1. Popište, co přináší implementace a integrace podnikového informačního systému pomocí struktury SOA (oproti situaci bez použití SOA). Soustřeďte se zejména na dekompozici systému (tj. co jsou jednotlivé komponenty), na životní cyklus komponent systému (např. pro údržbu systému), na komunikaci mezi komponentami systému a na přístup k ukládání dat (např. databáze)

6 bodů

Hodnocení: popis dekompozice za 2 body, popis životního cyklu za 2 body, popis komunikace za 1 bod a popis přístupu k ukládání za 1 bod.

Vzorová odpověď (8. a 9. slid přednášky): V SOA dekompozice na jednotlivé služby (zatímco bez SOA jsou kompozicemi většinou monolitické informační systémy pro jednotlivé procesy podniku), každá služba má samostatný životní cyklus (celý systém lze udržovat/aktualizovat po menších částech, než bez použití SOA), služby spolu přímo komunikují (na rozdíl od komunikace přes speciální rozhraní či datová uložiště pokud není SOA) a data jednotlivých částí úložišť jsou manipulována/spravována pro ně specifickými službami (např. s daty manipuluje jen služba reprezentující k datům příslušný business proces, např. správa zákazníků).

---

2. Uveďte alespoň tři doporučované praktiky pro agilní modelování (Agile Modeling) týkající se tvorby modelů při návrhu a pro dokumentaci software. Popiše, jak a proč modelovat v souladu s těmito doporučeními (výhody a rizika ve srovnání s neagilními přístupy).

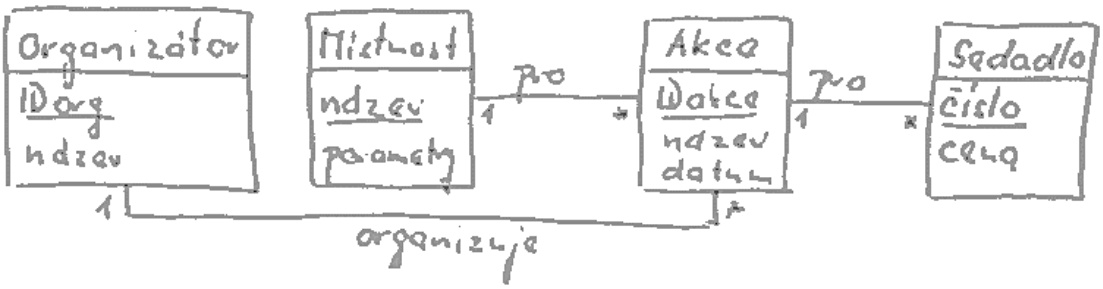
7 bodů

Hodnocení: každá praktika s vysvětlením za 2 body, srovnání za 1 bod

Vzorová odpověď (11. až 19. slid přednáška): cokoliv, např. Active Stakeholder Participation (modelování se zástupci uživatelů), Document Continuously/Late (průběžné modelování se vznikem finálního kódu vs. až zdokumentování výsledku), Just Barely Good Enough/Just In Time (modelování pro pochopení, ne pro dokumentaci, až je to potřeba a jen dokud je to potřeba), Multiple Models/Single Source Information (více modelů jako více pohledů, ale s jedním původem, nejčastěji kódem).

---

3. Předpokládejte, že vyvíjíte software ro zákazníka, který vlastní a pronajímá místnosti pro organizování kulturních akcí. Pokud si někdo chce pronajmout nějakou místnost, dodá objednávku, na základě které majitel nebo jím pověřená osoba vloží příslušné informace o organizátorovi a akci do systému a vytvoří organizátorovi v systému účet (pokud ho ještě nemá). To mu umožní provést tzv. konfiguraci místnosti (pro účely tohoto příkladu uvažujte, že akce se vždy koná v jedné místnosti), tj. určit ceny vstupenek pro jednotlivá sedadla v místnosti. Tato informace je pak využita při rezervacích a nákupech vstupenek. Vaším úkolem je realizovat základní scénář případu použití *konfigurujMistnost*, který rozšiřuje případ použití *prohlizeniAkci*, který umožňuje prohlížet detaily akcí přihlášeného organizátora a zvolit pro aktuální zobrazenou funkci konfigurace místnosti. K dispozici máte již vytvořený dílčí model domény, který ukazuje koncepty a jejich vztahy, důležité pro vaše návrhová rozhodnutí. Atribut parametry třídy Místnost zapouzdřuje hodnoty reprezentující rozmístění sedadel v místnosti. Třída *Sedadlo* reprezentuje sedadlo pro danou akci, její atributy *cena* cenu vstupenky na tuto akci pro toto sedadlo (právě toto definuje organizátor při konfiguraci). Atributy s unikátními hodnotami jsou podtrženy.



U scénáře *konfigurujMistnost*, jehož realizaci navrhujete, je primárním aktérem organizátor akce, podmínkou je, že je přihlášený a má následující kroky:

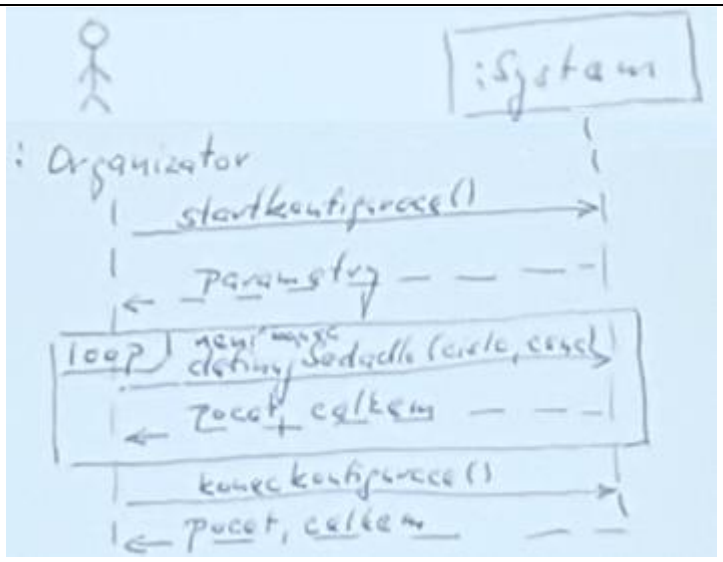
1. Organizátor zahajuje výběrem funkce konfigurace místnosti.
2. Systém zobrazí rozmístění sedadel v místnosti.
3. Dále se do ukončení konfigurace opakuje:
   3.1. Organizátor vybere sedadlo a zadá cenu.
   3.2. Systém zaznamená sedadlo a jeho cenu pro aktuálně zpracovávanou akci a zobrazuje celkový počet již „oceněných“ sedadel a průběžnou potenciální tržbu ze vstupenek na tato sedadla.
4. Organizátor ukončí konfiguraci a systém zobrazí celkový počet oceněných sedadel a celkovou potenciální tržbu ze vstupenek.

Nezabývejte se tím, jestli by takový průběh konfigurace byl uživatelsky přívětivý nebo ne, berte to jako požadavek daný tímto zadáním.

a) Nakreslete systémový diagram sekvence scénáře, který má být realizovaný.

3 body

Řešení:



b) Popište kontrakty systémových operací pro kroky 1 až 3 výše popsaného scénáře ve tvaru:
název_operace (parametry): předpoklad; stav_po_vykonání_operace

3 body

startKonfigurace(): je prohlížena organizátorem akce; je vytvořena instance třídy Mistnost na základě identifikátoru akce (hodnota atributu IDakce), je inicializována a asociována s prohlíženou akcí.

definujSedadlo(číslo, cena): sedadlo označeno a „oceněno" organizátorem; je vytvořena instance třídy Sedadlo, inicializována hodnotami argumentů a je asociována s akcí.

c) Pro systémové operace z bodu b) **navrhněte realizaci** (**řešíte pouze vrstvu domény**, pro výstupy systémových operací jen dostupnost informace). Protože případ použití rozšiřuje jiný případ použití, jsou už navrženy některé třídy jejich atributy dle modelu domény a operacemi (přinejmenším gettery a setter pro přístup k atributům) a vytvořeny jejich instance. Ty můžete využívat, jsou-li vhodnými kandidáty na přiřazení zodpovědnosti, a přidávat u nich další potřebné atributy a operace. Jedná se o tyto třídy a instance (viz též diagram v zadání k bodu d) níže):
AkceCTR – řadič případu použití *prohlizeniAkci*, vytvořena instance;
Akce – reprezentuje objednanou akci, vytvořena instance, včetně prázdné kolekce typu *List<Sedadlo>*, a asociována s řadičem;
Sedadlo – reprezentuje sedadlo s cenou pro konkrétní akci;
Organizator – reprezentuje organizátora akce, vytvořena instance;
DataMGR – třída zodpovědná za přístup k perzistentním datům a vytvoření instancí odpovídajících tříd, vytvořena instance (ta vytvořila instance tříd *Akce* a *Organizator*), je asociována s řadičem;

Nejprve strukturovaným způsobem podobným CRC štítkům rozhodněte o **přiřazení zodpovědnosti třídám** a o **spolupracujících třídách** (opět přiřazení zodpovědností a jim odpovídajících operací). Každé přiřazení zodpovědnosti zdůvodněte (PROČ jste přiřadili danou zodpovědnost právě této třídě, ne co operace dělá). Pokud použijete některý princip GRASP, nestačí uvést jenom jeho název, ale i tady musí být zřejmé, proč. Není zde nutné uvádět detailně zodpovědnost za vytvoření kolekce, zařazení do kolekce, čtení z kolekce, čtení a nastavování hodnot atributů (stačí až v diagramu v d)) a není třeba uvádět typy parametrů operací, stačí vhodná jména.

8 bodů

Zodpovědnost, operace	Třída	Zdůvodnění	Spolupracující třídy
startKonfigurace()	AkceCTR	Již vybraná jako řadič případu použití	Akce, DataMGR
Získání identifikátoru *getIDakce()*	Akce	Vlastní atribut	
Vytvoření instance třídy Mistnost a její inicializace *vytvorMistnost(IDakce)*	DataMGR	Zodpovědnost za přístup k datům a vytváření instancí již přiřazena	
Vytvoření asociace s akcí setMistnost(mistnost)	Akce	Vlastní atribut	
definujSedadlo(číslo,cena)	AkceCTR	Je řadičem	Akce
Vytvoření instance třídy Sedadlo a zařazení do kolekce sedadel *vytvorSedadlo(číslo,cena)*	Akce	Vlastní kolekci sedadel	List<Sedadlo>, Sedadlo
Dostupnost návratových hodnot			
Dostupnost parametrů místnosti *ziskejParametry()*	Akce	Má asociaci na objekt třídy Mistnost, kterou hodnotu vlastní	Mistnost (get)
Dostupnost počtu sedadel a celkové ceny	Akce	Vlastní kolekci sedadel, snadno zjistí	List<Sedadlo>, Sedadlo

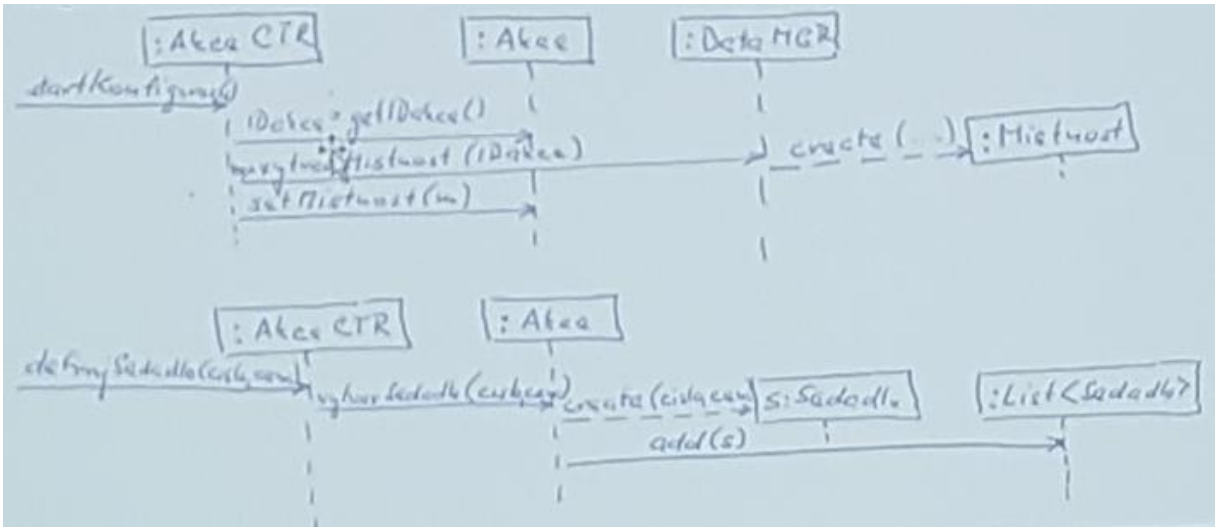
d) Doplňte **návrhový diagram tříd**, obsahující již navržené třídy, o část, kterou jste navrhli na základě tabulky z c). Musí obsahovat všechny navržené operace (parametry a návratové hodnoty není třeba uvádět) a atributy. (4 body z 13 celkem)

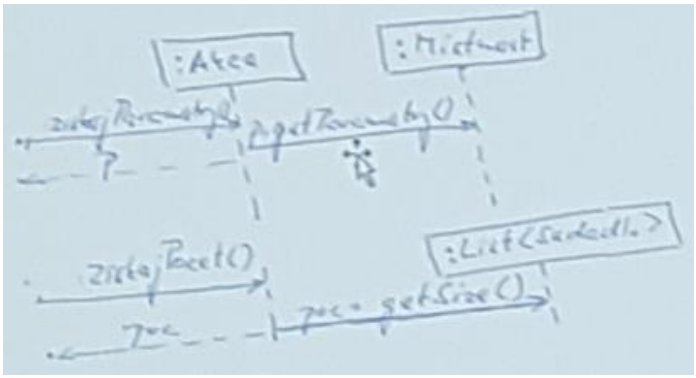
Černě bylo předpřipraveno zadání, modře doplněno řešení.

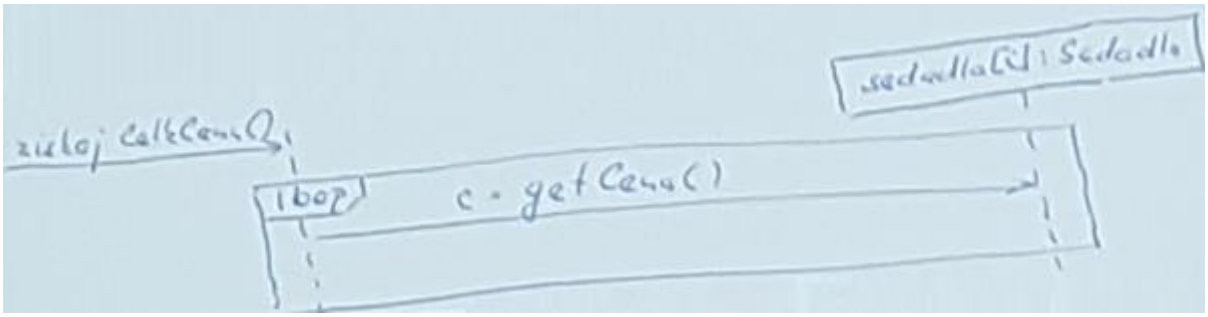
Dále nakreslete **diagramy sekvence nebo komunikace** pro navržené systémové operace a zajištění dostupnosti jejich návratových hodnot. (9 bodů z 13 celkem)

13 bodů

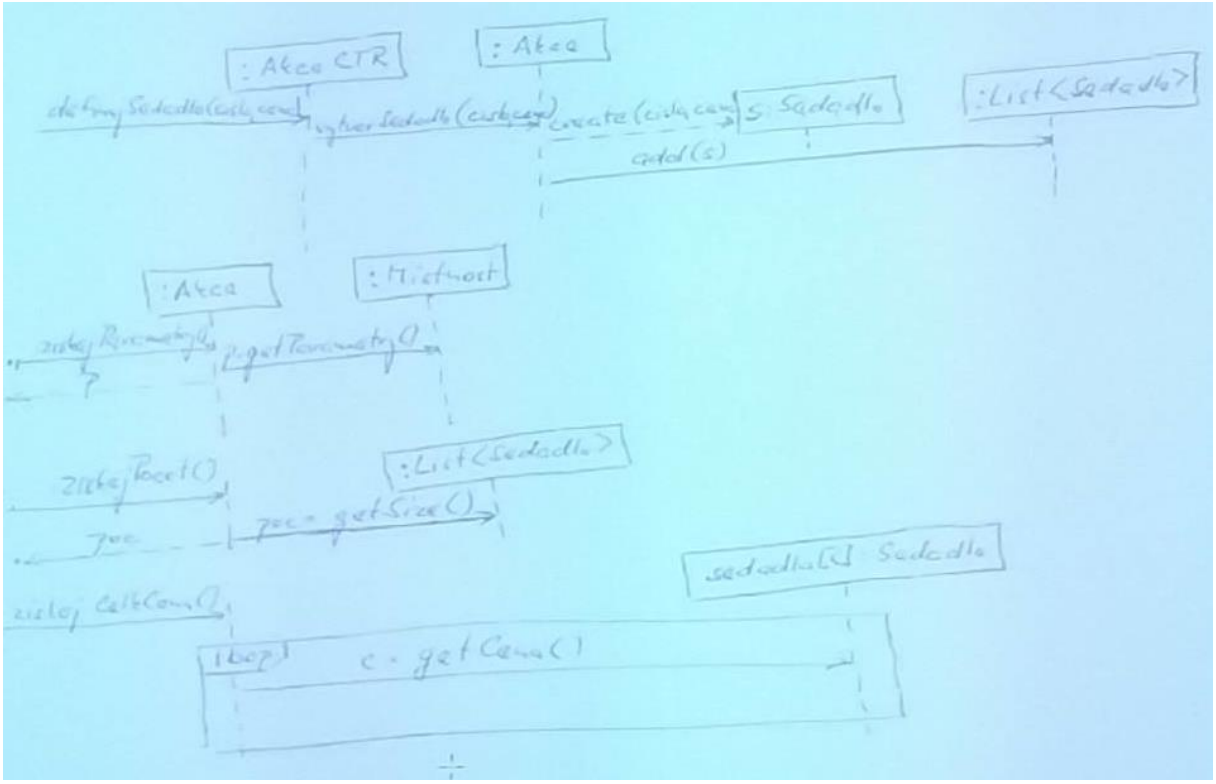
Řešení:





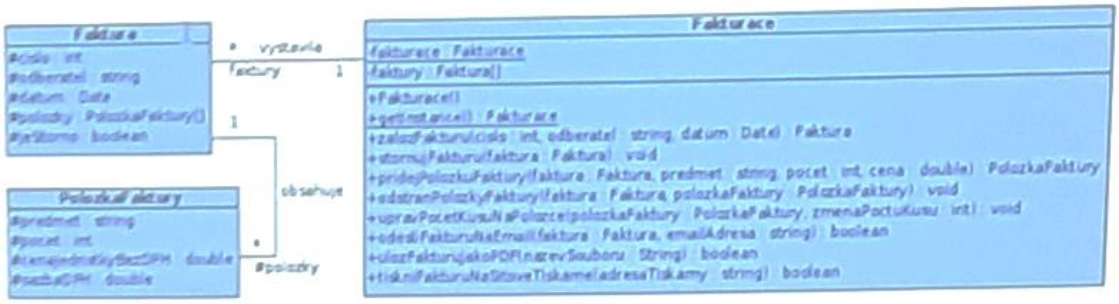


Jiný zdroj (stejné řešení):



---

4. Předpokládejte, že vyvíjíte aplikaci, jejíž součástí je modul pro vystavování faktur. Toto je realizováno třídou *Fakturace*, která uchovává seznam faktur a provádí jejich založení, stornování, nastavení jejich položek a jejich odeslání emailem, uložení do PDF souboru, či tisk na sdílené tiskárně. Kromě metod pro uvedené operace obsahuje třída i bezparametrický konstruktor a statickou metodu *getInstance()*, který vrací při volání vždy jednu a tutéž instanci (objekt) třídy *Fakturace*.



a) Uvedený návrhový diagram tříd obsahuje použití (nejméně) jednoho návrhového vzoru. Pojmenujte tento vzor (česky nebo anglicky) a formou diagramu sekvence objasněte, jak je zde implementován.

2.5 bodu

b) Přesné zadání bohužel chybí (včetně řešení a)); pouze přibližně:

- Něco s antivzory, podobně jako a) – ukázat, jaký/jaké antivzor(y) v tom příkladu byl(y) použit(y), jak to vyřešit...

---

5. Něco o stereotypu v UML, co to je, jak to funguje... (nejspíš 8.5 bodu za 4b a 5 dohromady...)