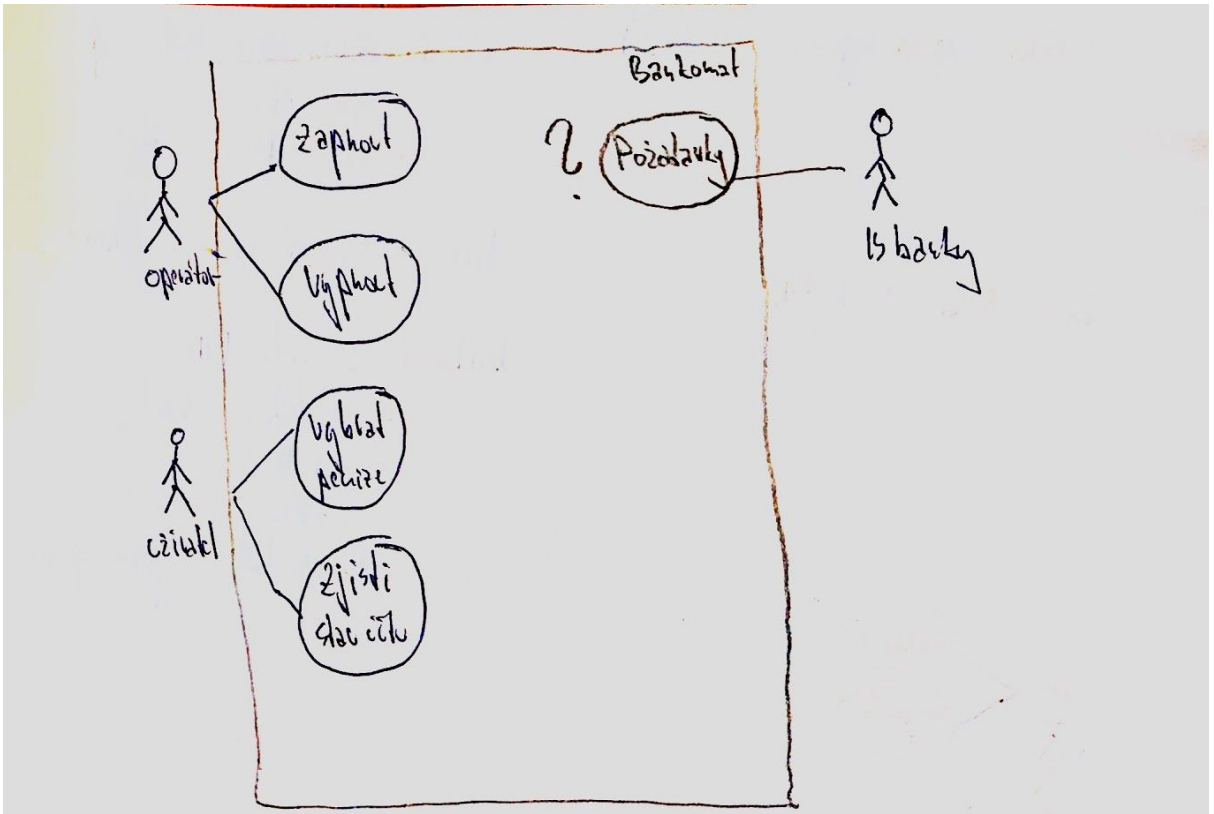
AIS priprava na semestralku

Otazky Riadny termin 2013/2014

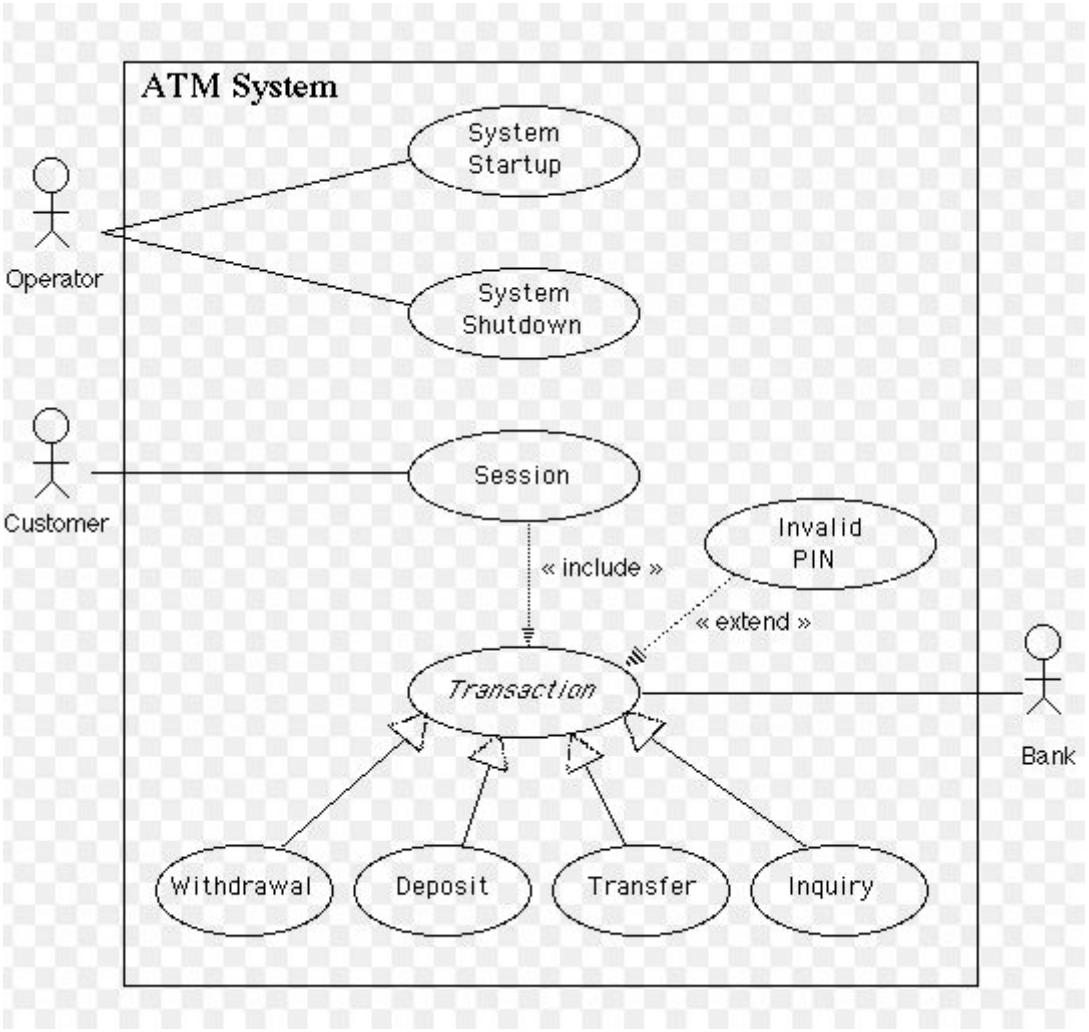
1. Uvazujte bankomat s displayom, klavesnicou, snimacom karty a zberac bankoviek. Bankomat zapina a vypina operator. Pouzivatel vlozi kartu a zadava pin. Vlozenim karty zacina transakcia. Karta obsahuje ID karty, ID uctu, a PIN. Bankomat komunikuje s IS banky, kde posiela PIN a udaje o karte pripade, ze zacne prebiehat transakcia, tak nie je mozne vypnut automat. Mozne 3 razy zadat zly pin potomdojde k zablokovaniu automatu a automat nevyda kartu. Pouzivatel moze v ramci transakcie prezerat stav uctu a vyberat peniaze. Pri vybere penazi zadava sumu, ktora musi byt nasobkom 100 Kc. Automat pri transakcii posiela poziadavky na IS banky. Po vybere penazi, automat vyda peniaze a vrati kartu pouzivatelovi.
a. Nakreslite Use-Case diagram a pre kazdy pripad pouzitia definujte pociatocne a koncove obmedzenia. 3b



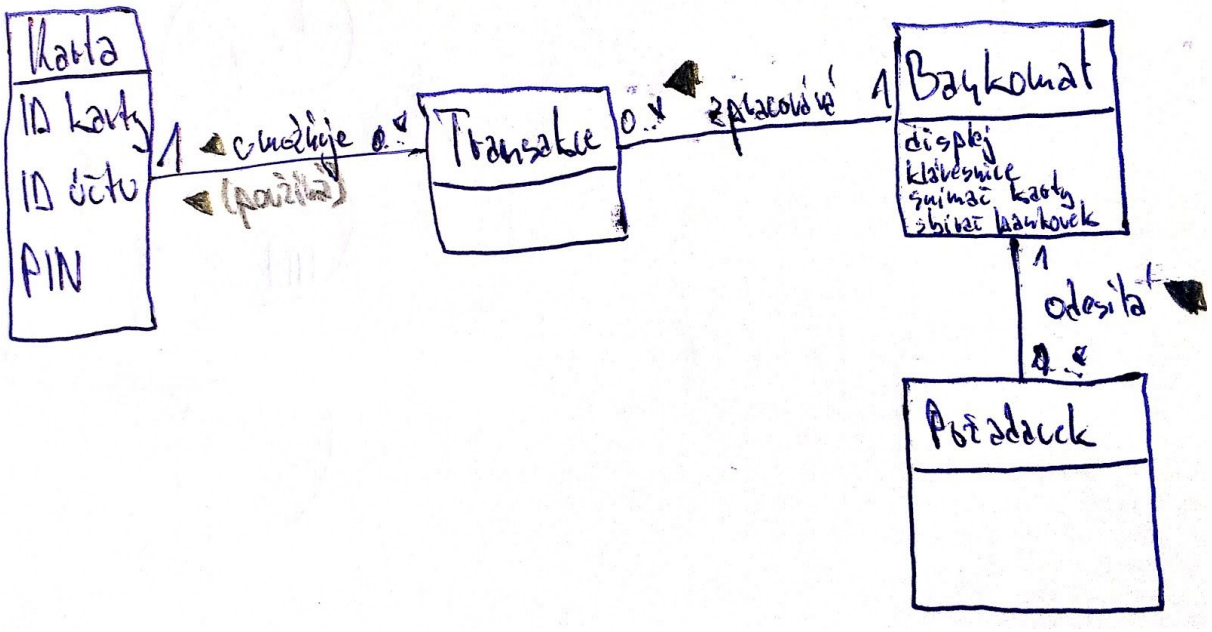
		Počáteční omezení (Předpoklad)	Koncové „omezení" (Následek podm.)
operátor	Zapnout	Bank. je vypnut.	Bank. je zapnut
	Vypnout	Bank. je zapnut. Neprobíhá transakce	Bank. je vypnut
uživatel	Vybrat peníze	Vložil kartu. Zadal správně pin. Zadal správně sumu.	Peníze vydány. Karta vydána.
	Zjistit stav účtu	Účet stejně bez kontroly sumy.	Karta vydána.

Nějaký názor??? => Nice try.

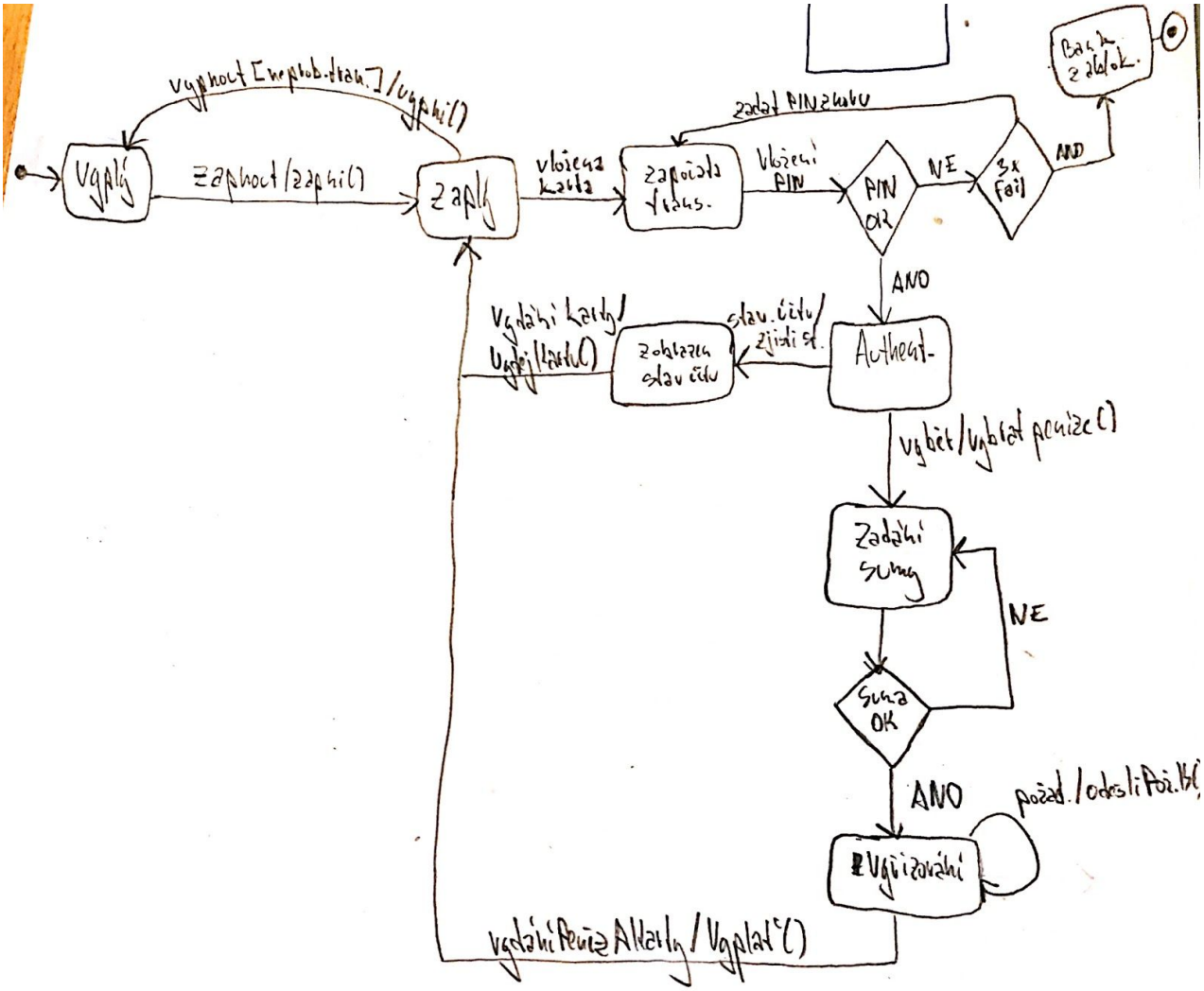
Na netu je podobný příklad, který nemá všechny formální náležitosti tohoto božského zadání, ale je řešen celkem chytře, včetně “invalid pin” stavu, viz, níže.



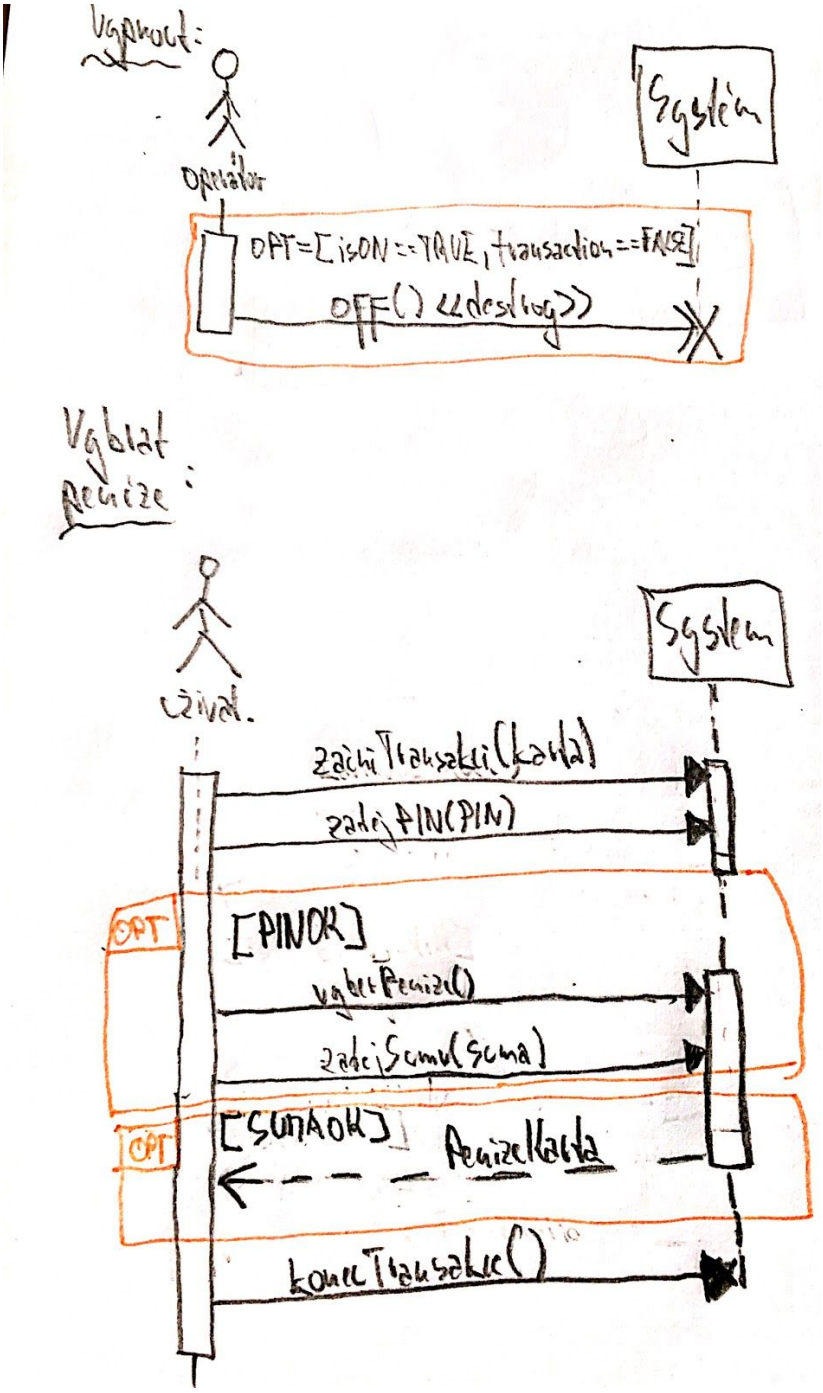
b. Nakreslite model domeny (nic nepridavajte). 2b

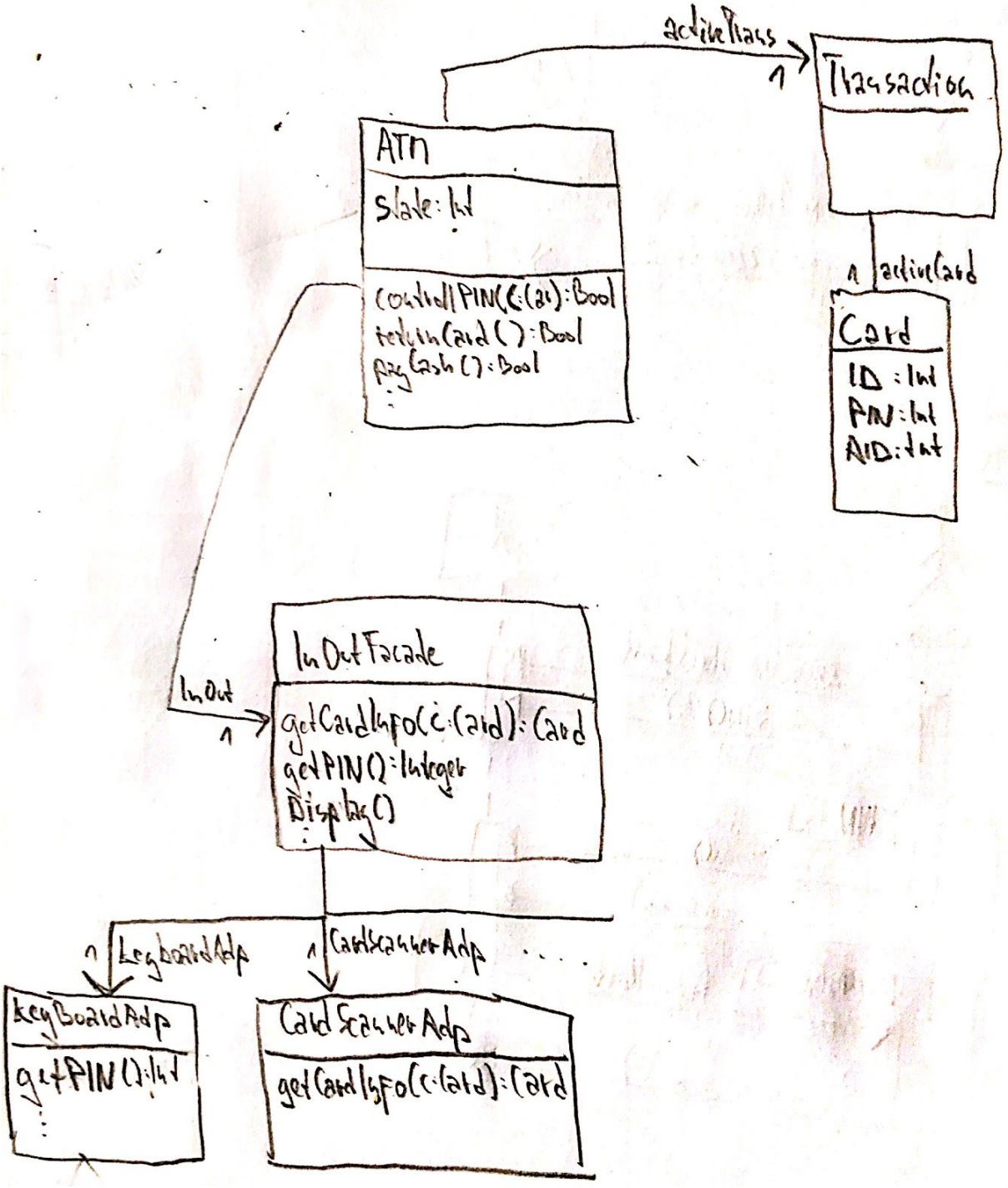


c. Nakreslite diagram chovania pre dane zadanie a napiste nazov vhodneho diagramu chovania (Prvdepodobne stavovy diagram). 8b



d. Nakreslit SSD, vyznacit nejake kontrakty pre systemove operacie, zostrojit diagram tried a este nieco




activeTrans
Transaction
ATM
state: Int
controllPIN(c:Card):Bool
returnCard():Bool
dispCash():Bool
activeCard
Card
ID: Int
PIN: Int
AID: Int
InOutFacade
InOut
getCardInfo(c:Card):Card
getPIN():Integer
Display()
KeyboardAdp
CardScannerAdp
KeyBoardAdp
getPIN():Int
CardScannerAdp
getCardInfo(c:Card):Card

1. Uvažujte, že jste členem týmu, který vyvíjí část informačního systému firmy, která bude poskytovat podporu pro řízení kritických technologických procesů firmy. Vaším úkolem je navrhnout řešení související se zpracováním hlášení o výjimečných stavech procesů. Předpokládejte, že potřebné informace o takovém stavu jsou zapouzdřeny v objektu třídy *Error*. Pro jednoduchost předpokládejte, že část systému, ve které mohou být tyto výjimečné stavy detekovány, je reprezentována objektem třídy *Detector*. Ta tedy při detekci vytvoří instanci třídy *Error* a předá ji ke zpracování skupině tříd, kterou máte navrhnout vy. Výjimečné stavy lze klasifikovat do několika úrovní významnosti (předpokládejte odpovídající atribut třídy *Error* s názvem *level*). Může existovat řada způsobů hlášení podle významnosti výjimečného stavu, např. výpis na konzolu, poslání SMS, uložení detailní informace do logovacího souboru apod. Je požadováno, aby navržené řešení bylo snadno rozšiřitelné i pro případné další typy hlášení (minimální dopad na kód ostatní části systému). Uvažujte, že zpracování může být složitější a je proto žádoucí, aby byl každý typ zpracování zapouzdřen v samostatné třídě. Navíc to, jestli daný typ zpracování proběhne, je dáno úrovní pro zpracování daného typu – proběhne pouze, je-li větší nebo rovna hodnotě *level* objektu třídy *Error*.
Při vzniku výjimečného stavu se musí objekt třídy *Error* vytvořený objektem třídy *Detector* dostat ke každému objektu, který zodpovídá za zpracování, a ten podle významnosti stavu a své úrovně zpracování požadavek zpracuje nebo ne.
Nakreslete diagram tříd a diagram sekvence nebo komunikace a komunikaci stručně popište. V diagramu tříd musí být uvedeny operace, které navrhujete pro realizaci požadovaného chování. Tyto operace popište pseudokódem nebo slovně (kde by byl popis složitý). Pokud vaše řešení vychází z nějakého návrhového vzoru, napište jeho název a popište podstatu. Pokud ne, diskutujte, proč se žádný ze vzorů probíraných v předmětu AIS nehodí. **18 bodů**

Jsou dvě řešení Observer a Chain of responsibility. Věděl by někdo nakreslit? -> riesil to na prednaske (v které přednášce?)

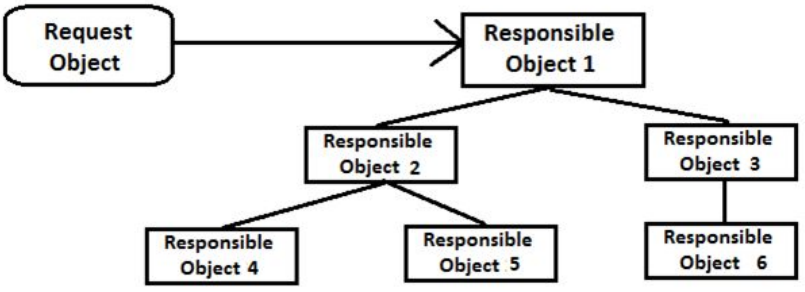
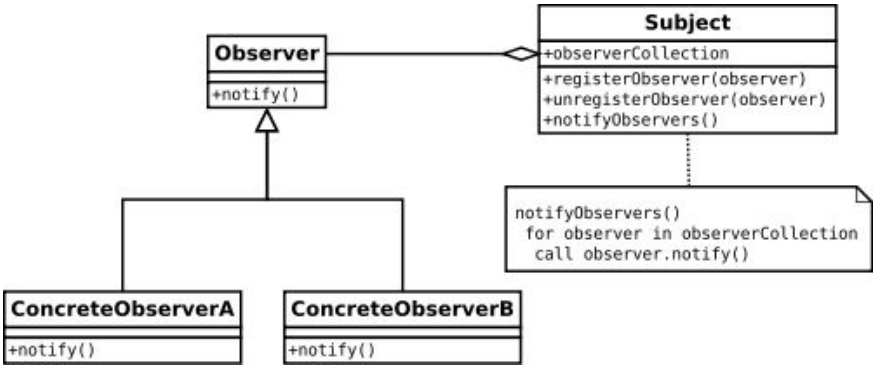


Fig: Tree Chain of Responsibility



1. Monopoly
a) nakresli domenovy model
b) rozpis vyuzitych metod a jejich prirazeni do trid a zapis zavislosti (doporuceno pouzit vzory GRASP)
c) nakreslit diagram trid a diagram interakce
https://www.fit.vutbr.cz/study/courses/AIS/private/prednasky/13_monopoly.pdf

2. Co znamena ze prvek B je zavisly na prvku A?
Co je mysleno prvky?
Jaky maji mezi sebou vztahy?
Vypsat alespon tri typy takovych zavislost + demonstrovat diagrmama trid

Závislost prvku B na prvku A znamená, že změna v prvku A bude mít vliv na funkcionalitu prvku B. Závislost se standardně značí přerušovanou šipkou, která ukazuje směr závislosti. Příkladem může být třída, která implementuje datový typ. Například třída “Potravina”. Jiná třída pak tento datový typ používá, tudíž je na ní závislá. Závislostí je také dědění, či realizace rozhraní. Dalším příkladem je zobrazení závislosti v případě, kdy třída implementuje nějakou metodu, která ve svém těle volá metodu jiné třídy.

3. Popis refaktorizace a alespoň 3 její důvody včetně podmínky. Co znamená a k čemu se používají Extract Method, Introduce Explaining Variable a Replace Constructor With Factory Method.

Disciplinovaná metoda přepisu nebo restrukturalizace existujícího kódu beze změny vnějšího chování, aplikace malých transformací kombinovaných s opakovaným spouštěním testů. Vede ke snadnějšímu pochopení kódu a usnadnění dalších úprav.

Důvody:
1. Refaktorizujeme proto, abychom zlepšili návrh software.
2. Refaktorizace vede k lepší srozumitelnosti software.
3. Refaktorizace pomáhá hledat chyby -> lepší pochopení kódu při refaktorizaci.

Extract Method:
Transformuj dlouhou metodu na kratší vyčleněním logické části do privátní pomocné metody.

Introduce Explaining Variable:
Ulož výsledek výrazu nebo části výrazu do přechodné proměnné se jménem vysvětlujícím smysl

Replace Constructor With Factory Method:
V Javě se nahradí použití new a konstruktoru voláním pomocné metody, která skryje detaily vytvoření.

4. Popsat Singleton. O co se jedná a k čemu slouží. Popsat na příkladu pomocí diagramu tříd a diagramu sekvence.

V UML se značí jedničkou pravém horním rohu. Ať už v diagramu tříd, tak v diagramu sekvence.

Jedná se o návrhový vzor, kdy chceme, aby pro danou třídu mohla být vytvořena jen jedna instance v rámci celého programu s globálním přístupem. Příkladem použití může být například dialogové okno.

5. Popsat kontejner a na co slouží. Dále popsat vztahy a ukázat příklady v JavaEE.
Části aplikačního serveru podle specifikace Java EE, které poskytují prostředí pro běh aplikačních komponent a zprostředkovávají obsluhu jejich rozhraní s okolním prostředím (spravují systémové zdroje).

Celkove cca pulka bodu byla za Monopoly, druha pulka za zbytek

1. termín:
Refaktorizace kdy jak proc 3 (alespon) typy... 10b
Viz výše

Registr sluzeb + jaka sluzba (nevim co) 6b
SOA je paradigma předepisující způsob uspořádání a použití distribuovaných služeb, které mohou být spravovány různými vlastníky. Definuje jednotný způsob inzerce, hledání, vzájemné komunikace a spotřeby služeb s cílem dosáhnout požadovaných výsledku v souladu s měřitelnými předpoklady a očekáváním.

Vsechny UML2.0 diagramy popis kde kdy co zobrazuji 11b
Modelování statické struktury

- diagram tříd
  - *Vizualizuje třídy (a rozhraní), jejich interní strukturu a vztahy k ostatním třídám.*
- diagram objektů
- diagram balíčků
- diagram komponent
- diagram složené struktury
- diagram nasazení

Modelování dynamické struktury (chování)

- diagram případů použití
  - *Zobrazuje chování systému (nebo jeho části) z hlediska uživatele.*
- diagramy interakce
  - *Modelování interakce tříd spolupracujících k dosažení požadované funkcionality*
  - diagram sekvence
  - diagram komunikace
  - přehledový diagram interakce
  - diagram časování
- diagram stavového automatu
  - *Zachycení stavů objektu a aktivit objektu v těchto stavech, událostí, které vedou ke změně stavu, akcí, které se změnou stavu souvisí*
- diagram aktivity
  - *Modelují chování bez nutnosti specifikovat statickou strukturu tříd a objektů*

UP modely a artefakty vsechny kdy kde proc 10b
odpoved

artefakty == milníky? zahájení - cíle, rozpracování - architektura, konstrukce - počáteční provozní způsobilost, zavedení - prod. verze
Ne, artefakty jsou prvky vyvíjeného systému, jejichž rozmístění se ukazuje diagramem nasazení a artefakt jako modelovací prvek se zpravidla vyskytuje v diagramu komponent (kde ukazuje, které komponenty implementuje).
Je to nějaký výsledek SW projektu.
Např.: Soubory se zdrojákem, EXE soubory, JAR archívy, DB tabulky, skripty, dokumenty... prostě když děláte školní projekt, tak to, co odevzdáváte do WISu, jsou artefakty.
Balíčky jako stereotyp artefaktu se v UML značí jako <<file>>.
Samotný frenkt pak <<artifact>>.
Kdyžtak mě opravte, pokud to chápu špatně.

- Model případů použití - fáze zahájení, rozpracování
- Doplňující specifikace

    - typicky nefunkční požadavky a co není/nelze vyjádřit v případech použití(také rysy (features), např. logování).
- Slovník
    - definice významných pojmů;zahrnuje také datový slovník (omezení, rozsahy atd.).
- Vize
    - shrnuje požadavky vysoké úrovně, rozpracované v případech použití a Doplňující specifikaci. Shrnuje hodnotu projektu (business case).
- Pravidla(business/domain rules)
    - rpavidla aplikační domény, případně i legislativa.

Priklad na navrhove vzory - bylo zadani ze ktereho se melo poznat kteremu NV vyhovuje a aplikovat ho - diagram komunikace/sekvence + trid 14b
odpoved

2. termín:
Diagram aktivit
odpoved

Validace a verifikace
Verifikace je kontrola, zda vyvíjený systém vyhovuje specifikacím (našemu návrhu). Máme nějaký model/návrh a podle něj jsme vyvinuli systém. Verifikací ověřujeme, že námi vyvinutý systém odpovídá návrhu, čili jsme vyrobili to, co jsme vyrobit chtěli (**vytváříme produkt správně**). Mezi metody verifikace patří většina testování, kterého se neúčastní zákazník (jednotkové testy, integrační testy apod.), inspekce kódu, simulace či různé formální analýzy a verifikace.

Validace je kontrola, zda vyvíjený systém splňuje to, co od něj zákazník očekává. Dal nám požadavky a zajímá jej, zda to, co dostane, je to, co chce. Je to tedy ověřování, že děláme (vyvíjíme) to, co máme (**vytváříme správný produkt**). Mezi metody validace patří např. akceptační testy, které se provádí u zákazníka (pokud např. zákazník požadoval, že operátor systému musí být schopen vykonat určitou činnost do půl minuty, tak se zkouší, zda to možné je).

Typy testování

Software Quality Assurance - SQA
Ve slajdech jsem nenašel...

Příklad na návrhový vzor - Composite

Problém: Jak zpracovat skupinu nebo složenou strukturu objektů stejně (polymorfně), jako by byly atomickým objektem?

Řešení: Vytvořit třídu pro kompozit a atomické objekty tak, aby implementovaly totéž rozhraní.

Dobře vysvětleno zde:
https://cs.wikipedia.org/wiki/Composite

Kontejner, komponenta - typy
Komponenty:
Části aplikace implementované vývojářem aplikace pomocí aplikačních technologií Java EE. (klientské aplikace, applety, JSP a servlety, a komponenty EJB)

Kontejnery:
Části aplikačního serveru podle specifikace Java EE, které poskytují prostředí pro běh aplikačních komponent a zprostředkovávají obsluhu jejich rozhraní s okolním prostředím (spravují systémové zdroje).

3. termín:

1) Co je to polymorfismus, rozhraní, abstraktní třída a zapouzdření (charakteristika, plus pokud to šlo, tak nakreslit v UML)

Polymorfismus
Prostředek, jehož pomocí může být jediný název operace nebo atributu definován na základě více než jedné třídy a může nabývat různých implementací v každé z těchto tříd

Rozhraní
Rozhraní je (opakovaně použitelná) skupina operací, která specifikuje určitý aspekt chování třídy a které třída používá ve vztahu k jiným třídám

Abstraktní třída
Abstraktní třída je třída, kterou je možno použít jako nadtřídu, ale nelze vytvářet její instance.

Zapouzdření
Zabalení operací (metod) a atributů představující nějaký stav do jednoho objektu, takže daný stav je přístupný či upravitelný pouze prostřednictvím rozhraní poskytovaného zapouzdřením (tedy pomocí operací či metod)

2) FURPS+ (popsat složky + metriky)

- Funkční
  - rysy, schopnosti, bezpečnost
- Použitelnost (Usability)
  - lidské faktory, nápověda, dokumentace, …
- Spolehlivost (Reliability)
  - frekvence výpadků, zotavitelnost, …
- Výkonnost (Performance)
  - doba odezvy, propustnost dostupnost, využití zdrojů, …
- Podporovatelnost(Suportability)
  - přizpůsobitelnost, udržovatelnost, konfigurovatelnost, internacionalizace, …

3) kontejnery a komponenty v java EE (stejná jako na prvním opravném)
Viz výše.

4) Příklad na uml diagram -Zjistit který použít, popsat jeho dvě hlavní části a jejich vlastnosti, nakreslit pro daný scénář

5) Příklad na návrhové vzory, máme třídu Clock, která představuje hodiny, a její metodu tick(), kterou volá interní časovač (posun o sekundu). Pak jsou různé třídy, které různě zobrazují čas pomocí gui (digitalClock, analogClock).

Observer, viz:
https://cs.wikipedia.org/wiki/Observer

dalsi 3 termin:

1) Zvolit vhodny UML diagram, popsat jej a nakreslit podle scenare - jednalo se o aktivitu (16b)

2) FURPS+ a jeho metriky (10b)

Viz výše.

3) Popsat model domeny a systemovy sekvencni diagram vzhledem k unifikovanemu procesu, co to je, kam bychom je zaradili, zda jsou povinne atp. (8b)

Model domény

- Ukazuje koncepty (proto také konceptuální model) dané aplikační domény, ne softwarové objekty.
- **Je nepovinný**, ale může být užitečný (má vliv na softwarové objekty vrstvy domény v modelu návrhu).
- Má podobu diagramu tříd.
- Je třeba se vyvarovat nadměrného úsilí na tvorbu dokonalého modelu domény (typické pro model vodopád).
- Používá se ve fázi rozpracování (elaboration).

Systémový sekvenční diagram

- Ukazuje vstupní (generované aktérem, vyžadující nějakou systémovou operaci) a výstupní události systému.
- Pro konkrétní scénář ukazuje aktéry, generované události a jejich pořadí.
- Kreslí se pro hlavní úspěšný scénář každého případu použití a časté nebo komplikované alternativy
- Používá se ve fázi rozpracování (elaboration).

4) (14b)
a) Vysvetlit princip proc se oddeluje vrstva modelu od pohledu, k cemu slouzi ty 2 casti

Důvody:

- UI se často mění oproti logice aplikace
- data často zobrazována různě (podle role uživatele, HW atd.)
- odlišnost návrhu UI oproti návrhu logiky aplikace (vyžaduje jiné dovednosti)
- Obtížnější testování UI

Oddělení aplikační logiky od uživatelského rozhraní. Jednodušší správa, možnost rozdělení na dvě vývojové skupiny, pokud se dodrží rozhraní. Pohled slouží jako grafické uživatelské rozhraní, model pak jako aplikační logika.

b) Popsat navrhovy vzor observer/publish-subscribe, k cemu se pouziva, nakreslit jeho diagram trid, diagram sekvence

https://cs.wikipedia.org/wiki/Observer

5) Co je to refaktorizace, k cemu se pouziva (3b)
Viz výše.

loni 1.termin:
1) Docela zdlouhavy popis problemu, pro ktery je potreba navrhnout reseni -> nakreslit diagram trid, sekvencni a pridat popis plu pouzite navrhove vzory (tady jich bylo mozne vymyslet vice, ale docela se nabizel chain of responsibility)

2) Popis sekvencniho diagramu a diagramu komunikace, nakreslit priklad, jejich rozdily, pouziti atd.

Diagram sekvence
Zachycuje časově uspořádanou posloupnost zasílání zpráv mezi objekty. Sekvenční diagram nejčastěji znázorňuje spolupráci několika vzorových objektů v rámci jednoho případu užití.

Diagram komunikace

Zdůrazňuje statickou strukturu, která se využije při interakci k dosažení požadovaného chování. Syntaxe podobná diagramu sekvence, ale bez „dvoudimenzionálnosti“. Opět možnost vyjádřit podmínky a iterace –větší nebezpečí snížení přehlednosti. Užitečné pro počáteční „brainstorming“ a agilní modelování (snadnější kreslení).

3) Postup pri navrhu SOA, popsat jednotlive fáze

Identifikace „kandidátních“ služeb

Tři přístupy:

- analýza shora-dolů, (dekompozice business procesů na nové služby)
- analýza zdola-nahoru, (kompozice existujích služeb do business procesů)
- modelování cílových služeb (angl. goal-service modeling). (ohodnocení služeb a výběr dle jejich business přínosu)

Využívá diagramy případů užití (UML) a popisy business procesů (BPMN), tj. aktivity procesů rozdělené mezi za ně zodpovědné aktéry.

Klasifikace služeb

Testuje „kandidátní“ služby na soulad s vlastnostmi SOA. (znovupoužitelnost, abstrakce, bezstavovost, atd.)

Analýza podsystémů

(rozkládá „kandidátní“ služby na komponenty, navazuje předchozí na dekompozici)

- funkční komponenty
    - poskytují business funkce, (např. výpis odebraných položek objednávky pro fakturu)
- technické komponenty
    - poskytují podpůrné funkce, (např. konverze měn, obecný přístup do databáze, atd.)
- „služby“ jak komponenty
    - realizují aktivity business procesů. (např. vystavení faktury)

Specifikace komponent

- vzniká datový model pro rozhraní komponent (rozhraní), (např. jako konceptuální diagram tříd v UML)
- probíhá návrh spolupráce a komunikace komponent (protokol). (např. jako UML sekvenční diagram volání komponent)

Sestavení služeb

- přiřazení komponent do příslušných vrstev architektury, (sdílení technických komponent, bezstavovost funkčních komponent, atd.)
- sestavení komponent do výsledných služeb.

Realizace služeb

- rozhodnutí o způsobu realizace služby, (konkrétní technologie)
- řešení otázek bezpečnosti, správy a monitorování služeb.

4) Nejaka otazka na externi prednasku

5) Vysvetlit, nakreslit vhodny diagram pro polymorfismus, zapouzdreni, interface, abstraktni tridu

1. opravny:

1.Activity diagram

2.SQA, testovani, verifikace, validace, Black-box, White box

Black box testování
https://en.wikipedia.org/wiki/Black-box_testing

White box testování
https://en.wikipedia.org/wiki/White-box_testing

Verifikace a validace
Viz výše.

3.JAVA EE

4.MVC

Původně pro Smalltalk na konci 70. let (pasivní a aktivní model) –odděluje modelování domény, prezentace a akcí založených na uživatelských vstupech.

- Model – odpovídá vrstvě domény, obsahuje objekty domény, přidává k datům aplikační logiku (např. výpočet sumy prodeje u třídy Sale).
- View –zodpovědný za prezentaci modelu (objekty UI, dynamické HTML stránky).
- Controler –zpracovává a reaguje na události (typicky akce uživatele), může měnit stav objektů balíčků Model i View.

5.Priklad - nakreslit diagram (tried a sekvencni)

2 + V-Model
3) Kontajnery a komponenty
Viz výše.

5 + nakreslit navrhhovy vzor a spravne pomenovat, diagram tried s operaciami popisanymi + diagram sekvencny alebo komunikacie

1)Vytvorit diagram tried, sekvencny diagram podla dlhsieho zadania, kde ale vysvetlil, ze co tam chce. Ak sa dalo, tak pouzit vhodne navrhove vzory.

2) Vysvetlit zavislost tried a dalsie boli asi zavislosti komponent a zavislosti vrstiev, nakreslit a vysvetlit typy zavislosti, ktore sa mozu vyskytnut.
Viz výše.

3) Nieco s EJB

4) Test driven development, vysvetlit + jeho vyhody

5) Co je zodpovednost vzhladom navrhu. Napisat dva principy GRASP aj popis,
k
Co se týče toho EJB, tak chtěl napsat, co je EJB komponenta a EJB kontejner. Dále typy EJB beans a v čem se liší.

2. Pouzit navrhovy vzor a zostrojit diagram tried a komunikacie. - Co3mposite 12b

Viz výše.

3. SOA - Popiste ako sa odlisuje IS s SOA architekturou od IS bez SOA architektury. Zamerte sa na dekompoziciu, zivotny cyklus komponent, komunikovanie sluzieb a databazove ulozisko.

Informační systém – bez architektury SOA

- podsystémy jsou monolitické (komplikovaný přístup k jednotlivým funkcím, nejasné rozhraní, obtížná integrace celých podsystémů)
- podsystémy jsou striktně oddělené, každý má samostatný životní cyklus (implementací, správou, možností modifikace, atd.)
- komunikace podsystémů přes společné úložiště dat (problematická návaznost procesů, např. „Order Status“, nejasná zodpovědnost za data)

Informační systém – s architekturou SOA

- dekompozice na úroveň jednotlivých služeb (jasně definovaná rozhraní služeb, podsystémy vznikají kompozicí)
- podsystémy se prolínají sdílenými službami, každá má samostatný životní cyklus (sdílení společných částí podsystémů, např. „Check Order Status“)
- přímá komunikace služeb, úložiště dat pro dokumenty (služby komunikují napříč podsystémy, s daty manipuluje služba reprezentující k datům příslušný business proces)

4. Agilny vyvoj. Popiste 2 najdolezitejsie charakteristiky modelu agilneho vyvoja. Popiste co je to TDD a ake prinosy ma vzhladom ku agilnemu vyvoju. Popiste co je Refaktorizacia a ake prinosy ma vzhladom ku agilnemu vyvoju.

Agilní vývoj

- komunikace (komunikace je důležitá a častá, uvnitř týmu i ven mezi týmem a zákazníkem)
- jednoduchost (tvorba jednoduchých modelů, spíš pro porozumění, než pro dokumentaci)
- zpětná vazba (rychlá a častá zpětná vazba, reakce na předložené modely a interakce)
- odvaha (dělat rychlá rozhodnutí, zkoušet nové, zahazovat či měnit stávající)
- pokora/respekt (naslouchat ostatním vývojářům i zákazníkům, přijímat jejich nápady)

1. Programování řízené testy (*Test-driven development* (**TDD**)) je přístup k vývoji software, který je založen na malých, stále se opakujících krocích, vedoucích ke zefektivnění celého vývoje.

Prvním krokem je definice funkcionality a následné napsání testu, který tuto funkcionalitu ověřuje. Poté přichází na řadu psaní kódu a nakonec úprava tohoto kódu (Refaktorizace).

Refaktorizace
Viz výše.

Vrstvy abstrakce SOA

Vrstva business procesů
BP je posloupnost kroků respektující business pravidla a vedoucí k zisku (hmotnému i nehmotnému), reprezentován sekvencí provedení několika služeb (choreografie služeb),

Vrstva služeb
Rozhraní jednotlivých komponent sjednocena do služeb, služba za běhu sestavuje komponenty a přeposílá jim požadavky, služba na rozhraní zpřístupňuje své funkce (popis služby),

Vrstva komponent
Základní stavební kameny služeb, realizace funkčnosti služeb a zajištění požadované kvality služeb (QoS), komponaenty jsou černé skříňky a jejich funkce jsou přístupné pouze přes rozhraní.

Cykly mezi třídami (řešení pomocí rozhraní, co je rozhraní, k čemu je, k čemu je v OO)

podle pripadove studie Monopoly nakreslit model domeny ... pocitalo se s ruznymi typy policek, vyplnit tabulku podobnou CRC stitkum a podle ni nakreslit diagram trid a interakce

Singleton - co je to, kdy pouzit, prakticky priklad pouziti, diagram trid a interakce …
Viz výše.

1) Popis a vztah architektur: logicka vs. nasazeni vs. softwarova. Popsat architekturu s vrstvami.

Softwarová architektura je definována jako „základní organizace softwarového systému zahrnující jeho komponenty, jejich vzájemné vztahy a vztahy s okolím systému, principy návrhu takového systému a jeho vývoje“. Návrh softwarové architektury informačního systému je jedním klíčových kroků vývojového procesu softwarového produktu. Architektura informačního systému leží na vyšší úrovni abstrakce tak, že zahrnuje jak pohled na aplikační doménu (tj. „pohled zákazníka“), tak pohled vývojáře na globální strukturu systému a chování jeho částí, jejich propojení, synchronizaci, rozmístění zdrojů a schéma jejich rezervace, přístup k datům a toky dat v systému, fyzické rozmístění komponent a další.

Logické architektury (Architecture)

vrstvené architektury (vertikální a horizontální cˇleneˇní),
(peer-to-peer, klient-server, referencˇní model ISO/OSI, . . .)
architektonické vzory,(Model-View-Controller, Presentation-Control-Mediator-Entity-Foundation, . . .)
návrhové vzory,(GoF: adapter, factory, singleton, . . . ; GRASP: inf. expert, creator, controller, . . .)
využívající neˇjaké paradigma.(architektura orientovaná na služby (SOA), komponentové systémy, . . .)

Architektury nasazení (Deployment)

dané prostˇredím (síťové topologie), (hveˇzda,strom,ad-hoc,...)
dané platformou,(BEA WebLogic, IBM WebSphere, JBoss, . . .)
dané implementacˇní technologií. (EJB, WebServices, CORBA, . . .)

2) Popsat jednotlive faze UP

- docela zdlouhavy popis problemu, pro ktery je potreba navrhnout reseni -> nakreslit diagram trid, sekvencni a pridat popis plu pouzite navrhove vzory (tady jich bylo mozne vymyslet vice, ale docela se nabizel chain of responsibility)
- popis sekvencniho diagramu a diagramu komunikace, nakreslit priklad, jejich rozdily, pouziti atd.
- postup pri navrhu SOA, popsat jednotlive faze
- nejaka otazka na externi prednasku
- vysvetlit, nakreslit vhodny diagram pro polymorfismus, zapouzdreni, interface, abstraktni tridu

Ta otázka týkající se externí přednášky byla: "Uveďte typy Open Source projektů z hlediska vývojového cyklu. "

2) Vysvetlit zavislost tried a dalsie boli asi zavislosti komponent a zavislosti vrstiev, nakreslit a vysvetlit typy zavislosti, ktore sa mozu vyskytnut.
Viz výše.

3) Nieco s EJB

4) Test driven development, vysvetlit + jeho vyhody

Programování řízené testy (*Test-driven development* (**TDD**)) je přístup k vývoji software, který je založen na malých, stále se opakujících krocích, vedoucích ke zefektivnění celého vývoje. Prvním krokem je definice funkcionality a následné napsání testu, který tuto funkcionalitu ověřuje. Poté přichází na řadu psaní kódu a nakonec úprava tohoto kódu (Refaktorizace).

Použití automatických testů při vývoji přináší **tu výhodu**, že je možné opětovné spouštění velkých množství testů, které znovu a znovu ověřují veškeré předem definované funkcionality a požadavky. Pokud nějaký zásah do kódu způsobí někde nějakou chybu, automatické testy tuto chybu odhalí.

Podobná situace nastává i při změně požadavků na dosavadní funkcionalitu. Pokud je potřeba někde něco změnit, stačí upravit nebo napsat nové testy a následně i kód. Pokud by tyto změny vedly k nefunkčnosti v jiných částech programu, opět lze snadno a rychle zjistit, co je špatně. Protože tento přístup skládá program z malých částí, je mnohem jednodušší, vypořádat se z různými vnějšími zásahy do návrhu programu.

5) Co je zodpovednost vzhladom k navrhu. Napisat dva principy GRASP aj popis,