Úvod do softwarového inženýrství

IUS 2024/2025

6. přednáška

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

21. a 25. října 2024

Téma přednášky

- **Architektonické vzory**
  - Model-View-Controller
  - vrstvená architektura
  - klient-server
- **Komplexní modelování systému**
  - doménový model
  - model architektury
  - modely chování
  - modely interakce
  - modely struktury
  - datový model

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

- dekompozice – jedna z důležitých aktivit při analýze a specifikaci požadavků
- dekompozice je důležitá pro organizaci specifikace a rozdělení práce na specifikaci požadavků
- dekompozice do komponent či podsystémů je základem abstraktního návrhu architektury

Architektonické vzory

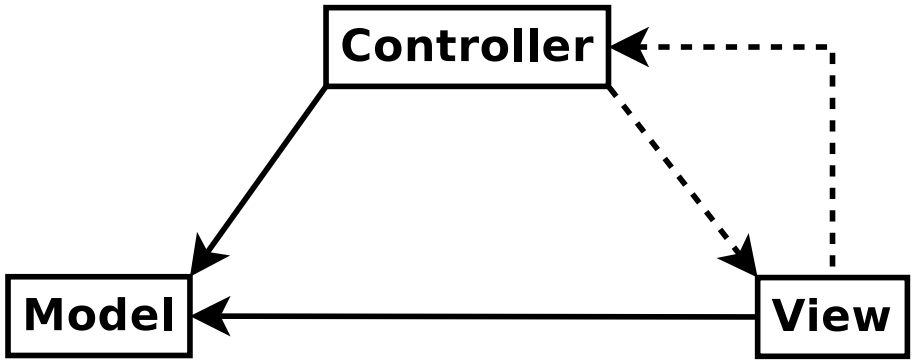
Architektonické vzory

- 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
- ...

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

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

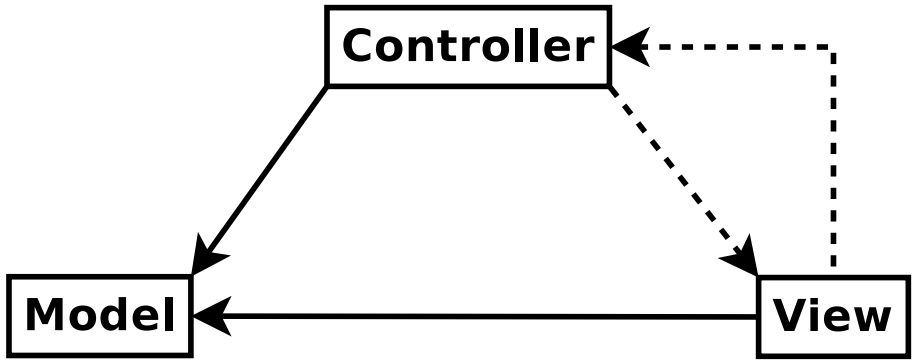
Výhody

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

Nevýhody

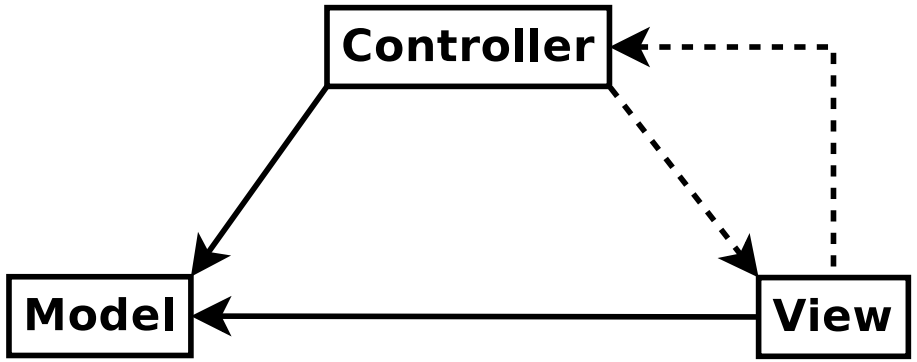
- navýšení režie pro jednoduché modely a interakce

Model-View-Controller



- MVC 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



- 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*

Návrhový vzor Observer

Účel

- definuje závislost 1 ku N mezi objekty
- vzor chování

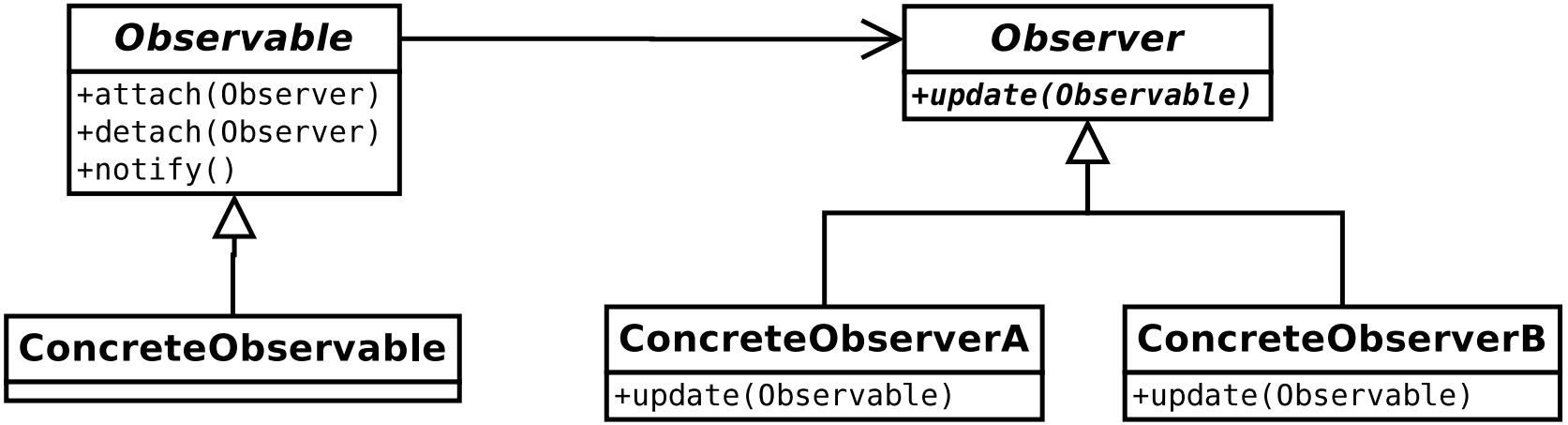
Motivace

- při změně stavu objektu jsou automaticky informovány všechny závislé objekty

Důsledky

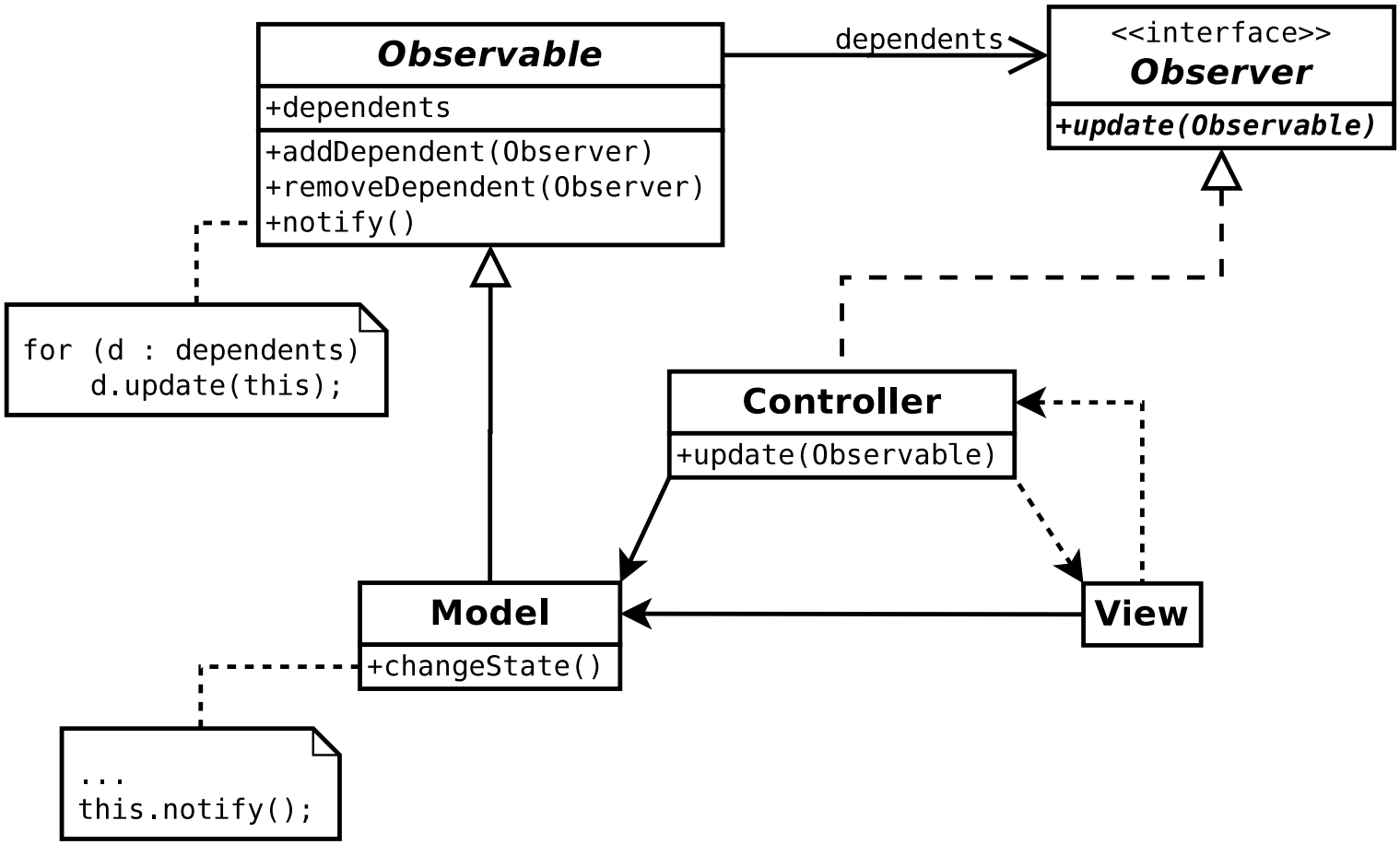
- konkrétní klient nemusí znát závislé objekty
- ...

Observer – Struktura



Model-View-Controller

- MVC s využitím vzoru *Observer*



Vrstvená architektura

Případně *vícevrstvá architektura* podle anglického *multi-tier architecture*

Koncept

- rozdělení systémů do vrstev – rozdělení zodpovědností
- vrstva poskytuje služby nadřazené vrstvě, nejnižší vrstva reprezentuje jádro systému
- každá vrstva odděluje elementy systému a lze je modifikovat nezávisle
- přidání či změna vrstvy je možná bez modifikace vrstev nižší úrovně
- inkrementální vývoj – vrstvenou architekturu lze snadněji upravovat

Vrstvená architektura

Příklad

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

Vrstvená architektura

Výhody

- snadnější údržba díky nízké závislosti na ostatních vrstvách
- vyšší znovupožitelnost
  lze znovupoužít celé vrstvy
  lze nahradit celou vrstvou jinou implementací
- vývoj lze jednoduše rozdělit do několika týmů
  každý tým se věnuje jedné vrstvě

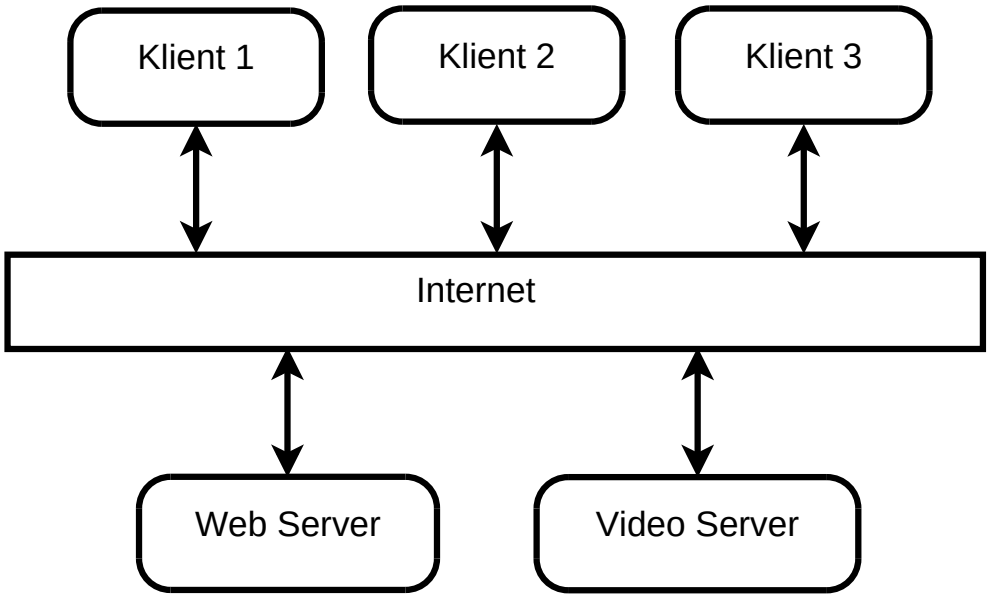
Nevýhody

- čisté oddělení vrstev je v praxi náročné
  vrstva vyšší úrovně může potřebovat komunikovat s vrstvami nižší úrovně přímo, ne jen prostřednictvím bezprostředně navazující vrstvy
- (opakované) zpracování požadavku na různých vrstvách může zpomalovat aplikaci

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



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

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

Komplexní modelování v procesu vývoje

Konceptuální modely

- doménový model
  - zachycuje koncepty (prvky/pojmy/objekty) problémové domény (nalezení abstrakcí, *slovníček pojmů*)
  - diagram analytických (konceptuálních) tříd
  - další modely používají pojmy doménového modelu

- 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

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

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 $\Rightarrow$ třídy, atributy
- analýza sloves $\Rightarrow$ odpovědnosti tříd
- metoda *CRC štítků* (*Class*, *Responsibilities*, *Collaborators*)
  - štítek reprezentuje třídu
  - obsahuje seznam odpovědností
  - obsahuje seznam spolupracovníků (jiné třídy) – hledání vztahů

CRC Cards

Class–Responsibilities–Collaborators

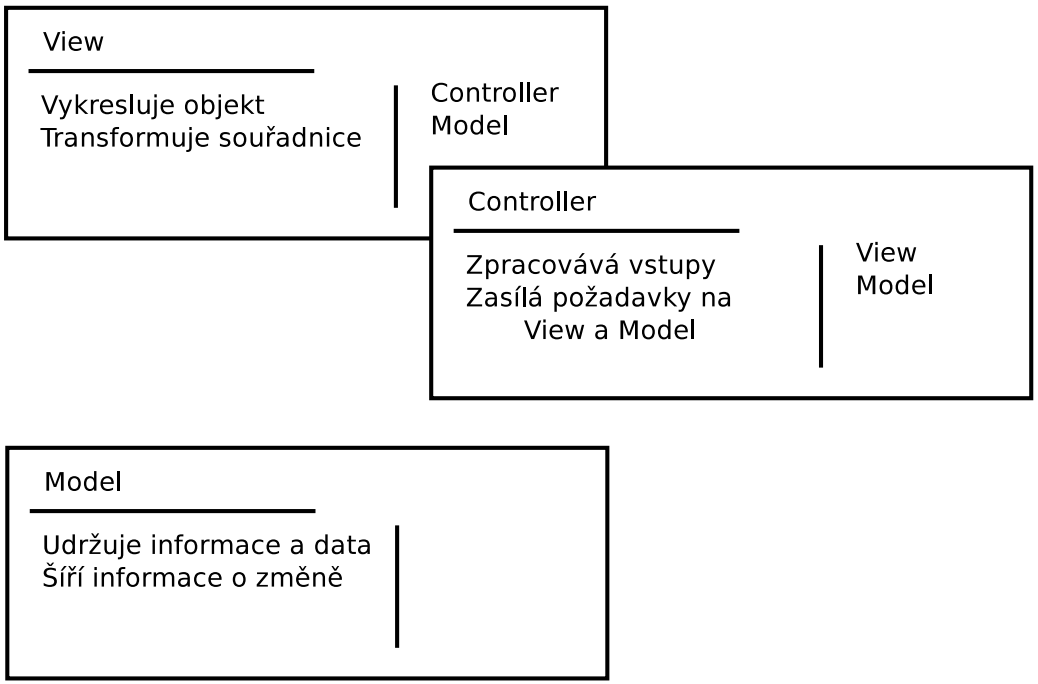
- představeny Kentem Beckem a Wardem Cunninghamem v roce 1989
- původně pro výuku objektově orientovaných paradigmat
- identifikace tříd, jejich zodpovědností a spolupracujících tříd
- bez počítačové podpory, flexibilní práce

Class name:	
Superclasses:	
Subclasses:	
Responsibilities:	Collaborators:

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

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*

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, . . .)

Komplexní modelování v procesu vývoje

Modely struktury

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

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

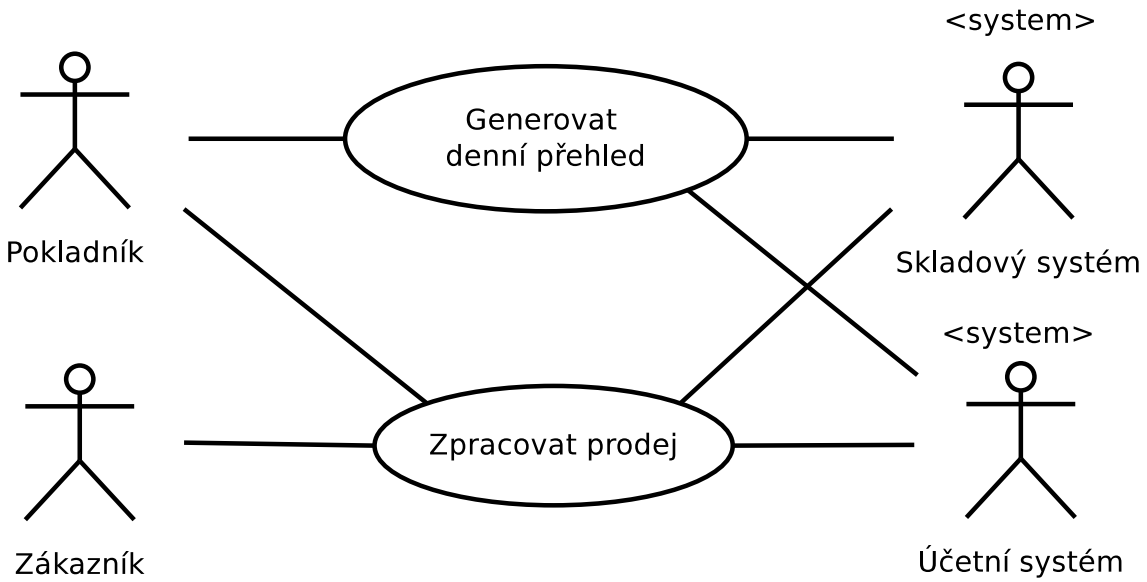
Poznámky

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

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.

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.

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.

Identifikace konceptuálních tříd

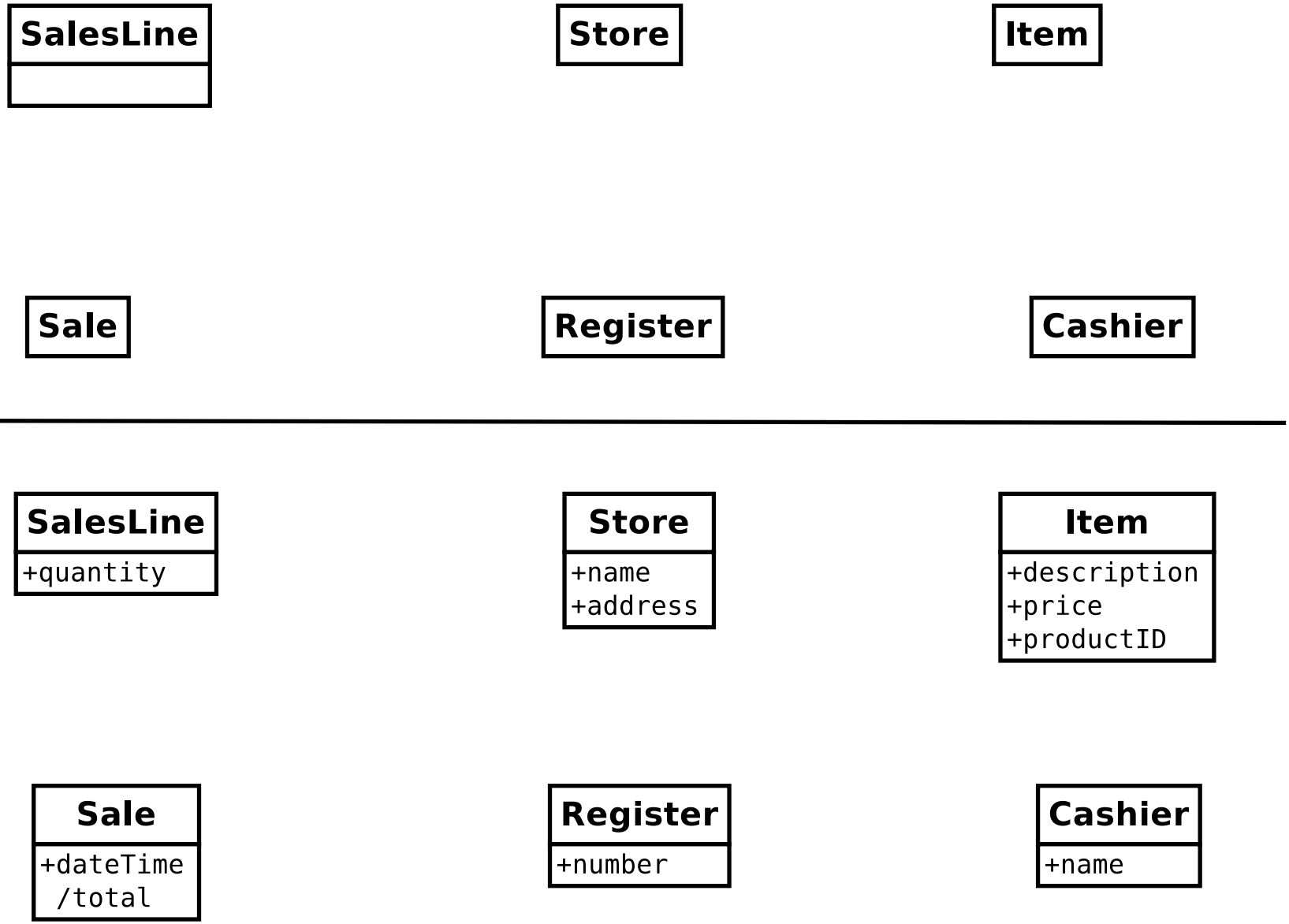
Seznam kategorií konceptuálních tříd

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

Kategorie	Příklady
Obchodní transakce *Guideline:* kritické	Prodej (Sale) Položka prodeje (SalesLine) 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)
...	...

Příklad: Konceptuální model

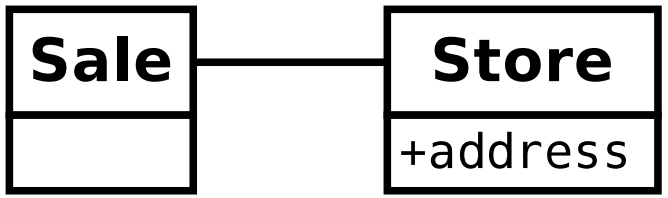
První verze + atributy



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**).

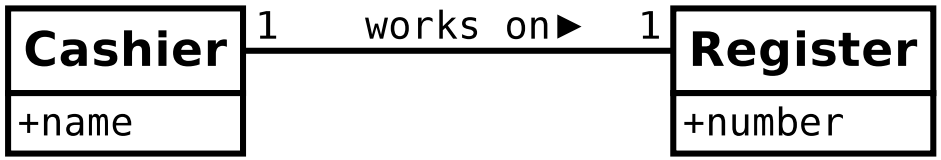


- **Store** je reálná entita, organizace mající svou adresu, je to *konceptuální reprezentace* prvku doménového systému
- pokud si nemůžeme představit konceptuální třídu jako číslo či řetězec v doménovém systému, jde skutečně o třídu, ne atribut

Spojení konceptuálních tříd asociací

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

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

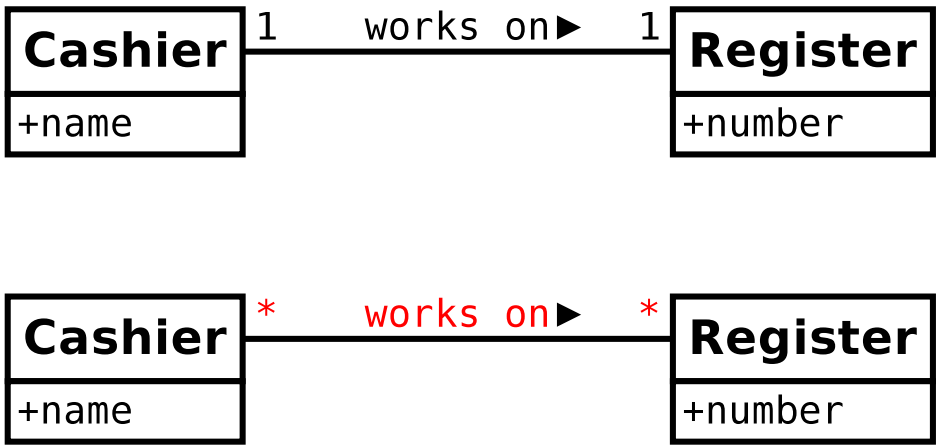


- pokladník pracuje na konkrétní pokladně
- pokladna má svou konceptuální třídu, existuje tedy *asociace* mezi třídami

Konceptuální model není datový model

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

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

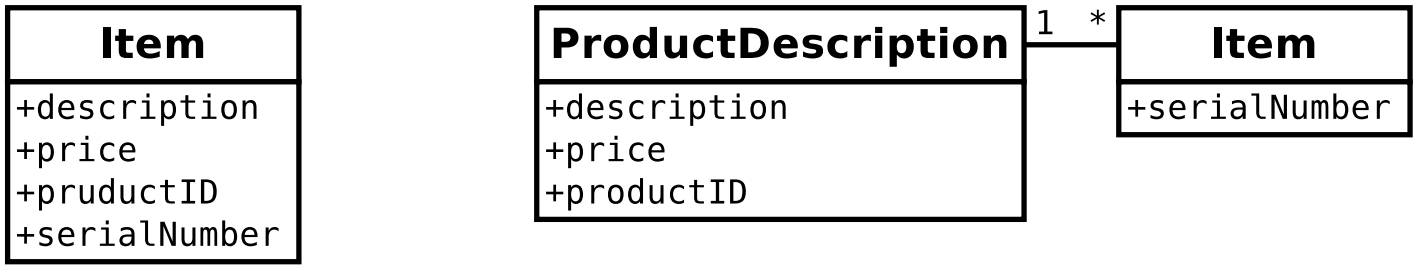


- konceptuální model: aktuálně pracuje jeden pokladník na jedné konkrétní pokladně
- datový model: zachycuje, kdy a na jaké pokladně pokladník pracoval v průběhu času

Description Classes

Description class obsahuje informace popisující skupinu jiných objektů.
Otázka, zda skupinu atributů vyjmout a modelovat jako samostatnou třídu.

Příklad: Koncept položky (**Item**)



- **Item** zachycuje jednu položku zboží (skutečný kus)
  - má svůj popis, cenu, produktový kód a může mít např. sériové číslo
  - totéž zboží (lednička XYZ) má více reálných kusů
  - informace o zboží se duplikují a, pokud neexistuje na skladě žádný kus, nejsou informace o zboží žádné
- **ProductDescription** zachycuje společné informace
  - $\Rightarrow$ *Description class*
  - **Item** pak zachycuje pouze informace jedinečné pro daný kus

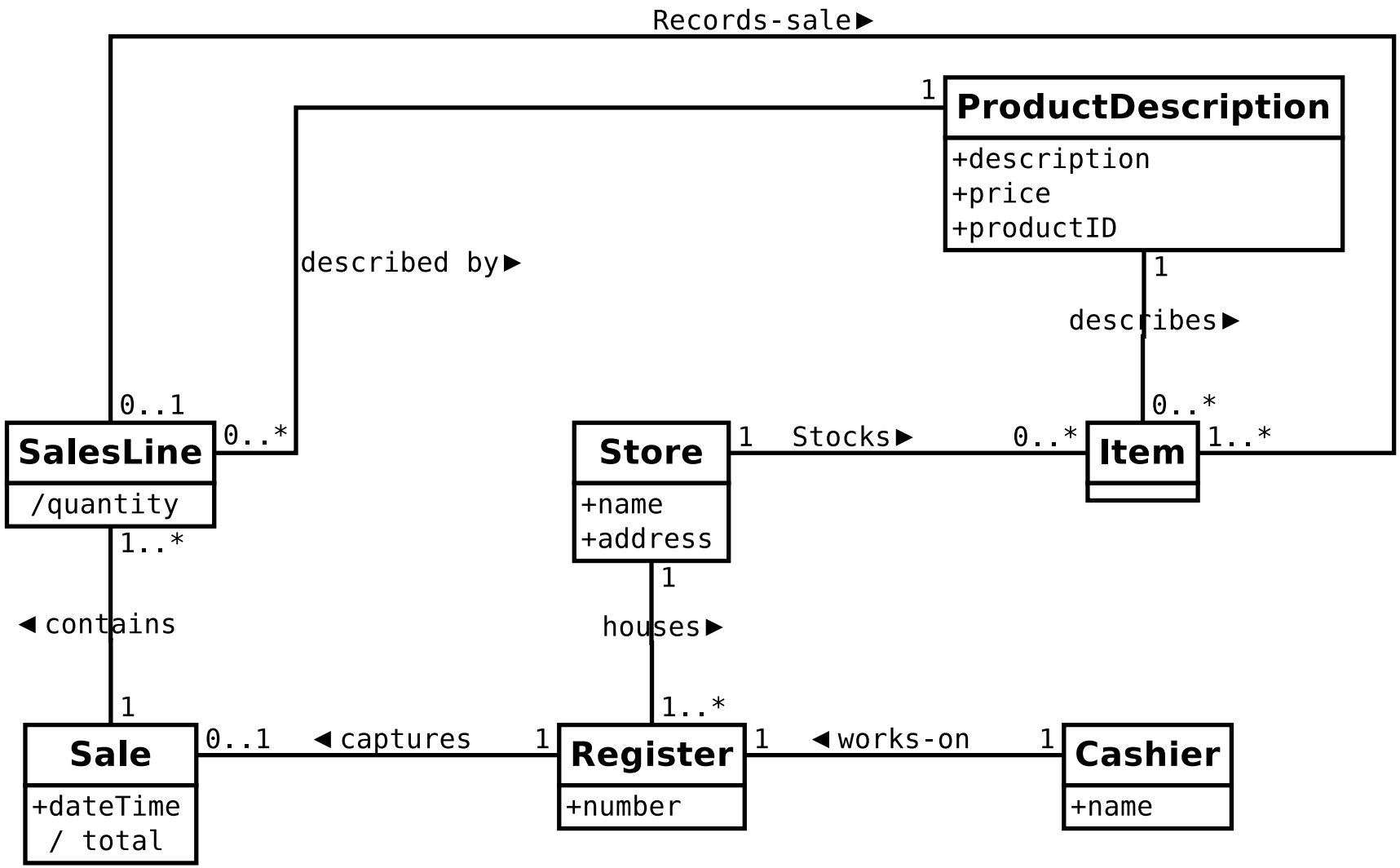
Identifikace asociací

Seznam kategorií asociací

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

Kategorie	Příklady
A je logickou součástí B	SalesLine – Sale
A je fyzicky umístěna v B	Register – Store
A je obsažena v B	ProductDescription – Catalog
A je popisem B	ProductDescription – Item
A používá/spravuje B	Cashier – Register
...	...

Příklad: Konceptuální model



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

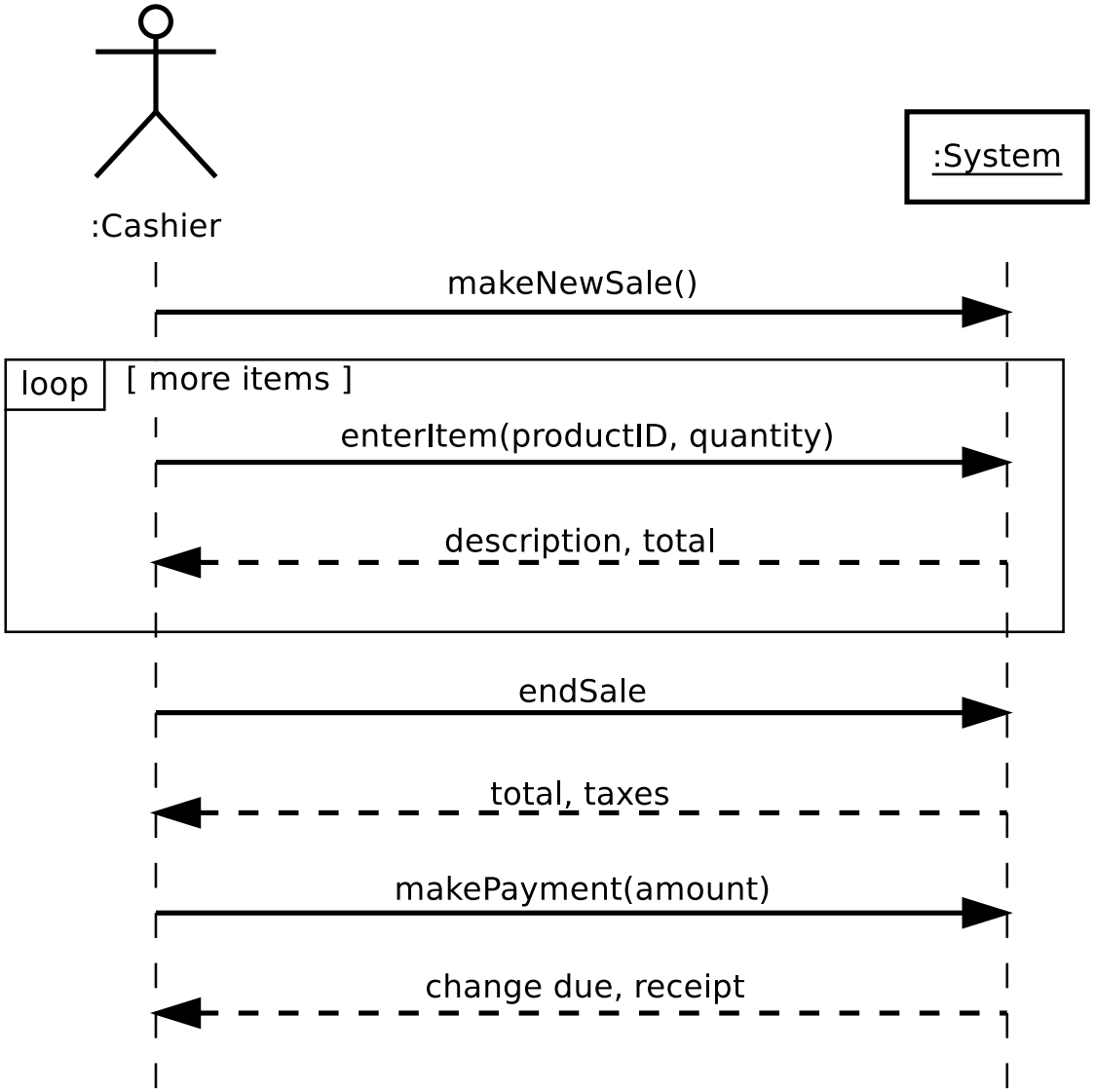
- zobrazuje systém jako černou skříňku
- důležitá součást analýzy chování systému – identifikuje události přicházející do systému

Účel

- validace požadavků / verifikace návrhu
- nalezení klíčových operací a atributů tříd a jejich vztahů
- možnost nalezení nových tříd

Systémový sekvenční diagram

Scénář **Zpracovat prodej**



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 kontrakt**
  - definuje chování pro operaci vázanou na případ užití;
    operace je součástí objektu (třídy) doménového modelu
  - popisuje změny pomocí *pre-conditions* a *post-conditions*

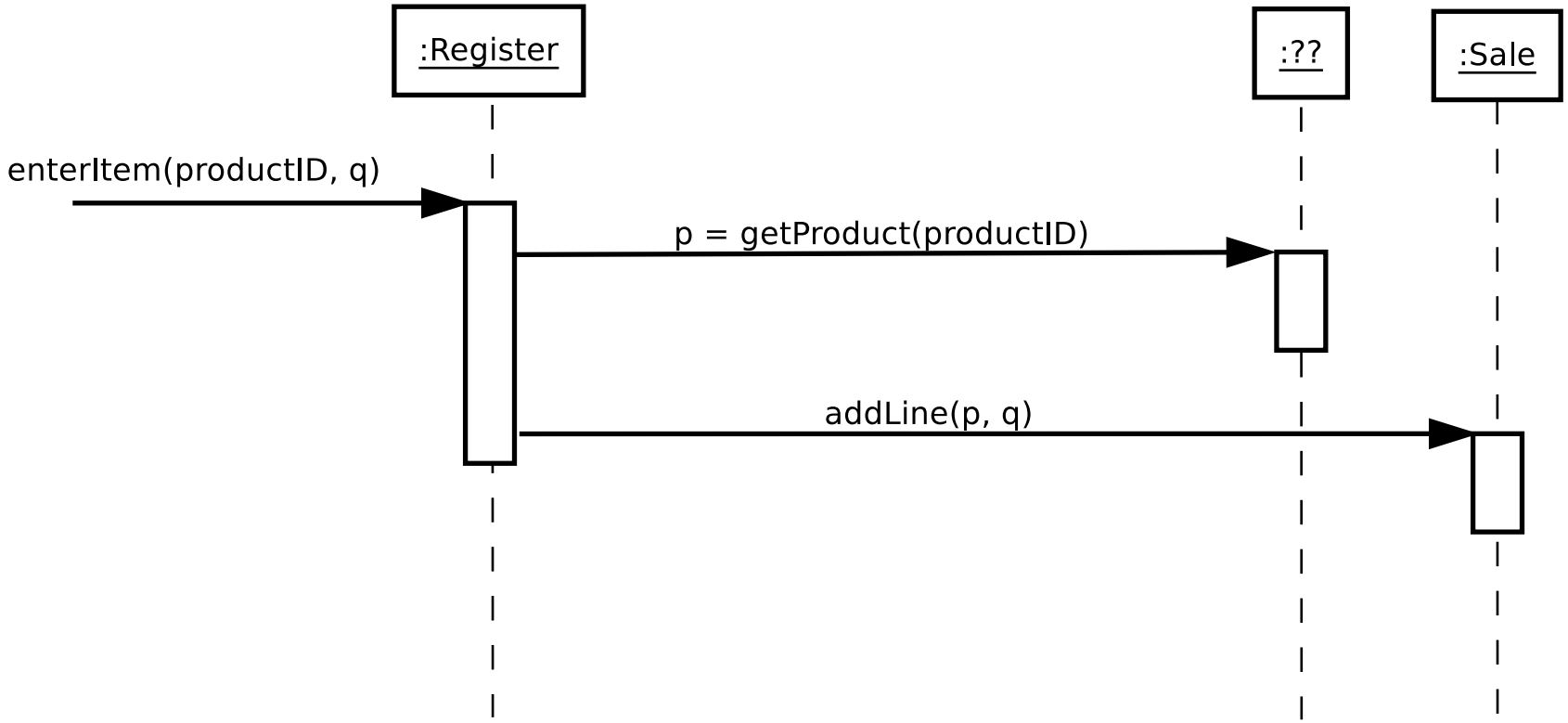
Popis kontraktu

- **Operace**
  - enterItem(productID, 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 productID*
  - sl byla asociována s příslušným počtem *Item* na základě quantity*

  * *může být součástí jiných kontraktů*

- **Otázka:** který objekt definuje tuto operaci?
  - iniciátorem je pokladník, nabízí se tedy pokladna (*Register*)

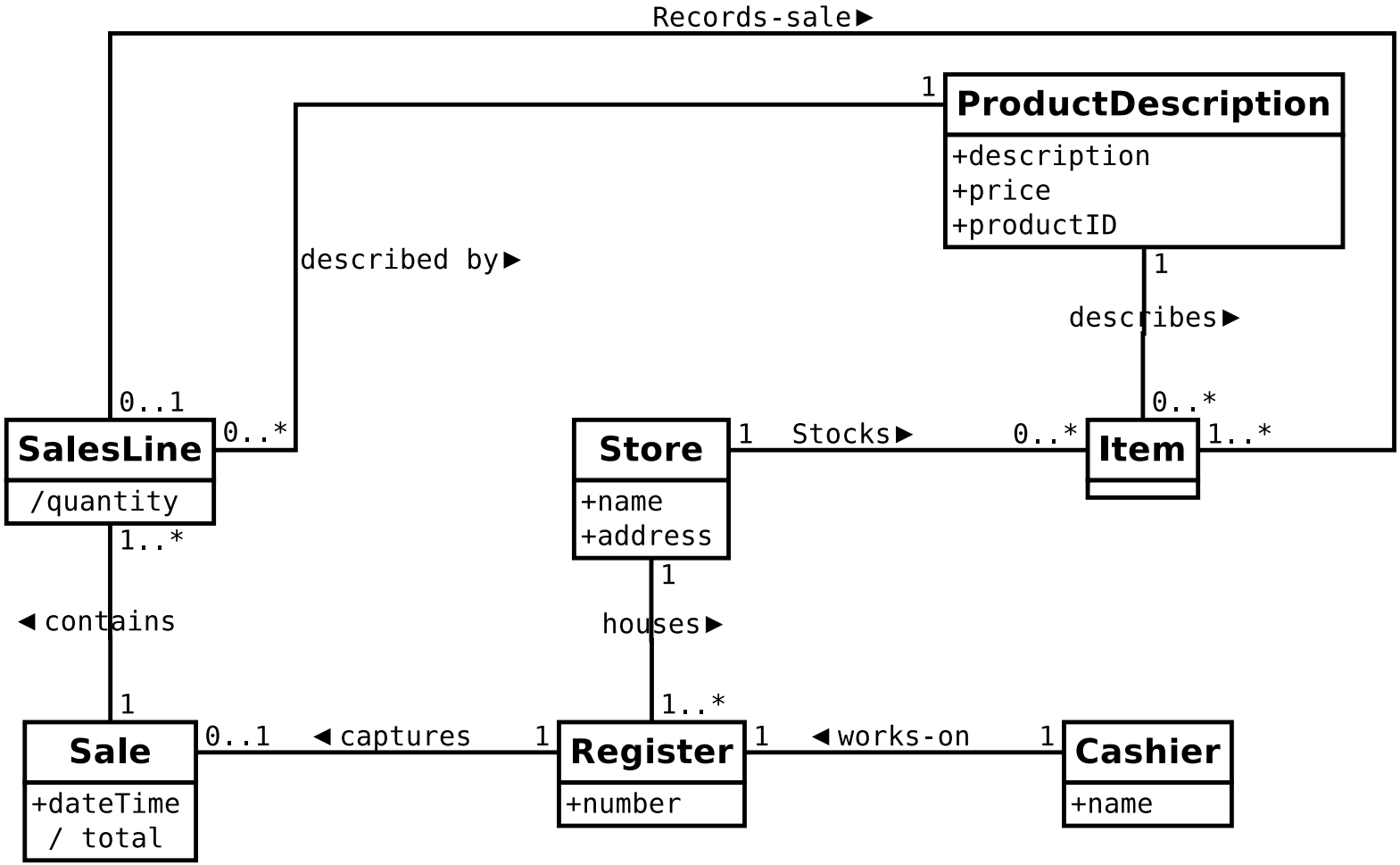
Sekvenční diagram pro kontrakt enterItem



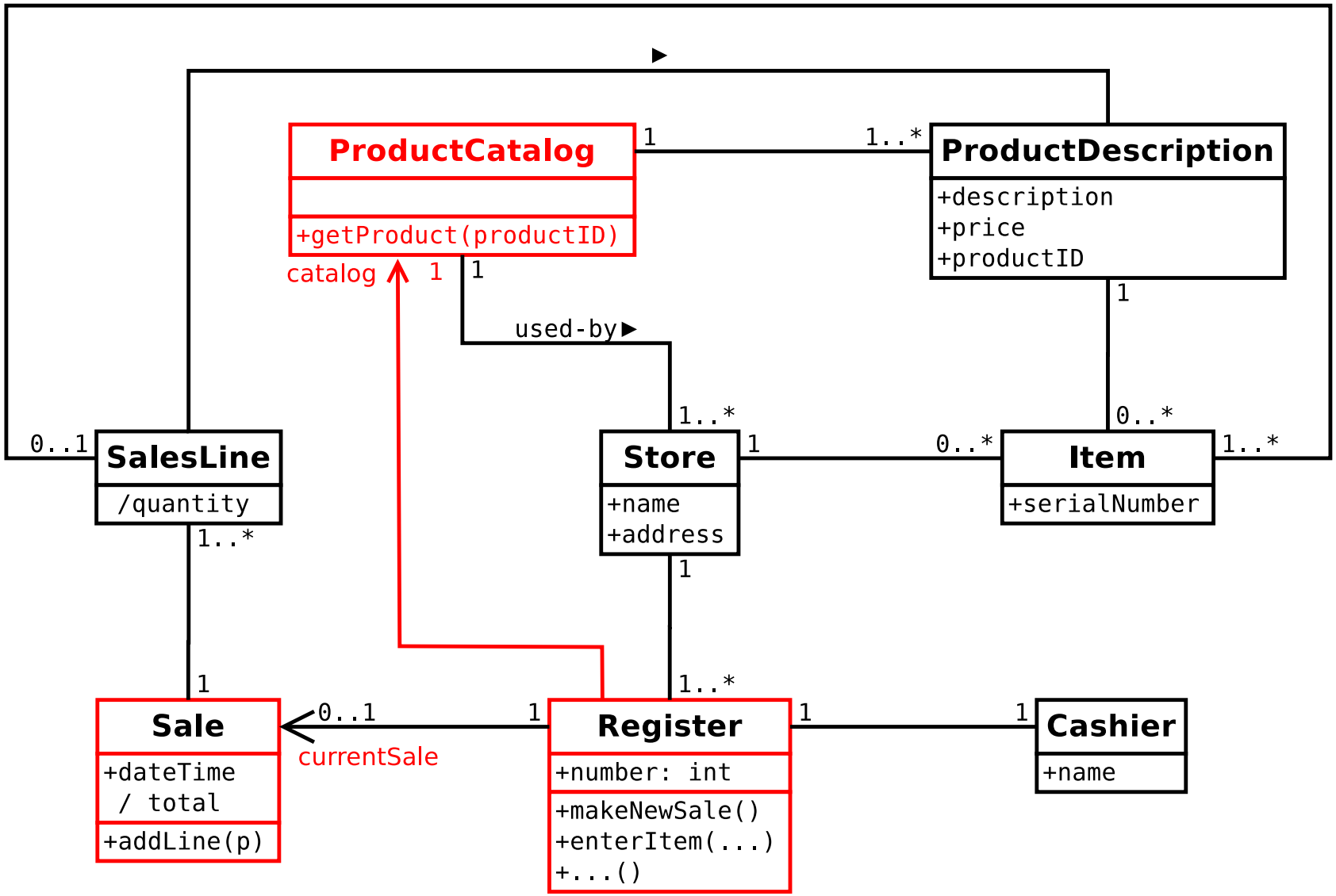
- je nutné vyhledat *ProductDescription* podle productID
- který objekt zpracuje zprávu *getProduct*?

Konceptuální model

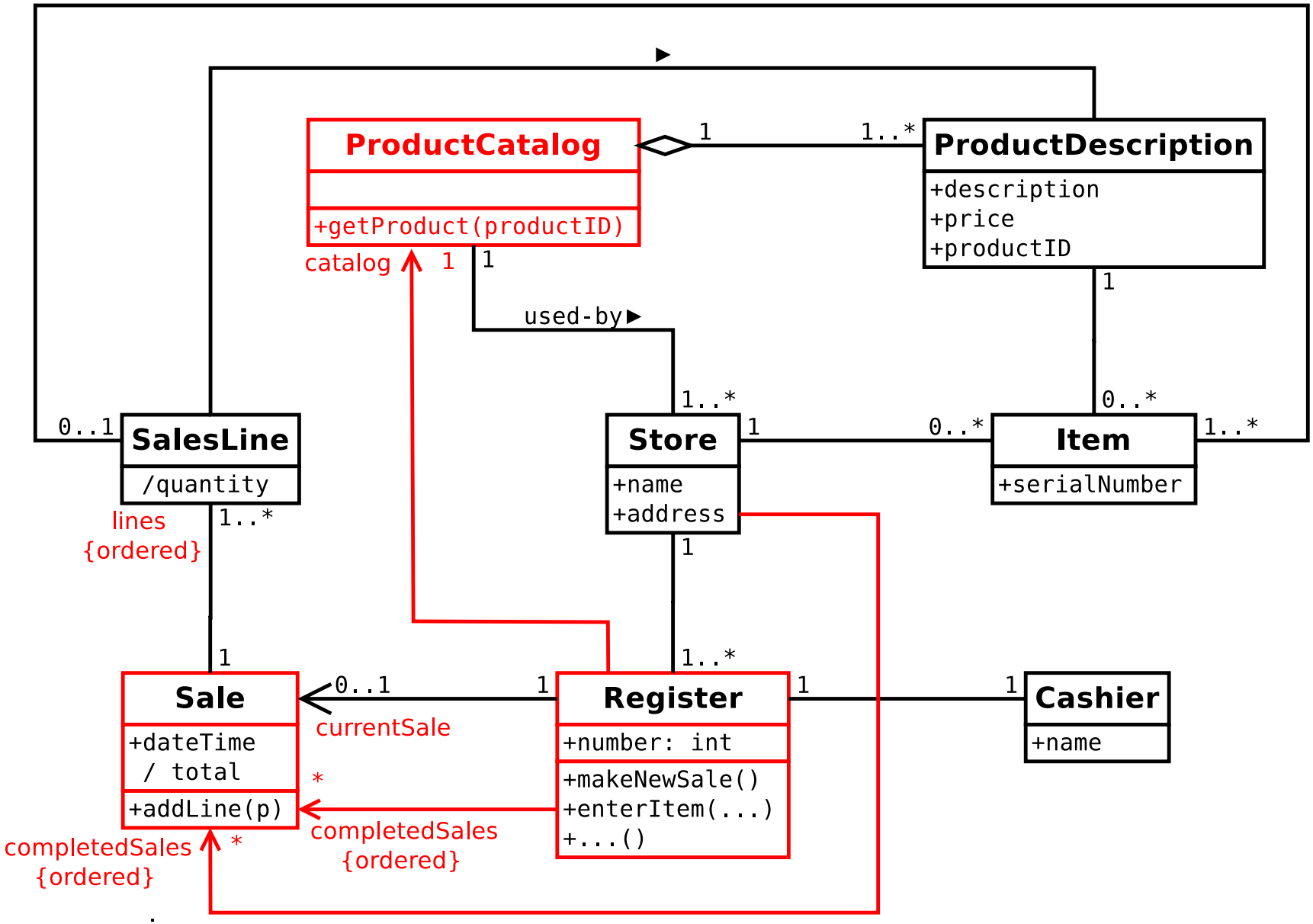
- Který objekt zpracuje zprávu *getProduct*?



Přechod k diagramu návrhových tříd

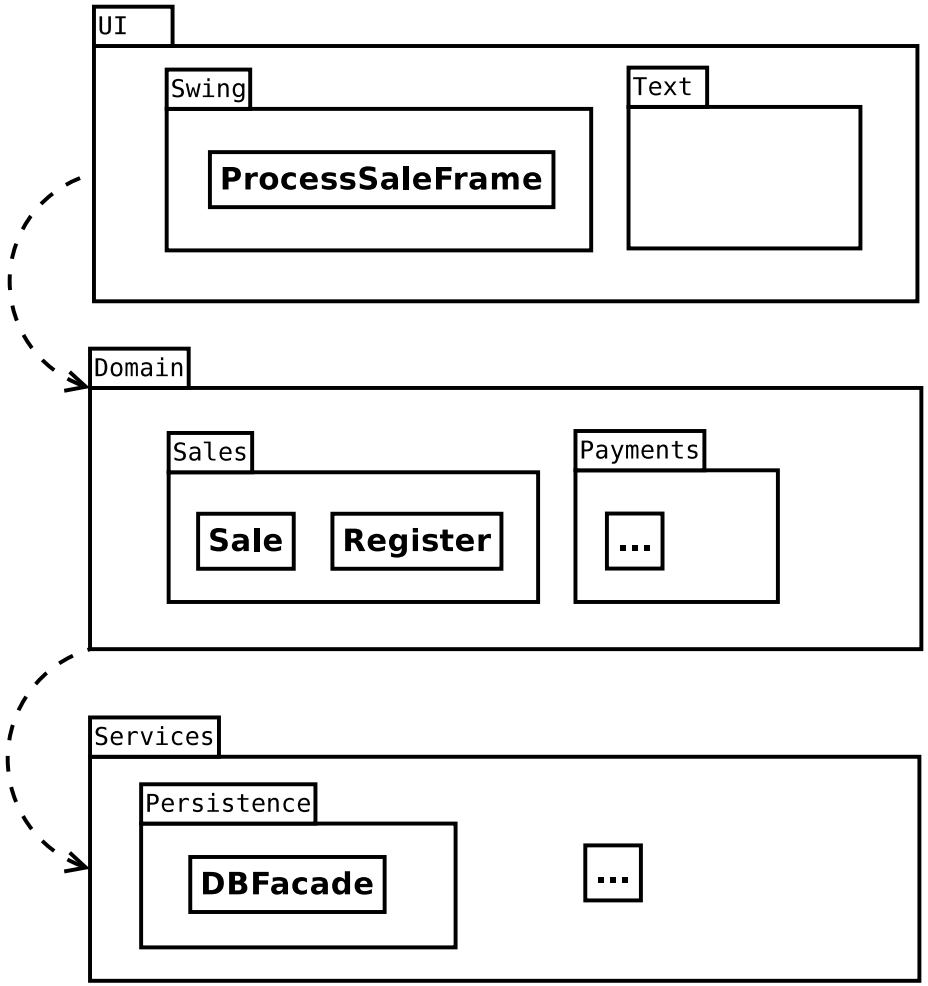


Přechod k diagramu návrhových tříd



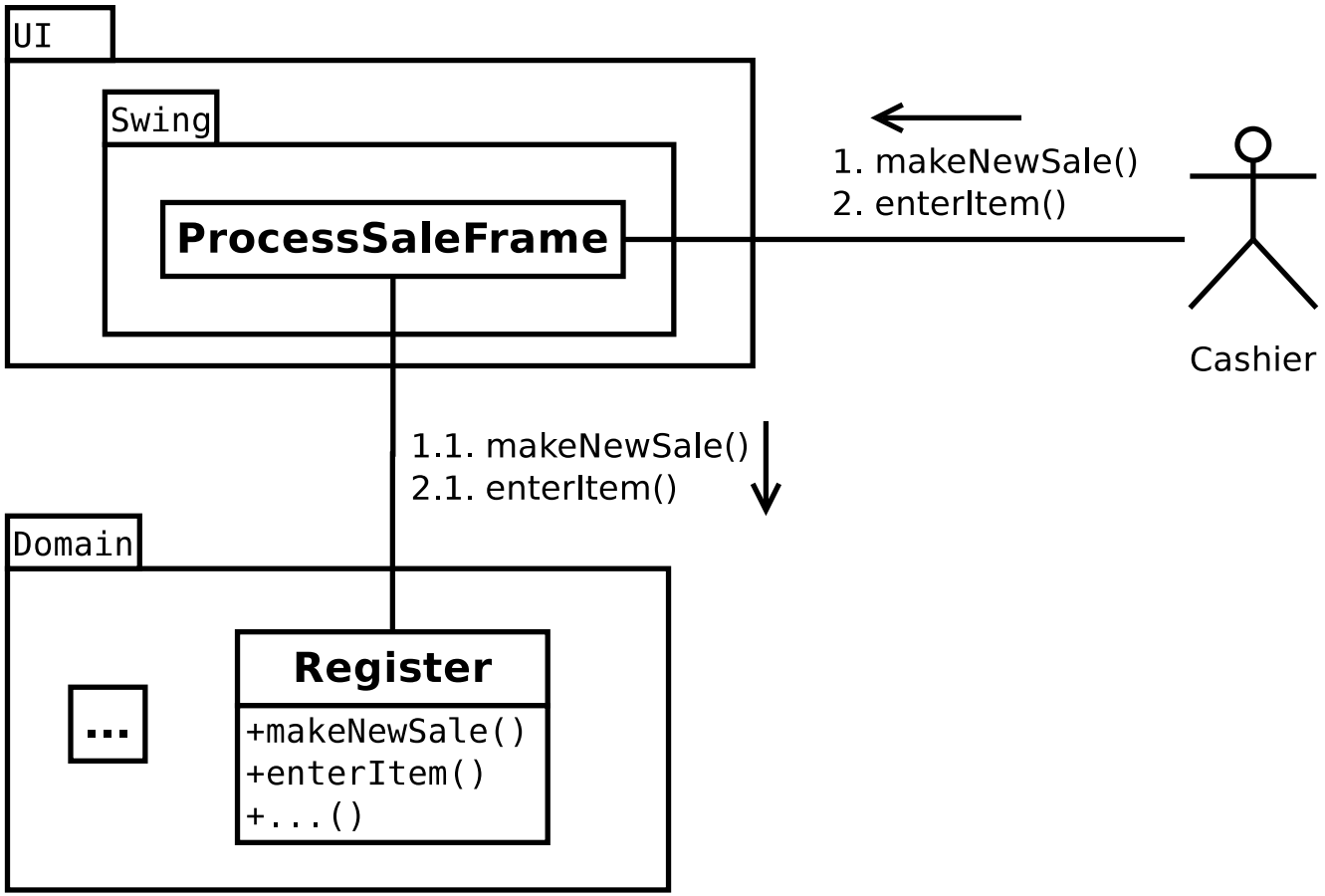
Model architektury

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



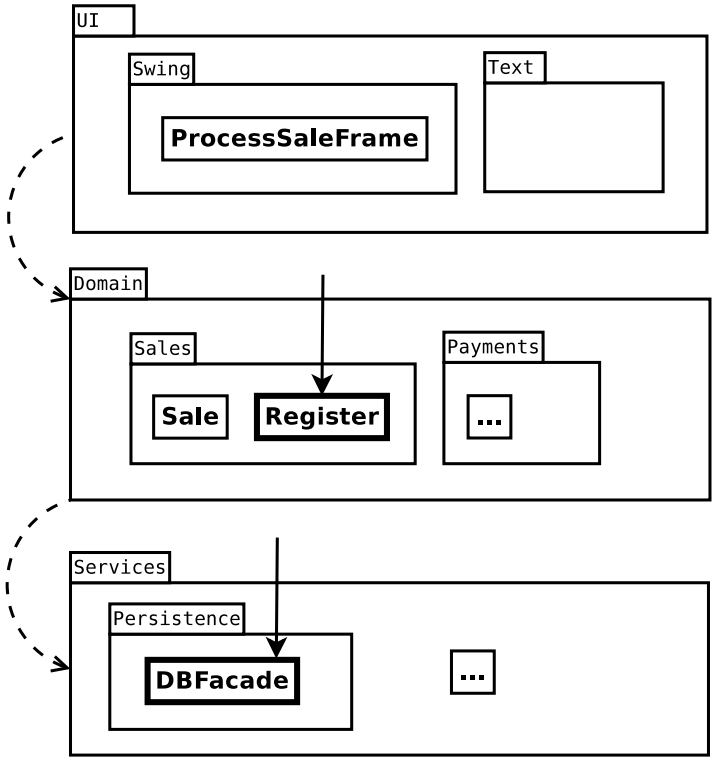
Model architektury

- model použití architektury pro námi analyzovaný scénář



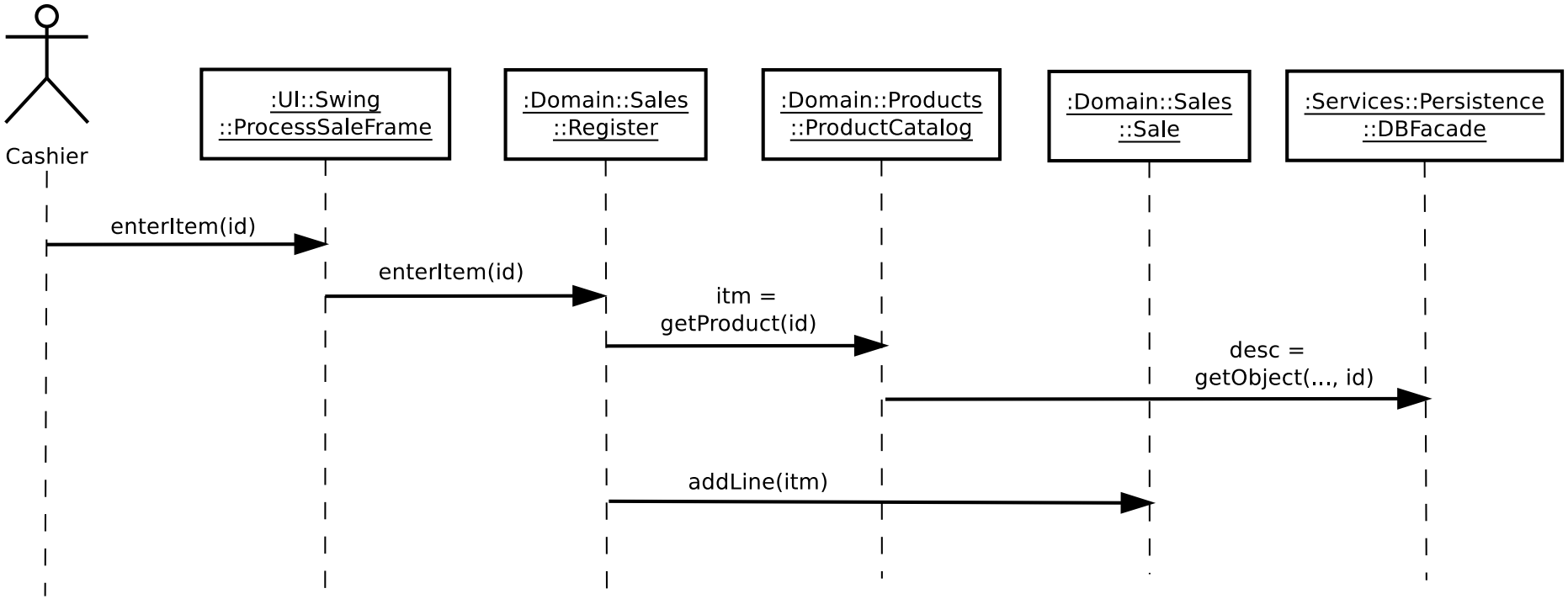
Model architektury

- rozhraní vrstev je definováno objekty (resp. třídami)
- lze aplikovat návrhový vzor *Fasáda* (*Facade*)
  - fasáda je jeden objekt (třída), která je jediná vidět z venku
  - zjednodušené rozhraní; možnost výměny vrstvy za fasádou beze změny uživatelských tříd



Architektura: Sekvenční diagram

- zachycení komunikace mezi vrstvami



Studijní koutek – Důvody ukončení studia

Studijní důvody

- alespoň 15 kreditů v 1. semestru studia
- alespoň 30 kreditů za každý rok studia
  nebo nejméně polovina zapsaných kreditů
- opakovaný povinný předmět
- státní závěrečná zkouška
  Státní závěrečnou zkoušku nebo kteroukoli její část lze jednou opakovat.
- překročení maximální doby studia (SZŘ VUT, čl. 4)
  Maximální doba studia je dvojnásobkem standardní doby studia.
- (opakovaná neomluvená neúčast v kontrolované výuce)

Kázeňské důvody

- vyloučení ze studia za závažný nebo opakovaný disciplinární přestupek

Formální důvody

- nezapsání se do dalšího ročníku

Zanechání studia písemným oznámením

Studijní koutek – Poplatek za studium

§ 58 odst. 3 Zákona č. 111/1998 O Vysokých školách (...)

- *Studuje-li student ve studijním programu déle, než je* **standardní doba studia zvětšená o jeden rok** *v bakalářském nebo magisterském studijním programu,* **stanoví mu veřejná vysoká škola poplatek za studium**, *který činí za každých dalších započatých šest měsíců studia nejméně jedenapůlnásobek základu;* **do doby studia se započtou též doby všech předchozích studií** *v bakalářských a magisterských studijních programech,* **které byly ukončeny jinak než řádně** *podle § 45 odst. 3 nebo § 46 odst. 3, nejde-li o předchozí studium, po jehož ukončení student řádně ukončil studijní program stejného typu. Období, ve kterém student studoval v takovýchto studijních programech, nebo v takovýchto studijních programech a v aktuálním studijním programu souběžně, se do doby studia započítávají pouze jednou. Od celkové doby studia vypočtené podle tohoto odstavce se však nejdříve odečte uznaná doba rodičovství.*

Příloha č. 4, článek 2, odst. 2 Statutu VUT

- *Výše poplatků za prodlouženou dobu studia za každých započatých 6 měsíců studia činí: a) trojnásobek základu v prvním roce, b) šestinásobek základu ve druhém roce, c) dvanáctinásobek základu ve třetím a dalších akademických rocích.*

Studijní koutek – Poplatek za studium

Výše základu vyhlášeného MŠMT pro akademický rok 2024/2025

- 4.838 Kč

Výše poplatku za každých započatých 6 měsíců studia
po dobu 12 měsíců pro akademický rok 2024/2025:

- 14.700 Kč pokud studium přesahuje standardní dobu zvětšenou o 1 rok
- 29.100 Kč pokud studium přesahuje standardní dobu zvětšenou o 2 roky
- 58.200 Kč pokud studium přesahuje standardní dobu zvětšenou o 3 a více let

Vizte Rozhodnutí rektora č. 3/2024.

`https://www.vut.cz/uredni-deska/vnitrni-predpisy-a-dokumenty/-d253477`