Namjena UML-a

UML je grafički jezik za:

- Vizuelizaciju
- Specifikaciju
- Konstruisanje
- Modelovanje

UML može da posluži u modelovanju konkretnih stvari kao što su baze podataka, tipovi podataka, klase podataka, kao i konceptualnih stvari kao što su proces poslovanja i funkcije sistema. UML nije programski jezik ali se njegovi rezultati mogu preslikati u programske jezike kao što su C++, Java, Visual Basic.

Blokovi za izgradju UML-a

Blokove za izgradju UML-a dijelimo na:

- Opšta sredstva
- Relacije (odnose)
- Dijagrame

Relacije spajaju opšta sredstva dok dijagrami grupišu opšta sredstva.

1. Opšta sredstva UML-a

Opšta sredstva UML-a se dijele na:

- Strukturna opšta sredstva
- Opšta sredstva za opis ponašanja
- Grupišuća opšta sredstva
- Anotaciona opšta sredstva

1.1 Strukturna opšta sredstva predstavljaju statički (nepromjenljivi) dio koji uključuje kako konceptualne tako i fizičke elemente modela. Ima ih ukupno sedam. Osnovni strukturni elementi su klasa, interfejs, društvo saradnika, korisnička funkcija, aktivna klasa, komponenta, čvor.

Klasa je opis skupa objekata koji imaju iste atribute i operacije, veze i semantiku.

Interfejs je kolekcija opisa operacija koje može da izvrši (usluga koje može da pruži) klasa. To je skup poruka koji se može poslati klasi, namenjen je korisnicima klase i ne uključuje implementaciju tih operacija.

Društvo saradnika definiše kooperaciju pojedinih djelova sistema. Jedna klasa može pripadati više društava saradnika.

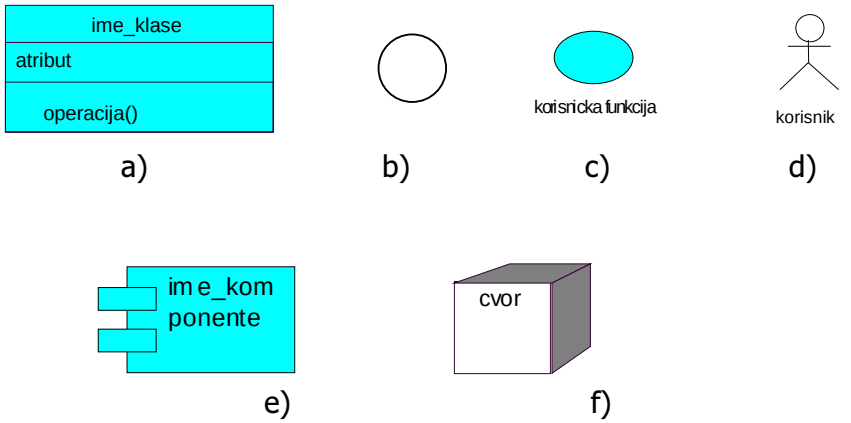
Korisnička funkcija (use case) prikazuje jednu funkciju sistema kako je vidi spoljni korisnik (actor). To je opis skupa akcija koje sistem izvršava da bi proizveo ponašanje koje želi specifični korisnik. Korisnička fnkcija, dakle, služi da struktuira ponašanje u modelu.

Aktivna klasa je strukturno opšte sredstvo tipa klase koje može inicijalitzovati upravljačku aktivnost. Grafički se predstavlja kao i klasa ali sa podebljanim linijama.

Komponenta je fizički dio sistema koji je saglasan sa nekim skupom interfejsa i realizuje ga. Komponenta može biti izvorni ili izvršni program. Komponente mogu da se grupišu u pakete.

Čvor je fizički element koji postoji u vrijeme izvršavanja i predstavlja računarski resurs, sistem ili uređaj. Skup komponenti može da bude smešten na jednom čvoru a može i da se premesti sa jednog na drugi čvor.

Grafička prezentacija strukturnih elemenata UML-a data je na slici 1.

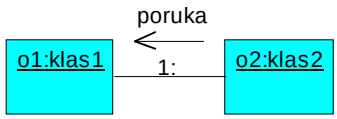


Slika 1: a) klasa b) interfejs c) korisnička funkcija d) korisnik
e) komponenta f) čvor

1.2. Elementi ponašanja

Ovaj skup elemenata predstavlja dinamički dio UML modela. To su, prije svega, interakcija i konačni automat.

Interakcija je ponašanje koje uključuje skup poruka koje se razmjenjuju među objektima u specifičnom kontekstu. Grafički, poruka se predstavlja usmjerenom linijom duž linka između objekata (slika 2).

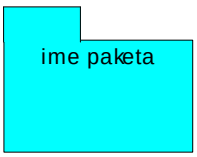


Slika 2: poruke među objektima

Konačni automat je ponašanje koje opisuje niz stanja kroz koja prolazi objekat ili interakcija u toku svog života. Stanje se grafički predstavlja zaobljenim pravougaonikom.

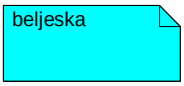
1.3. Grupišuća opšta sredstva

Ovi elementi predstavljaju organizacioni dio UML modela, tj. dekompoziciju modela, i uključuju koncept paketa. U paket se mogu smjestiti strukturni elementi, elementi ponašanja, pa i drugi grupišući elementi - paketi. Paketi postoje samo u vrijeme razvoja sistema i predstavljaju, za razliku od komponenti, konceptualni element.



1.4. Elementi označavanja

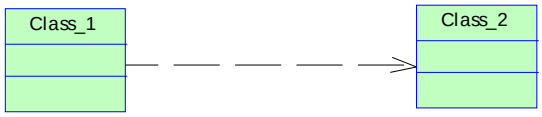
Elementi označavanja odnose se na dio UML modela za objašnjenja. To su komentari koji opisuju, rasvetljavaju, i uvode napomene i ograničenja o elementima modela. Osnovni element označavanja je bilješka koja se pridružuje elementu ili kolekciji elemenata.



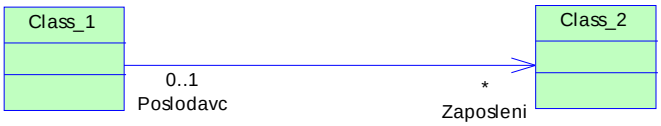
1.5. Relacije

U UML-u ima tri osnovne vrste elemenata povezivanja: zavisnost, asocijacija i generalizacija.

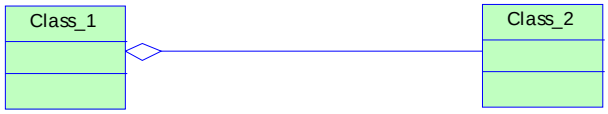
Zavisnost je semantička relacija između dva elementa u kojoj promjena jednog (nezavisnog) elementa može da utiče na semantiku drugog (zavisnog) elementa.



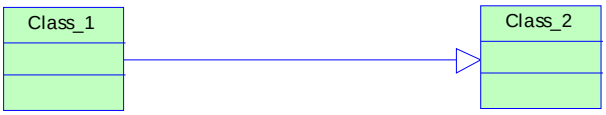
Asocijacija je strukturna veza koja opisuje skup linkova (veza između objekata). Posebna vrsta asocijacije je agregacija, koja predstavlja strukturnu vezu cjeline i njenih djelova. Asocijacija može da bude usmjerena, može da uključi ime, uloge elemenata koje povezuje i kardinalnost veze.



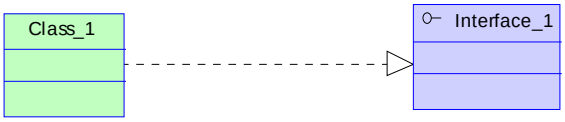
Poseban slučaj asocijacije je **agregacija**, koja predstavlja strukturnu vezu cjeline i njenih djelova.



Generalizacija je veza posebno/opšte u kojoj objekti elementa koji se specijalizuje (posebno) mogu u svakom trenutku zamijeniti objekte elementa koji je njihovo uopštenje. Ova veza može da se uspostavi među raznim elementima modela - npr. među klasama, korisnicima, itd.



Realizacija je semantička relacija između klasifikatora, gdje jedan klasifikator specifikuje ugovor čije izvršavanje garantuje drugi klasifikator. Realizacija se javlja u dva slučaja: kao veza između interfejsa i klase ili komponente koja ga realizuje; između korisničke funkcije i društva saradnika koji nju realizuje.



2. Dijagrami

Dijagrami su grafičke strukture koje opisuju pojedine djelove ili aspekte sistema koristeći grafički prikaz elemenata UML modela i obično se pridružuju jednom pogledu. Pogledi predstavljaju različite aspekte sistema koji se modelira. U UML-u postoji devet vrsta dijagrama:

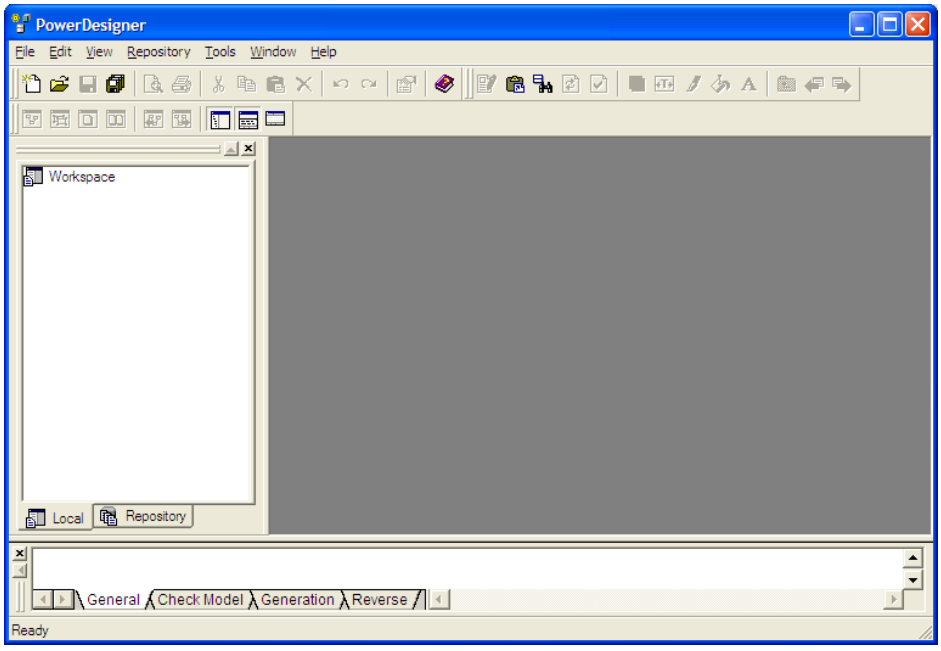
1. Dijagram klasa
2. Dijagram objekata
3. Dijagram korisničkih funkcija
4. Dijagram sekvenci (redosleda)
5. Dijagram stanja
6. Dijagram aktivnosti
7. Dijagram komponenti
8. Dijagram društva saradnika
9. Dijagram implementacije

Svaki pogled na sistem koristi više vrsta dijagrama za prikazivanje svog sadržaja a, sa druge strane, jedna vrsta dijagrama može da se koristi za prikazivanje djelova modela u raznim pogledima na sistem.

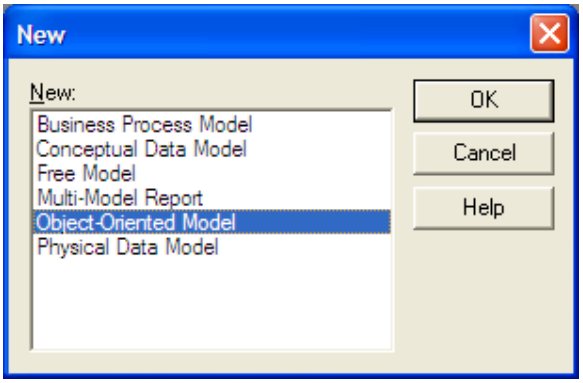
Za kreiranje ovih dijagrama koristiće se program PowerDesigner 9.5 firme Sybase.

PowerDesigner služi za objektno orijentisano modelovanje (OOM). Pored Power Designera često se za modelovanje koriste i programi Erwin i Rational Rose.

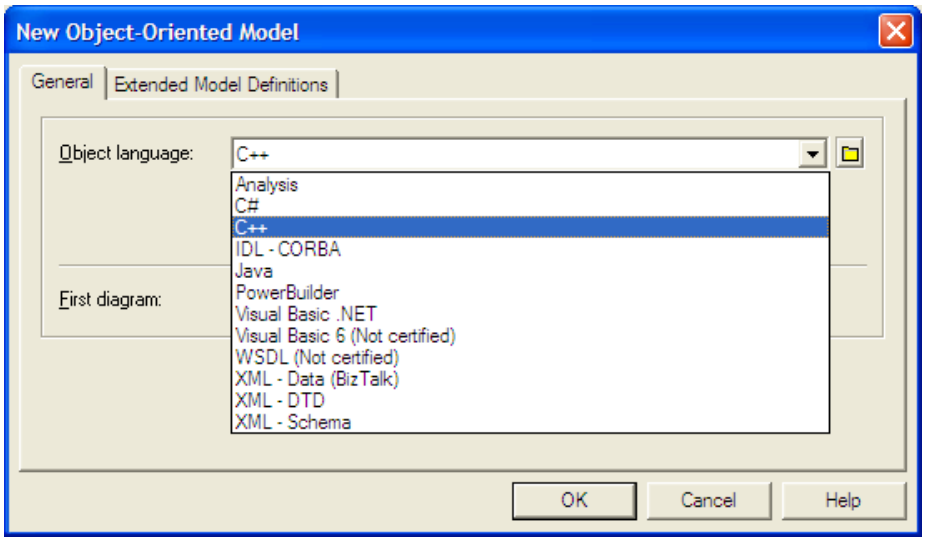
Pokretanjem programa dobija se uvodni ekran:



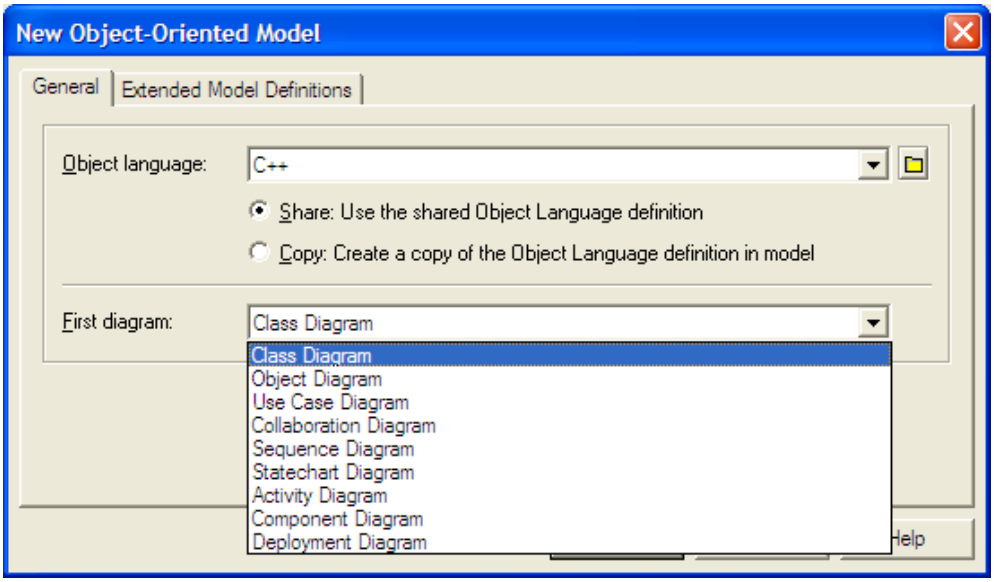
Otvaranje novog modela vrši se sa **File/New** i biramo **Object-Oriented Model**



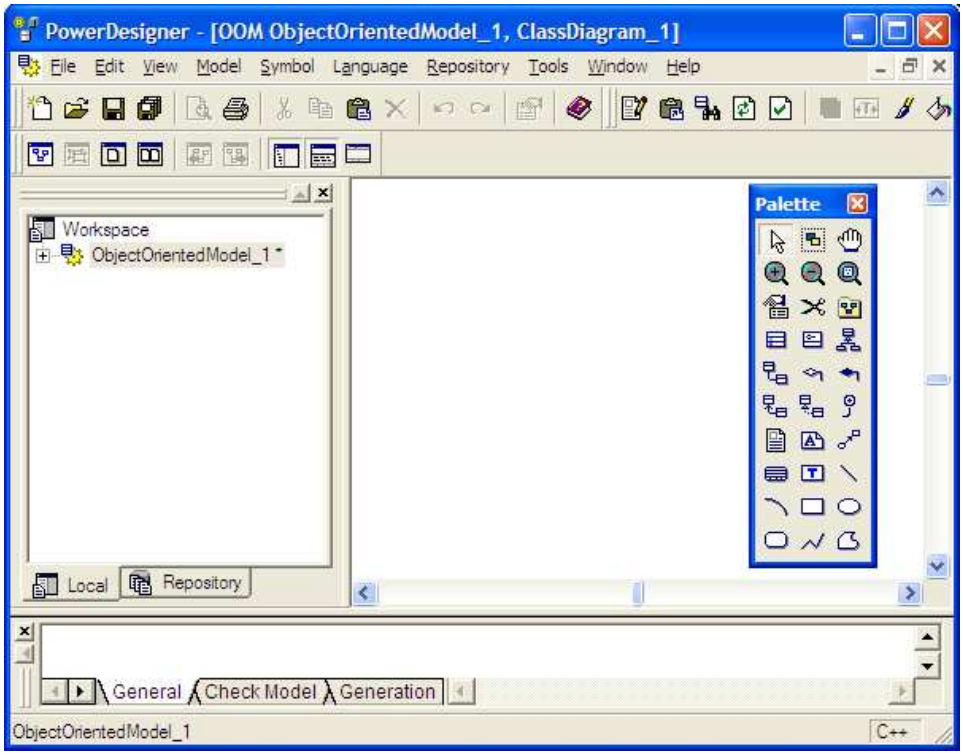
U polju Object language: bira se programski jezik u kojem se želi dobiti kod.



a u polju Frst Diagram: bira se jedan od ponudjenih 9 tipova dijagrama.



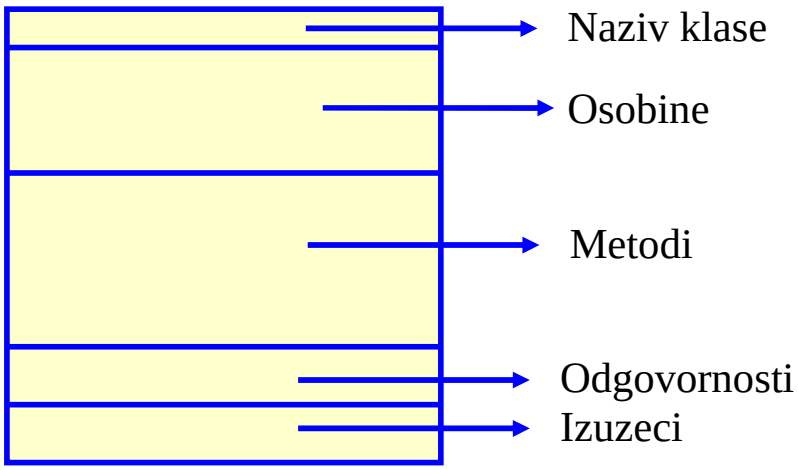
Ako se odabere **Class Diagram** dobija se sljedeća radna površina

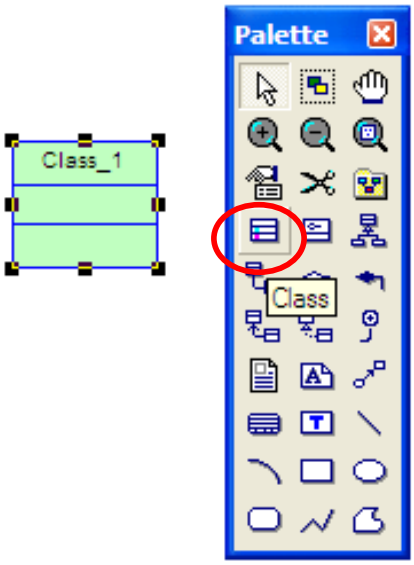


Bitni elementi klasnog dijagrama iz palete sa elementima, prikazani su u tabeli.

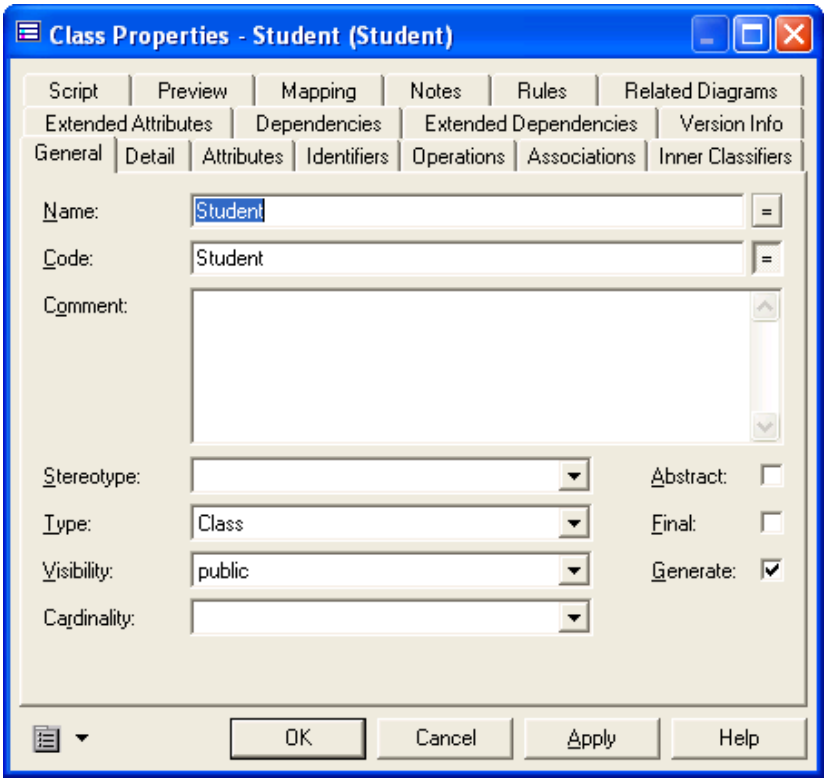
Ime	Opis
Class	**Klasa**
Inerface	**Interfejs**
Generalization	**Generalizacija**
Association	**Asocijacija**
Aggregation	**Agregacija**
Composition	**Kompozicija**
Dependency	**Zavisnost**
Realization	**Realizacija**
Note	**Komentar** je anotaciono opšte sredstvo
Title	**Naslov** unosi naziv autora projekta, tip dijagrama i vrstu modela.
Text	Unos teksta (pojašnjenja i sl.)

2.1. Realizacije klasa



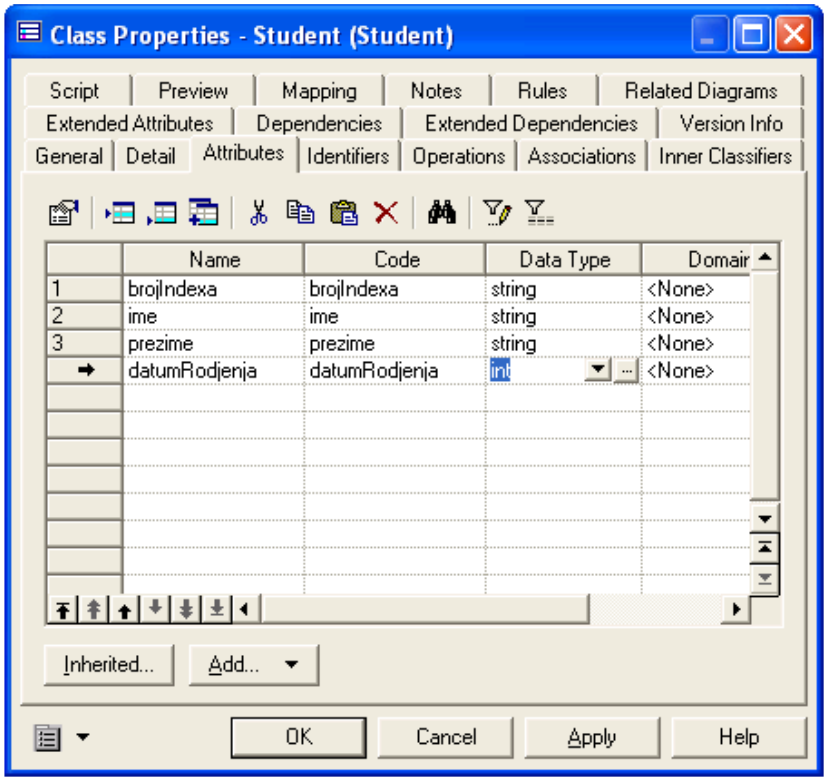


Sa palete se izabere simbol **Class** i lijevim klikom na radnu površinu dobija se **Class_1**. Desni klik miša pa **properties** dobija se meni kao na slici gdje se odredjuju atributi klase, operacije itd.

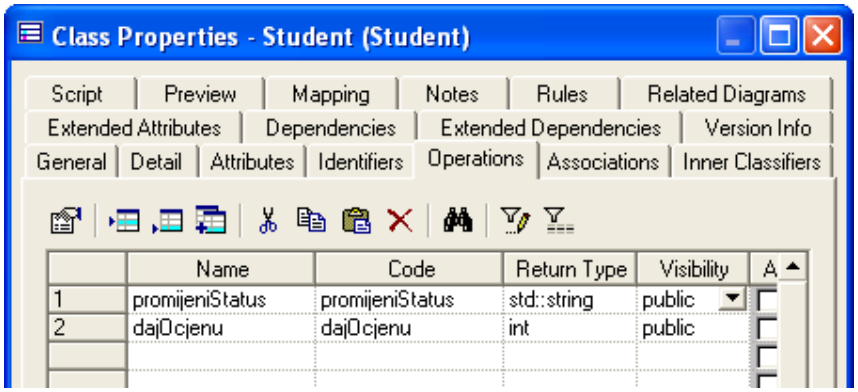


Kao primjer kako se pravi klasa biće objašnjeno na klasi Student.

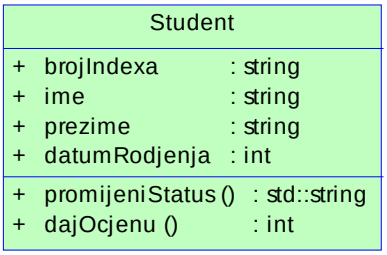
Kad smo u kartici General dali klasi ime, prelazimo na karticu Attributes gdje uisujemo atribute klase i odedjujemo tip tih atributa.



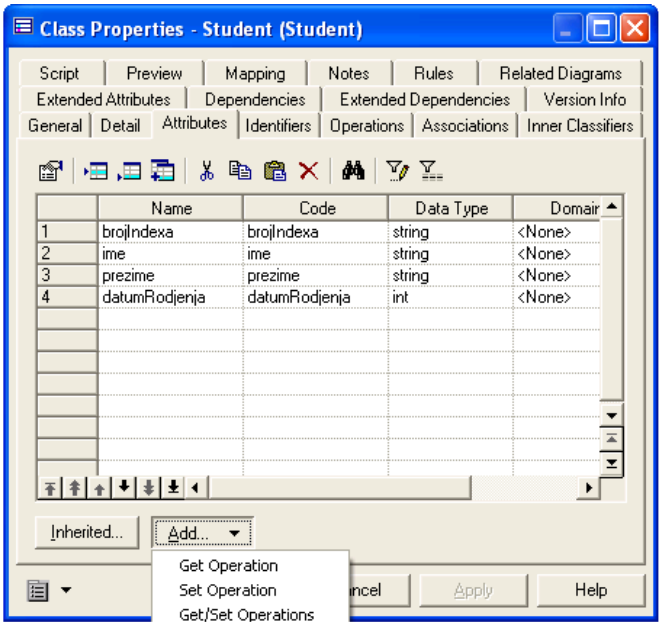
U katrici Operations osmislili smo da se upisuju ocjene i da student može promijeniti status tokom studiranja (budžet, samofinansirajući, stipendista...)



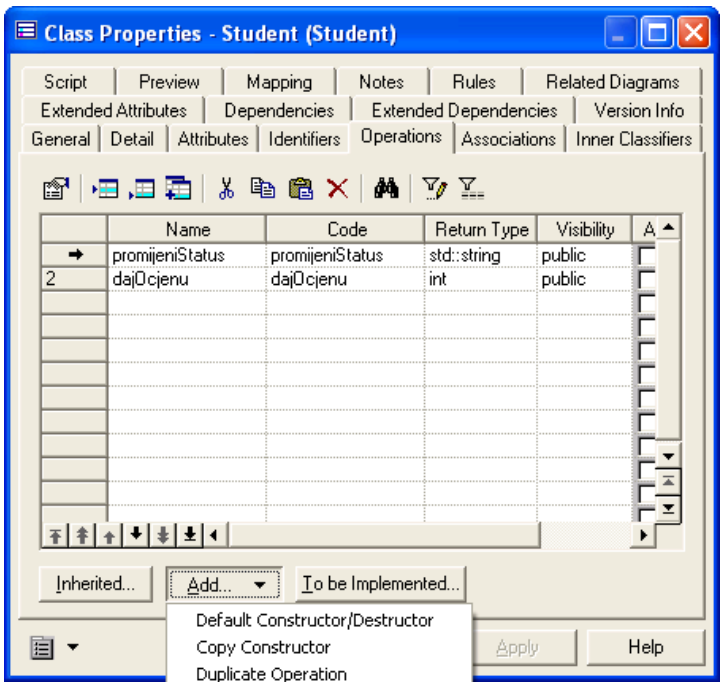
Kad smo ovo odradili dobijamo klasu Student kao na slici



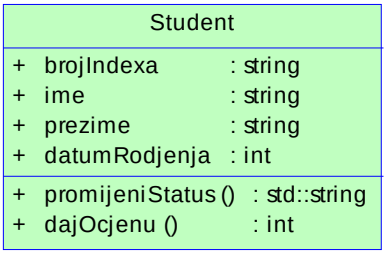
Još jedan primjer klase kod koje ćemo uključiti Get/Set operacije. One nam obezbjedjuju enkapsulaciju (učaurivanje). To je u terminologiji objekto orijentisanog programiranja, zaštita objekta od pristupa neke druge rutine (vrlo značajno za timski rad gdje svaki programer odradjuje svoj dio posla). Na ovaj način programer ne mora direktno pristupati podatku na objektu. Na donjoj slici se vidi da se Get/Set dodaju tako što se u kartici Attributes sa Add dodaju. Mogu se selektovati svi atributi ili samo neki od njih.



Konstructor/Destructor se dodaje sa kartice Operations. Konstructor obezbjedjuje prostor u RAM memoriji da se kreira klasa i njeni atributi i rezerviše slobodan prostor za te podatke. Tako npr. za neki podatak integer tipa rezerviše se 8 bita.



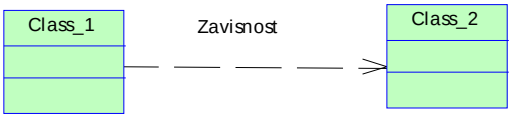
Objektno orijentisano programiranje dopušta dva tipa metoda: statičke i dinamičke, ali suština je upravo u korišćenju virtualnih metoda (dinamičkih). Tabela virtualnih metoda (TVM) je tabela adresa koja ukazuje na virtualne metode u okviru objekta. Struktura TVM počinje sa dvije riječi. Prva riječ sadrži veličinu objekta izraženu u bajtima, čija adresa se smješta na TVM. Naredna riječ sadrži negativnu vrijednost ove veličine. Pri inicijalizaciji virtuelne metode, vrši se provjera da li je ovaj zbir nula. Ako nije nula, generiše se kod greške u trenutku izvodjenja programa. Objekat koji sadrži virtualnu metodu, mora da se inicira Constructor metodom. Ova metoda formira i postavlja početne vrijednosti u TVM. Destructor koristi TVM da odredi potrebne parametre za brisanje objekata i time oslobadja rezervisanu memoriju.



UBACITI ovu sliku sa GET SET I KONSTRUKTOR DESTRUKTOR

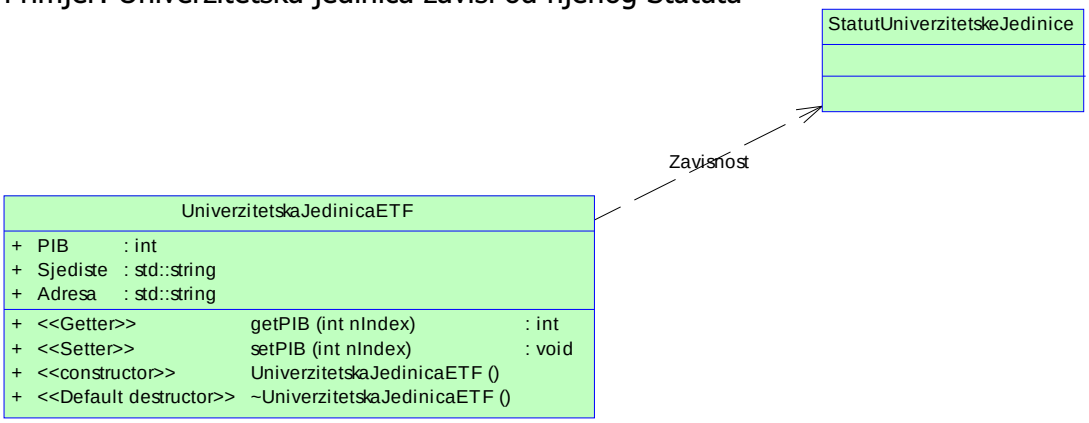
2.2. Realizacije relacija

a) Zavisnost; uopšteno:



(Zavisnost je relacija između dva opšta sredstva za koje važi da promjena jednog (nezavisnog) opšteg sredstva može uticati na semantiku drugog (zavisnog) opšteg sredstva)

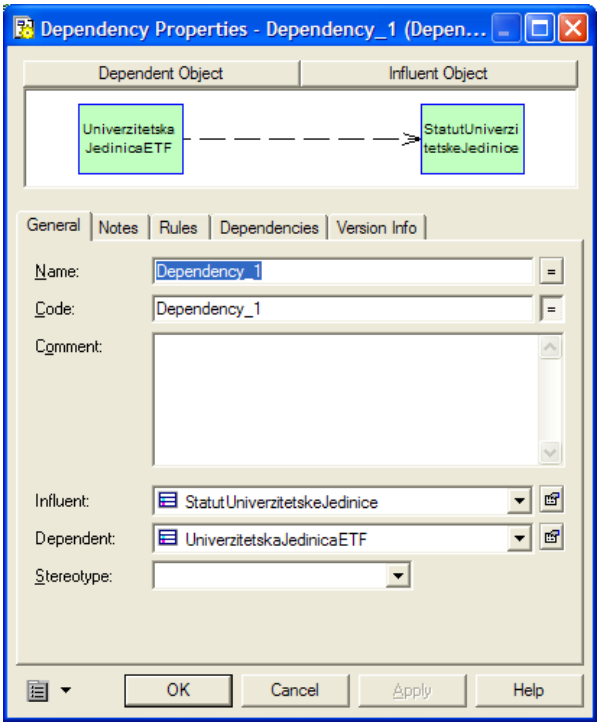
Primjer: Univerzitetska jedinica zavisi od njenog Statuta



Da bi napravili vezu zavisnosti, iz Palette izaberemo vezu zavisnosti (dependency)

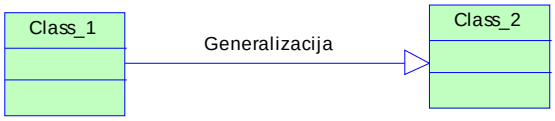
i kliknemo lijevim tasterom miša na jednu klasu i prevučemo do druge klase i pustimo lijevi taster miša i veza je kreirana.

Desni klik na vezu zavisnosti **/properties** dobija se sljedeći prozor



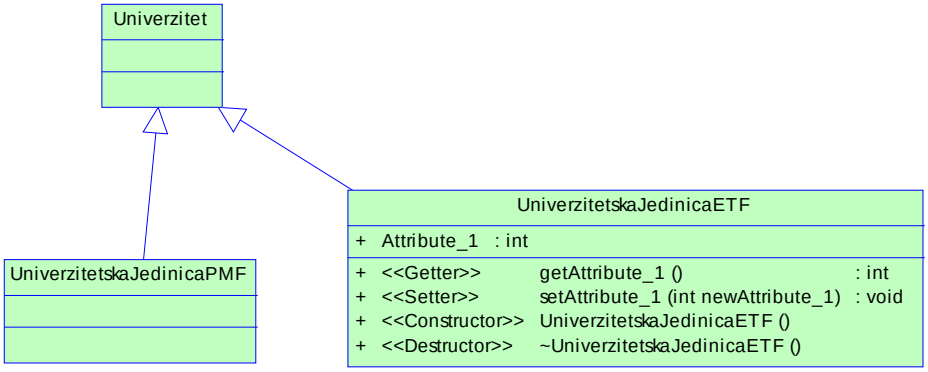
i u polju Name: upišemo kako želimo da nam se zove veza zavisnosti.

b) Generalizacija; uopšteno :

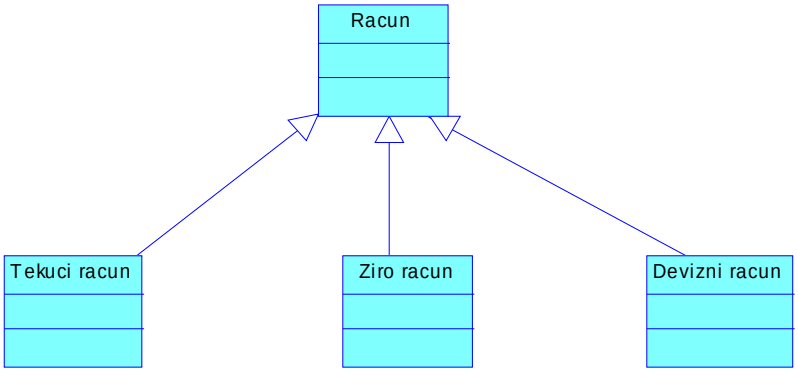


(Veza između opšteg sredstva roditelja i opšteg sredstva djeteta kod koje dijete nasljeđuje sve osobine roditelja i ima neke specifičnosti).

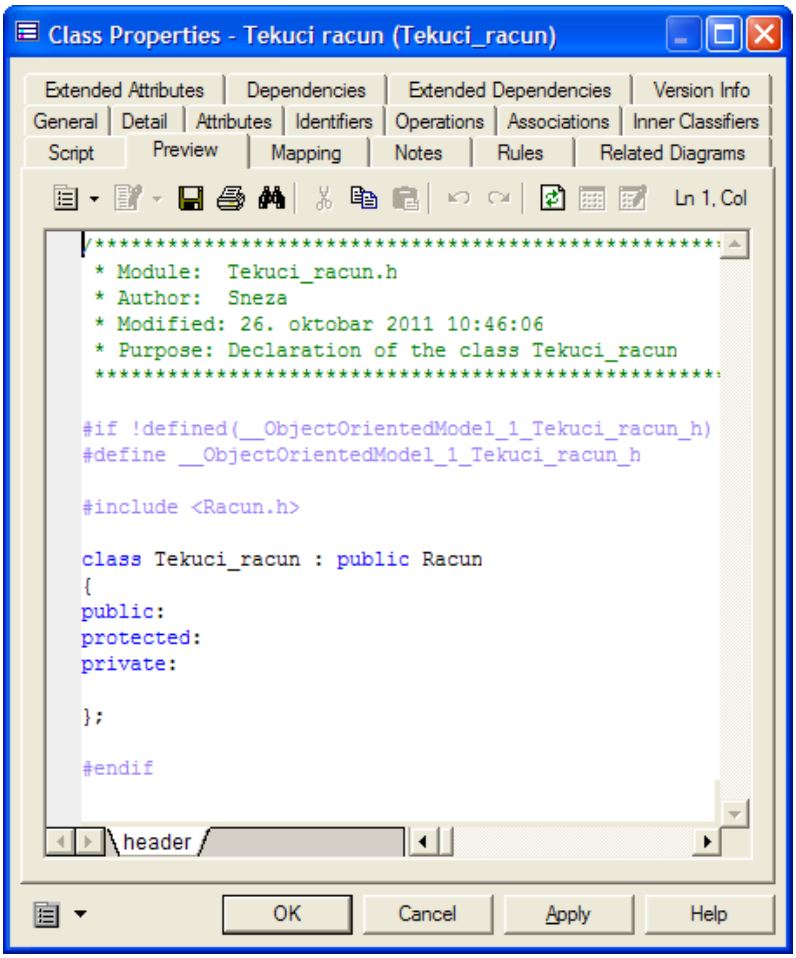
Primjer:



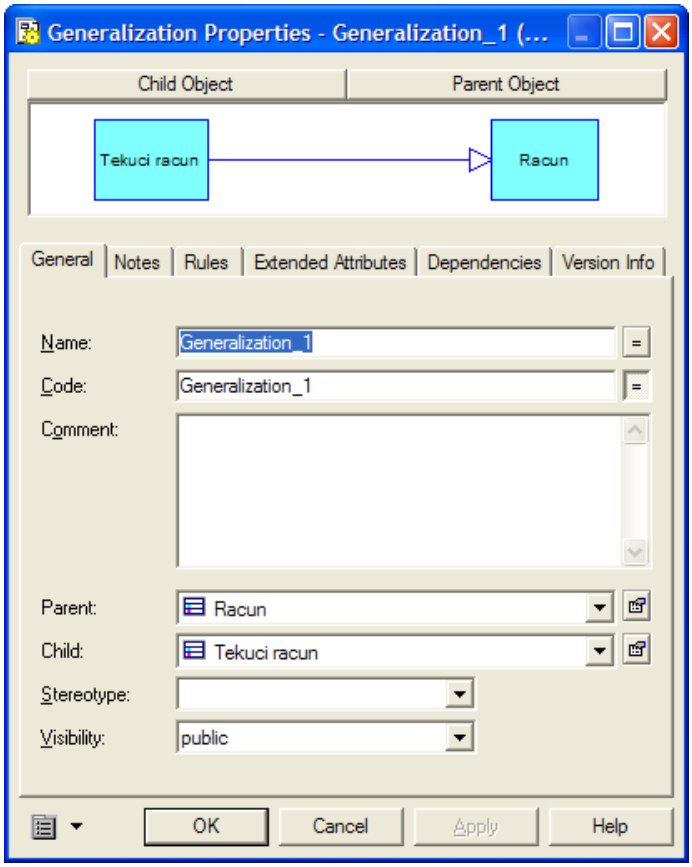
Pregledanjem koda (npr u C++) može se provjeriti da klasa "child" nasljedjuje osobine klase "parent"



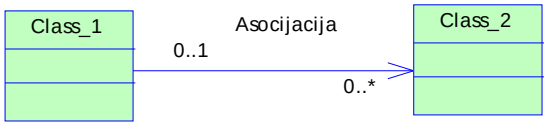
U kartici Preview na klasi TEKUĆI RAČUN vidi se da se koristi i klasa RAČUN



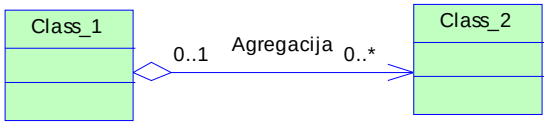
Desni klik na vezu generalizacije **/properties** dobija se sljedeći prozor i ovdje se može dati ime generalizaciji.



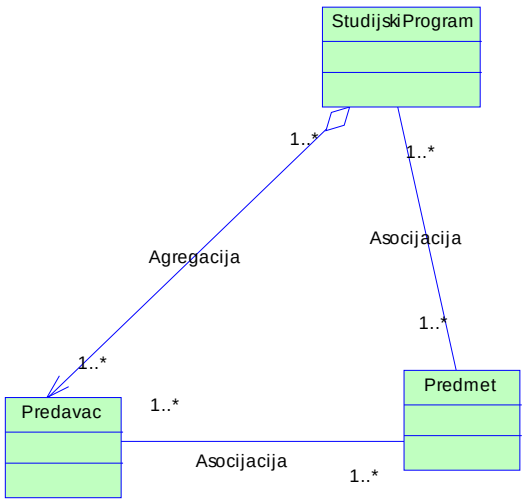
c) Asocijacija; uopšteno :



agregacija

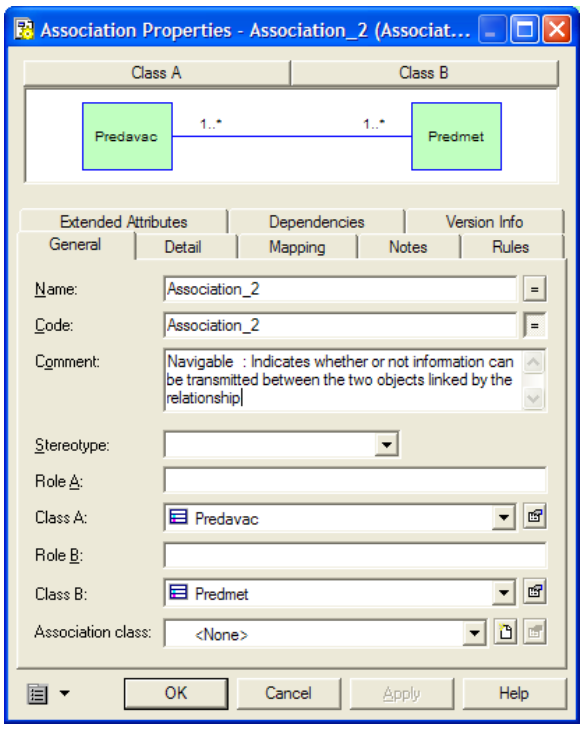


Primjer:



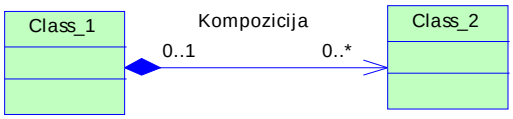
Agregacija je specijalan sličaj asocijacije.

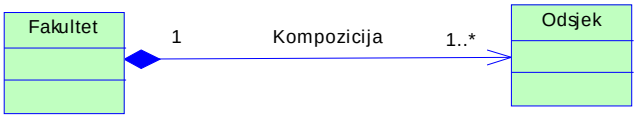
Desni klik na vezu asocijacije izmedju Predavača i Predmeta **/properties** dobija se sljedeći prozor



Desni klik na vezu agregacije izmedju Predavača i Studijskog programa **/properties** dobija se sljedeći prozor

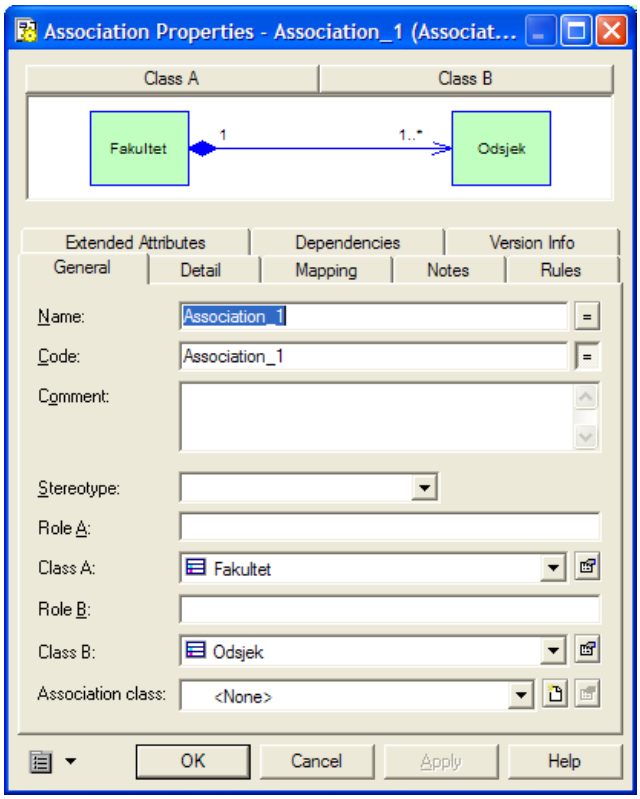
d) Kompozicija; uopšteno :



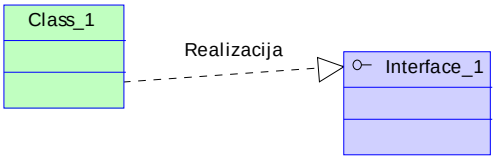


Svaki fakultet je sačinjen od jednog ili više odsjeka, a svaki odsjek pripada samo jednom fakultetu.

Desni klik na vezu kompozicije **/properties** dobija se sljedeći prozor

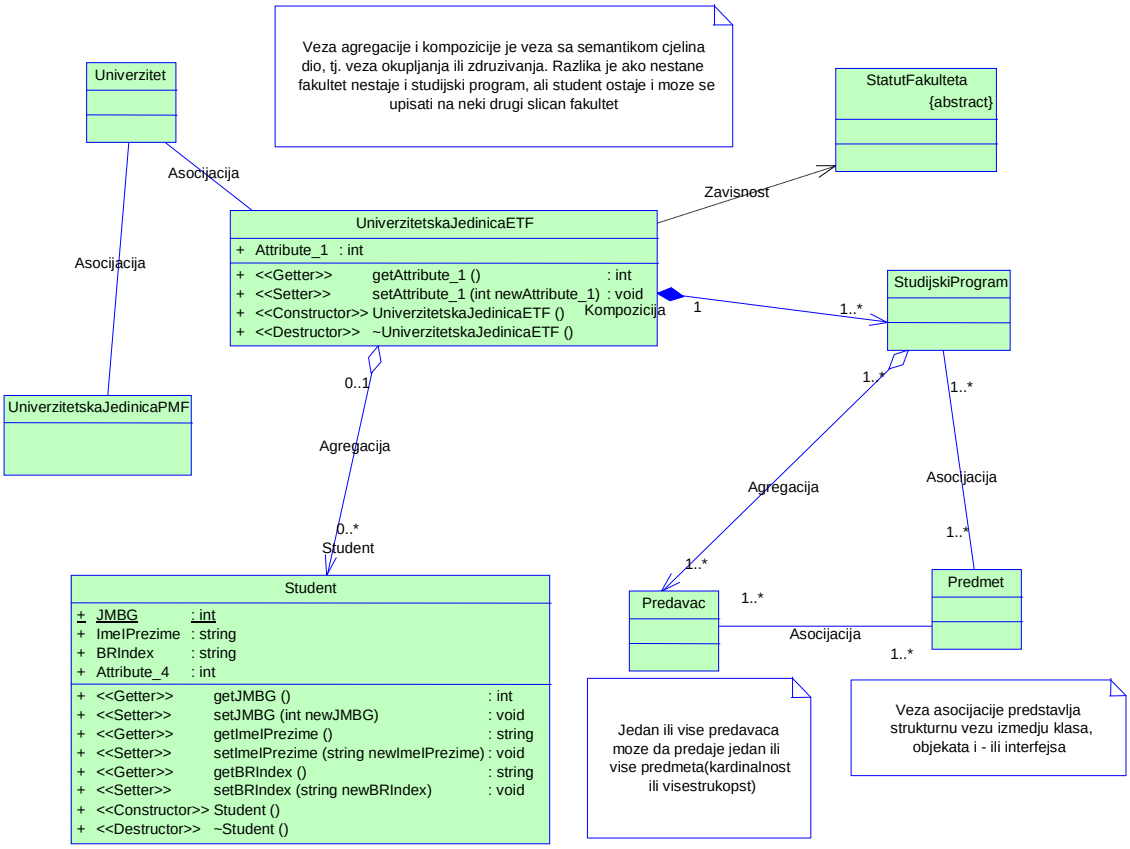


d) Realizacija:



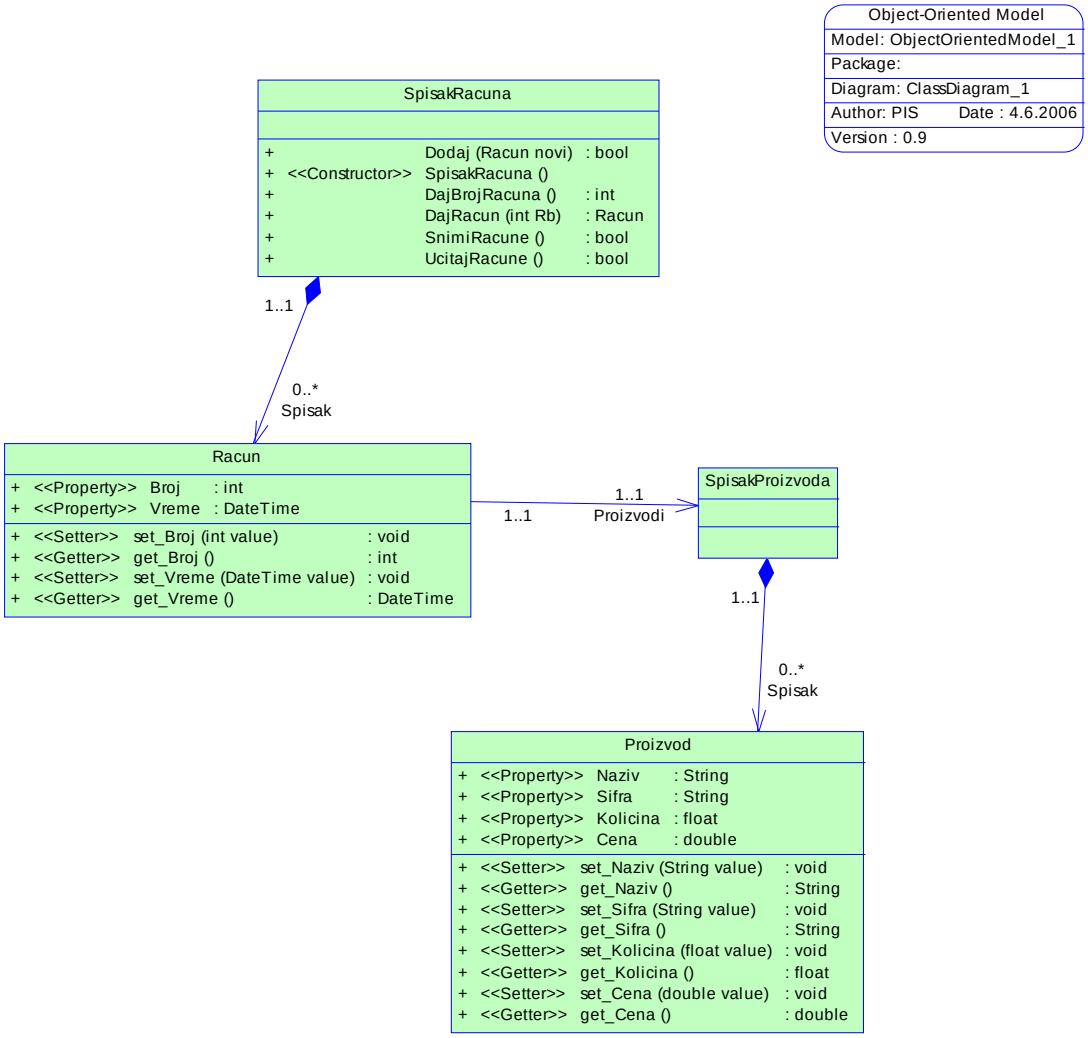
2.3. Dijagram klasa prikazuje statičku strukturu klasa u sistemu. Klase predstavljaju objekte koje sistem obrađuje, a mogu se nalaziti u različitim tipovima međusobnih odnosa: asocijaciji, agregaciji, zavisnosti, generalizaciji. Opis sistema obično sadrži više dijagrama klasa.

Primjer dijagrama klasa prikazan je na slici. Uradjen je uprošćen primjer Univerzitetske jedinice ETF.



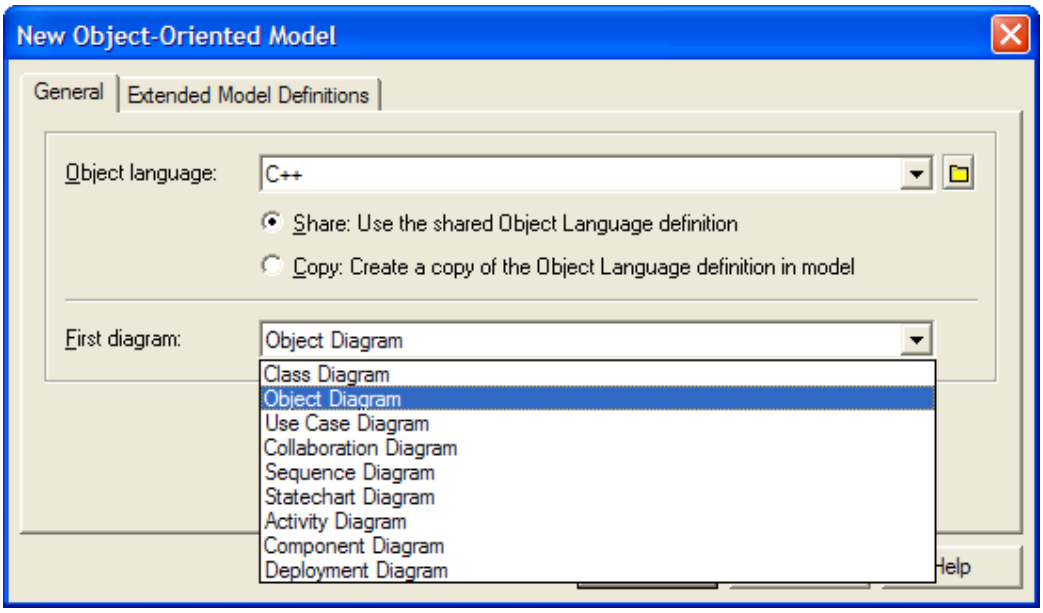
Prikazane su razne vrste veza i objašnjenja što te veze predstavljaju. Ranije je objašnjeno pravljenje klasa sa atributima i operacijama. Pojedine klase su detaljno uradjene a za druge samo su dati nazivi klasa.

Primjer dijagrama klasa fiskalne kase.



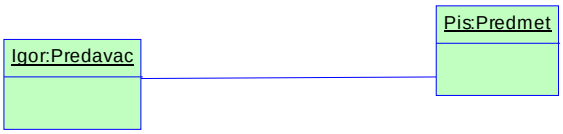
2.4 Dijagram objekata

U dijagram objekata ulazimo slično kao i u diagram klasa, samo se izabere **Object Diagram**



Paleta kod dijagrama objekata je slična sa paletom kod dijagrama klasa.

Primjer:



2.5 Sekvencijalni dijagram

Dijagram sekvenci (Sequence Diagram) opisuje vrijeme trajanja poruke i način na koji objekti u sistemu međusobno komuniciraju, ostvarujući očekivano ponašanje. Dakle, prikazuje se vrijemenska komponenta i poruke koje se prosljeđuju između objekata u cilju izvršenja posmatrane operacije. Objekti su imenovane ili neimenovane instance klasa, ali mogu da budu i instance drugih stvari, kao što su saradnja, komponente ili čvorovi. Dijagram sekvenci je grafička ilustracija dinamičke interakcije, gdje objekti komuniciraju preko sekvenci poruka, tj. prikazuje dinamičku saradnju između objekata u vrijemenu.

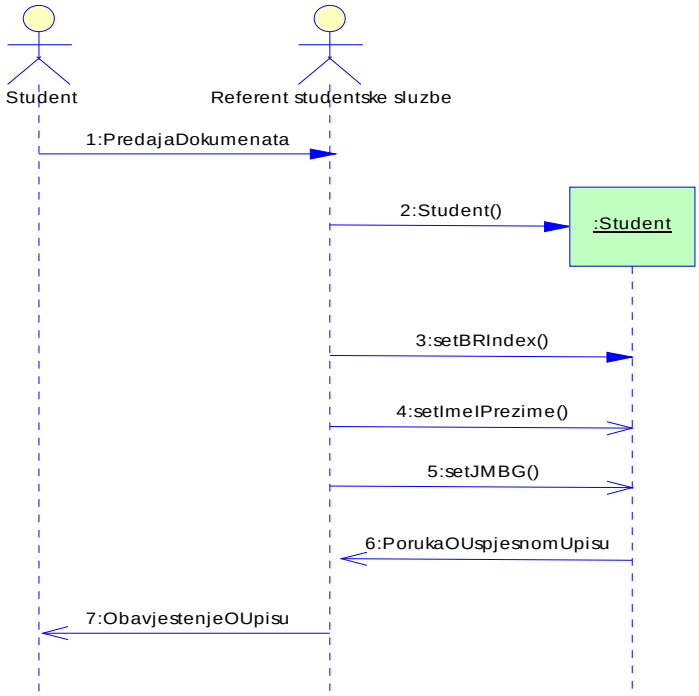
Paleta izgleda kao na slici

Bitni elementi sekvencijalnog dijagrama iz palete su

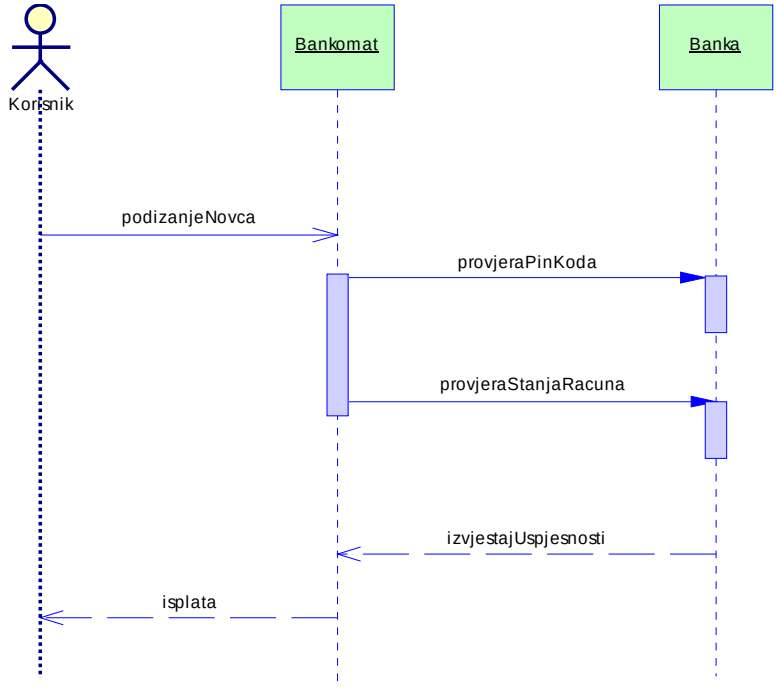
Ime	Opis
Package	Paketi čine grupu više elemenata modela
Actor	Predstavlja eksternu osobu, učesnika, izvođača koja je u nekoj interakciji sa sistemom, pod-sistemom ili klasama.
Object	Instanca klase sa plivajućom stazom. Plivajuća staza je dio dijagrama interakcije za organizovanje dužnosti akcija. Često odgovaraju org. jedinicama u poslovnom modelu.
Activation	Aktivnosti predstavljaju izvršne procedure, uključujući i ugnježdene procedure koje čekaju na izvršenje.
Message	Poruka koja prenosi informaciju sa očekivanim događajima.
Self Message	Samo poruka je rekurzivna poruka gdje su pošiljalac i primalac isti objekti.
Call Message	Poruka zvanja je rekurzivna poruka gdje su pošiljalac i primalac isti objekti.
Self Call Message	Procedura zvana samo poruka zvanja koja se po defaultu aktivira odnosno izvršava.
Return Message	Procedura zvana rekurzivna poruka koja se po defaultu aktivira odnosno izvršava.

Self Return Message	Genaralna asocijacija sa zvanjem procedure gdje vraćena poruka može da izostavlja implicitno izvršavanje poruke.

Dijagram nazovimo **Upis na fakultet**

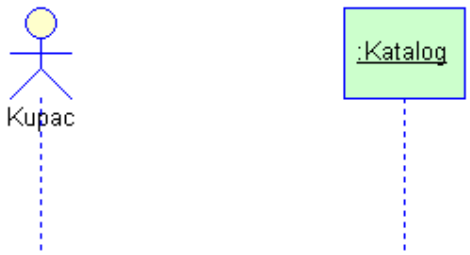


Dijagram **podizanja novca sa bankomata**

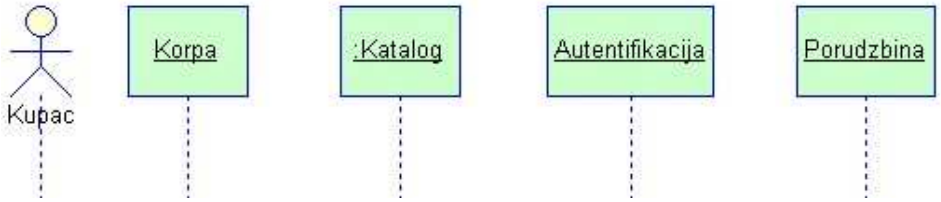


Dijagram **Procesiranje porudzbine.**

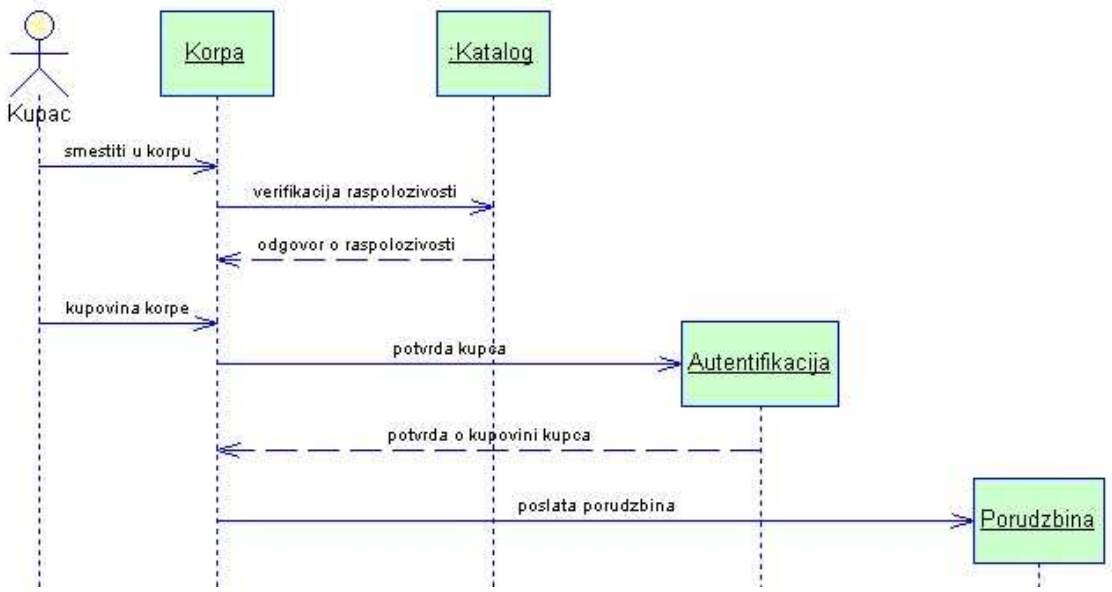
Na dijagram iz ***Broswer prozora*** ubacimo učesnika **Kupac** i objekat po imenu **Katalog**.



Da bi odradili kupovinu ubacimo još elemenata

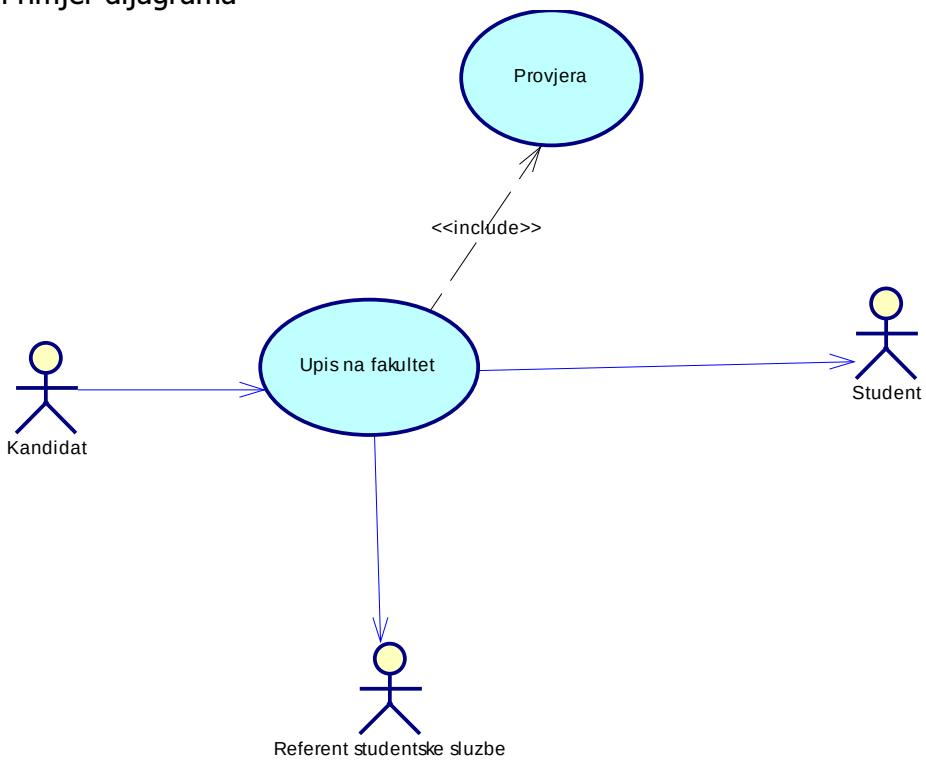


Zatim koristeći poruke ***Message*** i ***Return*** **Message** kreira se dijagram.



2.6 Dijagram korisničkih funkcija prikazuje spoljne učesnike, odnosno korisnike sistema i njihove veze sa korisničkim funkcijama koje sistem omogućava. Model korisničkih funkcija obično se sastoji od više dijagrama korisničkih funkcija. Osnovne komponente su: korisničke funkcije, učesnik (korisnik) i sistem koji se modelira, kao i različite veze: asocijacija, generalizacija i zavisnost između elemenata. Korisnička funkcija je opis pojedine funkcije koju sistem obezbjeđuje, viđen iz perspektive korisnika, bez opisa unutrašnje funkcionalnosti. Detaljnije se opisuje tekstom ili dijagramom aktivnosti. Učesnik je bilo koji spoljni entitet, čovjek, drugi sistem ili neki hardverski uređaj, koji treba da komunicira sa sistemom koji se modelira. Sam proces kreiranja modela čini: definisanje sistema, uočavanje učesnika i korisničkih funkcija, opisivanje korisničkih funkcija, definisanje veza između funkcija i na kraju validacija modela.

Primjer dijagrama

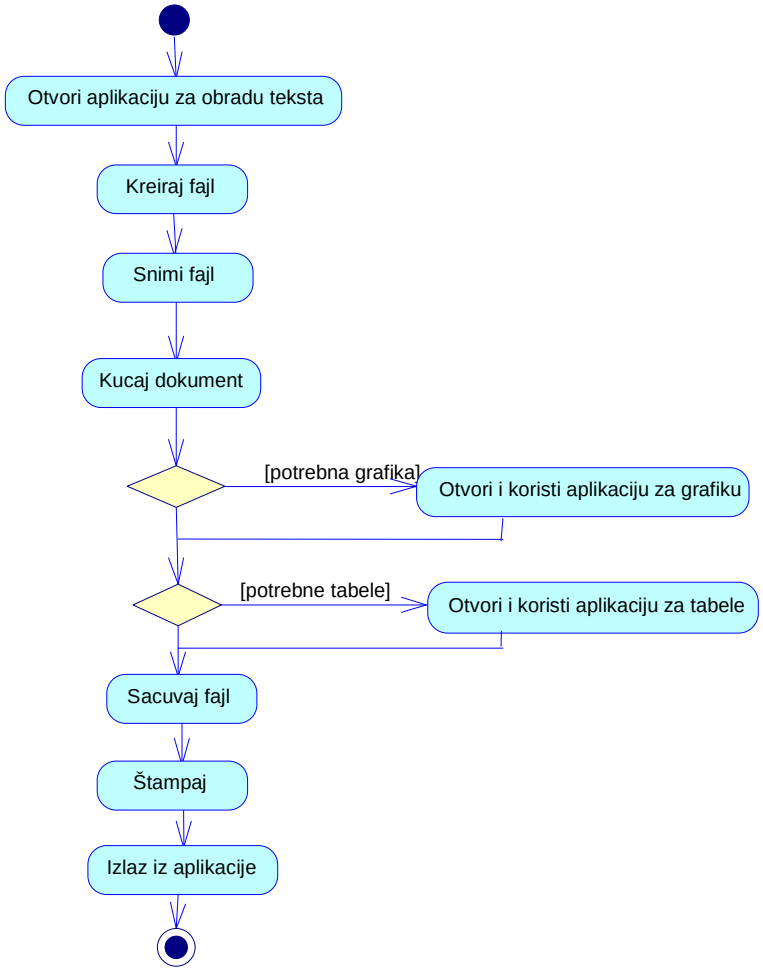


2.7 Dijagram aktivnosti (Activity Diagram) prikazuje komunikaciju izmedju objekata ali je akcenat na aktivnostima koje se izvršavaju tokom posmatrane komunikacije. Posmatraju se aktivnosti koje objekti izvršavaju kao i redosljed tih aktivnosti. Dijagramom aktivnosti opisuju se aktivnosti koje se izvršavaju u okviru jedne operacije. Opisuje se takodje redosljed izvršavanja tih aktivnosti odnosno sam algoritam operacija.

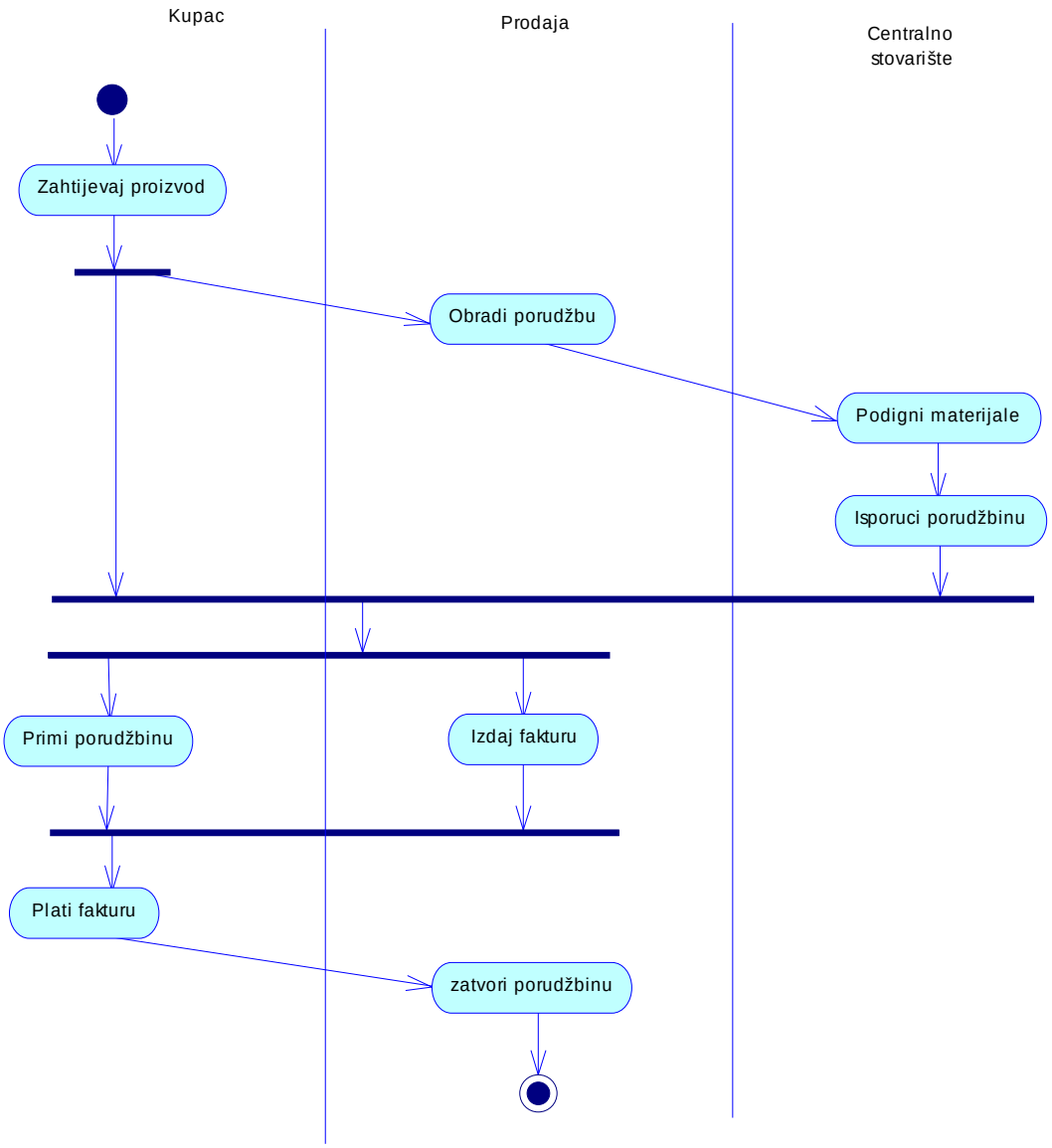
Paleta izgleda kao na slici:

Ime	Opis
Package	Paketi čine grupu više elemenata modela.
Start	Start - Tačka startovanja svih aktivnosti prezentovani u dijagramu aktivnosti.
Activity	Aktivnost – Prizivanje bilo koje aktivnosti. Ponašanje koje se dešava u nekom stanju. Aktivnost može biti prekinuta događajem tranzicije.
Composite Activity	Kompozitna aktivnost – Kompleksna, složena aktivnost koja može da se dekompunuje na detaljne aktivnosti.
Object state	Objekat stanja – Reprezentuje specifično stanje bilo koje aktivnosti. Objekat koji poseduje nit i koji može inicijalizovati kontrolnu aktivnost. Instanca aktivne lase.
Organization unit	Organizaciona jedinica – Elelement koji reprezentuje, predstavlja kompaniju, sistem, servis ili organizaciju, korisnike sa svojim ulogama (role).
Transition	Tranzicija – Relacija između dva stanja koja pokazuje da jedan objekat u prvom stanju će izvesti glavne specificifirane akcije i unijeti drugo stanje kada je speficirano stanje zadovoljeno. Na ovakvoj promjeni stanja tranzicija se zove paljba.
Decision	Tačka Odluke – Tačka u dijagramu aktivnosti u kojoj se koriste zaštitni uslovi da ukažu na različite moguće tranzicije.
Synchronization	Sinhronizacija – Zahtev koji se šalje objektu pauze da sačeka rezultate ili da se izvrši sinhronizacija dveju ili više konkuretnih aktivnosti.
End	Kraj - Tačka završetka svih aktivnosti koji su opisani u dijagramu aktivnosti.

Primjer dijagrama aktivnosti:



Primjer dijagrama aktivnosti sa plivačkim stazama:



2.8. Dijagram stanja (State Diagram) je konačni automat koji sadrži stanje, prelaze, događaje i aktivnosti. Dijagram promjene stanja je dinamički dijagram koji prikazuje sekvencu stanja kroz koje objekat prolazi tokom vremena (tokom životnog vijeka), a kao reakcija na spoljne ili unutrašnje pobude (vezan za samo jedan objekat i određenu operaciju unutar njega za određenu klasu). Opis stanja obuhvata aktivnosti koje se izvršavaju u pojedinim stanjima, akcije koje se izvršavaju pri prelasku iz jednog stanja u drugo, kao i poruke koje uslovljavaju promjenu stanja posmatranog objekta. Kreiranjem dijagrama promjene stanja prikazuju se reakcije sistema izazvane događajima. Dijagram promjene stanja se može prevesti u dijagram aktivnosti koji se fokusira na tok kontrole (i obrnuto).

Paleta:

Ime	Opis
Package	Paketi čine grupu više elemenata modela.
Start	Start - Tačka startovanja svih stanja prezentovani u dijagramu stanja.
State	Stanje – Akumulirani rezultati ponašanja nekog objekta; jedno od mogućnosti stanja (situacija) u kome objekat može da postoji.
Action	Akcija – Ponašanje koje se nadovezuje na neki događaj tranzicije. Za akciju se smatra da se izvršava trenutno i da se ne može prekinuti.
Event	Događaj, slučaj – Specifikacija važnog pojavljivanja koje ima mjesto u vremenu i prostoru. U kontekstu dijagrama stanja, jedan slučaj je jedno pojavljivanje koje se može označiti kao izmjena stanja.
Transition	Tranzicija – Relacija između dva stanja koja pokazuje da jedan objekat u prvom stanju će izvesti glavne specificifirane akcije i unijeti drugo stanje kada je speficirano stanje zadovoljeno.
Junction point	Tačka spajanja – Tačka spajanja je slična tački odlučivanja i dijagramu aktivnosti, ali sa mogućnošću višestrukih ulaznih i izlaznih tranzicija.
End	Kraj - Tačka završetka svih stanja koji su opisani u dijagramu stanja.

Pri opisivanju dinamike sistema preko dijagrama prelaza stanja koriste se sljedeći pojmovi:

(1) **Sistem (objekat)** je skup objekata, njihovih atributa i njihovih veza. Struktura sistema - odnos njegovih objekata, veza i atributa opisuje se preko modela opisanih u prethodnom dijelu.

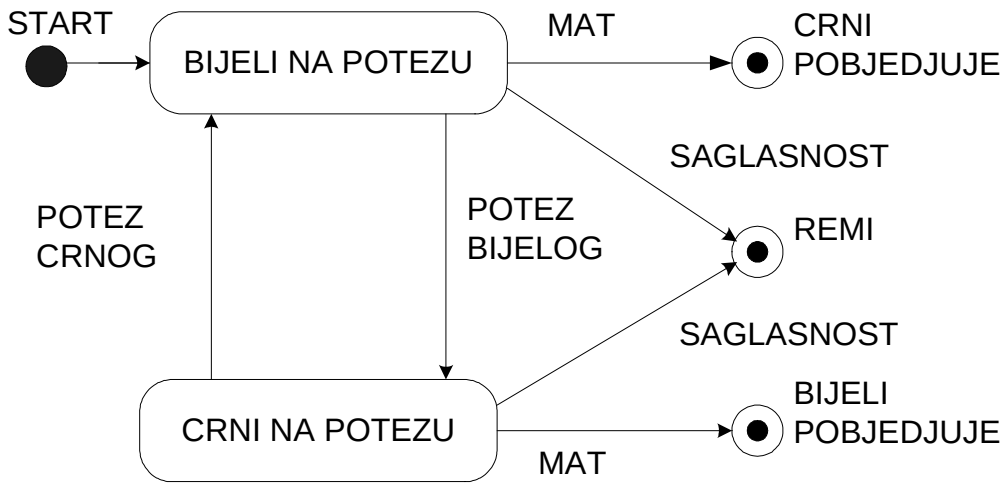
(2) **Stanje sistema (objekta)** u jednom trenutku vremena predstavlja skup vrijednosti atributa svih objekata i "vrijednosti" svih veza u tom trenutku. Termin "vrijednost veze" opisuje par (za binarne veze ili n-torku uopšte) identifikatora pojavljivanja objekata koji su u vezi.

(3) **Događaji** iniciraju promjene stanja sistema. Odziv sistema na neki događaj zavisi od stanja u kome se on nalazi. Događaj može da prouzrokuje promjenu stanja sistema i/ili da indukuje novi događaj. Događaj se zbiva u jednom trenutku vremena, događaj nema trajanje (u vrijemenskoj skali u kojoj se posmatra dati sistem). Ponekad se događaj i poruka tretiraju kao sinonimi. Međutim, precizno, poruka je pojavljivanje događaja.

(4) **Dijagram prelaza** (promjene) stanja je apstrakcija koja pokazuje stanja, događaje i prelaze (tranzicije) iz stanja u stanje kao mogući odziv na događaje.

(5) **Dijagram promjene stanja** povezuje stanja (konkretna, imenovana) sa događajima u sistemu. Promjena stanja izazvana događajem naziva se tranzicija (prelaz). Dijagram promjene stanja je usmjereni graf u kome su čvorovi stanja, a grane tranzicije, sa usmjerenjem od polaznog do prouzrokovanog stanja. Granama grafa daju se nazivi događaja koji prouzrokuju tranziciju. (Jedan događaj može da prouzrokuje više tranzicija, pa više grana može da ima isto ime).

(6) **Početna i krajnja stanja.** Može se uvesti i koncept početnog i krajnjeg stanja, za objekte (sisteme) koji imaju “ograničen život”. U tom slučaju, početno stanje je rezultat kreiranja odgovarajućeg objekta, a krajnje podrazumeva njegovo “uništenje” (nestanak). Početna i krajna stanja na grafu imaju specijalne oznake, a mogu imati i imena.



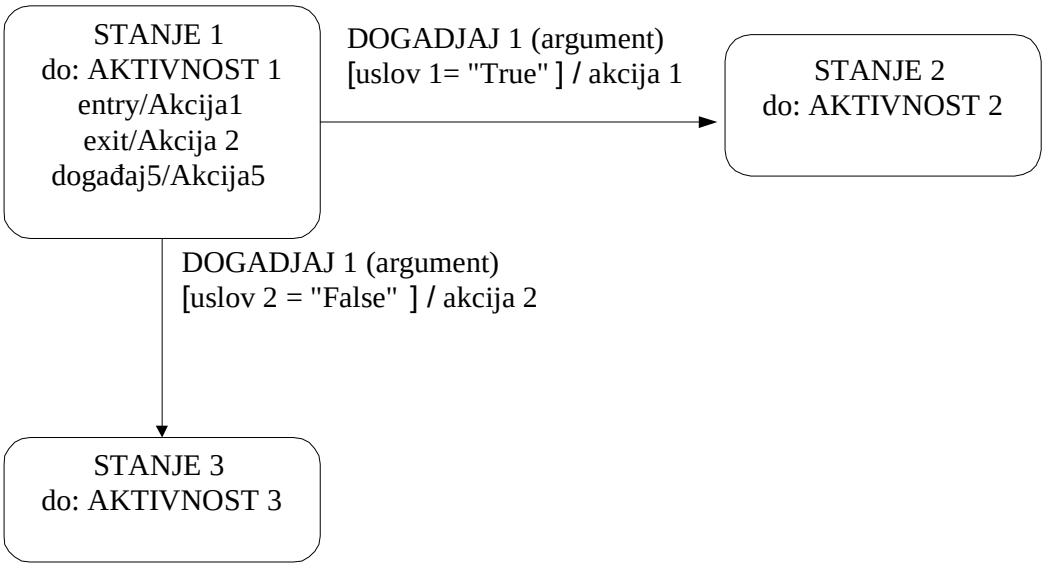
(7) **Uslovi**. Uslov je Bulova funkcija nad vrijednostima atributa i veza. Stanja sistema se mogu opisati preko uslova. Iskaz da je objekat u nekom stanju je uslov. Pored toga, uslovi se mogu koristiti da ograniče tranzicije prouzrokovane događajima. Ponekad, za prelaz sistema iz jednog stanja u drugo potrebno je, pored događaja, da bude ispunjen i neki uslov. Na dijagramu prelaza stanja uslov se iskazuje uz naziv događaja, unutar uglaste zagrade.

(8) **Akcija** pretstavlja jedno "atomsko sračunavanje" koje prouzrokuje promjenu stanja sistema ili vraća neku vrijednost. Neka akcija okida događaj koji će sistem prevesti iz jednog u drugo stanje. Akcija može da pozove operaciju nekog objekta, da

kreira ili uništi neki objekat ili da pošalje signal nekom objektu. Akcije se mogu pridružiti stanjima i tranzicijama. Ako se akcije pridružuju stanjima one mogu biti:

- "entry" – akcija koja se obavlja uvek pri ulazu u stanje, bez obzira koja je tranzicija to prouzrokovala;
- "exit" - akcija koja se obavlja uvek pri napuštanju stanja, bez obzira koja je tranzicija to prouzrokovala;
- "inerna tranzicija" – akcija koja ne menja stanje sistema.

Akcija se može obaviti i pri prelazu iz jednog u drugo stanje. Na tranziciji akcije se iskazuju uz naziv događaja, iza uslova i oznake "/".



(9) **Sinhronizacija konkurentnih aktivnosti**. U jednom stanju se može obavljati više konkurentnih aktivnosti.. Ove aktivnosti ne moraju biti sinhronizovane, mogu se obavljati bilo kojim redom, ali sve one moraju biti obavljene prije nego što se izvrši tranzicija u drugo stanje. Konkurentne aktivnosti u jednom stanju se prikazuju podjelom stanja (čvora) na djelove razmaknute isprekidanom linijom.

(10) **Neoznačena ili automatska tranzicija**. Koristi se da bi se prikazala automatska tranzicija iz jednog stanja u drugo koja se obavlja čim se aktivnost u nekom stanju obavi. Kraj aktivnosti u nekom stanju može se tretirati kao neimenovani događaj. Taj neimenovani događaj "okida" neimenovanu tranziciju u drugo stanje.

(11) **Dekompozicija dijagrama prelaza stanja.** Dijagrami prelaza stanja se mogu dekomponovati na sljedeće načine:

(i) Kompozitno stanje, odnosno sekvencijalna podstanja.
(ii) Generalizacija stanja.
(iii) Agregacija stanja - agregaciona konkurentnost.