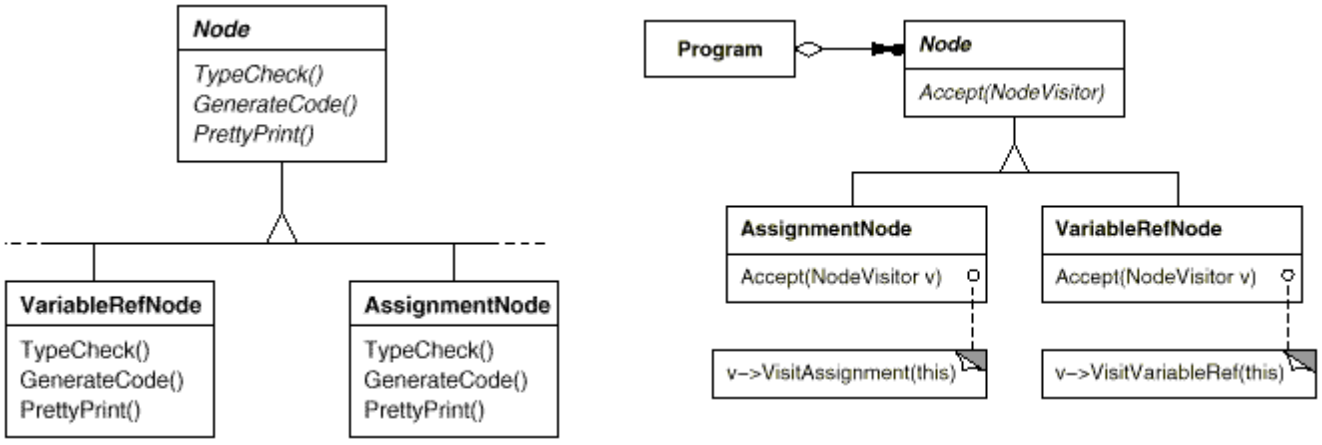
Důsledky (***adapter třídy***)

- Přizpůsobujeme Adaptovaného k Cíli připojením se ke konkrétní třídě Adapter. Adapter *třídy* nebude funkční, pokud chceme adaptovat třídu a všechny její podtřídy
- Umožňuje, aby Adapter změnil chování Adaptovaného, protože Adapter je podtřídou Adaptovaného
- Zavádí pouze jeden objekt a není tedy nutné použít nepřímé ukazatele k Adaptovanému

Důsledky (***adapter objektu***)

- Umožňuje jednomu Adapteru pracovat s mnoha Adaptovanými - tj. nejen s Adaptovaným, ale i s jeho podtřídami
- Chování Adaptovaného lze změnit obtížněji. Vyžaduje vytvoření podtřídy a odkaz Adapteru na podtřídy Adaptovaného

27. Návrhový vzor Visitor



Reprezentace operace, která má být provedena na elementech struktury. Definice nové operace bez změny tříd elementů, se kterými pracuje.

Aplikovatelnost

- strukturovaný objekt obsahuje mnoho tříd/objektů s různými rozhraními; chceme provádět operace, které závisejí na tom, ke kterým třídám objekty patří
- mnoho různých a nesouvisejících operací je třeba provést s objekty různých tříd ve struktuře a chceme se vyhnout „přeplnění“ těchto tříd novými operacemi
- třídy, které definují strukturu, se mění méně často; častěji chceme definovat nové operace nad strukturou

Důsledky

- Návštěvník usnadňuje přidávání nových operací.
- Návštěvník vidí související operace a separuje nesouvisející.
- Související chování je lokalizováno u návštěvníka.
- Přidávání nových tříd „KonkrétníElement“ je obtížné. Je třeba modifikovat všechny konkrétní návštěvníky.
- Návštěva v hierarchii tříd (na rozdíl od Iterátoru, kde nelze)
- Možnost akumulace stavů.
- Porušené zapouzdření. Často je potřeba více „otevřít“ rozhraní navštěvovaného elementu, aby mohl návštěvník s využitím vnitřního stavu elementu plnit požadovanou operaci.

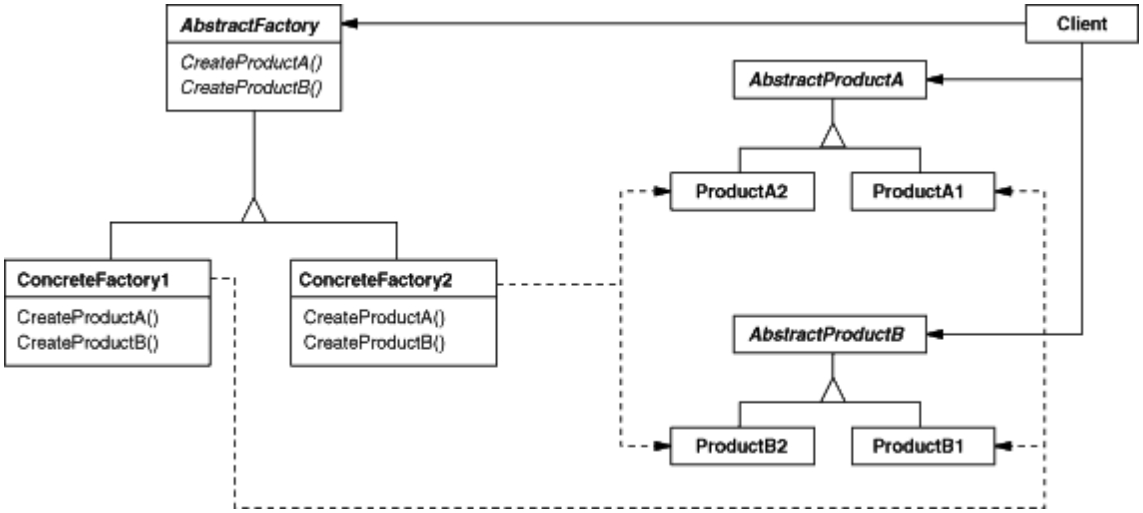
28. Návrhový vzor Abstraktní továrna

Aplikovatelnost

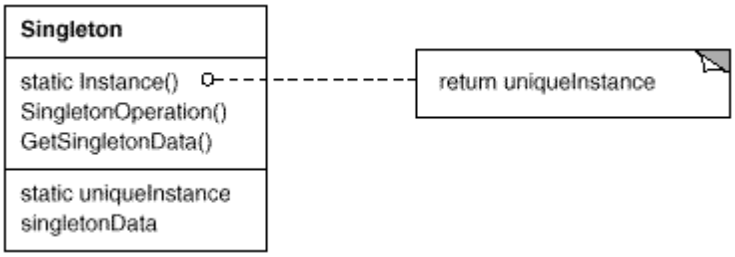
- Systém nemá být závislý na tom, jak jsou výrobky vytvářeny, sestavovány a reprezentovány
- Systém bude konfigurován pro jednu z mnoha rodin výrobků
- Rodina souvisejících výrobků bude používána společně a toto je potřeba zajistit (vynutit)
- Vytváříme objektovou knihovnu výrobků a chceme zveřejnit pouze jejich rozhraní, ne implementaci

Důsledky

- Izolace konkrétních tříd - produkty jsou implementovány vně klienta, ten používá jen obecné rozhraní.
- Snadná záměna rodin výrobků
- Podpora pro konzistenci výrobků
- Podpora nových druhů výrobků je obtížná.



29. Návrhový vzor Singleton



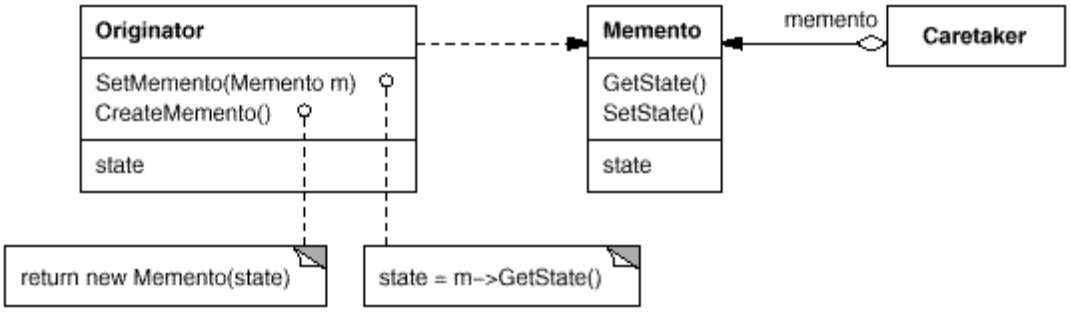
Aplikovatelnost

- Musí existovat právě jedna instance třídy a musí být přístupná klientům ve známém přístupovém bodě
- Instance má být rozšiřitelná mechanismem podtříd a klienti by měli mít možnost používat rozšířené instance bez modifikace vlastního kódu

Důsledky

- Řízený přístup k jediné instanci.
- Redukce jmenného prostoru
- Náhrada mnoha globálních proměnných jedním singletonem
- Povoluje úpravu operací
- Chování singletonu může být změněno jeho podtřídou. Konfigurace aplikace pro použití podtřídy může být dokonce provedena za běhu.
- Povoluje změnu počtu instancí (úprava vzoru)
- Pružnější, než pomocí operací třídy
- Téhož chování lze dosáhnout pomocí statických metod, u kterých ale obvykle chybí polymorfismus. Stejně tak je obtížné pracovat s více instancemi.

30. Návrhový vzor Memento



Slouží k zachycení určitého stavu objektu pro pozdější použití (např. obnovení)

Aplikovatelnost

- snímek (části) objektu musí být uchován pro pozdější obnovení stavu
- přímé rozhraní pro získání stavu by odhalilo implementační detaily a porušilo zapouzdření objektu

Důsledky

- Zachování hranic zapouzdření
- Zjednodušení „původce“ mementa.
- Použití mement může být nákladné.
- Definice úzkých a širokých rozhraní (jazykové problémy omezení přístupu k mementu výhradně pro původce).
- Skryté náklady pro zpracování mement

31. Byly dane navrhove tridy: - k prvým dvom kód, poslednú dekomponovať (aby nebolo M:N)

```
+-----+          +-------+
|House|1------>1|Address|
+-----+          +-------+

+-----+          +-------+
|House|1------>*|Address|
+-----+          +-------+

+-----+          +-------+
|House|*------>*|Address|
+-----+          +-------+
```

- Prva – kod kompozicie
- Druha – vedie k agregacii
- Posledna – treba vlozit nejaku pomocnu triedu, ktorou dekomponujeme na M:N

Co mi este ostalo a ani to nemienim vypracovat:

4. sequence diagram : do cestovky pride zakaznik, chce zajazd. zamestnanec
vyhlada zajazd, ak zakaznik este nieje v systeme tak ho tam prida a priradi mu vyhladany zajazd. Mal tam byt 1 aktor a minimalne 3 rozne objekty

4. V kartoteke su vedene zaznamy o zaverecnych pracach. Kazda zaverecna praca je bud bakalarska alebo diplomova, specializovana alebo nespecializovana. Kazda zaverecna praca ma svojho riesitela, veduceho, oponenta a ak je specializovana tak aj garanta specializacie. Zaverecna praca sa prihlasuje na obhajobu pred komisiou podla rozvrhu obhajob.

4. sekvenčný diagram, oblasť: kataster nehnuteľností - správa pozemkov

4) Systém katastrálního úřadu - Diagram tříd

4. Nakreslit sekvenční diagram k systému rezervace letenek podle uvedeného toku událostí a přiloženého diagramu tříd

3) stavovy diagram pro objekty tridy "byt". Zadane stavy byly "vyprojektovan", "postaven", "prodan", "v rekonstrukci", "neobyvatelny (po havarii)", "zbouran" + doplnit pripadne dalsi stavy.

3. Nakreslit stavovy diagram pro tridu dum. Dum ma nejake takove stavy: vyprojektovan, postaven, prodan, obyvan, neobyvatelny kvuli havarii, zbouran. Za ukol bylo jeste nejake rozumne stavy pridat.

2. Stavovy diagram - kde se pouziva, jakymi tremi zakladnimi prvky je vzdy tvoren, nakreslit stavovy digram z libovolne aplikacni oblasti

3. Sekvencni diagram pro nasledujici situaci. Pracovnik dopravniho odboru zadava do systemu informace o novych vozidlech a jejich majitelich. Pote je kazdemu zaregistrovanemu vozidlu prirazena jedna z volnych SPZ. (Sekvecni diagram ma obsahovat mimo aktera (= pracovnik DO) minimalne dalsi dva objekty).

4. Bol daný systém leteniek. Zákazník si povie požadavky a zamestnanec mu vybere podle zadaných požadavku volnu letenku. Pokud zákazník není v systému, zamestnanec zanese jeho údaje do systému. Potom k nemu pripojí rezervaci letenky. Nakreslete sekvenční diagram, musí obsahovat 1 aktéra a alespoň 3 další elementy.

4) Neco ve stylu:
Mame registr vozidel, ktery eviduje automobily a majitele. Ke kazdemu automobilu je vystaven technicky prukaz obsahujici informace o vozidle (typ, barva, datum vyroby) a SPZ. Kazdy automobil ma majitele, jeden majitel muze mit vic automobilu. Automobil si pamatuje i predchozi majitele. Automobily se deli na osobni a nakladni - osobni automobily jsou bud jednostope nebo dvoustope, nakladni jsou vzdycky dvoustope. Nakreslete diagram trid v notaci UML a uvedte nezbytne atributy.