Návrh architektury

Návrh architektury

- zaměřuje se na otázku jak má být systém organizován
- vytváří se na počátku vývoje; v iterativním vývoji většinou po první iteraci
- spojuje návrh se specifikací požadavků
- identifikuje komponenty, jejich vztahy a komunikaci

Vztah mezi specifikací a architekturou

- ideálně by se specifikace neměla architekturou zabývat
- nerealistický požadavek
- dekompozice je důležitá pro organizaci specifikace a rozdělení práce na specifikaci požadavků
- dekompozice do komponent či podsystémů je základem abstraktního návrhu architektury

Úvod do softwarového inženýrství

IUS 2019/2020

6. přednáška

Ing. Radek Kočí, Ph.D.
Ing. Bohuslav Křena, Ph.D.

4. listopadu a 8. listopadu 2019

Architektonické vzory

Architektonické vzory

- abstraktní popis dobrých vyzkoušených praktik
- ověřeno na různých systémech a v různých prostředích
- každý vzor by měl obsahovat informace o vhodnosti použití, slabé a silné stránky

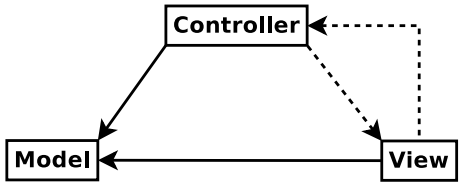
Přehled architektonických vzorů

- Model-View-Controller
- Vrstvená architektura
- Klient-Server
- ...

Téma přednášky

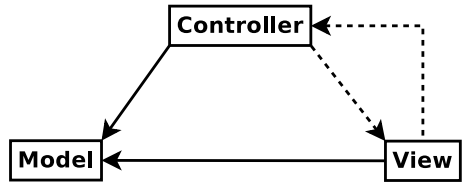
- **Architektonické vzory**
- **Komplexní modelování systému**
  - doménový model
  - model architektury
  - modely chování
  - modely interakce
  - modely struktury
  - datový model
- **Organizace 8. přednášky – Provoz a servis IT**
  - Ing. Martin Koloušek, IBM GSDC Brno
  - pondělí 18. 11. 2019
  - pátek 22. 11. 2019

Model-View-Controller



- MVC odděluje model a pohled na model.
- Jak ale zajistit změnu pohledu při změně modelu, pokud model nic neví o pohledu ani kontroleru?

Model-View-Controller



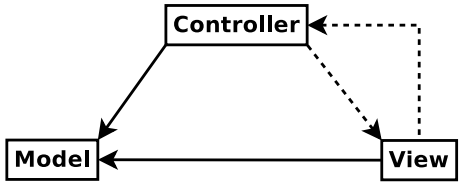
Konceptuální pohled

- *Model* – zapouzdřuje data a stav aplikace, informuje *View* o změnách stavu
- *View* – zobrazuje model, vyžaduje změny modelu, posílá uživatelské události *Controlleru*
- *Controller* – zajišťuje změny modelu na základě uživatelských akcí a změny *View* na základě změny modelu, vybírá *Views*

Konkrétní pohled – webové aplikace

- *Model* – databáze, business logika
- *View* – dynamické stránky, formuláře
- *Controller* – zpracování HTTP protokolu, validace dat

Model-View-Controller



- MVC odděluje model a pohled na model.
- Jak ale zajistit změnu pohledu při změně modelu, pokud model nic neví o pohledu ani kontroleru?

$\Rightarrow$ návrhový vzor *Observer*

Model-View-Controller

Popis

- odděluje prezentaci a interakci od systémových dat

Kdy použít

- různé způsoby zobrazení a interakce nad stejným modelem
- budoucí požadavky na zobrazení a interakce nejsou známé

Výhody

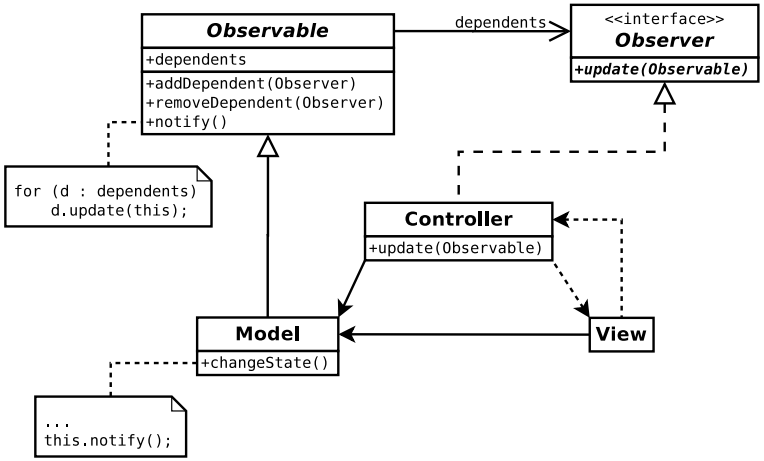
- data mohou být měněna nezávisle na jejich reprezentaci (pohledu) a naopak
- podpora prezentace dat různými způsoby

Nevýhody

- navýšení režie pro jednoduché modely a interakce

Model-View-Controller

- MVC s využitím vzoru *Observer*



Návrhový vzor Observer

Účel

- definuje závislost 1 ku N mezi objekty
- vzor chování

Motivace

- při změně stavu objektu jsou automaticky informovány všechny závislé objekty

Důsledky

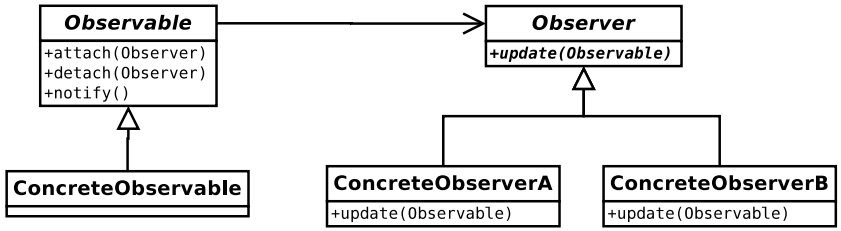
- konkrétní klient nemusí znát závislé objekty
- . . .

Vrstvená architektura

Koncept

- rozdělení systémů do vrstev
- každá vrstva odděluje elementy systému a lze je modifikovat nezávisle
- přidání či změna vrstvy je možná bez modifikace vrstev nižší úrovně
- lepší podpora inkrementálního vývoje – vrstvenou architekturu lze snadněji upravovat

Observer – Struktura



Vrstvená architektura

Výhody

- nahrazení celé vrstvy za novou
- redundantní služby (autentizace apod.) na každé vrstvě pro zvýšení spolehlivosti

Nevýhody

- čisté oddělení vrstev je v praxi náročné
- vrstva vyšší úrovně může potřebovat komunikovat s vrstvami nižší úrovně přímo, ne jen prostřednictvím bezprostředně navazující vrstvy
- výkon aplikace – potřeba vícenásobná interpretace požadavku na různých vrstvách může zpomalovat výkon aplikace

Vrstvená architektura

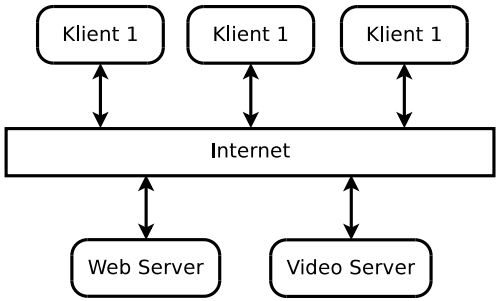
Příklad

- knihovní systém řídící přístup k chráněným elektronickým zdrojům
- pětivrstvá architektura, poslední vrstva představuje jednotlivé databáze

Architektura Klient-Server

Popis

- funkcionalita je rozdělena do služeb, každá služba (či množina služeb) je poskytována nezávislým serverem
- klient je uživatel služeb, přistupuje na servery



Vrstvená architektura

Popis

- organizace systému do vrstev, každá vrstva má přidělenu svou zodpovědnost (funkcionalitu)
- vrstva poskytuje služby nadřazené vrstvě, nejnižší vrstva reprezentuje jádro systému

Kdy použít

- přidání nových vlastností nad již existujícím systémem
- vývoj je rozdělen do několika týmů, každý se věnuje jedné vrstvě
- požadavek na víceúrovňovou bezpečnost (*security*)

Komplexní modelování v procesu vývoje

Konceptuální modely

- doménový model
  - zachycuje entity a pojmy problémové domény
  - diagram analytických tříd
- model architektury
  - zachycuje dekompozici systému a jeho budoucí architekturu
  - diagram tříd / balíčků
- modely chování
  - zachycují uživatelské a funkční požadavky
  - mohou modelovat i některé nefunkční požadavky (doba odezvy apod.)
  - diagramy případů užití, aktivit a stavový diagram

Architektura Klient-Server

Kdy použít

- data ve sdílené databázi musí být přístupná pro velký počet lokací (konkrétních míst)
- servery mohou být replikovány – lze využít, pokud je zatížení systému proměnlivé

Výhody

- servery mohou být distribuovány na síti
- služby jsou dostupné všem klientům a nemusí být implementovány všemi uzly

Nevýhody

- služba je jeden bod na síti, je náchylnější na útoky typu *denial of service*
- výkon aplikace je těžko predikovatelný, závisí na vytížení sítě
- problémy se správou, pokud jsou servery vlastněny jinou organizací

Komplexní modelování v procesu vývoje

Konceptuální modely

- modely interakce
  - zachycují interakci modelovaných elementů, např. objektů a aktérů participujících na případu užití
  - sekvenční diagram, diagram komunikace
- modely struktury
  - zachycují strukturální vazby mezi elementy systému
  - modely reflektují principy návrhu architektury
  - diagram návrhových tříd
- datový model
  - zachycuje perzistentní data systému
  - „odlehčený“ diagram tříd, ERD

Komplexní modelování v procesu vývoje

Pojmy

- *problémová doména*
  - reprezentuje reálný systém (*system-as-is*), jehož model máme vytvořit a následně implementovat
  - z problémové domény vycházejí obchodní požadavky, uživatelské požadavky, funkční a nefunkční požadavky
- *doména řešení*
  - reprezentuje vyvíjený systém (*system-to-be*), který odpovídá doménovému systému
  - modely systému, návrh, způsob řešení

CRC Cards

Class–Responsibilities–Collaborators

- představeny Kentem Beckem a Wardem Cunninghamem v roce 1989
- původně pro výuku objektově orientovaných paradigmat
- identifikace tříd, jejich zodpovědností a spolupracujících tříd
- bez počítačové podpory, flexibilní práce

Class name:	
Superclasses:	
Subclasses:	
Responsibilities:	Collaborators:

Komplexní modelování v procesu vývoje

Doménový model

- model konceptuálních (analytických) tříd
- nalezení abstrakcí v problémové doméně (např. *zákazník*)
  - abstrakce by měly odpovídat problémové doméně (*slovníček pojmů*)
  - model případů užití používá pojmy z doménového modelu
- zobrazovat jen věci podstatné z hlediska problémové domény
  - třídy *Zákazník*, *Košík*, ...
  - třída pro přístup k databázím patří do domény řešení

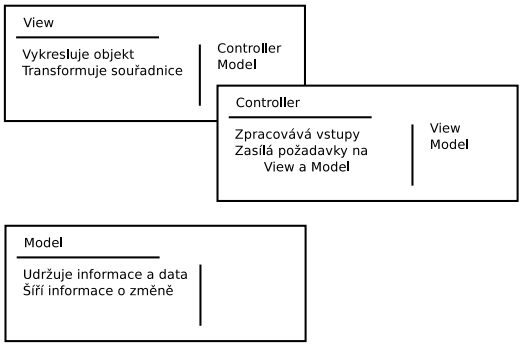
⇒ doménový model *není* datový model; konceptuální třída nemusí mít atributy ani nemusí reprezentovat perzistentní data

⇒ doménový model pojmenovává koncepty doménového systému a zmenšuje tak propast mezi softwarovou reprezentací a naším mentálním modelem systému

CRC Cards

Příklad – MVC

- *View* a *Controller* se překrývají, existuje úzká spolupráce
- *View* a *Controller* jsou umístěny nad *Model*, neboť *Model* neiniciuje žádnou spolupráci
- uspořádání karet často reflektuje princip probublávání abstraktnějších konceptů na vrchol



Konceptuální třídy

Konceptuální třída

- obsahuje jen nejpodstatnější atributy a operace
- obsahuje malou a správně definovanou množinu odpovědností
- obsahuje minimum vazeb na jiné analytické třídy

Hledání konceptuálních tříd

- využití *existujících modelů*
- využití *seznamu kategorií*
- analýza podstatných jmen ⇒ třídy, atributy
- analýza sloves ⇒ odpovědnosti tříd
- metoda *CRC štítků* (*Class*, *Responsibilities*, *Collaborators*)
  - štítek reprezentuje třídu
  - obsahuje seznam odpovědností
  - obsahuje seznam spolupracovníků (jiné třídy) – hledání vztahů

Komplexní modelování v procesu vývoje

Modely struktury

- modely návrhových tříd
- vychází z doménového modelu, modelů chování a interakce
- seskupení tříd reflektuje zvolenou architekturu

Konceptuální třídy

Co by měly konceptuální třídy splňovat

- třída má 3 až 5 odpovědností
- každá třída spolupracuje s jinými třídami (není osamocena)
- pozor na příliš mnoho malých tříd nebo malý počet obsáhlých tříd
- pozor na hlubokou hierarchii ve stromu dědičnosti (typicky 3 a více úrovní) – může signalizovat nevhodné použití dědičnosti
- název třídy by měl vymezovat její účel
  - *NakupniKosik*
  - *NavstevnikWeboveStranky* – spíše se jedná o roli, ve které může vystupovat *Zakaznik*

Ukázkový příklad

Postup

- vyjdeme ze specifikace požadavků
- navrhneme doménový model
- vytvoříme modely chování
- vytvoříme modely interakce
- zvolíme model architektury
- vytvoříme model struktury
- vytvoříme model dat

Poznámky

- příklad nebude úplný, pouze demonstrační
- nebudeme pracovat s úplnou specifikací, ale vyjdeme z tzv. *scénářů*
- *Scénář* = textová strukturovaná specifikace případu užití

Komplexní modelování v procesu vývoje

Modely interakce

- modelují interakce konceptuálních tříd
  - možnost nalezení nových konceptuálních tříd
- identifikují zasílané zprávy mezi objekty (instancemi tříd)
  - nalezení klíčových operací a atributů konceptuálních tříd a vztahů mezi konceptuálními třídami
- během procesu modelování se mohou aktualizovat stávající doménový model a modely chování
- obdobně je aplikováno i na návrhové diagramy (diagramy struktury, stavové diagramy, . . .)

Ukázkový příklad

Alternativní tok případu Zpracovat prodej

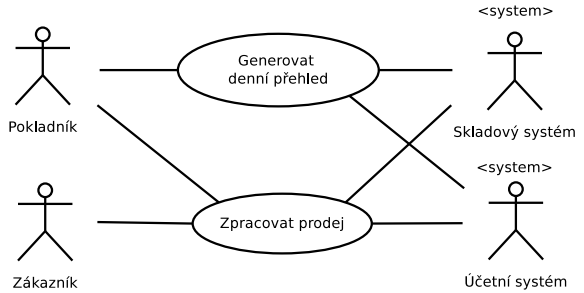
8a. Hotovostní platba

1. Pokladník zadá do systému přijatou částku.
2. Systém zobrazí rozdíl a uvolní pokladní zásuvku.
3. Pokladník uloží přijatou částku a vrátí rozdíl.
4. Systém zaznamená hotovostní platbu.

Ukázkový příklad

Základní specifikace

Vytvořte systém pro pokladny v supermarketu (*point-of-sale*, POS). POS je počítačová aplikace zaznamenávající prodej a spravující platby. Obsahuje hardwarová zařízení (čtečky kódu, displej apod.) a software. Komunikuje s dalšími systémy, jako např. řízení zásob. Systém musí být odolný vůči výpadkům systémů třetích stran; např. pokud není dočasně k dispozici systém pro řízení zásob, musí být systém schopen zaznamenat prodej a přijmout alespoň hotovostní platbu.



Inspirováno knihou C. Larman: Applying UML and Patterns.

Identifikace konceptuálních tříd

Seznam kategorií konceptuálních tříd

- seznam kandidátů konceptuálních tříd
- vychází z obecných kategorií stojících za zvážení při návrhu

Kategorie	Příklady
Obchodní transakce *Guideline:* kritické	Prodej (Sale) Platba (Payment)
Kde je transakce uložena *Guideline:* důležité	Pokladna (Register) Účetnictví (Ledger)
Role lidí nebo organizací *Guideline:* potřebujeme znát strany zainteresované na transakci	Pokladník (Cashier) Zákazník (Customer) Obchod (Store)
Reálné objekty *Guideline:* relevantní při návrhu řídicího softwaru nebo simulaci	Položka (Item) Pokladna (Register) Účtenka (Receipt)
...	...

Ukázkový příklad

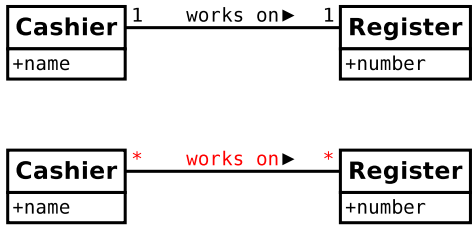
Scénář případu Zpracovat prodej

1. **Zákazník** přichází k POS zařízení se **zbožím**.
2. **Pokladník** začíná nový **prodej**.
3. **Pokladník** vloží identifikaci **položky**.
4. Systém zaznamená **položku** prodeje a zobrazí **popis položky**, její **cenu** a **aktuální součet**.
5. Kroky 3 a 4 se opakují, dokud je nějaké zboží na pásu.
6. Systém zobrazí součet včetně vypočtené **daně**.
7. Pokladník oznámí částku zákazníkovi a požádá o **platbu**.
8. Zákazník zaplatí platební kartou a systém zaznamená platbu.
9. Systém zaznamená kompletní **prodej** a zašle informace do externích systémů **Účetnictví** a **Řízení zásob**.
10. Systém tiskne **účtenku**.
11. Zákazník odchází s účtenkou a zbožím.

Konceptuální model není datový model

Konceptuální model nezachycuje *statická data*, ale objekty, které reprezentují *běh aplikace*.

Příklad: Koncept pokladníka (**Cashier**) a pokladny (**Register**).



- konceptuální model: aktuálně pracuje jeden pokladník na jedné konkrétní pokladně
- datový model: zachycuje, kdy a na jaké pokladně pokladník pracoval v průběhu času

Atribut nebo třída?

Jeden z největších problémů je správná identifikace konceptuálních tříd a zejména rozhodnutí, zda určitý element je třída nebo jen atribut třídy.

Příklad: Koncept prodeje (**Sale**) a obchodu (**Store**).

- **Store** je reálná entita, organizace mající svou adresu, je to *konceptuální reprezentace* prvku doménového systému
- pokud si nemůžeme představit konceptuální třídu jako číslo či řetězec v doménovém systému, jde skutečně o třídu, ne atribut

Description Classes

Description class obsahuje informace popisující jiné objekty. Typicky se jedná o otázku, zda určitou skupinu atributů vyjmout a modelovat jako samostatnou třídu.

Příklad: Koncept položky (**Item**).

- **Item** představuje jednu položku zboží, jeden skutečný kus; má svůj popis, cenu a sériové číslo
- totéž zboží (lednička XYZ) má více reálných kusů; informace se tedy duplikují a pokud neexistuje na skladě reálný kus, nejsou informace žádné
- společné informace modelujeme pomocí *Description class*

Spojení konceptuálních tříd asociací

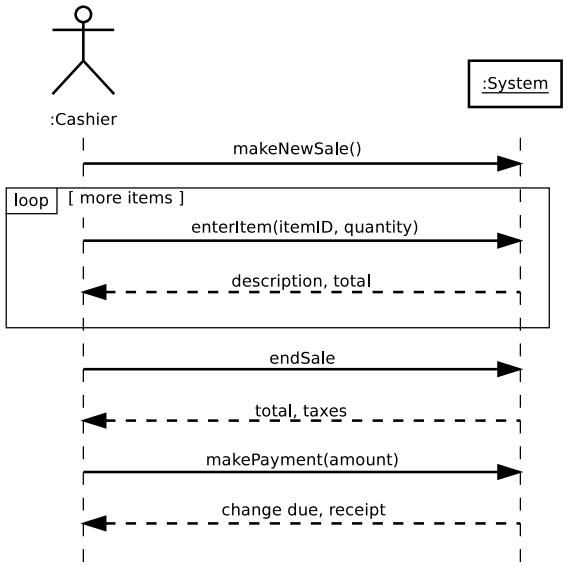
Dvě konceptuální třídy, které spolu souvisejí, spojujeme asociací, nikoliv atributy (tzv. cizími klíči).

Příklad: Koncept pokladníka (**Cashier**) a pokladny (**Register**).

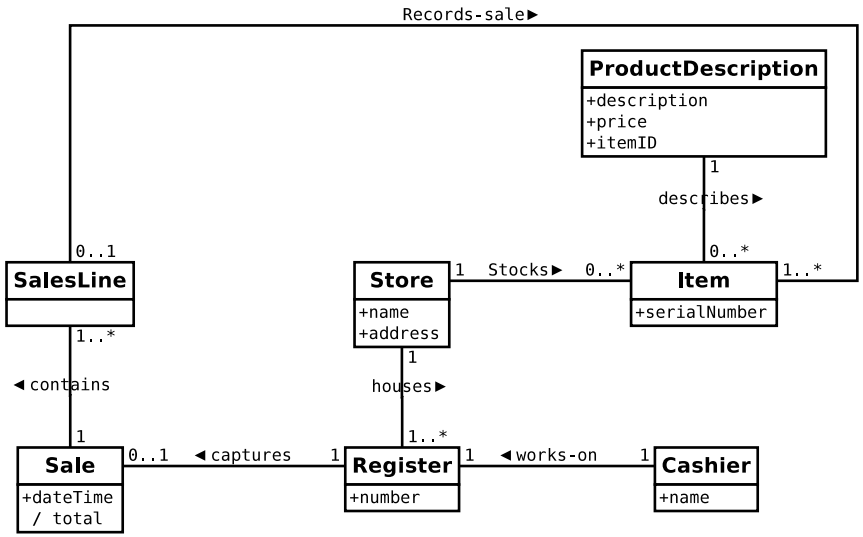
- pokladník pracuje na konkrétní pokladně
- pokladna má svou konceptuální třídu, existuje tedy *asociace* mezi třídami

Systémový sekvenční diagram

Scénář **Zpracovat prodej**



Příklad: Konceptuální model



Modely chování

Diagramy případů užití a scénáře jsou hlavním způsobem zachycení chování systému. V některých případech je vhodné použít podrobnější popis.

- diagram aktivit
  - popisuje scénář prostřednictvím toku událostí, lze zachytit i události
- stavový diagram
  - popisuje změny objektu doménového modelu v reakci na události
- **operační kontrakty**
  - popisuje změny objektu doménového modelu pomocí *pre-conditions* a *post-conditions*

Modely interakce

Sekvenční diagram

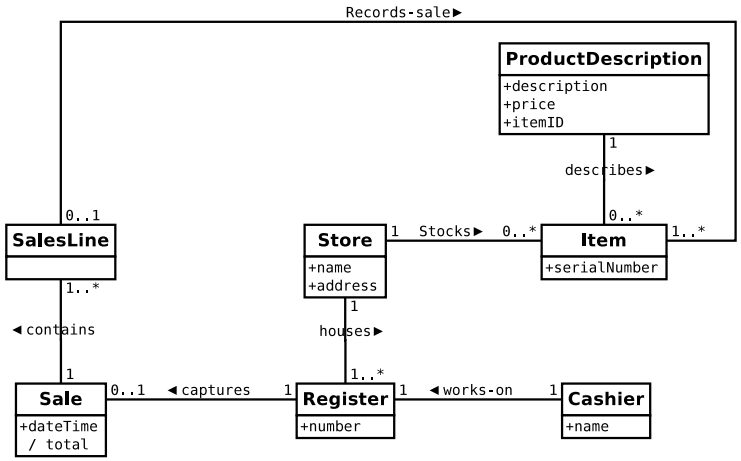
- zobrazuje objekty systému, externí aktéry a interakci mezi nimi
- zachycuje události pro jeden scénář případu užití, vychází se z jeho inspekce
- interakce jsou zachyceny pomocí zasílání zpráv

Systémový sekvenční diagram

- zobrazuje systém jako černou skříňku
- důležitá součást analýzy chování systému – identifikuje události přicházející do systému

Konceptuální model

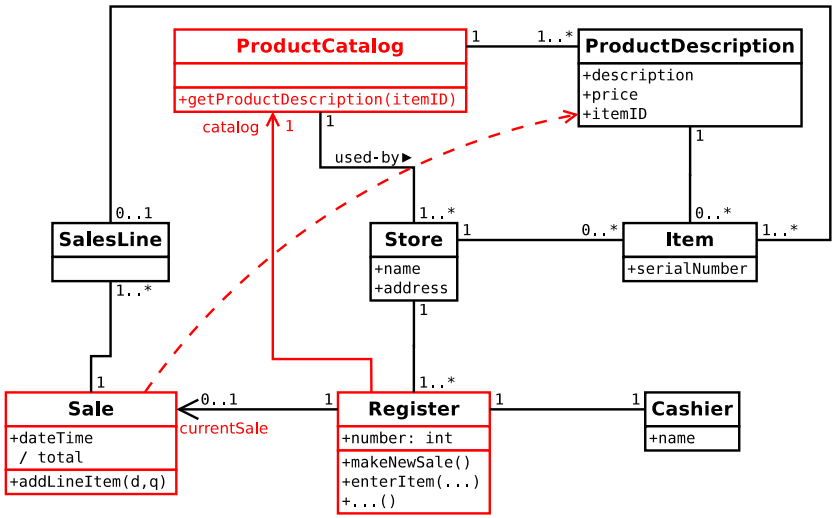
- který objekt zpracuje zprávu *getProductDescription*?



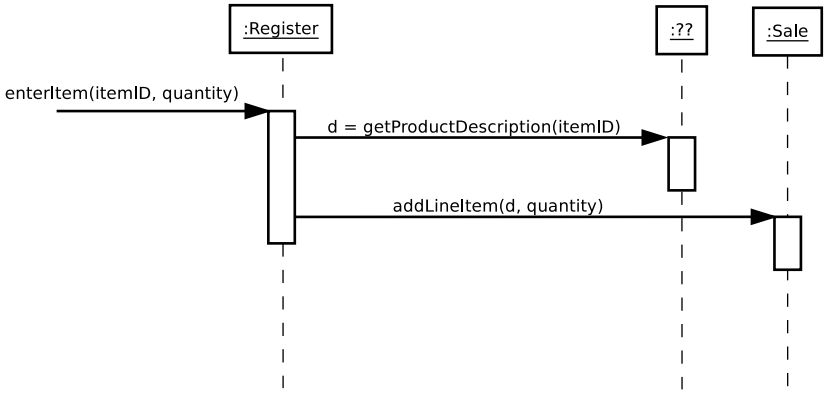
Popis kontraktu

- **Operace**
  - enterItem(itemID, quantity)
- **Reference**
  - případ užití *Zpracovat prodej*
- **Pre-conditions**
  - prodej byl zahájen
- **Post-conditions**
  - byla vytvořena instance sl třídy *SalesLine*
  - sl byla asociována s aktuální *Sale*
  - sl byla asociována s *ProductDescription* na základě itemID
- **Otázka**
  - který objekt definuje tuto operaci?
  - iniciátorem je pokladník, nabízí se tedy pokladna (*Register*)

Přechod k diagramu návrhových tříd



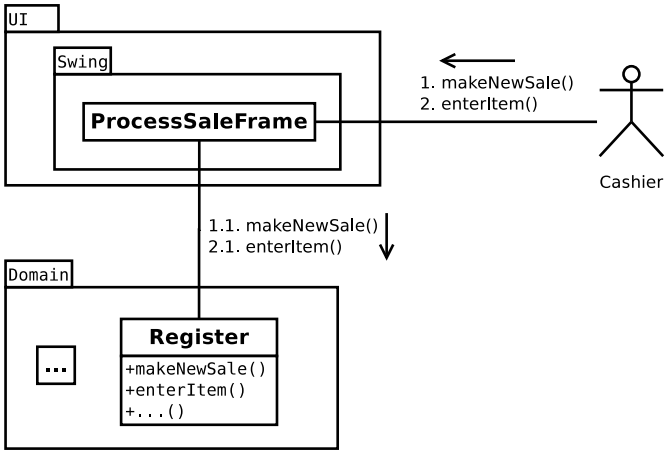
Sekvenční diagram pro kontrakt enterItem



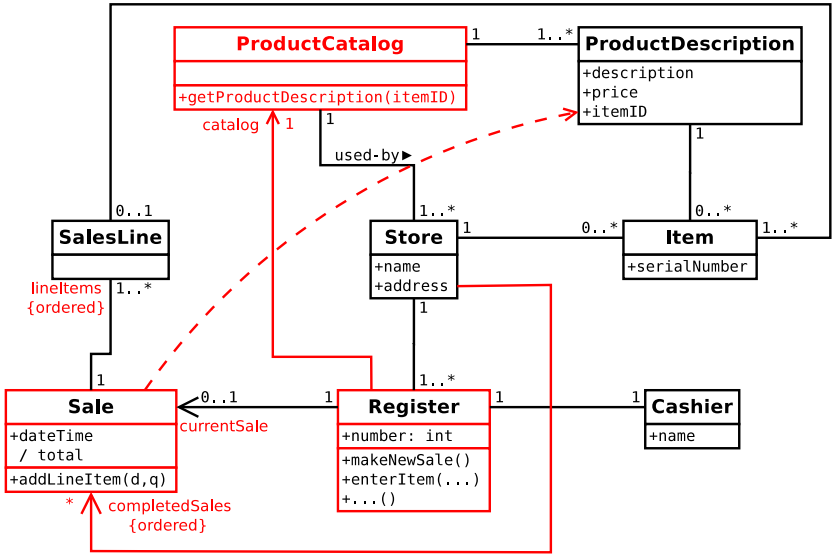
- je nutné vyhledat *ProductDescription* podle itemID
- který objekt zpracuje zprávu *getProductDescription*?

Model architektury

- model použití architektury pro námi analyzovaný scénář

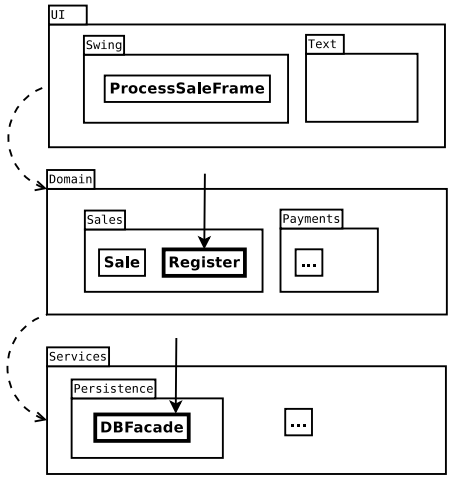


Přechod k diagramu návrhových tříd



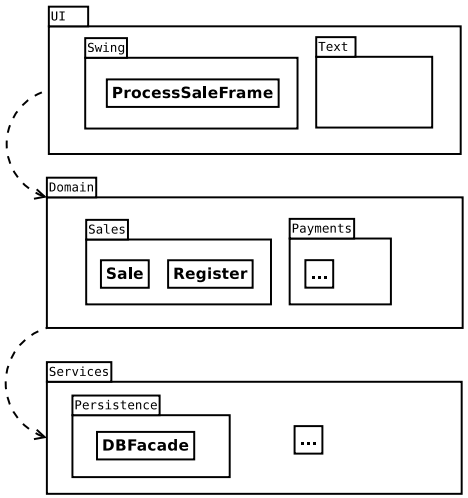
Model architektury

- rozhraní vrstev je definováno objekty (resp. třídami)
- lze aplikovat vzor *Fasáda* (*Facade*); fasáda je jeden objekt (třída), která je jediná vidět z venku a zajišťuje rozhraní k ostatním objektům



Model architektury

Zvolíme vícevrstvou architekturu, případně model MVC. Důležité je oddělení uživatelského rozhraní, aplikační logiky, databáze, ...



Návrhový vzor Fasáda (Facade)

- Ukázka vlivu na kód

```
// Bez aplikace vzoru
JFrame frame = new JFrame(...);
frame.setSize(...);
JLabel label = new JLabel(...);
frame.getContentPane().add(label, ...);
frame.setVisible(true);
```

```
// Aplikace vzoru
JOptionPane.showMessageDialog(...);
```


Návrhový vzor Fasáda (Facade)

Účel

- zjednodušení komunikace mezi prvky systému
- zjednodušení práce se složitějšími požadavky a systémy

Motivace

- jednoduché rozhraní ke komplexnímu systému
- prostředník oddělující implementační třídy a jejich použití (= vrstvy)

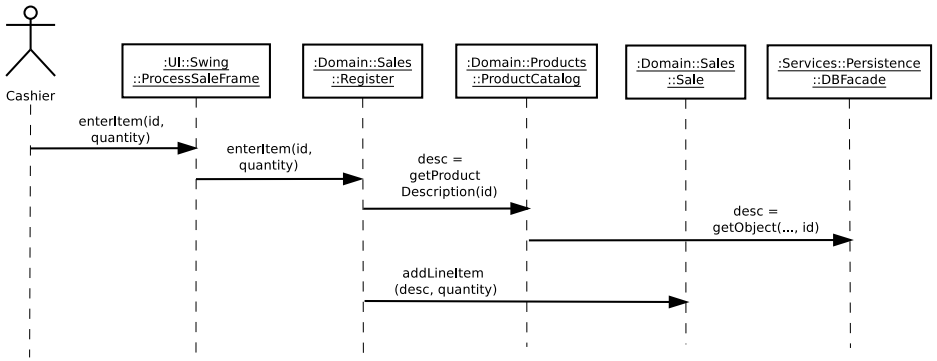
Důsledky

- zjednodušené rozhraní
- možnost výměny vrstvy za fasádou beze změny uživatelských tříd
- ...

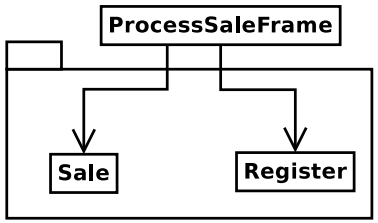
Příbuzné vzory: Abstraktní továrna (Abstract Factory), Prostředník (Mediator)

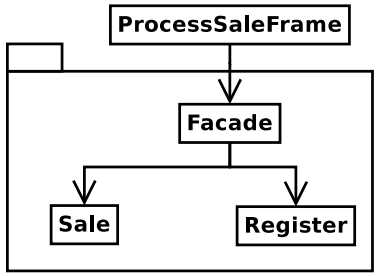
Architektura: Sekvenční diagram

- zachycení komunikace mezi vrstvami



Návrhový vzor Fasáda (Facade)





Studijní koutek – Projektová praxe

- Možnost zapojit se do řešení výzkumných projektů fakulty
- Individuální práce pod vedením zkušeného školitele
- 5 kreditů za semestr
  - IP1 až IP3 – 3. až 5. semestr Bc.
  - PP1 a PP2 – 2. a 3. semestr Ing.
- Očekávané pokračování v bakalářské/diplomové práci
- Určeno výborným studentům
  - Absolvování všech povinných předmětů 1. ročníku Bc.
  - Přihlašování do výběrového řízení v IS FIT
  - Postup do dalšího předmětu podmíněn úspěšnou prací
- Další výhody
  - Možnost mimořádného stipendia
  - Možnost cestovat