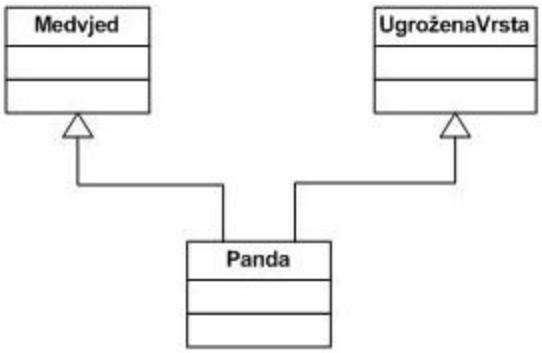
ZAD1

Pri razvoju neke aplikacije za praćenje životinja, u objektnom modelu su definirane klase Medvjed i Panda. Budući da je panda vrsta medvjeda, između klasa je ustanovljen odnos nasljeđivanja tako što je Panda izvedena iz Medvjeda. Nakon nekog vremena se pojavio zahtjev da se u aplikaciji prati i, odnosno pravi razlika između ugroženih vrsta životinja i neugroženih (ie. svih ostalih). Developer je to modelirao na sljedeći način:



a) (*5 bodova*) Koji je osnovni problem ove hijerarhije, odnosno njome ustanovljenih klasa i njihovih veza?
b) (*10 bodova*) Kako bi poboljšali ovaj dizajn (radi općenitosti pretpostavite da postoji klasa Životinja, iz koje je izvedena klasa Medvjed, a i ostale životinje relevantne za aplikaciju)? (hint: uvedite baznu klasu ŽivotinjskaVrsta, iz nje izvedite UgroženaVrsta i NeugroženaVrsta i onda … sad vi)

a) Osnovni problem je u tome što pande mogu prestati biti ugrožene, pa stoga klasa Panda treba prestati nasljeđivati klasu UgroženaVrsta, a može i medvjed postati ugrožen. (mixed-role kohezija)

by Deathclaw

Ovdje ne kuzim bas kak panda nasljeduje iz dve klase, po meni bi bolje bilo imat Zivotinja abstract class ko kak je napiso i onda da se iz nje izvode Medvjed itd... I neka sučelja IJeUgrožena i INijeUgrožena

Jos jedno moguce rjesenje, za kasniji laksi rad sa vrstama, ako koja postane ugrozena:
Normalna hijerarhija zivotinjskog svijeta, npr: Animalia -> Chordata -> Mammalia -> Carnivora -> Ursidae -> Ailuropoda -> Ailuropoda melanoleuca (Panda).
Bazna klasa definira, ajmo-tako-rec strategiju postupanja sa zivotinjom, tj., takvim rjesenjem ostavljamo zajednicke karakteristike zivotinja na svom mjestu, dok svakoj zivotinji dajemo mogucnost dinamicke promjene karakteristike ugrozenosti.

```
public class ZivotinjskaVrsta {

	protected AnimalStrategy strategy;

	public ZivotinjskaVrsta(AnimalStrategy strategy) {
		setStrategy(strategy);
	}

	public int status {
		return this.strategy.status();
	}

	public void setStrategy(AnimalStrategy strategy) {
		this.strategy = strategy;
	}

	public void execute () {
		this.strategy.execute();
	}

	//the rest of the class

}

public class AnimalStrategy {
	//defines abstract methods for handling animals
}

public class EndangeredStrategy extends AnimalStrategy {
	//defines methods for handling endangered animals
}

public class SafeStrategy extends AnimalStrategy {
	//defines methods for handling safe animals
}
```

by Lopina

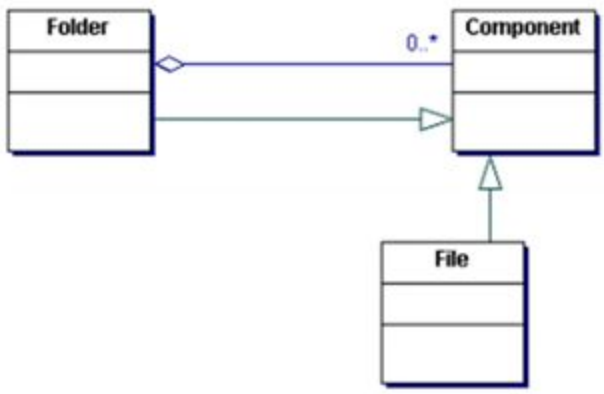
---

http://books.google.hr/books?id=iNAezyMExBkC&pg=PA303&lpg=PA303&source=bl&ots=BKp-84D9vq&sig=YQmef3kEuIzhVW76WvwTIlaxm9U&hl=hr&ei=947FTo_KIMKP4gTn-KHDDQ&sa=X&oi=book_result&ct=result&resnum=1&ved=0CB8Q6AEwAA#v=onepage&q=bear%20panda%20species%20object%20class%20inheritance&f=false

Evo rješenja u knjizi Fundamentals of object-oriented design in UML
(Alan A.)

ZAD2

(*10 bodova*) Dan je sljedeći dizajn kojim su modelirani fajlovi i folderi (direktoriji) u nekom operativnom sustavu.



n

Željeli bi u klasu Folder dodati funkcionalnost za izračunavanje ukupne veličine svih fajlova i foldera unutar tog foldera.
Pitanje za vas je što bi sve trebalo dodati u navedene klase kako bi se ta funkcionalnost ugradila (pretpostavite da postoji funkcija u dostupnom API-ju getFileSize(FILEHANDLE fh) i da klasa File u sebi sadrži člansku varijablu FILEHANDLE koja predstavlja handle na fajl).
Napišite implementaciju svih funkcija koje ste dodali u dizajn (C#/Java/C++, sasvim svejedno, ali naznačite što ste odabrali).

Tu je component valjda apstraktna klasa, bilo bi dobro da ima apstraktnu metodu koja izračunava svoju veličinu nešto tipa size(). Folder će onda implementirat tu metodu tak da će pozivat size() nad svim komponentama u listi komponenta koje se nalaze u tom folderu. File će je implementirat tak da koristi ovaj getFileSize. Desit će se to da kad će se pozvat size() nad folderom on će pozvat size() nad svim komponentama unutar tog foldera (znači nad svim folderima i fajlama). Kod ispod:

Composite pattern - Lopina

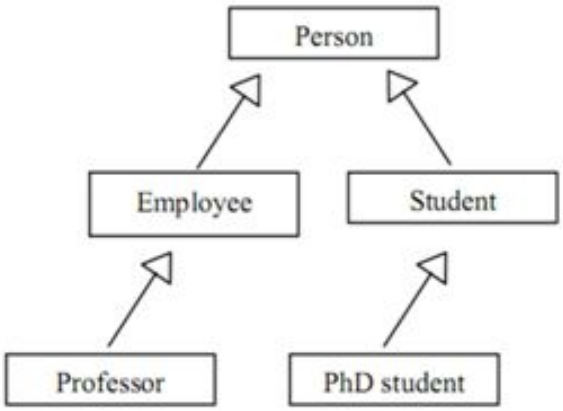
```
Component#public abstract int size();

Folder#public int size() {
	int size = 0;
	for (Component c : this.components) {
		size += c.size();
	}
	return size;
}
```

```
File#public int size() {
	return getFileSize(this.HANDLE)
}
```

ZAD3

(*10* bodova) Dana je sljedeća hijerarhija klasa:



Svaka osoba (Person) ima metodu *startWorking()* i *stopWorking()* koje se pozivaju kad osoba počinje i završava s radom (odnosno, definirane su kao virtualne u klasi Person i prekrivene u izvedenim klasama).
Potrebno je modificirati navedenu hijerarhiju kako bi se dodalo *runtime* praćenje (logiranje) rada osobe. Zahtjev je (hard constraint) da se praćenje može uključivati/isključivati tijekom izvršavanja (odnosno, ne podešava se samo na početku izvođenja) za svaku pojedinačnu klasu i to na način da se može uključiti/isključiti i za cijelu klasu i za pojedinačno instancirane objekte (ukoliko se za neki objekt desi konfliktna situacija, prednost imaju postavke za klasu!). Kada je praćenje uključeno, potrebno je u globalno dostupni log zapisivati kada je osoba počela i kada je završila s radom.
Koje bi sve klase trebalo dodati (ako ikoje!?) i kako bi ih trebalo integrirati u gore navedeni dizajn? Što bi trebalo (ako išta!?) promijeniti u implementacijama postojećih funkcija? Da li se vaše rješenje mijenja (i kako) ukoliko se zna da će se u navedenu hijerarhiju dodavati i nove klase i da je lakoća budućeg održavanja važan kriterij kvalitete rješenja?
Napomena: Pretpostavite da postoji globalno dostupna instanca (singleton) klase Logger, svkoja ima o i logStop(Person) kojima peracije logStart(Person)se obavlja konkretno zapisivanje u log.

Spomenuto na F2N, moguce rjesenje sa statickom varijablom, uz odgovarajuci mehanizam promjene i to na ovaj nacin:
Kada se zeli ukljuciti log za klasu objekata, staticka zastavica se digne na true.
Kada se zeli iskljuciti log za klasu objekata, staticka zastavica se spusti na false.

Dio koji mi nije jasan je sa onim konfliktnim situacijama, gdje kaze da u slucaju konflikta, postavke za klasu uvijek imaju prioritet. U binarnoj tablici, to bi izgledalo ovako nekako:

C	*O*	*f*
0	*0*	*0*
0	*1*	*0*
1	*0*	*1*
1	*1*	*1*

Mislim da nam ne treba Quine-McClusky da se iz ovakve definicije zadatka iscita ovaj rezultat. Iako, mozda je Zvone htio reci ovo:
Kad stisnemo prekidac za logiranje za klasu, svi objekti te klase zasvijetle na kontrolnoj ploci. Medjutim, mi mozemo pojedinacno paliti i gasiti logiranje za objekte te klase.

Primjer:
A, B, C je element Klase 1
Log Start Class 1 => Log Running A, Log Running B, Log Running C
Log Stop Object A => Log Stopped A, Log Running B, Log Running C
Log Stop Object C => Log Stopped A, Log Running B, Log Stopped C
Log Start Object A => Log Running A, Log Running B, Log Stopped C
Log Stop Class 1 => Log Stopped A, Log Stopped B, Log Stopped C

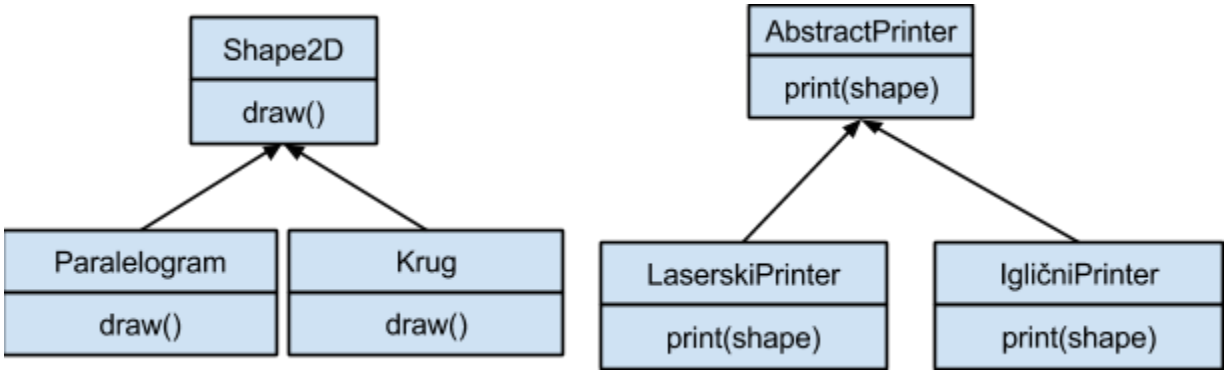
To je barem moja interpretacija - Lopina

ZAD4

(*15 bodova*) Zadana je hijerarhija dvodimenzionalnih likova (Paralelogram, Krug, Trokut) koji su svi izvedeni iz razreda Lik2D. Naknadno se pojavila potreba proširenja funkcionalnosti s mogućnošću iscrtavanje likova (metoda Print) na različitim tipovima printera: matrični, laserski, inkjet.
Naravno, algoritam za iscrtavanje npr. pravokutnika razlikuje se od algoritma za iscrtavanje kruga ali i algoritmi za iscrtavanje istog lika na različitim printerima su posve različiti. Koje bi sve klase, metode trebalo dodati da se omogući iscrtavanje zadanog lika na zadanom printeru. Nacrtajte dijagram razreda za vaše rješenje.
Da li se vaše rješenje mijenja (i kako) ukoliko se zna da će se u navedenu hijerarhiju dodavati

novi oblici i novi tipovi printera i da je lakoća budućeg održavanja važan kriterij kvalitete rješenja?

Strategy pattern
Primjerice PrinterStrategy apstraktna klasa koja sadrzi apstraktnu metodu Print koja prima objekt tipa Lik2D. Onda sve vrste printera nasljeduju PrinterStrategy i implementiraju metodu Print koristeći algoritam ispisa (koji ovisi o tipu printera) i objekt tipa Lik2D koji ima implementirano kako se iscrtati. (lik2d apstraktna klasa ima apstraktnu metodu IscrtajSe() a pojedinacni likovi je implementiraju svaki na svoj nacin). Onda preko nekog Konteksta pozivamo to sve, primjerice konstruktor kontekst klase prima instancu tipa PrinterStrategy i instancu tipa Lik2D i onda pomoću polimorfizma se na željenom printeru ispiše željeni lik.



ZAD5
(*45 bodova*) – *Sustav za planiranje i praćenje realizacije projekata*
Poslovna potreba
Poduzeće Light Speed Technologies Inc. se bavi razvojem softverskih projekata i zbog brzog razvoja i rasta, planiranje i praćenje razvojnih aktivnosti preko Excel tablica više nije dovoljno. Stoga je uočena potreba za razvojem informacijskog sustava kroz koji bi se obavljale aktivnosti planiranja i praćenja realizacije postavljenih ciljeva, kao i praćenje dnevnih aktivnosti developera, što bi uz kvalitetan sustav izvještavanja omogućilo daljni kvalitetan razvoj i rast firme.
Opis domene
Svi projekti u firmi su inicirani Nalogom za izvršenje (NZI), odnosno za sav posao koji se obavlja u firmi mora postojati izdan odgovarajući radni nalog koji je definiran nazivom, imenom klijenta

za kojeg se radi, te datumom početka i predviđenim datumom završetka. Firma je organizirana kroz više odjela s time da su Odjel za razvoj i Odjel za implementaciju ključni odjeli koje treba pokriti novim sustavom (Odjel računovodstva i Odjel nabave svoje aktivnosti obavljaju kroz ERP sustav). U svakom od tih odjela se nalazi više timova, od kojih svaki ima određeni broj članova (developera, odnosno općenitije članova tima – koji mogu biti „stariji“ (senior) ili „mlađi“ (junior) inženjeri) i jednog voditelja (Team leadera).

Planiranje se radi na tjednoj, mjesečnoj i kvartalnoj bazi definiranjem odgovarajućih ciljeva. Za svaki cilj se u sustav unosi kratki naziv, opis te jedan ili više članova tima odgovornih za realizaciju a također se za ciljeve definiraju i procjene – i to procjene utroška vremena, te procjene materijalnih troškova. Svaka vrsta cilja ima definiran i prioritet (low, medium ili high). U sustavu mora postojati mogućnost za hijerarhijsku organizacija ciljeva što podrazumijeva da se mjesečni ciljevi povezuju (odnosno „pripadaju pod“) kvartalni cilj, a tjedni ciljevi pripadaju pod mjesečni cilj. Kvartalni ciljevi su na vrhu hijerarhije i oni se vezuju direktno na NZI. No, mora postojati i mogućnost vezivanja tjednih i mjesečnih ciljeva direktno na NZI (odnosno, da se NE vežu na viši cilj u hijerarhiji).

Pored planiranja, koje se obavlja kroz definiranje ciljeva, prati se i vrijeme potrošeno na njihovu realizacija na način da se zapisuju dnevne aktivnosti developera i povezuju s ciljevima za koje su odrađene. Svaki developer može tijekom dana unijeti proizvoljan broj odrađenih aktivnosti. Za svaku aktivnost se unosi njen kratki naziv, detaljan opis, potrošeno vrijeme te cilj za koji je aktivnost realizirana. Aktivnost se može povezati bilo s tjednim, mjesečnim ili kvartalnim ciljem, bilo direktno na NZI (npr. ukoliko je „uletjelo“ nešto iz prošlosti a što nije bilo planirano).

Konkretna realizacija svih ciljeva se prati na način da se svaki cilj može označiti bilo kao 25%, 50%, 75% ili 100% finished. Markiranje mjere završenosti ciljeva odrađuju oni koji su za cilj zaduženi a bitno je pri tome zapisati i timestamp kada je cilj markiran, kako bi se kasnije moglo]napraviti graf napretka realizacije ciljeva. Cilj se može označiti i kao „Not finished“ u kojem slučaju se mora moći unijeti i bilješka s opisom razloga zašto cilj nije završen.

Napomene:

- Da li će aplikacija biti realizirana kao klijent-server ili web aplikacija za vas i vaše zadatke nije (odnosno ne bi trebalo biti) bitno
- Autorizacijsko-security aspekte informacijskog sustava možete zanemariti
- Možete zanemariti i sustav izvještavanja jer se on bazira na samoj domeni, odnosno ne doprinosi složenosti domene već samo iskorištava definirane podatke

Vaš zadatak je:

a) (*10 bodova*) Identificirati 5-6 **glavnih** use case-ova (znači, navesti ih imenom - uz kratko objašnjenje, ako iz samog imena nije sve jasno)
b) (*10 bodova*) Detaljno razraditi dva osnovna use case-a (obavezno s alternativnim scenarijima!)
c) (*25 bodova*) Izraditi model domene bazirano na danom design story-u koji će biti osnova programskog rješenja

Unos radnog naloga

1. Korisnik odabire...

2. Korisnik unosi naziv radnog naloga, odabire klijenta na kojeg se radni nalog odnosi, unosi datum početka i predviđeni datum završetka

Unos ciljeva

1. Korisnik odabire...
2. Korisnik unosi naziv i opis cilja
3. Korisnik odabire članove tima odgovorne za realizaciju cilja
4. Korisnik unosi procjenu vremena izvršavanja cilja
5. Korisnik unosi procjenu materijalih troškova potrebih za izvršavanje cilja
6. Korisnik odabire prioritet izvršavanja cilja
7. Korisnik odabire unutar koje skupine ciljeva spada novi cilj
8. Korisnik odabire hijerarhijsku viši cilj unutar kojeg novi cilj spada

Unos aktivnosti:

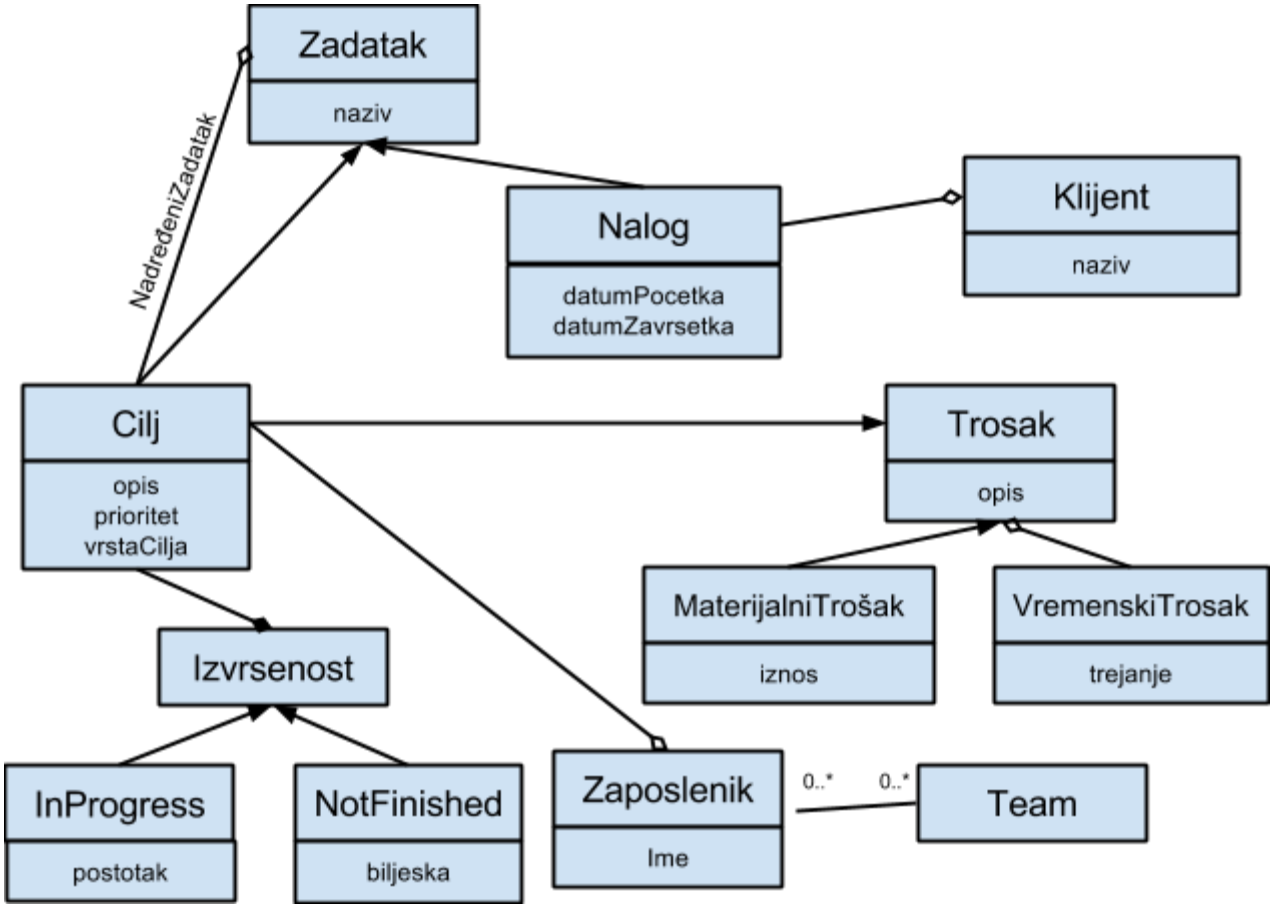
1. Korisnik odabire...
2. Korisnik odabire developera koji je odradio aktivnost
3. Korisnik odabire cilj za na koji se odnosi aktivnosti
4. Korisnik unosi naziv aktivnosti, detaljan opis aktivnosti i vrijeme utrošeno za izvršavanje aktivnosti

Markiranje izvršenosti cilja

1. Korisnik odabire
2. Korisnik odabire postotak izvršenosti cilja
3. Korsisnik odabire datum kada je došlo do promjene izvršenosti

2.a Korisnik je odabrao da cilj nije izvšen
1. Korisnik unosi unosi bilješku sa opisom zašto cilj nije izvršen

CRUD za developere



ovdje fale aktivnost, vrste ciljeva i odjeli

ZAD6

(*20 bodova*) U poduzeću „Pero i sinovi d.o.o“ su odlučiti implementirati i uvesti informacijski sustav za praćenje „imovine“ (eng. Asset – i u ovom kontekstu se misli na širi pojam imovine poduzeća u smislu svega onoga u što firma ulaže svoje resurse). S obzirom da je poduzeće orijentirano na visokotehnološku SW i HW proizvodnju, glavni Asseti firme su SW i HW proizvodi, a esencijalno važan aspekt cijele priče je i Release Management, odnosno praćenje različitih izdanih verzija SW i HW proizvoda.

Svaki Asset ima svoje ime, id, i mora se znati popis svih izdanih verzija Asseta (verzije su opisane kasnije!). Pored toga, za svaki Asset je definiran i odgovarajući tip (Asset Type) kojim je opisana vrsta Asseta. Asset Type ima svoj naziv i definiran odgovarajući skup atributa kojima su definirane vrijednosti relevantne za takav tip Asseta. Atributi (Asset Attribute) opisuju određena

svojstva vrste Asseta na način da svaki atribut ima definiran svoj naziv i vrstu pripadne vrijednosti (Asset Attribute Type). Predviđeno je da u sustavu postoji ograničen broj tipova atributa koji se mogu koristiti za opis Asseta i oni obuhvaćaju: value (broj), string, link (http) i document (PDF ili DOC format). ENUM?
Npr. za Asset Type „SW komponenta“, nekoliko tipičnih atributa bi bilo „Release directory“ (tip atributa string), „SVN source code path“ (tip atributa link), dok bi za Asset Type „SPZ znak“ tipični atributi bili „luminoznost“ (value), „potrošnja struje“(value) i sl.
Za svaki Asset se prati popis izdanih verzija (Asset Version), i za svaku verziju Asseta je definirano: oznaka verzije, datum izdavanja verzije, kratki opis promjena te popis (konkretnih) vrijednosti atributa definiranih za takvu vrstu Asseta. Kako se ne bi prilikom svakog izdavanja nove verzije morale nanovo unositi sve potrebne vrijednosti atributa, potrebno je omogućiti da se za svaki pojedinačni Asset mogu definirati default vrijednosti tih atributa, a koje se prilikom izdavanja nove verzije mogu editirati. Svako izdavanje verzije Asseta se mora planirati unaprijed na način da se odredi planirani datum izdavanja Asseta, te definira pripadni broj verzije izdanog Asseta.
ZADATAK: Nacrtajte detaljni dijagram klasa, s potrebnim pojašnjenjima, kojim bi realizirali definirani model domene.

rješenje ovdje...

http://materijali.fer2.net/File.8119.aspx

ZAD7 - (Baze podataka i XML mapiranje nismo radili)
(*25 bodova*) Dana vam je hijerarhija razreda kojom su modelirane različite vrste dokumenata u nekom uredu. Vaš zadatak kao „Chief Architecta“ :-) je opisati realizaciju O/R mapiranja uz korištenje NHibernate-a.
Opis:
Bazni razred hijerarhije je klasa *Document*, sa svojstvima *DocID* (koji predstavlja urudžbeni broj – ie. interni identifikator u uredu), *SlucajID* (referenca/strani ključ na instancu slučaja uz koji je dokument vezan), *Autor* i *DatumKreiranja*.
Iz tog razreda su izvedeni razredi koji predstavljaju konkretne vrste dokumenata, a specifični su po svojim svojstvima i poslovnim procesima u kojima sudjeluju.
Nekoliko važnih arhitekturnih napomena:

- Očekuje se da će se tijekom života sustava dodavati još različitih vrsta dokumenata
- Važna je mogućnost polimorfnih upita
- Izvedeni dokumenti imaju bogati skup specifičnih svojstava (članskih varijabli!)

Zadatak:

1) Specificirajte i argumentirajte na koji način bi obavili mapiranje hijerarhije klasa u tablice u relacijskoj bazi podataka
2) Uzevši kao primjer dva konkretna dokumenta *Narudžbenicu* i *Fakturu* (za svaki definirajte nekoliko svojstava po volji), napišite kako bi izgledalo mapiranje korištenjem XML-a

rješenje ovdje..

ZAD8 - (MVP Pattern nismo radili)

(25 *bodova*) U ovom zadatku potrebno je opisati konkretnu realizaciju MVP patterna na primjeru Transakcije iz case study-a FinAssist koji smo razrađivali na predavanjima.

Opis konteksta:

· Postoji klasa *Transaction* koja modelira jednu transakciju (sa svojstvima: datum, iznos, opis – zanemarujemo kategorizaciju i sve ostale stvari radi jednostavnosti).

· Postoji *TransactionRepository* (sa standardnim sučeljem: *Add*(), *Update*(), *Remove*(), a pretpostavite da ima i funkciju *GetLastNTransactions*(int N)) koji predstavlja standardni repozitorij transakcija.

· Pregled svih transakcija (View) je realiziran preko Windows Forme koja ima *ListView* kontrolu za prikaz 50 zadnjih transakcija (s kolonama za prikaz odgovarajućih svojstava) te button *Add transaction* kojim se inicira dodavanje nove transakcije.

· U glavnoj formi postoji izbornik u kojem je kroz opciju *View transactions* dostupna funkcionalnost za pregled transakcija.

Zadatak:

Opišite kako bi korištenjem MVP pattern-a realizirali funkcionalnost za pregled transakcija.

Stvari o kojima trebate razmisliti (i koje morate imati u svom rješenju):

1) Koje dodatne objekte bi trebalo definirati, kakvu funkcionalnost bi trebali imati i kako bi trebali biti povezani s ostalim objektima u sustavu
2) Detaljno napišite kako bi u vašem rješenju izgledao kôd u event-handleru za opciju View transactions u glavnoj formi

Središnji dio vašeg rješenja mora biti uredno napravljen (i konciznim komentarima pojašnjen) class diagram u kojem će biti nacrtane sve klase i odnosi među njima, a napišite i relevantne (znači važne) odsječke programskog kôda (ali, ako imate u svom rješenju više od 20-ak LOC, previše detaljizirate :-).

Bitna napomena: prozor View transactions je modeless dijalog, što znači da može ostati otvoren

u pozadini! (a u aplikaciji postoje i drugi načini za dodavanje nove transakcije, pored buttona *Add transaction* na formi).

rješenje ovdje ...
