Aspektno orijentiran razvoj softvera

Duketis, Deodato

Sveučilište Jurja Dobrile u Puli

Fakultet ekonomije i turizma

„Dr. Mijo Mirković“

Deodato Duketis

ASPEKTNO-ORIJENTIRAN RAZVOJ SOFTVERA

Diplomski rad

Pula, 2015.

Sveučilište Jurja Dobrile u Puli

Fakultet ekonomije i turizma

„Dr. Mijo Mirković“

Deodato Duketis, 329-ED, redovni student

Smjer: Poslovna informatika

ASPEKTNO-ORIJENTIRAN RAZVOJ SOFTVERA

Diplomski rad

JMBAG: 0016052778, redoviti student

Smjer: Poslovna informatika

Kolegij: Softversko inženjerstvo

Mentor: Doc. dr. sc. Krunoslav Puljić

Pula, 2015.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Deodato Duketis, kandidat za magistra poslovne ekonomije ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student:

U Puli, 23. rujna 2015.

SADRŽAJ

1. Uvod

Informacijski sustavi današnjice su glavni oslonac poslovanja diljem svijeta. Kako se poslovne politike mijenjaju iz dana u dan, tako se mijenjaju i zahtjevi na informacijske sustave koji im pružaju potporu u obavljanju poslovnih aktivnosti i nemoguće je zamisliti tvrtku koja bi danas mogla funkcionirati bez potpore informacijskog sustava. Širenjem poslovanja i promjenom poslovnih politika raste složenost zadaće koju jedan informacijski sustav mora odraditi, a samim time, raste i direktan trošak pisanja kôda za takav sustav kao i njegovo daljnje održavanje i razvoj. Javlja se potreba za novim načinima proizvodnje informacijskih sustava – novim načinima koji programerima olakšavaju razvoj velikih sustava.

Osnovna hipoteza ovog rada jest da aspektno-orijentiran razvoj softvera omogućuje lakši razvoj te brži i jednostavniji rast složenih informacijskih sustava te samim time smanjuje troškove proizvodnje i održavanja jednog takvog informacijskog sustava.

Cilj ovog rada je pokazati potrebu za novim načinima razvoja softvera, objasniti logiku u novom načinu razmišljanja kod razvoja jednog aspektno-orijentiranog sustava, te prikazati primjer jedne aspektno-orijentirane aplikacije. Biti će objašnjeni problemi u razvoju softvera današnjice i pokazana potreba za novim metodama, zatim će biti prikazane koristi kod aspektno-orijentiranog razvoja softvera. Nakon toga rad usmjerava na korake probleme i rješenja tih problema pomoću aspektno-orijentiranog programiranja. Zatim slijedi prikaz aspektno-orijentiranog programskog jezika AspectJ koji je u stvari nadogradnja na Java programski jezik. Na kraju se navodi i nekoliko jednostavnih konkretnih primjera u AspectJ-u.

Rad se sastoji od 14 poglavlja. Nakon prvog poglavlja uvoda, u drugom poglavlju se govori o povijesti aspektno-orijentiranog razvoja softvera, objašnjava se pojam dužnosti i navode se razlike između osnovnih i presijecajućih dužnosti. Usporedbom sa objektno-orijentiranim programiranjem, objašnjava se potreba za novim metodama. Objašnjava se što je to zapravo aspektno-orijentirano programiranje, uvodi se pojam aspekta i objašnjava se njihova priroda i svrha kao i razlozi zašto su oni potrebni. Navodi se jednostavan primjer koji prikazuje razliku u korištenju objektno-orijentiranog programiranja i aspektno-orijentiranog programiranja. Objašnjeno je kako se teorija AOP-a prevodi u praksu kroz specifikaciju i implementaciju jezika. Specifikacija opisuje sintaksu, a implementacija provjerava pridržavanje pravila specifikacije. Također, navedene su neke od koristi u korištenju metodologije aspektno-orijentiranog razvoja softvera nad korištenjem dosadašnjih metoda razvoja softvera.

Treće poglavlje opisuje ideju o razdvajanju dužnosti na osnovne i isprepletene, te opisuje vrste dužnosti podijeljene po načinu organiziranja zahtjeva. Razdvajanje dužnosti popraćeno je objašnjenjem uz analogiju prizme. Također, u poglavlju se opisuju problemi koje metodologija AOP-a može rješavati – zaplitanje kôda i raspršivanje kôda.

Četvrto poglavlje prikazuje proces razvoja softvera pomoću aspekata: počevši od utvrđivanja zahtjeva (koje je prilagođeno aspektno-orijentiranom programiranju), preko oblikovanja i implementacije, pa do verifikacije i validacije jednom gotovog sustava. Također, navode se i poteškoće kod testiranja jednog aspektno-orijentiranog sustava.

U petom poglavlju navedeni su konstrukti koje uvodi AOP. Opisuju se aspekti, točke spajanja, točke presjeka i savjeti. Ti novi pojmovi, odnosno apstrakcije, potrebne su za razumijevanje logike koja stoji iza aspektno-orijentiranog razvoja softvera.

Šesto poglavlje uvodi alat za razvoj aspektno-orijentirane aplikacije – AspectJ. Taj program je nadogradnja na Java programski jezik, a u poglavlju se opisuje njegova povijest, koristi, specifikacija i implementacija. U ovom je poglavlju pobliže objašnjena sintaksa točke presjeka u jeziku AspectJ. Navode se njeni sastavni dijelovi (specifikator pristupa, ključna riječ, ime točke, vrsta točke, potpis), objašnjava se uloga višeznačnika u definiciji točke presjeka. Također, objašnjavaju se obrasci potpisa: obrazac potpisa tipa, obrazac potpisa metode, obrazac potpisa polja te njihova uloga.

Sedmo poglavlje ukratko objašnjava logiku u pisanju jednog aspektno-orijentiranog programa. Navodi se da se jedan takav program razvija u dvije faze: pisanje osnovnog kôda pomoću osnovnog jezika i pisanje aspektnog kôda pomoću nadogradnje jezika kao što je to AspectJ. Ovdje su navedeni jednostavni primjeri korištenja aspekata u programiranju. Prvi primjer objašnjava poznati „Hello World“ slučaj kao uvod. Drugi primjer uvodi tzv. „*around*“ vrstu savjeta koja omogućuje izmjenu konteksta izvođenja metode.

Osmo poglavlje navodi par pitanja sa ciljem objašnjavanja da li aspektno-orijentirani pristup razvoju softvera uopće ima smisla ili je to samo nova složena metoda.

U devetom poglavlju dan je zaključak koji je rezultirao pisanjem ovog rada. Navode se problemi koji ukazuju na potrebu nastajanja nove metode kao što je to AOP, pokazuju se najvažniji koraci u pisanju aspektno-orijentirane aplikacije. Za kraj, navode se razlozi otpora u prihvaćanju metodologije AOP-a.

U zadnjim poglavljima navodi se popis literature, popis slika, popis tablica i popis primjera. Pred kraja rada dodan je prilog radu – primjer aplikacije za pristup bazi podataka i izvoz podataka u HTML datoteku. Na samom kraju rada nalazi se sažetak na hrvatskom i engleskom jeziku.

2. O aspektno-orijentiranom programiranju

Složene poslovne sustave današnjice je teško prikazati na jednostavan način. Razlog tome su skupovi poslovnih zahtjeva koje softver mora ispuniti. Softverski sustavi nisu sastavljeni samo od jednog modula koji se nalazi u jednoj datoteci. Oni su glomazni, sastavljeni su od mnogobrojnih modula koji u sinkroniziranom radu pružaju određenu funkcionalnost definiranu poslovnim zahtjevima. Razvoj softverskog inženjerstva omogućio je da se danas programeri mogu suočavati sa puno većim problemima nego što su se suočavali godinama unazad. Nekad su se programeri brinuli o strojnim instrukcijama, a danas se na poslovni sustav gleda kao na skup objekata koji međusobno surađuju.

Kod oblikovanja složenog poslovnog sustava potrebno je uzeti u obzir trenutne zahtjeve ali i planirati eventualne buduće zahtjeve koji se mogu pojaviti širenjem poslovanja i pojavom novih poslovnih procesa. Takvim pristupom povećava se složenost sustava, a programeri rješenje nalaze u modularizaciji – problem se rastavlja na manje cjeline (potprobleme) i svakoj od tih cjelina se pristupa pojedinačno, te se na taj način, rješavajući dio po dio problema, stvara oblik cjelokupnog sustava.

Svaki potproblem predstavlja jednu funkcionalnost/dužnost (eng. ***concern***) koju sustav mora ispuniti u svrhu obavljanja svoje funkcije. Funkcionalnosti sustava se mogu podijeliti u dvije skupine:

- Centralne dužnosti – *core concerns*
- Presijecajuće dužnosti – *crosscutting concerns*.

Centralne su dužnosti one koje opisuju glavnu funkciju jednog modula i karakteristične su samo za taj modul, dok presijecajuće dužnosti opisuju sporedne dužnosti sustava koje su istovremeno prisutne u više modula.

Složen sustav moguće je pojednostaviti tako da se identificiraju dužnosti te da se one modulariziraju (razdvajanje dužnosti – eng. ***separation of concerns***). OOP je uspješna metoda za modulariziranje centralnih sistemskih dužnosti ali ne i za rasprostranjene / presijecajuće dužnosti. OOP se pokazalo kao odličan pristup za opisivanje centralnih dužnosti sustava time što omogućuje dekompoziciju sustava na manje dijelove (objekte) i modeliranje ponašanja svakog od tih objekata. Koristeći OOP moguće je centralne module labavo vezati koristeći sučelja (eng. ***interface***), ali, korištenjem OOP-a presijecajuće dužnosti nije moguće labavo

vezati. OOP se dobro nosi sa modeliranjem zajedničkog ponašanja ali ima poteškoća kod modeliranja ponašanja koja su zajednička mnoštvu nepovezanih modula.

U OOP-u svaka klasa mora biti svjesna svoje okoline, i to je razlog pojave kopiranja dijelova kôda u aplikacijama (eng. ***code scattering*** – rasipanje/raspršivanje kôda). Radi toga OOP kod presijecajućih dužnosti dovodi do toga da se kôd potreban za implementaciju te dužnosti raspršuje u druge module. Tu dolazi do tzv. čvrste veze (eng. ***tight coupling***) među presijecajućim i centralnim dužnostima. Dodavanjem jedne nove presijecajuće dužnosti ili modificiranjem postojeće, može doći do potrebe za modificiranjem relevantnog centralnog modula, a kao rezultat pojavljuje se nemodularnost.

Pojavilo se nekoliko metodologija kao rješenje modularizacije presijecajućih dužnosti: generativno programiranje, meta-programiranje, reflektivno programiranje, kompozicijsko filtriranje, adaptivno programiranje, subjektno-orijentirano programiranje i aspektno-orijentirano programiranje. Od navedenih, jedino je AOP postao općeprihvaćeno rješenje.

Metodologija AOP-a uvodi novu razinu apstrakcije – ***aspekt***. Aspekt predstavlja modul koji centralizira presijecajuću dužnost – omogućava centralnim modulima da se razvijaju nezavisno o presijecajućim dužnostima. AOP uvodi pojam aspektnog tkalca (eng. ***weaver***) koji ima zadaću sličnu kao i kompajler – kombinira centralne i presijecajuće dužnosti, i odgovoran je za kompoziciju završnog sustava kroz proces nazvan tkanje (eng. ***weaving***). Uvođenjem modularizacije u presijecajuće dužnosti, AOP u velikoj mjeri rješava postojeće probleme, omogućava lakše održavanje sustava, povećava ponovnu iskoristivost kôda (eng. ***code-reuse***), te kao završni rezultat isporučuje sustav koji je lakši za oblikovanje, te daljnje razvijanje i održavanje.

AOP je metodologija koja je zamišljena kao nadogradnja OOP-a sa svrhom identificiranja jasne uloge presijecajućih dužnosti u sustavu, kao i izdvajanje te uloge u zaseban modul (labavo vezanje uz samo par drugih modula).

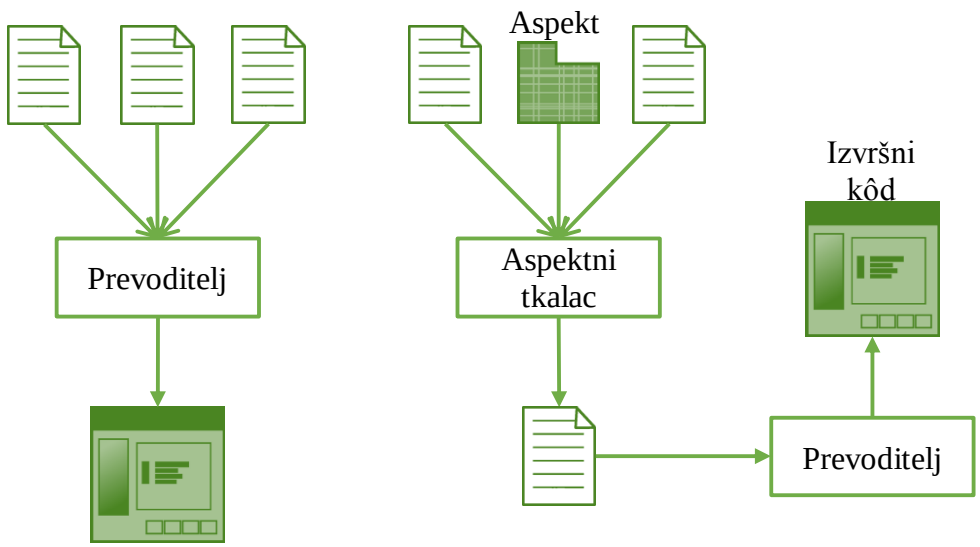
Ako za primjer uzmemo bankarski sustav, njegove bi centralne dužnosti bile vođenje računa, računanje kamata, obavljanje transakcija među bankama, i slične. Osim navedenih centralnih dužnosti, postoje i sporedne dužnosti koje aplikacija mora zadovoljiti kako bi sustav funkcionirao. Te sporedne dužnosti se presijecaju sa centralnim dužnostima pa se stoga nazivaju „presijecajuće“ dužnosti, a u bankarskom sustavu možemo uočiti: autentifikacija (identifikacija korisnika), administracija, upravljanje transakcijama (kod svakog pristupanja bazi podataka), upravljanje bazom podataka, provjera grešaka, i dr.

Moguće je stvoriti module koji sadrže jednostavne funkcije, ali ipak, često je potrebno stvaranje modula koji sadrže miješane ciljeve: funkcija, kao npr. „prijava korisnika“, može biti raspodijeljena među više modula u jednoj aplikaciji. Tako se neka svojstva mogu raspodijeliti po cijelom sustavu, a problem sa kojim se onda programeri suočavaju je u održavanju i proširivanju takvog sustava.

AOP nije zamjena za trenutno rasprostranjen OOP – još je uvijek potrebno koristiti OOP metodologiju oblikovanja da se napravi solidna temeljna arhitektura. Ono što AOP nudi nije potpuno novi proces oblikovanja, već nudi dodatna sredstva koja omogućavaju programeru da već sada riješi potencijalne buduće zahtjeve bez da ugrozi osnovnu arhitekturu sustava, i da provede manje vremena na presijecajućim dužnostima tijekom početnih faza oblikovanja budući da se one mogu ugraditi (utkati) u sustav onda kada se za to pokaže potreba.

2.1 Metodologija AOP-a

Slika 1. Tradicionalni (lijevo) i aspektno orijentirani (desno) pristup izradi aplikacija.



Izvor: Lee, K.W.K. An Introduction to Aspect-Oriented Programming, 2002., str. 4.

U poslovnim sustavima, odnosi između programskih modula i zahtjeva su složeni. Svaki modul može sadržavati elemente iz više zahtjeva, a jedan se zahtjev može implementirati pomoću više modula. To za posljedicu ima da ako dođe do promjene zahtjeva moguće je da će biti potrebno razumjeti i izmijeniti više modula. Aspektno-orijentiran razvoj softvera je pristup koji za cilj ima razvoj takvih programa koji su jednostavniji za održavanje i ponovno korištenje.

Funkcionalnosti koje mogu biti potrebne na više različitih mjesta u programu implementiraju se pomoću aspekta – apstrakcije na kojoj se temelji aspektno-orijentiran razvoj softvera. Oni se koriste zajedno sa već poznatim apstrakcijama: objektima i metodama.

Aspekti sadrže definiciju o tome **gdje** se oni trebaju uključiti (utkati) u program. Potrebno je odrediti da li će se kôd (koji implementira presijecajuću dužnost) uključiti **prije** ili **poslije** određenog poziva metode ili **u trenutku** pristupa nekom atributu.

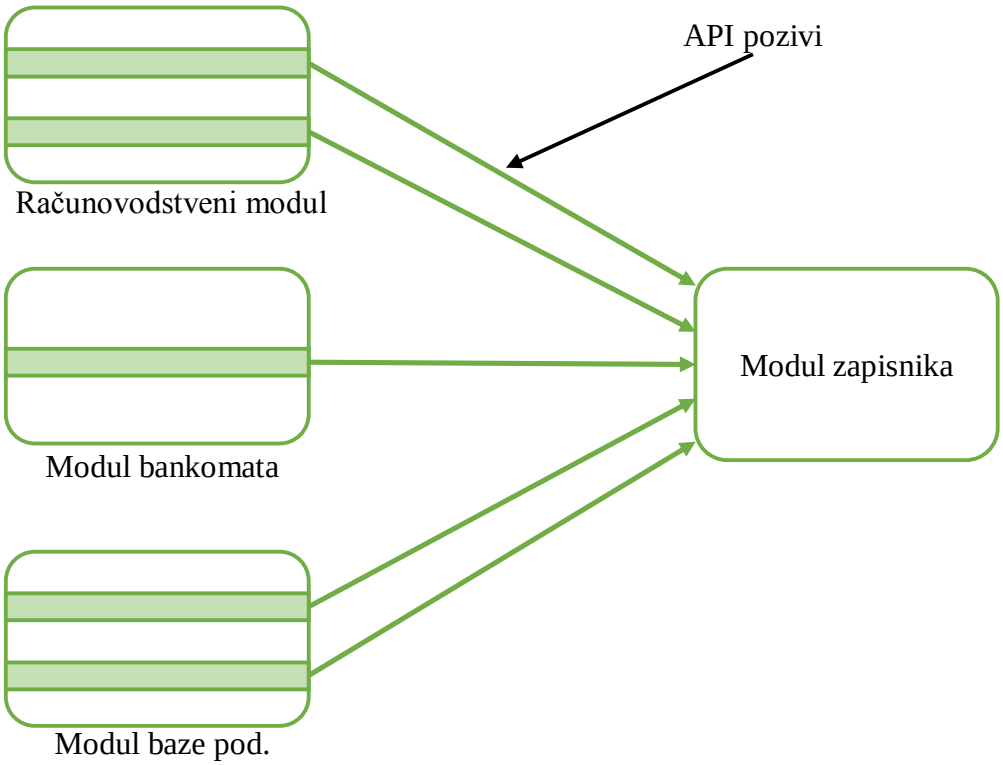
Glavna korist AOP-a je da omogućuje razdvajanje dužnosti te njihov prikaz pomoću aspekta čime se te funkcionalnosti mogu ponovo koristiti i nezavisno modificirati.

Za primjer možemo uzeti zahtjev za autentikacijom korisnika koja je obavezna prije nego se izmijene osobni podaci u bazi podataka. Zahtjev za korisničkim imenom i lozinkom možemo prikazati kao aspekt u kojem možemo specificirati da se kôd za autentikaciju uključi prije svakog poziva metoda koje izmjenjuju osobne podatke. Ako se zahtjev promijeni i želi uključiti na svaku izmjenu baze podataka to se jednostavno može postići tako da se izmijeni aspekt, tj. promijeni se definicija o tome gdje se kôd treba utkati u sustav. To uvelike olakšava dodavanje izmjena u sustav i smanjuje vjerojatnost da se napravi greška te samim time i smanjuje neželjene sigurnosne propuste u sustavu.

Još jedan primjer implementacije presijecajuće dužnosti korištenjem OOP-a: modul za autentikaciju koji pruža uslugu putem sučelja (eng. ***interface***). Korištenje sučelja oslabljuje vezu između klijenta i implementacija sučelja. Klijenti koriste usluge autentikacije putem sučelja pa radi toga ne znaju kakav način implementacije oni koriste. Ako dođe do promjene implementacije autentikacije to neće zahtijevati bilo kakve promjene na klijentima. To omogućuje da se jedna implementacija zamijeni drugom uz nikakve ili minimalne promjene na modulima klijenata. Ovaj način ipak zahtjeva da svaki klijent sadrži kôd potreban za pozivanje API-ja (eng. ***Application Programming Interface***) a taj kôd se samim time miješa sa osnovnim kôdom (logikom) modula.

Na slici broj 2 prikazan je primjer implementacije funkcije zapisnika (eng. ***logging***) uz korištenje konvencionalnih tehnika (OOP).

Slika 2. Implementacija dužnosti zapisnika korištenjem konvencionalnih tehnika.

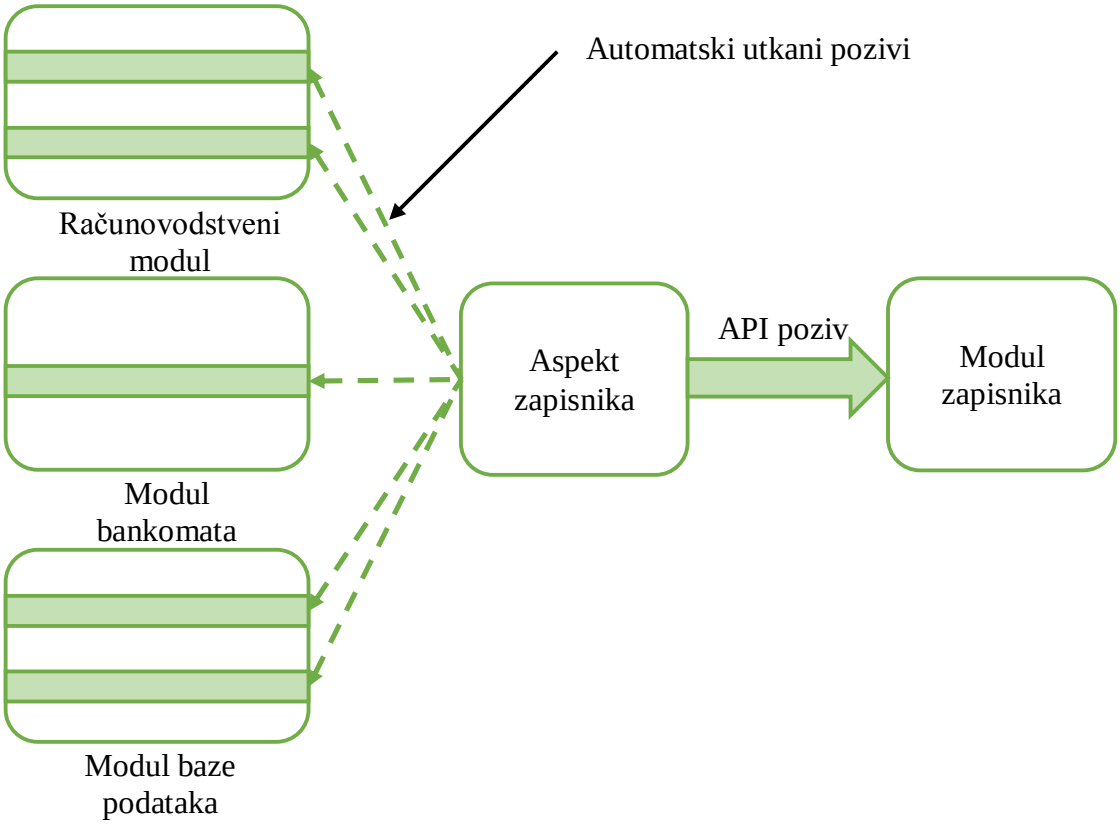


Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 12.

Kada bi za isti zadatak (zapisnik) koristili AOP, moduli ne trebaju sadržavati kôd za pozivanje zapisnika.

Na slici broj 3 je prikazana AOP implementacija zapisnika gdje se kôd zapisnika nalazi zasebno u modulu i aspektu zapisnika. Time je postignuta potpuna izolacija klijenata te će bilo kakva promjena u zapisniku utjecati samo na njegov aspekt.

Slika 3. Implementacija dužnosti zapisnika korištenjem AOP tehnika.



Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 13.

Razvijanje sustava korištenjem metodologije AOP-a slično je razvijanju sustava korištenjem drugih metodologija: identifikacija dužnosti, implementacija dužnosti, formiranje završnog sustava kombinirajući implementirane dužnosti. Istraživačka zajednica AOP-a ta tri koraka definira ovako[1]:

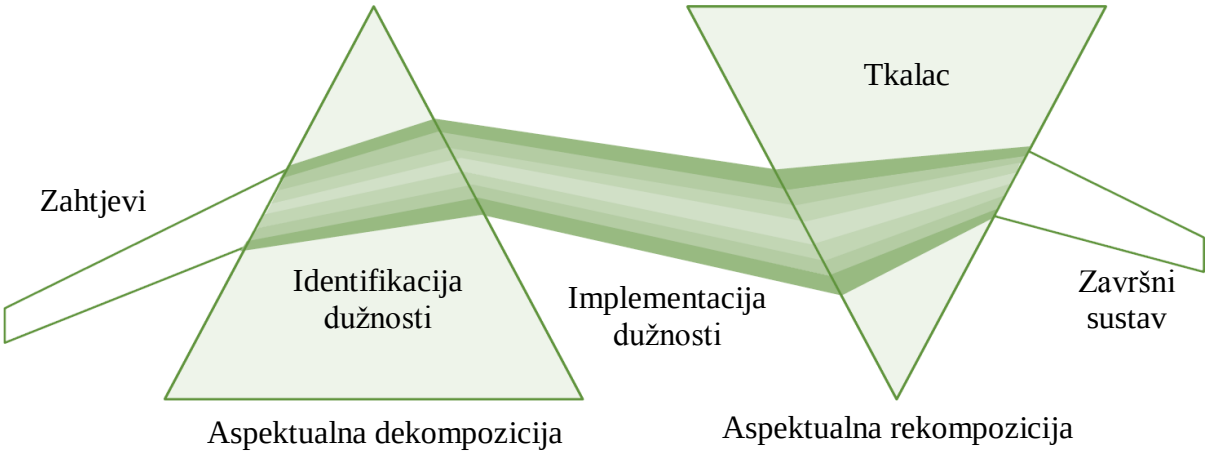
- **Aspektualna dekompozicija** – u ovom se koraku identificiraju osnovne i presijecajuće dužnosti tako da se zahtjevi dekomponiraju.
- **Implementacija dužnosti** – ovaj korak implementira svaku dužnost zasebno. Za osnovne dužnosti koriste se proceduralne ili OOP tehnike. Možemo uočiti da pojam „osnovni“ (eng. ***core***) ima relativno značenje. Osnovna dužnost u modulu autorizacije je povezivanje korisnika sa akreditacijama i provjera da li su te

[1] Prema: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 21.

akreditacije dovoljne za pristup autoriziranom servisu. Međutim, za modul poslovne logike, dužnost autorizacije je vanjska dužnost i kao takva neće biti implementirana u istom modulu.

- **Aspektualna rekompozicija** – u ovom se koraku stvaraju aspekti te se time specificiraju pravila rekompozicije sustava. Taj proces rekompozicije završnog sustava naziva se „tkanje“.

Slika 4. Faze razvoja sustava korištenjem metodologije AOP-a.



Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 22.

Na slici broj 4 prikazane su faze razvoja AOP-a uz analogiju prizme. Ulazni zahtjevi su rastavljeni na spektar dužnosti. Zatim se te dužnosti implementiraju te rekomponiraju u završni sustav korištenjem aspektnog tkalca. Temeljna promjena AOP-a jest očuvanje međusobne nezavisnosti pojedinačnih dužnosti nakon njihove implementacije, pa je takav sustav lakše razumjeti, jednostavnije implementirati te jednostavnije prilagođavati promjenama.

2.2 Anatomija aspektno-orijentiranog jezika

Aspektno-orijentirano programiranje je samo metodologija, teorija, i da bi bila od ikakve koristi mora se prevesti u praksu, odnosno, mora biti implementirana. Implementacija

aspektno-orijentiranog programiranja se sastoji od specificiranja jezika i određenog alata za rad sa tim jezikom[2]:

- Specifikacija jezika – opisuje jezične konstrukte i sintaksu koja će se koristiti u realizaciji logike osnovnih dužnosti kao i u tkanju presijecajućih dužnosti.
- Implementacija jezika – verificira da li se kôd pridržava specifikacije i prevodi kôd u izvršivi oblik. To uobičajeno obavlja kompajler ili interpreter.

2.2.1 Specifikacija jezika aspektno-orijentiranog programiranja

Svaka realizacija aspektno-orijentiranog programiranja mora imati jezik koji će se koristiti za implementaciju pojedinačnih dužnosti te jezik koji će se koristiti za implementaciju pravila za tkanje implementacija dužnosti.

Dužnosti su implementirane u modulima koji sadrže podatke i ponašanje potrebno za pružanje svojih usluga. Za implementaciju osnovnih i presijecajućih dužnosti koriste se jezici kao što su C, C++ i Java.

Pravila tkanja određuju kako će se implementirane dužnosti integrirati u završni sustav. Za primjer uzmimo dužnost zapisnika – nakon što je zapisnik (eng. ***log***) implementiran, potrebno ga je uvesti u sustav. U ovom slučaju, pravilo tkanja specificira točke zapisnika, informaciju koja će se bilježiti, itd. Moguće je, u samo par linija kôda, odrediti da se u sustavu moraju bilježiti sve javne operacije, što je puno kraće nego da se svaka operacija zasebno modificira kako bi joj se dodao kôd za zapisnik.

Pravila tkanja mogu se specificirati u jeziku koji je proširenje osnovnog jezika. Npr. ako je za implementaciju AOP-a korištena Java, mogu se uvesti proširenja koja se dobro uklapaju u osnovni Java jezik. Ali, moguće je također i koristiti odvojeni jezik koji je, primjerice, temeljen na XML-u.

2.2.2 Implementacija jezika aspektno-orijentiranog programiranja

Prvi korak u implementaciji jezika aspektno-orijentiranog programiranja naziva se tkanje. Ovdje tkalac kombinira pojedinačne dužnosti uz korištenje pravila tkanja. Nakon toga, dobiveni rezultat se pretvara u izvršni kôd.

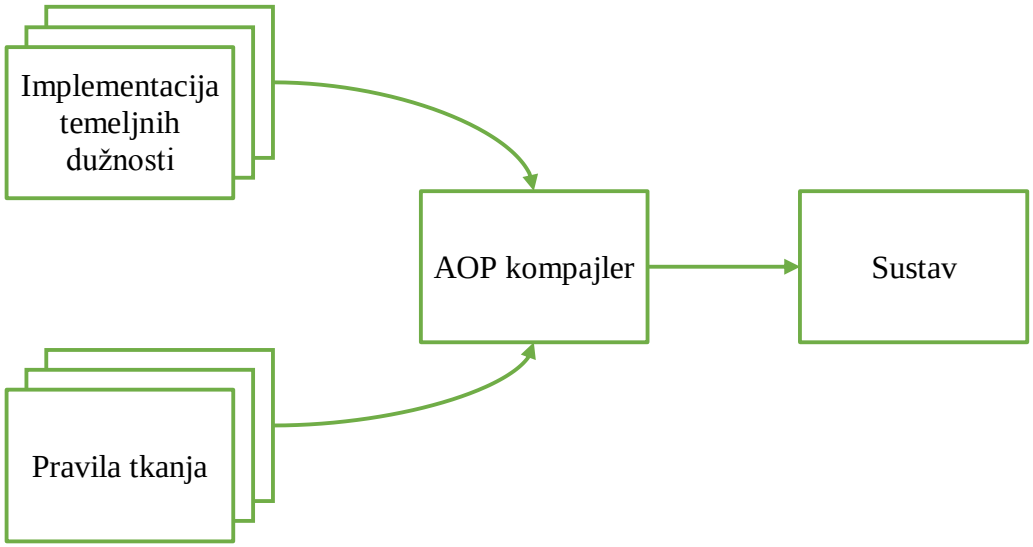
---

[2] Prema: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 22-23.

Korištenjem pravila tkanja definiranih u aspektima se iz osnovnih modula stvara završni sustav kroz proces nazvan tkanje. Izmjenom pravila tkanja u aspektima moguće je jednostavno promijeniti sustav.

Slika broj 5 prikazuje shemu AOP implementacije temeljene na kompajleru.

Slika 5. AOP implementacija koja pruža tkalca u obliku kompajlera. Kompajler prima implementaciju temeljnih i presijecajućih dužnosti i spaja (tka) ih u završni sustav.



Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 25.

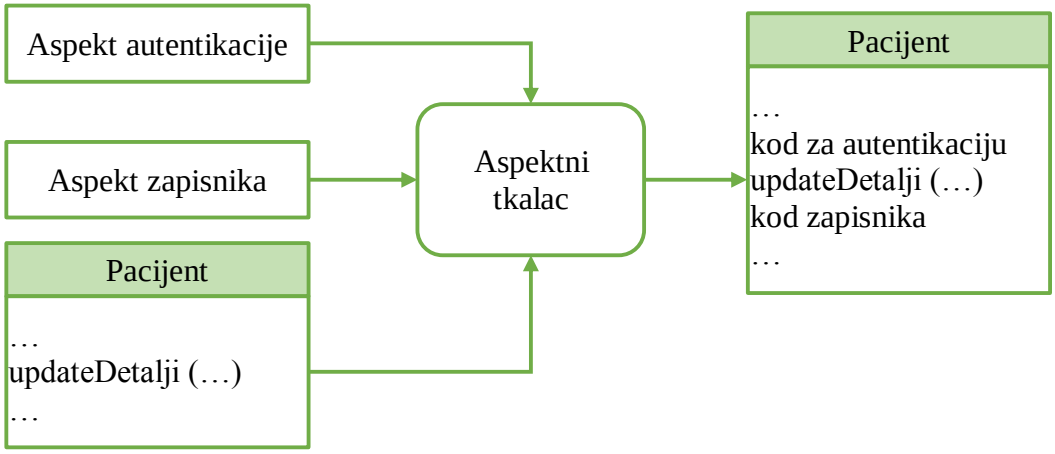
2.2.3 Aspektni tkalac

Aspektni tkalac je proširenje kompajlera i on obavlja tkanje – uključuje savjete na točke spajanja koje su određene u točkama presjeka. Moguće ga je realizirati na više načina. Neovisno o tome koji se način koristi, aspektni tkalac ne modificira originalni izvorni kôd:

- Pretprocesuiranje izvornog kôda (sa izvora na izvor) – izvorni kôd se prvo pretprocesuira aspektnim kompajlerom i stvara se „istkani“ izvorni kôd koji se onda osnovnim kompajlerom jezika pretvori u završni izvršivi kôd. Takav pristup koristi jezik AspectX i njegov tkalac XWeaver (Birrer et al., 2005).
- Istovremeno tkanje (eng. ***link time weaving***) – spajanje aspektnog tkalca sa osnovnim kompajlerom. Aspektno-orijentiran jezik (npr. AspectJ) se obrađuje i generira se standardni Java bytecode koji se onda može izvršiti direktno putem Java interpretera ili se može dalje procesuirati da se generira izvorni strojni kôd.

- Dinamičko tkanje tijekom izvršenja – točke spajanja se nadgledaju tijekom izvršavanja i kada se pojavi neki događaj na kojeg pokazuje određena točka presjeka, na to mjesto u program se integrira odgovarajući savjet.

Slika 6. Aspektno tkanje.



Izvor: Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 575.

2.3 Prednosti AOP-a

AOP je često kritiziran kao težak koncept za shvaćanje, što je djelomično i istina radi vremena koje je potrebno za svladavanje logike. Dio otpornosti leži i u novom načinu razmišljanja o oblikovanju i implementaciji sustava. Neke od prednosti AOP-a[3]:

- *Jasnije odgovornosti individualnog modula* – moduli imaju odgovornost samo za svoju osnovnu dužnost, što za rezultat ima jasniji raspored odgovornosti.
- *Veća modularizacija* – omogućava obradu svake dužnosti zasebno čak i kod presijecajućih dužnosti, što za rezultat ima puno manje redundancije i dupliciranog kôda te smanjenu zapetljanost kôda (eng. ***code clutter***).
- *Lakši razvoj/evolucija sustava* – dodavanje nove funkcionalnosti je pitanje dodavanja novog aspekta i ne zahtjeva promjene na osnovnim modulima. Ako se u sustav doda novi osnovni modul, postojeći aspekti će presijecati i taj novi modul što rezultira bržim odgovorom na nove zahtjeve.

[3] Prema: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 28-29.

- *Naknadno vezivanje (eng.* ***binding****) odluka o oblikovanju* – nije više nužno da dizajner na početku donese odluke o oblikovanju budućih zahtjeva. Može ih naknadno donositi jer je te zahtjeve moguće implementirati kao zasebne aspekte, te time može više pažnje posvetiti trenutnim zahtjevima.
- *Više ponovnog korištenja kôda* – AOP povećava modularizaciju kroz aspekte i tako omogućava slabije vezivanje modula (eng. ***loose coupling***) nego kod konvencionalnih tehnika, a to je ključ u ponovnoj uporabi kôda. Osnovni moduli nisu svjesni jedni drugih – jedino su moduli sa specifikacijom pravila tkanja svjesni bilo kakvog vezivanja/uparivanja.
- *Skraćivanje vremena za isporuku* – naknadno vezivanje odluka o oblikovanju ubrzava proces oblikovanja. Ponovno korištenje kôda skraćuje vrijeme za razvoj. Lakši razvoj sustava ubrzava odgovor na nove zahtjeve. Sve to vodi sustavu kojeg je lakše razviti i isporučiti.
- *Smanjeni troškovi implementacije novih mogućnosti* – AOP pojeftinjuje implementaciju određene presijecajuće dužnosti time što izbjegava trošak za modifikaciju višestrukih modula.

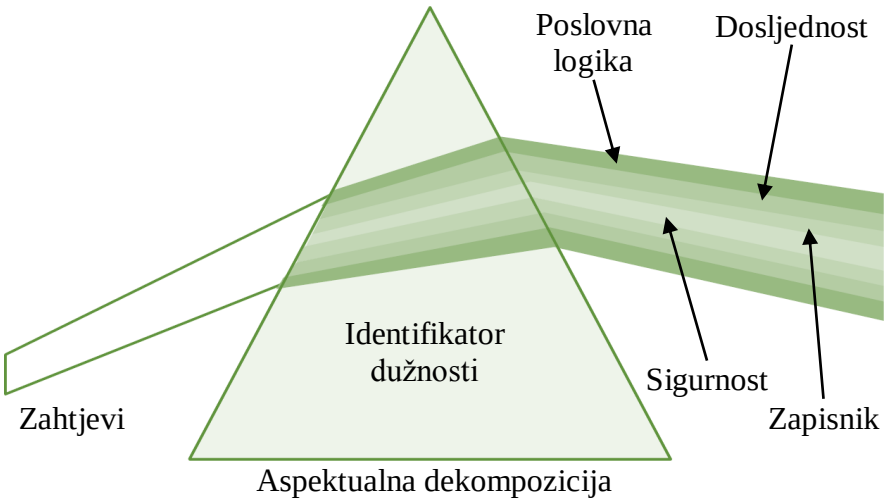
3. Razdvajanje dužnosti

U softverskom inženjerstvu dobra je praksa da se dužnosti (funkcionalnosti) razdvoje. Razdvajanje dužnosti (eng. ***separation of concerns***) podrazumijeva da je svaki element u programu zadužen samo za jednu stvar. Pojam „dužnost“ se može objasniti kao funkcionalni dio – to je neki element funkcionalnosti u sustavu. Opširnije je dužnost moguće opisati kao „bilo koji dio interesa ili fokusa u programu“. U praksi se dužnosti nalaze negdje između prve i druge definicije[4].

Dužnost možemo definirati kao specifičan zahtjev ili razmatranje koje se mora riješiti kako bi se zadovoljio opći cilj sustava. Softverski sustav je realizacija skupine dužnosti[5].

Razdvajanje dužnosti važan je korak u smanjenju složenosti dizajna i implementacije. Na slici broj 7 je pomoću analogije prizme prikazano razdvajanje dužnosti – zraka svjetlosti (zahtjevi) prolazi kroz prizmu (identifikator dužnosti) te se razdvaja na spektar (odvojene dužnosti). Nakon tog procesa, na svaki zahtjev možemo gledati kao na skup osnovnih i presijecajućih dužnosti.

Slika 7. Razdvajanje dužnosti uz pomoć analogije prizme.



Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 9.

[4] Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 567.

[5] Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 7.

Dužnost je moguće definirati kao refleksije zahtjeva sustava i prioriteta dionika u sustavu (Jacobson i Ng, 2004.). Neki dionici (npr. korisnici) žele imati brz odaziv sustava pa u tom slučaju, performanse sustava predstavljaju jednu dužnost, dok neki dionici mogu željeti neku specifičnu funkcionalnost. Tada, dužnost možemo definirati kao nešto što je od interesa ili važnosti jednom dioniku ili više dionika.

Na dužnosti možemo gledati kao na način organiziranja zahtjeva – dužnosti možemo pratiti izražene kao zahtjev ili skup zahtjeva, pa tako možemo izdvojiti nekoliko vrsta dužnosti (zahtjeva) dionika[6]:

- Funkcionalne dužnosti – odnose se na specifičnu funkcionalnost koja se treba uključiti u sustav.
- Dužnosti kvalitete usluge (QoS) – odnose se na nefunkcionalno ponašanje sustava, a uključuje karakteristike kao što su performanse, pouzdanost i dostupnost.
- Političke dužnosti – odnose se na cjelokupne politike koje upravljaju korištenjem sustava. Političke dužnosti uključuju dužnosti pouzdanosti i sigurnosti (eng. ***security and safety***) te dužnosti vezane za poslovna pravila.
- Sustavne dužnosti – odnose se na atribute sustava kao cjeline, kao npr. sposobnost održavanja i sposobnost konfiguriranja.
- Organizacijske dužnosti – odnose se na ciljeve i prioritete organizacije. To uključuje proizvodnju sustava unutar budžeta, korištenje već postojeće softverske imovine, i održavanje reputacije organizacije.

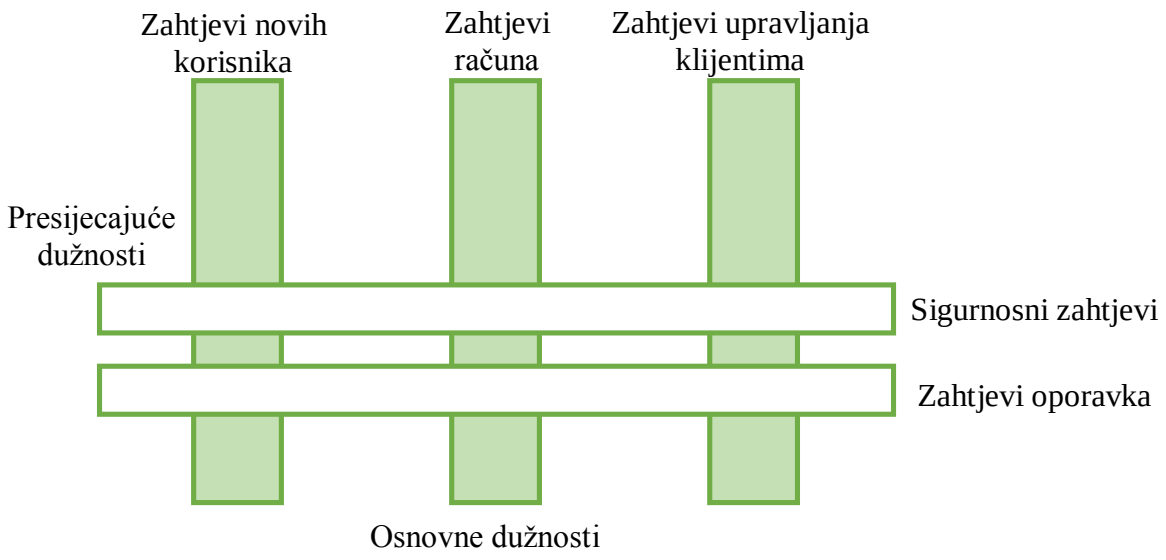
Osnovne (ključne) dužnosti sustava su one *funkcionalne dužnosti* koji se odnose na njegovu primarnu svrhu. Za bolnički informacijski sustav pacijenata centralne funkcionalne dužnosti su stvaranje, uređivanje, dohvaćanje i upravljanje zapisima o pacijentima, dok u sustavu kontrole vlaka specifična funkcionalna dužnost je kočenje vlaka.

Također, postoje i *sekundarne* funkcionalne dužnosti. Recimo da sustav ima zahtjev omogućiti istovremeni pristup zajedničkom/dijeljenom spremniku (eng. ***buffer***). Jedan proces dodaje podatke u spremnik a drugi ih vadi. U ovom primjeru, osnovna je dužnost održavanje dijeljenog spremnika. Međutim, kako se procesi ne bi miješali potrebna je sekundarna dužnost – sinkronizacija – proces koji dodaje podatke ne smije pisati preko podataka koji nisu iskorišteni, a proces koji vadi podatke ne može vaditi iz praznog spremnika.

[6] Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 568.

Postoje i druge dužnosti koje odražavaju zahtjeve sustava: kvaliteta usluge (QoS) i organizacijska politika – to su *sustavne dužnosti* jer se odnose na sustav kao cjelinu, a ne na pojedinačne zahtjeve. Kako bi ih razlikovali od centralnih dužnosti, nazivamo ih presijecajućim dužnostima.

Slika 8. Presijecajuće dužnosti.



Izvor: Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 569.

Sustav iz primjera na slici broj 8 kao primarnu svrhu ima proviziju bankarskog sustava, i uz nju su povezane centralne dužnosti. Osim zahtjeva za upravljanjem računima klijenata i zahtjeva za upravljanje postojećim klijentima, postoje i zahtjevi vezani za nove klijente: provjera adrese, provjera kreditiranja, i slični.

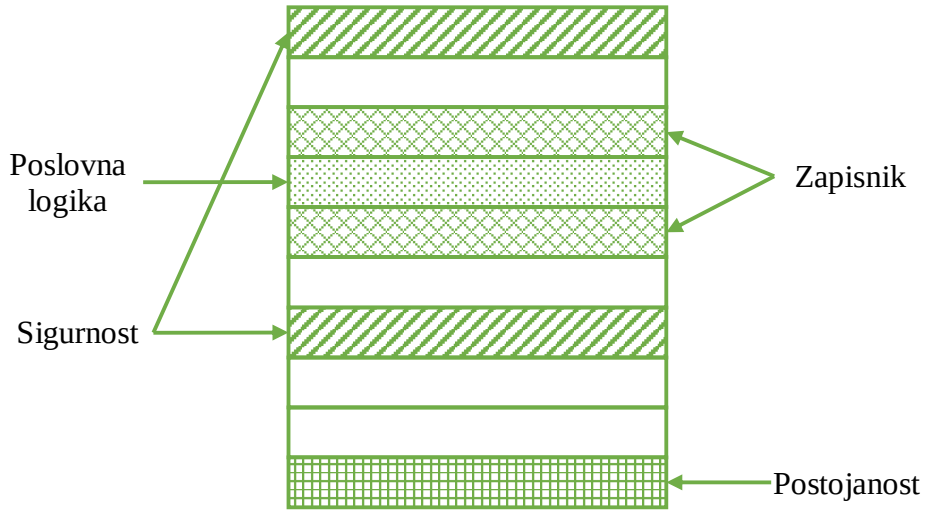
U tom sustavu postoje i dužnosti koje mogu utjecati na implementaciju ostalih zahtjeva, a to su primjerice sljedeće presijecajuće dužnosti: kako bi se osiguralo da se podaci ne izgube ako se dogodi pad sustava, prisutni su i zahtjevi oporavka (eng. ***recovery***), a radi politike sigurnosti banke prisutni su i sigurnosni zahtjevi.

Konvencionalne tehnike (OOP) dobro implementiraju osnovne dužnosti sustava, ali potreban je i dodatan kôd za implementaciju presijecajućih dužnosti, funkcionalnih dužnosti, dužnosti kvalitete usluge, te političkih dužnosti. Taj dodatan kôd unosi probleme u sustav: **zaplitanje** i **raspršivanje** (eng. ***tangling***, ***scattering***).

3.1 Zaplitanje koda

Do zaplitanja kôda (eng. ***code tangling***) dolazi kada modul sadrži kôd za implementaciju različitih zahtjeva sustava, odnosno kada modul odrađuje više dužnosti istovremeno. Osim kôda koji implementira osnovnu dužnost, u modulu se nalazi i kôd za implementaciju presijecajućih dužnosti. To se dešava kada se tijekom implementacije uzimaju u obzir dužnosti kao što su sigurnost, performanse, poslovna logika, sinkronizacija i dr. Primjer zaplitanja možemo vidjeti na slici broj 9.

Slika 9. Zaplitanje kôda uzrokovano simultanim implementacijama više različitih dužnosti.



Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 16.

Na kôdu iz primjera broj 1 je prikazano dodavanje stavke u spremnik pomoću operacije ***PUT***. Ako je spremnik pun, potrebno je čekati dok određena operacija ***GET*** ne isprazni spremnik. Za sinkronizaciju se koriste pozivi funkcija ***wait()*** i ***notify()***. Kôd za implementaciju funkcije spremnika je zapleten sa kôdom za implementaciju sinkronizacije koji mora biti prisutan u svim metodama koje pristupaju zajedničkom spremniku.

Primjer 1. Zaplitanje kôda za upravljanje spremnikom i kôda za sinkronizaciju.

```
synchronized void put (SensorRecord rec )
{
      if ( numberOfEntries == bufsize)
            wait () ;
      store [back] = new SensorRecord (rec.sensorId, rec.sensorVal) ;
      back = back + 1 ;
      if (back == bufsize)
            back = 0 ;
      numberOfEntries = numberOfEntries + 1 ;
      notify () ;
}
```

Izvor: Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 570.

3.2 Raspršivanje koda

Raspršivanje kôda (eng. ***code scattering***) se pojavljuje kada se jedna dužnost (zahtjev ili grupa zahtjeva) implementira kroz više modula, odnosno kada je raspršena preko više komponenata u programu.

Za presijecajuće dužnosti je normalno da budu raširene na više modula, pa su i njihove implementacije raširene kroz te module. Za sustav koji koristi bazu podataka, primjerice, dužnosti performansi mogu utjecati na sve module koji pristupaju bazi podataka.

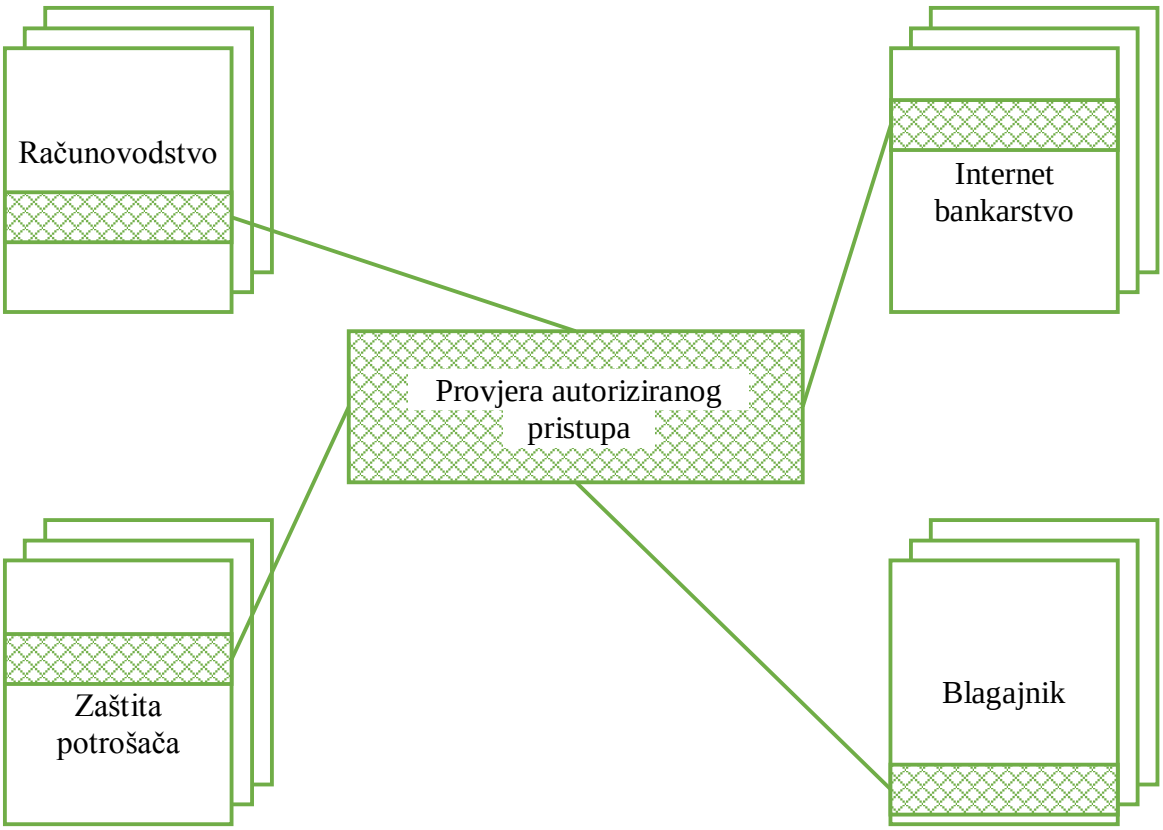
Raspršivanje kôda možemo podijeliti u dvije vrste:

- Dvostruki blokovi kôda
- Nadopunjavajući blokovi kôda.

3.2.1 Dvostruki blokovi kôda

Primjer za dvostruke blokove kôda je prikazan na slici broj 10 koja prikazuje sustav u kojem samo autorizirani korisnici smiju pristupiti servisu. Da bi se to postiglo, više modula u sustavu moraju sadržavati određeni kôd za provjeru autorizacije.

Slika 10. Raspršivanje kôda prouzrokovano kopiranjem skoro identičnih blokova kôda kroz više modula radi implementacije određene funkcionalnosti.

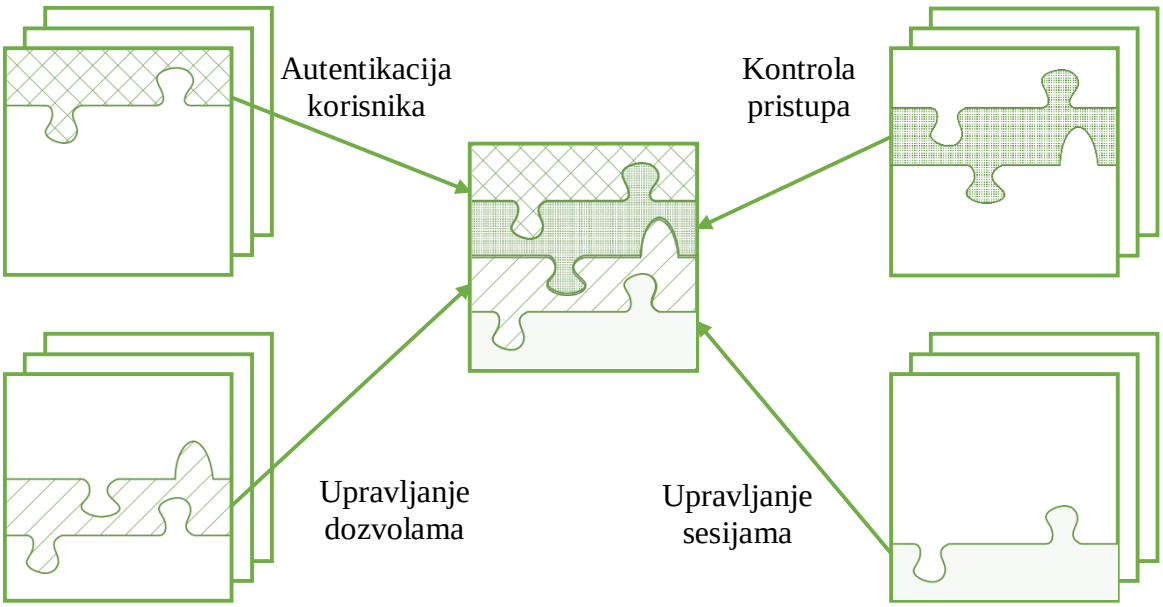


Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 17.

3.2.2 Nadopunjavajući blokovi kôda

Kod kontrole pristupa, autentikacija se odrađuje u jednom modulu, autenticiranog korisnika se prosljeđuje drugom modulu (kojem je potrebna autorizacija), pa zatim taj modul odrađuje autorizaciju. Tu nekoliko modula implementiraju nadopunjavajuće dijelove dužnosti kako bi se pružala određena funkcionalnost. Da bi autorizacija bila ispravno implementirana, moduli trebaju raditi zajedno, kako je to prikazano na slici broj 11.

Slika 11. Raspršivanje kôda prouzrokovano ugradnjom nadopunjavajućih blokova kôda kroz više modula.



Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 17.

Raspršivanje i zaplitanje kôda su pojave do kojih dolazi kada dođe do promijene početnih zahtjeva sustava. Ako imamo bankarski sustav koji bilježi svaku transakciju, te ako je u taj sustav potrebno ubaciti nove statističke podatke, promjene neće biti centralizirane samo na jednom mjestu. Potrebno je pronaći svaki element i zatim primijeniti svaki od pronađenih elemenata radi dodavanja izmjena. Iz navedenog, vidimo da taj postupak može biti skup, ali također dovodi do eventualnog izostavljanja dijela kôda i samim time neželjenih pojava greška u sustavu.

Za primjer uzmimo sustav medicinskih zapisa u kojem se održavaju informacije o pacijentu, lijekovima koji mogu biti prepisani, itd. Recimo da se sva ažuriranja u bazi podataka mogu obaviti metodama koje počinju sa „*update*“:

Primjer 2. Update.

```
updateOsobneInformacije (pacijentId, informacijaUpdate)
updateLijek (pacijentId, lijekUpdate)
```

Prema: Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 572.

U prvom parametru (*pacijentId*) se identificira pacijent, a u drugom (*informacijaUpdate*) su unesene potrebne promjene. Takvo ažuriranje može obavljati samo osoblje bolnice koje je prijavljeno u sustav. Ako imamo slučaj gdje je zlonamjerno promijenjena informacija o pacijentu, moguće je da se to desilo u trenutku kada se član osoblja udaljio od računala bez prethodne odjave. Kako bi se to spriječilo potrebno je uvesti novu sigurnosnu politiku – ponovna autentikacija prije bilo kakve izmjene informacija u bazi pacijenata uz bilježenje svih promjena u zapisnik (*log*). Dva su moguća načina provođenja takve politike:

- Modifikacijom „*update*“ metode u svakoj komponenti tako da se doda kôd za pozivanje metoda za autentikaciju i zapisnik
- Modifikacija sustava na način da se prije poziva „update“ metode doda poziv za autentikaciju, a nakon „*update*“ funkciju poziv za metodu zapisnika.

Prvi način dovodi do pojave ***zapetljanog*** kôda (dužnosti autentikacije, ažuriranja baze te zapisnika su nepovezane dužnosti i moraju se moći koristiti odvojeno), a drugi pristup dovodi do ***raspršivanja*** kôda (kôd za pozive metoda autentikacije i zapisnika se nalazi na više različitih mjesta u sustavu). Ti problemi se rješavaju dodavanjem aspekta kao u primjeru broj 3. Aspekt iz primjera će biti utkan u sustav na svako mjesto gdje se koristi poziv metode „*update*“.

Primjer 3. Aspekt autentikacije.

```
aspect autentikacija {
      before: call (public void update* (..)) // prije poziva metode
update
      {
            // kôd autentikacije
      }
}
```

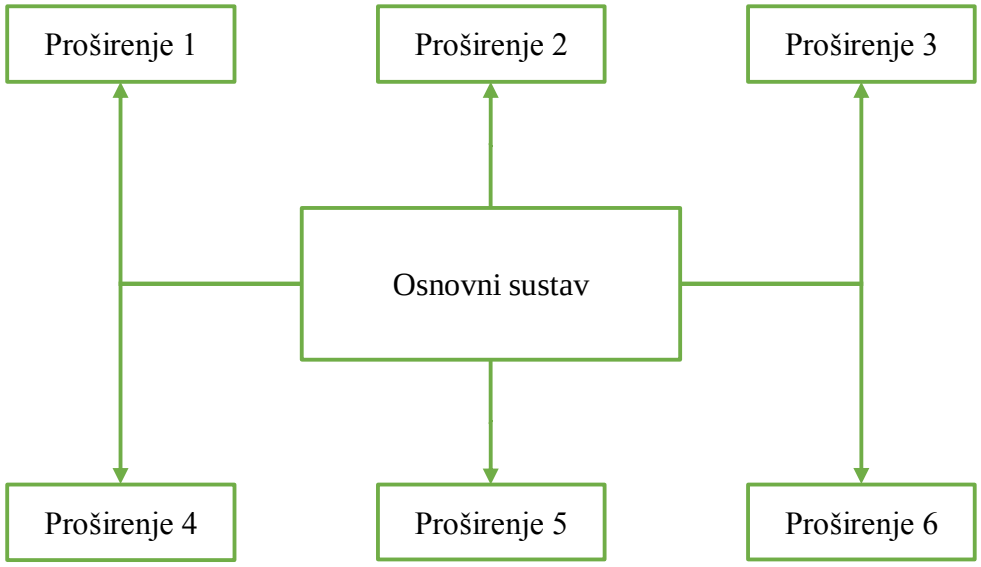
Prema: Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 573.

4. Softversko inženjerstvo pomoću aspekata

Dužnosti odražavaju zahtjeve, pa je logično da se aspektno-orijentiran pristup ne koristi samo u programiranju nego i u ranijim fazama procesa razvoja sustava. Aspektno-orijentirani jezici su samo tehnička podrška za dobro identificirane i modelirane zahtjeve te za implementaciju razdvojenih dužnosti.

Sustav koji podržava dužnosti različitih dionika možemo zamisliti kao centralni (eng. ***core***) sustav uz dodatke / proširenja (eng. ***extension***) kao što je prikazano na slici broj 12. Proširenja dodaju novu funkcionalnost na osnovni sustav, a AOP je jedan način implementacije tih proširenja jer omogućava da se spoje sa osnovnim sustavom pomoću tkanja.

Slika 12. Osnovni sustav sa proširenjima.



Izvor: Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 577.

U osnovni sustav ulaze dužnosti koje obavljaju glavnu svrhu sustava. Za primjer uzmimo informacijski sustav fakulteta čija je osnovna svrha održavanje informacija o studentima. Njegov osnovni sustav omogućavati će uređivanje i upravljanje bazom podataka zapisa o studentima. Jedno od proširenja moglo bi biti dužnost za enkripcijom i kontrolom pristupa.

Možemo izdvojiti nekoliko vrsta proširenja koja proizlaze iz ranije spomenutih vrsta dužnosti[7]:

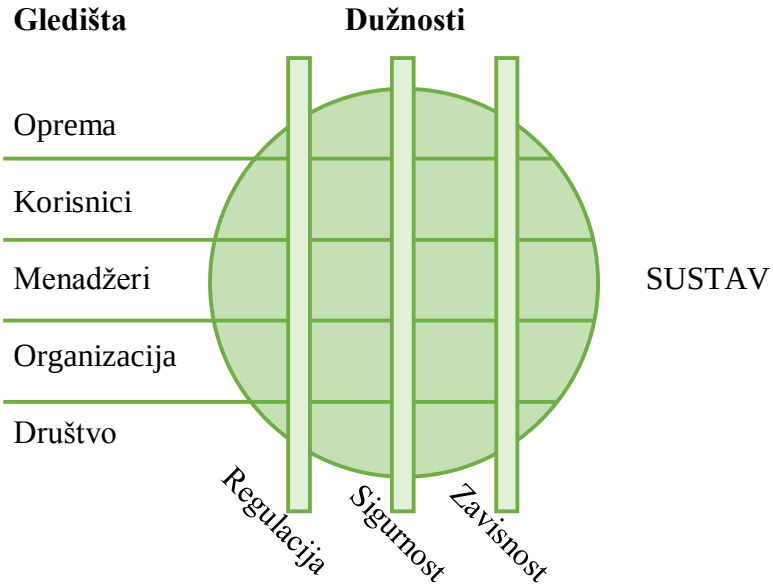
- Sekundarna funkcionalna proširenja – dodaju mogućnosti na postojeće funkcionalnosti osnovnog sustava. Primjer: generiranje izvještaja o položenim ispitima u IS-u fakulteta.
- Proširenja politike – dodaju funkcionalne mogućnosti za potporu organizacijskim politikama. Primjer: dodavanje sigurnosnih značajki kao što je izrada pričuvne kopije podataka (eng. ***backup***) kopije za slučaj pada sustava.
- Proširenja kvalitete usluge – dodaju funkcionalne mogućnosti kako bi se udovoljilo zahtjevima kvalitete koji su specificirani za sustav. Primjer: dodavanje predmemorije (eng. ***cache***) za brži pristup bazi podataka.
- Proširenje infrastrukture – dodaju funkcionalne mogućnosti koje podržavaju implementaciju sustava na neke specifične implementacijske platforme. Primjer: za postojeći sustav koji koristi bazu podataka, može se dodati sučelje (eng. ***interface***) za direktno upravljanje bazom podataka.

4.1 Dužnostima-orijentirano inženjerstvo zahtjeva

Kako bi cijeli AOP koncept imao smisla, potrebno je da se i u razmatranju zahtjeva koristi takav pristup koji će razdvojiti različite dužnosti svih dionika. Potrebno je identificirati zahtjeve osnovnog sustava i zahtjeve za proširenja sustava, odnosno definirati centralne i sekundarne dužnosti, i organizirati ih ovisno o točkama gledišta dionika. Točke gledišta predstavljaju skupove funkcionalnosti koji su potrebni određenim dionicima u sustavu, pa će tako postojati i zahtjevi koji se presijecaju kroz više dionika, kao što je prikazano na primjeru sa slike broj 13. Nakon sortiranja po gledištima, moguće je uočiti zahtjeve koji se protežu kroz sva gledišta, a oni čine centralnu funkcionalnost sustava. Oni zahtjevi koji se pojavljuju samo u nekim gledištima predstavljaju proširenja. Korištenjem takvog pristupa, proces inženjerstva zahtjeva rezultirati će zahtjevima sortiranim oko centralnog sustava sa proširenjima, i kao takvi su potpora daljnjem aspektno-orijentiranom razvoju sustava.

[7] Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 576.

Slika 13. Gledišta i dužnosti.

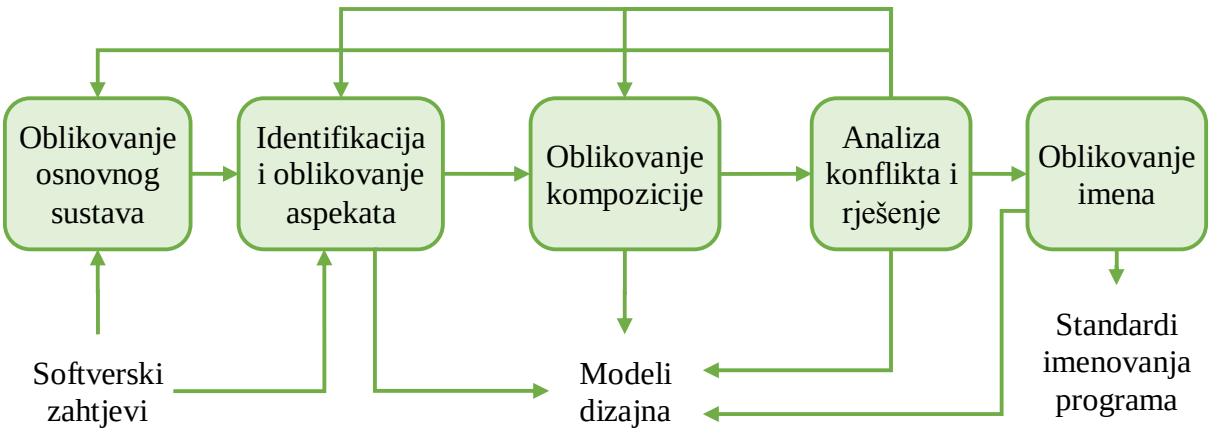


Izvor: Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 578.

4.2 Aspektno-orijentirano oblikovanje i programiranje

Nakon inženjerstva zahtjeva potrebno je oblikovati sustav koji će koristiti aspekte za implementaciju presijecajućih dužnosti i proširenja. U tom procesu stvaraju se prijedlozi dizajna koji se zatim mijenjaju ovisno o daljnjoj analizi. Tu je potrebno prevesti dužnosti u aspekte i pritom paziti da ne dođe do dvosmislenosti u sustavu.

Slika 14. Proces aspektno-orijentiranog oblikovanja.



Izvor: Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 582.

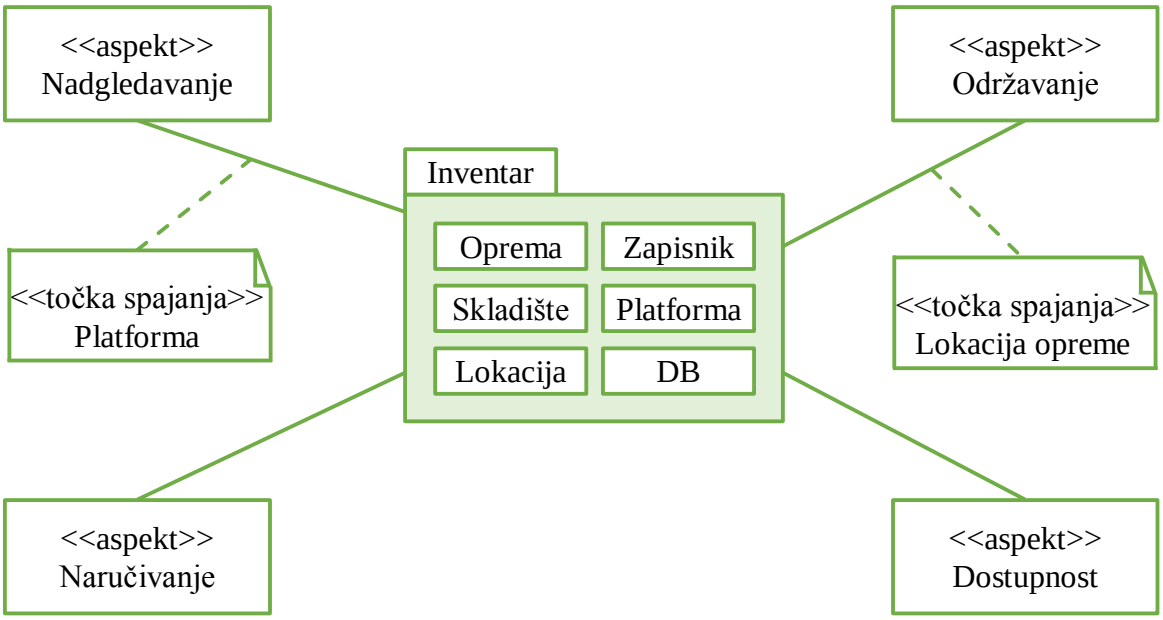
Kako je prikazano na slici broj 14, Sommerville predlaže sljedeće aktivnosti u aspektno-orijentiranom procesu oblikovanja[8]:

- **Oblikovanje osnovnog sustava** – oblikuje se arhitektura sustava koja podržava centralnu funkcionalnost sustava. Arhitektura mora uzeti u obzir zahtjeve za kvalitetom usluge (QoS) kao što su zahtjevi za performansama i pouzdanosti.
- **Identifikacija i oblikovanje aspekata** – analiza proširenja koja su identificirana kroz zahtjeve sustava kako bi se doznalo da li su sami po sebi aspekti ili ih je potrebno razdvojiti na više aspekata. Kada su aspekti identificirani, moguće ih je zasebno oblikovati.
- **Oblikovanje kompozicije** – identificiraju se točke spajanja kroz analizu osnovnog sustava i oblikovanje aspekta (određuje se gdje će se aspekti ugraditi u osnovni sustav).
- **Analiza konflikata i rješenje** – aspekti mogu međusobno utjecati jedni na druge (zapliću se). Do konflikta dođe ako postoji sudaranje točka presjeka: različiti aspekti imaju specificirano da se ugrade na isto mjesto (točku spajanja). Također, ako se oblikuju odvojeno, aspekti mogu donositi pretpostavke o funkcionalnosti osnovnog sustava koju je potrebno izmijeniti. Kada postoji više takvih aspekata, jedan može utjecati na funkcionalnost sustava na takav način kakvog ne očekuju drugi aspekti.
- **Oblikovanje naziva** – definira standarde za imenovanje entiteta u programu. Važan korak u izbjegavanju problema sa slučajnim točkama presjeka do kojih dolazi ako u nekim točkama spajanja naziv slučajno odgovara obrascu točke presjeka pa se tada savjet (*advice*) slučajno primijeni na tu točku i tako dolazi do neočekivanog ponašanja programa.

Rezultat tog procesa jest aspektno-orijentiran dizajn modela kojeg se može prikazati pomoću UML-a kao na primjeru sa slike broj 15 na kojoj je prikazan osnovni sustav za inventar hitne službe sa nekim aspektima koji bi mogli biti ukomponirani sa tim osnovnim sustavom.

[8] Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 582.

Slika 15. Aspektno-orijentiran dizajn modela.



Izvor: Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 583.

4.3 Verifikacija i validacija

Verifikacija je proces koji pokazuje da program odgovara zahtjevima korisnika (da odgovara specifikaciji) i daje odgovor na pitanje „da li dobro gradimo proizvod?“, a **validacija** je proces koji pokazuje da program odgovara stvarnim potrebama korisnika (program treba činiti ono što korisnik stvarno želi) i daje odgovor na pitanje „da li gradimo dobar proizvod?“. U procesima verifikacije i validacije postoje dvije metode:

- Statička verifikacija i validacija – kontrola softvera, tj. otkrivanje problema kroz analizu statičkog prikaza sustava. Provodi se bez izvođenja programa a provjerava se specifikacija zahtjeva (dokumenti), dijagrami oblikovanja te izvorni kôd.
- Dinamička verifikacija i validacija – testiranje sustava, tj. ispitivanje i promatranje ponašanja programa. Provodi se uz izvođenje programa na zadanim podacima.

Dvije često korištene metode testiranja su „black box“ i „white box“ testiranje. Proces validacije se provodi tako da se prema sustavu odnosi kao prema „crnoj kutiji“ (eng. ***black box***) te se testovima potvrđuje da li sustav odgovara zahtjevima. Međutim, poteškoće se pojavljuju kod provođenja pregledavanja programa i testiranja „bijelom kutijom“ (eng. ***white box***).

Kod pregledavanja programa, grupa čitača pregledava izvorni kôd od vrha prema dnu (sekvencijalno) i pronalazi eventualne greške u kôdu. Kod aspektno-orijentiranih sustava to nije moguće, već se prvo moraju proučiti svi aspekti, razumjeti njihove točke presjeka kao i model točaka spajanja tog jezika. Programer koji čita AOP kôd treba biti u mogućnosti pronaći svaku potencijalnu točku spajanja tako da provjeri kôd aspekta da utvrdi da li se taj aspekt može utkati na tu potencijalnu točku. Taj proces zahtjeva veliku koncentraciju i rasuđivanje, pa je testiranje jednog aspektno-orijentiranog programa moguće jedino uz alate za čitanje kôda koji „poravnavaju“ aspektni kôd u izvorni kôd koji sadrži već utkane aspekte i tako olakšava čitanje. Također, alat za čitanje mora znati kako će aspektni tkalac riješiti problem kada se pojave dva aspekta sa istom specifikacijom točke presjeka. Također, model točaka spajanja može biti dinamičan i nemoguće je pokazati da će se stvarni program izvršiti isto kao i „izravnani“ program.

White-box (strukturalno) testiranje, se provodi tako da se namjerno naprave defektni testovi za poznati izvorni kôd. Kod takvog testiranja koriste se analizatori programa kojima se pokušava pokriti svaki logični put kroz program i da se sve naredbe i uvjeti izvrše bar jednom. Međutim, postoje problemi kod tog načina testiranja aspektno-orijentiranih sustava. Jedan od problema je što za razliku od sekvencijalnih strukturiranih programa, kod aspektno-orijentiranog programa je teško dobiti graf toka podataka. Tijekom izvršenja programa postoji mogućnost da će se na određenoj točki spajanja uključiti neki aspekt, i samim time nije osigurano da će testom biti pokriven svaki logički put kroz program.

Kada je riječ o velikim sustavima, uobičajeno je da postoje timovi za osiguravanje kvalitete koji određuju testove i ocjenjuju uspješnost projekta po tim postavljenim testovima.

Velika prepreka u prihvaćanju aspektno-orijentiranog razvoja softvera su baš takve vrste testiranja i nemogućnost dobivanja testova, a oba problema nastaju jer se aspekti ne mogu zasebno testirati iz razloga što su oni usko vezani uz osnovni kôd. Jedan aspekt može biti utkan u program na više različitih mjesta ali se ne može reći da ako radi na jednoj točki spajanja da će isto raditi i na drugoj točki spajanja.

Sommerville navodi sljedeća pitanja koja ostaju predmet istraživanja u aspektno-orijentiranom razvoju softvera[9]:

[9] Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 586.

- Kako bi aspekti trebali biti specificirani da bi se mogli derivirati testovi za te aspekte?
- Kako se aspekti mogu testirati odvojeno od osnovnog sustava u kojem bi trebali biti utkani?
- Kako se može testirati interferencija (miješanje) aspekata?
- Kako treba oblikovati testove a da se izvrše sve točke spajanja i da se primijene testovi pripadajućih aspekata?

5. Konstrukti: aspekti, točke spajanja i točke presjeka

Terminologija koja se koristi u AOP-u danas nastala je iz AspectJ-a u kasnim 1990-ima. AspectJ koristi proširenja Java programskog jezika. AspectJ proširenja koriste konstrukte iz tablice broj 1 za specifikaciju pravila tkanja.

Tablica 1. Terminologija u aspektno-orijentiranom softverskom inženjerstvu.

Pojam	Definicija
Aspect	Aspekt – programska apstrakcija koja definira presijecajuću dužnost. Sadrži definiciju točke presjeka te savjet vezan za tu dužnost.
Advice	Savjet – kôd koji implementira dužnost.
Join point	Točka spajanja – događaj u izvođenom programu gdje se savjet vezan uz određenu dužnost može izvršiti.
Join point model	Model točka spajanja – skup događaja na koje može upućivati točka presjeka.
Pointcut	Točka presjeka – izjava, uključena u aspektu, koja definira točke spajanja na kojima se povezani savjeti aspekta trebaju izvršiti.
Weaving	Tkanje – ugradnja kôda savjeta na specificiranim točkama spajanja putom aspektnog tkalca.

Izvor: Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 572.

5.1 Aspekt

Glavni pojam u AOP-u jest aspekt. Kao što je klasa ključni element Jave, tako je Aspekt ključni element AspectJ-a. U aspektu se nalazi kôd kojim se opisuju pravila tkanja. Aspekt sadrži točke presjeka, savjete, uvode i deklaracije, kao i podatke, metode i ugniježdene članove klasa. Aspekt sadrži kôd za implementaciju neke presijecajuće dužnosti, ali on također ima i točku presjeka koja definira gdje se taj kôd treba utkati u program. Aspekt dakle zna **što** i **gdje** treba utkati, dok druge apstrakcije (npr. metode) ne mogu znati kada će one biti pozvane jer se kôd za njihovo pozivanje može nalaziti bilo gdje u programu. Posebnost aspekta je u mogućnosti da se odredi (pomoću točke presjeka), gdje će se kôd izvršiti.

Primjer 4. Aspekt

```
public aspect ExampleAspect {
      before() : execution(void Account.credit(float)) {
            System.out.println(„About to perform credit operation“);
      }
      declare parents: Account implements BankingEntity;
      declare warning: call (void Persistence.save(Object))
            : "Consider using Persistence.saveOptimized()";
}
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 36.

Kod oblikovanja aspekta, prvo je potrebno odrediti mjesta gdje se želi izmijeniti ponašanje i nakon toga se oblikuje novo ponašanje. Kod implementacije, prvi korak je napisati aspekt koji će enkapsulirati presijecajuću dužnost. Zatim se u taj aspekt dodaju točke presjeka koje će dohvaćati željene točke spajanja, a na kraju se piše savjet koji dodaje novu funkcionalnost.

5.2 Točka spajanja

Točka spajanja (eng ***join point***) je neko mjesto ili događaj kojeg se može identificirati tijekom izvršavanja programa. Točka spajanja je mjesto na koje se točka presjeka može referencirati. To može biti poziv metode, dodjela članu nekog objekta, inicijalizacija varijable, *return* izjava, usporedba, *handler* iznimka, ažuriranje polja, te *while* i *do-while* petlje. To su mjesta na koja će se utkati presijecajuće aktivnosti.

Primjer 5. Točke spajanja u kôdu.

```
public class Account {
      …
      void credit (float amount) {
            _balance += amount;
      }
}
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 34.

Točke spajanja iz gornjeg primjera u klasi **Account** sadrže izvršavanje metode **credit ()** i pristup **_balance** članu instance.

5.3 Točka presjeka

Točka presjeka (eng. ***pointcut***) je izjava kojom se definira gdje će se aspekt utkati u program. Točke presjeka služe kako bi se odredio tip točke spajanja. To je konstrukt (deklaracija) koji označava točke spajanja i sakuplja kontekst na tim točkama. Točka presjeka se sastoji od izraza za identificiranje točke spajanja u programu.

Točka presjeka može označiti točku spajanja (koja je poziv metode) i može također dohvatiti kontekst metode te argumente metode. Kontekst metode može biti npr. ciljani objekt na kojem je metoda pozvana. Točka presjeka jednoznačno identificira specifične događaje u programu na koje će se savjet utkati.

Primjer 6. Točka presjeka.

```
before() : call (public void update* (..))
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 35.

Ovaj primjer znači sljedeće: prije svakog izvršenja svake metode čije ime počinje sa riječi „update“ i nastavlja sa bilo kojom kombinacijom znakova (znak „*“ je višeznačnik i označava bilo koji dozvoljeni niz znakova), kôd u aspektu se treba izvršiti.

Razlika između točke spajanja i točke presjeka je u sljedećem: točka presjeka specificira pravila tkanja, a točka spajanja je situacija koja zadovoljava ta pravila.

5.4 Savjet

Savjetom (eng. ***advice***) se naziva kôd koji se treba izvršiti na točki spajanja koju je označila točka presjeka. Savjet je kôd za implementaciju presijecajuće dužnosti. Savjet je sličan metodi – on sadrži logiku koja će se izvršiti kada se stigne do točke spajanja.

Postoje tri vrste savjeta:

- ***Before*** – izvršava se prije točke spajanja
- ***After*** – izvršava se nakon točke spajanja
- ***Arround*** – okružuje izvršenje točke spajanja.

Primjer 7. Savjet "*before*".

```
before() : call(* Account.*(..)) {
            … autentikacija korisnika
      }
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 83.

Na gornjem primjeru, savjet će izvršiti autentikaciju korisnika prije izvršenja bilo koje metode iz klase Account. Ukoliko *before* savjet baci iznimku (eng. ***exception***), označena operacija (u gornjem primjeru metoda ***call()***) se neće izvršiti.

Primjer 8. Savjet "*after*".

```
after() : call(* Account.*(..)) {
      … bilježiti u zapisnik povratnu informaciju operacije
}
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 83.

Na gornjem primjeru, savjet će pozvati kôd za zapisnik nakon izvršenja metode ***call()*** iz klase Account. AspectJ još nudi i mogućnost da se savjet izvrši nakon uspješnog izvršenja točke spajanja. Pri tome se koristi ključna riječ ***after() returning*** kao u sljedećem primjeru.

Primjer 9. Ključna riječ "*after returning*".

```
after() returning : call(* Account.*(..)) {
      … bilježi uspješnu operaciju
}
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 24.

Zadnja vrsta savjeta je savjet ***arround***. Specifičnost savjeta *arround* jest da on može izmijeniti kôd koji se izvršava na točki spajanja, može ga zamijeniti ili zaobići. Točke presjeka i savjeti zajedno tvore pravila dinamičkog presijecanja.

5.5 Model točaka spajanja

Model točka spajanja (eng. ***joint point model***) je skup događaja na koje se može referencirati u programu. Nije se moguće referencirati na sve točke spajanja već samo na one koje su **izložene** (eng. ***exposed***). Primjerice, u AspectJ-u se ne možemo referencirati na petlje, a razlog tome jest što ih se lako može modificirati/pretvoriti (npr. *for* petlju u *while* petlju) pa tada svi savjeti na točku spajanja za *for* petlju više ne bi valjali jer sama *for* petlja više ne bi postojala. Iz primjera vidimo da modeli točka spajanja nisu standardni nego svaki programski jezik ima vlastiti model točaka spajanja. U AspectJ-u, model točka spajanja uključuje ove događaje[10]:

- Call events – događaji poziva. Poziv metode ili konstruktora.
- Execution events – događaji izvršenja. Izvršenje metode ili konstruktora.
- Initialization events – događaji inicijalizacije. Inicijalizacija klase ili objekta.
- Data events – događaji podataka. Pristupanje polju ili ažuriranje polja.
- Exception events – događaji iznimaka. Rukovanje iznimkom.

5.6 Uvod

Uvod (eng. ***introduction***) je statička uputa koja uvodi promjene u postojeće klase, sučelja i aspekte sustava. Daje novu funkcionalnost aplikaciji tako da definira nove atribute i metode u već postojeće klase. Uvod vrši statičke promjene koje ne utječu direktno na ponašanje modula, npr. dodavanje metode ili polja u klasu. Uvod iz sljedećeg primjera deklarira klasu **Account** koja implementira sučelje **BankingEntity**.

Primjer 10. Uvod sa statičkom promjenom.

```
declare parents: Account implements BankingEntity;
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 36.

Prilikom izvođenja, ponašanje klase koja već ne sadrži potreban atribut ili metodu, nije moguće izmijeniti bez korištenja deklaracija AspectJ uvoda.

[10] Sommerville, I. (2010.) Software Engineering. Ninth Edition. Boston: Addison-Wesley, str. 574.

6. O AspectJ-u

Aspektno-orijentirano programiranje je započelo 1997. godine u Xerox PARC laboratorijima razvojem programskog jezika AspectJ. Članovi Palo Alto istraživačkog centra (PARC – *Palo Alto Research Center* – dio Xerox korporacije), Christina Lopez i Gregor Kiczales bili su među prvima koji su uveli tu novu metodologiju. Godine 1996. je Gregor Kiczales tu metodologiju nazvao aspektno-orijentirano programiranje. Prvu implementaciju AspectJ-a su članovi PARC-a napravili krajem devedesetih, a prva javna verzija izašla je 2001. godine. Kasnije je AspectJ projekt, pod vodstvom Adriana Colyera, prebačen na Eclipse Open Source zajednicu koja dalje radi na poboljšanju projekta.

Osim AspectJ-a: postoje i druge implementacije AOP-a: HyperJ, AspectWerkz, Nanning, JBoss AOP. Postoje i implementacije za druge jezike: AspectC++, AspectC#, .NET framework jezici (C#, VB.NET) itd. AspectJ je najrasprostranjeniji jer se ističe kvalitetom i brojnom razvojnom zajednicom.

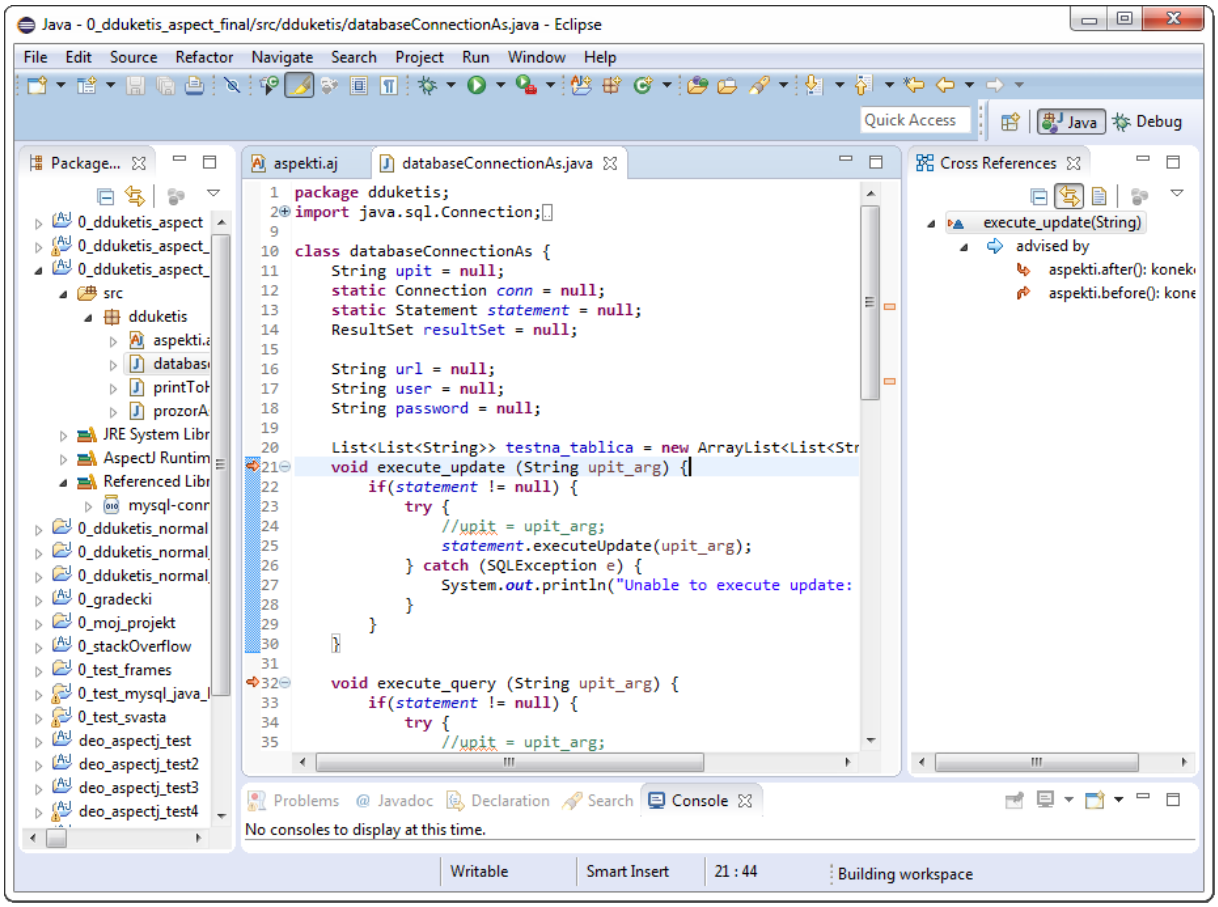
AspectJ je jezik otvorenog kôda koji funkcionira kao dodatak na Java jezik. Kao što su Java, C++, Visual Basic implementacije OOP-a, tako je AspectJ implementacija AOP-a. AspectJ je u potpunosti kompatibilan sa Javom: aspekti jezika AspectJ rade usporedno s Java klasama kako bi se stvorila opsežna i jasna aplikacija. Prednosti korištenja AspectJ uz Javu:

- Manje zapletenog kôda
- Kraći kôd
- Lakše održavanje i evolucija aplikacije
- Lakše otklanjanje neispravnosti (*debug*), refaktoriranje[11] i modifikacija
- Kôd koji je za višekratnu uporabu – programeri mogu stvarati biblioteke aspekata na isti način na koji se stvaraju biblioteke objekata u OOP.

AspectJ je aspektno-orijentirano proširenje Java programskog jezika. Budući da je AspectJ proširenje Jave, svaki ispravan Java program je ujedno i validan AspectJ program.

[11] Refaktoriranje – proces kojim se mijenja unutarnja struktura programa bez mijenjanja njegovog vanjskog ponašanja. Program se obično refaktorira radi poboljšanja čitljivosti te lakšeg održavanja kôda.

Slika 16. AspectJ IDE.



Na slici broj 16 prikazano je razvojno okruženje AspectJ-a. Na slici se vidi dio kôda aplikacije iz priloga: AspectJ unutar Eclipse sučelja pomoću strelica jasno prikazuje mjesta na koja se uključuje savjet.

AspectJ se sastoji od dva dijela: specifikacija jezika i implementacija jezika. Specifikacija jezika definira jezik u kojem se piše kôd – u AspectJ, centralne dužnosti se implementiraju koristeći Java programski jezik, a proširenje koje pruža AspectJ koristi se za implementaciju tkanja presijecajućih dužnosti. Implementacija jezika pruža alate za prevođenje (eng. ***compiling***), ispravljanje greška (eng. ***debugging***), i integraciju sa popularnim sučeljima integriranog razvoja (IDE – eng. ***integrated development environment***).

Presijecajuće dužnosti se u centralnu logiku mogu utkati korištenjem pravila tkanja. Pravila tkanja specificiraju „**koju**“ akciju izvršiti „**kada**“ se naiđe na određene točke u izvršenju programa. U AspectJ implementaciji AOP-a, AspectJ kompajler koristi module koji sadrže pravila tkanja, koja se odnose na presijecajuće dužnosti, kako bi dodao novo ponašanje u

module koji se odnose na centralne dužnosti – sve to bez modificiranja izvornog kôda centralnih modula – tkanje se događa samo na bajt-kôdu kojeg kompajler stvori.

6.1 Dinamičko i statičko presijecanje

U AspectJ-u, implementacija pravila tkanja se naziva presijecanje (eng. ***crosscutting***) – pravila tkanja „presijecaju“ višestruke module na sistematičan način kako bi modularizirali presijecajuće dužnosti. AspectJ definira dvije vrste presijecanja: statičko i dinamičko presijecanje.

Dinamičko presijecanje je tkanje novog ponašanja u izvršavanje programa. Većina presijecanja koje se događa u AspectJ-u je dinamičko. Dinamičko presijecanje povećava ili čak zamjenjuje osnovni programski tok izvršavanja na način da presijeca kroz module, te tako modificira ponašanje sustava. Primjerice, ako želimo odrediti da se neka aktivnost izvrši prije nego se izvrše određene metode ili *handler* iznimaka (eng. ***exception handler***) u skupu klasa, u zasebnom modulu možemo specificirati točke tkanja te aktivnost za izvršavanje kada se stigne do tih točaka.

Statičko presijecanje je tkanje modifikacija u statičku strukturu (klase, sučelja, aspekti) sustava. Samo po sebi ne modificira izvršno ponašanje sustava. Najčešća funkcija statičkog presijecanja je da podržava implementaciju dinamičkog presijecanja. Na primjer, ako se želi dodati nove podatke i metode u klase i sučelja kako bi se definirala specifična klasna stanja i ponašanja koja se mogu koristiti u aktivnostima dinamičkog presijecanja. Još jedan primjer korištenja statičkog presijecanja je deklariranje upozorenja i grešaka za vrijeme prevođenja kroz više modula.

6.2 Osnove sintakse točke presijecanja u jeziku AspectJ

AspectJ je jezik zamišljen kao prirodna nadogradnja na Javu. Točke presijecanja dohvaćaju i označavaju točke spajanja te na njih ubacuju određene izmjene. Izgled aspekta je sličan izgledu klase, točka spajanja ima izgled sličan deklaraciji metode, a savjet ima izgled sličan implementaciji metode.

Točka presijecanja može biti anonimna ili imenovana. Korištenje anonimnih točki presijecanja se izbjegava. Sintaksa imenovane točke presijecanja prikazana je u primjeru broj 11.

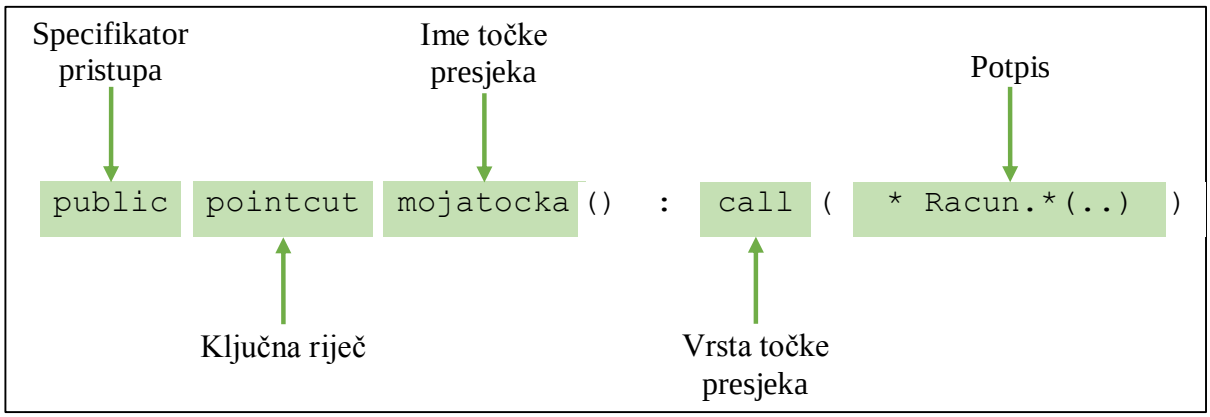
Primjer 11. Sintaksa točke presjeka.

```
[access specifier] pointcut pointcut-name([args]) : pointcut-definition
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 65.

Pri tome, specifikator pristupa (eng. ***access specifier***) ograničava pristup i može biti: ***public***, ***private*** i drugi. Definicija točke presjeka (eng. ***pointcut-definition***) identificira točke spajanja na koje će se dodati savjet.

Slika 17. Definicija točke presjeka.



Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 66.

Točka presjeka sa primjera na slici broj 17 nazvana „mojatocka“ zahvaća sve pozive svih metoda u klasi „Racun“. Tu točku presjeka možemo iskoristiti u savjetu kao što je to prikazano u sljedećem primjeru.

Primjer 12. Savjet za točku presjeka sa slike broj 17.

```
before() : mojatocka() {
      … ovdje dodati radnju savjeta
}
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 66.

6.2.1 *Višeznačnici i operatori točke spajanja*

U jeziku AspectJ koriste se određene oznake kako bi se identificiralo više točka spajanja koje imaju neka zajednička obilježja. Te oznake su:

- * – oznaka za bilo koji broj znakova osim interpunkcijske točke,
- .. – oznaka za bilo koji broj znakova uključujući bilo koji broj točaka,
- + – oznaka za bilo koju podklasu ili podsučelje bilo koje vrste.

AspectJ također koristi i neke operatore, i to:

- ! – jedini unarni operator (predznak) kojeg AspectJ podržava jest oznaka za negaciju. Koristi se u zahvaćanju svih točki spajanja ***osim*** onih specificiranih u točki presjeka.
- || i && – AspectJ podržava samo dva binarna operatora. Spajanjem dviju točka presjeka operatorom || rezultirati će takvim skupom točki spajanja koje odgovaraju jednoj i/ili drugoj točki presjeka. Spajanjem dviju točka presjeka operatorom && rezultirati će skupom točki spajanja koje odgovaraju i jednoj i drugoj točki presjeka.

6.2.2 *Obrazac potpisa*

Konstrukti u Javi imaju svoj potpis kojeg se koristi kod točka presjeka za specifikaciju mjesta na kojima je potrebno dohvatiti točke spajanja. Programer piše obrazac koji odgovara potpisima u točkama presjeka. Postoje tri tipa obrasca potpisa u jeziku AspectJ:

- Tip – obrazac potpisa tipa (vrste)
- Metoda – obrazac potpisa metode
- Polje – obrazac potpisa polja.

6.2.2.1 Obrazac potpisa tipa

Obrazac potpisa tipa u točki presjeka specificira točke spajanja po tipu (vrsti) po kojem je potrebno izvršiti neku akciju. Pojam „tip“ odnosi se na klase, sučelja i primitivne tipove.

Primjer 13. Obrazac potpisa tipa.

```
javax.swing.JComponent+
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 68.

Na gornjem primjeru, radi znaka „+“, potpis odgovara klasi ***JComponent*** zajedno sa svim njezinim podklasama.

Tablica 2. Primjeri potpisa tipa.

Obrazac potpisa	Odgovarajući tip
`Account`	Tip po imenu Account
`*Account`	Svi tipovi čije ime završava sa Account
`java.*.Date`	Tip „Date“ u bilo kojem direktnom podpaketu „Java“ paketa.
`java..*`	Svi tipovi unutar „java“ paketa
`javax..*Model+`	Svi tipovi u „javax“ paketu i njezinim direktnim ili indirektnim podpaketima čija imena završavaju sa „Model“ te njihovim podtipovima.

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 69.

6.2.2.2 Obrazac potpisa metode i konstrukta

Obrazac potpisa metode i konstrukta omogućuje da točke presjeka označe pozive i izvršenja točki spajanja u metodama. Ti obrasci trebaju sadržavati ime, povratni tip (samo metode), deklarirani tip, tipove argumenata, i modifikatore.

Primjer 14. Obrazac potpisa metode.

```
public boolean Collection.add(Object)
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 70.

Na gornjem primjeru, potpis označava metodu po imenu ***add()*** iz sučelja po imenu ***Collection***, a ta metoda ima argument tipa ***Object*** i vraća vrijednost tipa ***boolean***.

Tablica 3. Primjeri potpisa metoda.

Obrazac potpisa	Odgovarajuća metoda
`public void Account.set*(*)`	Sve javne metode u klasi Account čije ime počinje sa „set“ i prihvaća jedan argument bilo kojeg tipa
`public void Account.*()`	Sve javne metode u klasi Account koje vraćaju void i ne prihvaćaju nikakav argument
`public * Account.*()`	Sve javne metode u klasi Account koje ne prihvaćaju argumente a vraćaju bilo koji tip
`public * Account.*(..)`	Sve javne metode u klasi Account koje prihvaćaju bilo koji broj argumenata bilo kojeg tipa
`* Account.*(..)`	Sve metode u klasi Account, čak i one sa privatnim pristupom.

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 70.

Potpis konstruktora je sličan potpisu metode, a razlike su:

- ne sadrži povratnu vrijednost,
- umjesto imena koristi se ključna riječ „*new*“.

Tablica 4. Primjeri potpisa konstruktora.

Obrazac potpisa	Odgovarajući konstruktori
`public Account.new()`	Javni konstruktor klase Account koji ne prihvaća argumente
`public Account.new(int)`	Javni konstruktor klase Account koji prihvaća jedan argument tipa integer
`public Account.new(..)`	Javni konstruktor klase Account koji prihvaća bilo koji broj argumenata bilo kojeg tipa
`public Account+.new(..)`	Bilo koji javni konstruktor klase Account ili njezinih podklasa
`public *Account.new(..)`	Bilo koji javni konstruktor klasa čije ime završava sa Account.

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 72.

6.2.2.3 Obrazac potpisa polja

Obrazac potpisa polja može se koristiti za dohvaćanje točaka spajanja koje odgovaraju čitanju i pisanju u određeno polje. Potpis polja mora sadržavati tip polja, deklarirani tip, i modifikatore.

Primjer 15. Obrazac potpisa polja.

```
public int java.awt.Rectangle.x
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 72.

Gornji primjer određuje javno polje po imenu ***x*** tipa ***integer*** u klasi ***Rectangle***. U sljedećoj tablici navesti ćemo par primjera.

Tablica 5. Primjeri potpisa polja.

Obrazac potpisa	Odgovarajuća polja
`private float Account._balance`	Privatno polje _balance u klasi Account
`* Account.*`	Sva polja u klasi Account, neovisno o modifikatoru pristupa, tipu ili imenu
`!public static * banking..*.*`	Sva polja koja nisu javna, tipa static, iz paketa banking i njegovih direktnih i indirektnih podpaketa
`public !final  *.*`	Nezavršna javna polja bilo koje klase

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 72.

7. Kako krenuti – logika u stvaranju aspektno-orijentiranog programa

Postoje više načina kako razviti jednu AspectJ aplikaciju:

- Prvi način – prvo se napišu osnovne dužnosti korištenjem osnovnog jezika. Zatim se koristi AspectJ za pisanje aspektnog kôda za implementaciju presijecajućih dužnosti.
- Drugi način – nadodavanje novih dužnosti na postojeću aplikaciju korištenjem AOP-a.

Redoslijed stvaranja jednog aspektno-orijentiranog programa može se skratiti ovako[12]:

- Pisanje osnovnog kôda
- Pisanje aspektnog kôda
    - Identifikacija točke spajanja
    - Određivanje točke presjeka
    - Pisanje savjeta.

7.1 Osnovni kôd

Proces započinje dokumentom sa specifikacijama zahtjeva korisnika, koji po svojoj prirodi mogu biti obavezni u programu ili poželjni za dodati u program. Zahtjevi se implementiraju na konvencionalan način (OOP), nakon čega je potrebno implementirati zahtjeve za presijecajuće dužnosti.

7.2 Aspektni kôd

Presijecajuće dužnosti implementiraju se pomoću aspektnog kôda. U program se dodaju aspekti koji imaju specificirana mjesta na koja će se ubaciti u program (točke presijecanja) kao i radnju koja se treba izvršiti (savjet).

Nakon što je uspješno implementirana osnovna dužnost sustava korištenjem konvencionalnog jezika, potrebno je analizirati moguće dužnosti koje će presijecati sustav. Primjerice, ako imamo informacijski sustav pacijenata, i ako postoji zahtjev za autentikacijom osoblja u sustav, potrebno je identificirati „kada“ odnosno „gdje“ je potrebna autentikacija. U

[12] Prema: D. Gradecki, J. i Lesiecki, N. (2003) Mastering AspectJ – Aspect-Oritented Programming in Java. Indianapolis: Wiley Publishing, str. 32-36.

tom primjeru, autentikacija može biti potrebna prilikom prepisivanja lijeka, izmijene terapije, ili pak prilikom bilo kakve izmijene nad bazom podataka.

Sada je potrebno sve potrebne točke generalizirati pomoću jedne točke presjeka, pa će tako definirana točka presjeka tada predstavljati skup tih točki spajanja na kojima određena dužnost presijeca osnovni kôd, odnosno, točke spajanja nad kojima je potrebno izvršiti određenu radnju koja je definirana u savjetu aspekta.

Nakon što smo odredili presijecajuće dužnosti iz specificiranih zahtjeva, identificirali njihove točke spajanja te napisali točke presjeka, potrebno je napisati kôd koji zapravo i unosi nekakve promjene u sâm osnovni kôd – potrebno je napisati kôd koji implementira presijecajuću dužnost. Taj korak naziva se davanjem savjeta programu. Kôd savjeta koji implementira dužnost pisan je u osnovnom Java jeziku i kao takav sličan je tijelu metode.

7.3 Primjeri AspectJ programa

7.3.1 „Hello World“

Kod pisanja aspektno-orijentiranog programa potrebno je utvrditi osnovne dužnosti. U programu za naš primjer postoji samo jedna primarna dužnost: ispisati poruku „Hello World“.

Primjer 16. „*Hello World*“.

```
package deo_hello_world;

class Ispisivanje {
	public void Pisem() {
		System.out.println("Hello World");
	}
	public static void main(String args[]) {
		Ispisivanje nov_pisi = new Ispisivanje();
		nov_pisi.Pisem();
	}
}
```

Prema: D. Gradecki, J. i Lesiecki, N. (2003) Mastering AspectJ – Aspect-Oritented Programming in Java. Indianapolis: Wiley Publishing, str. 56.

Kôd sa Primjera broj 16 ispisuje poruku na sljedeći način: metoda `Pisem()` je odgovorna za ispisivanje poruke. Tu metodu se poziva iz `main()` metode iz klase `Ispisivanje` tako što se stvori nova instanca klase `Ispisivanje` po imenu `nov_pisi`.

U gornji primjer želimo dodati aspekte: dodati ćemo jedan aspekt koji obavlja radnju **prije** i jedan aspekt koji obavlja radnju **poslije** metode `Pisem()`.

Primjer 17. MojPrviAspekt.aj.

```
public aspect MojPrviAspekt {
	pointcut savjetBefore() : call(public void Pisem());
	before() : savjetBefore() {
		System.out.println("Jesi tu?");
	}
}
```

Prema: D. Gradecki, J. i Lesiecki, N. (2003) Mastering AspectJ – Aspect-Oritented Programming in Java. Indianapolis: Wiley Publishing, str. 56.

Primjer 18. MojDrugiAspekt.aj.

```
public aspect MojDrugiAspekt {
	pointcut savjetAfter() : call(public void Pisem());
	after() : savjetAfter() {
		System.out.println("Laku noć");
	}
}
```

Prema: D. Gradecki, J. i Lesiecki, N. (2003) Mastering AspectJ – Aspect-Oritented Programming in Java. Indianapolis: Wiley Publishing, str. 56.

Aspekti po imenu `MojPrviAspekt` i `MojDrugiAspekt` dodaju funkcionalnosti prije odnosno poslije glavne metode. Prvi aspekt definira točku presjeka po imenu `savjetBefore` i savjet koji ispisuje poruku „Jesi tu?“ prije nego se izvrši metoda iz definicije točke presjeka, odnosno metoda `Pisem()`. Analogno, drugi aspekt definira točku presjeka po imenu `savjetAfter` i savjet koji ispisuje poruku „Laku noć“ nakon izvršenja metode iz definicije točke presjeka, odnosno metode `Pisem()`.

7.3.2 *Korištenje „around“ savjeta – primjer*

U ovom poglavlju napraviti ćemo kratki primjer aplikacije. Stvoriti ćemo klasu koja sadrži dvije metode koje ispisuju poruke.

Primjer 19. Ispisivanje.java.

```
package deo_drugi_primjer;

	public class Ispisivanje {
		public static void pisem(String poruka) {
				System.out.println(poruka);
		}
		public static void pisem(String osoba, String poruka) {
			System.out.print(osoba + ", " + poruka);
		}
}
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 37.

Klasa **Ispisivanje** sadrži dvije metode: prva ispisuje poruku, a druga ispisuje poruku osobi. Sljedeći primjer prikazuje klasu kojom možemo testirati ispravnost klase **Ispisivanje**.

Primjer 20. Proba.java.

```
package deo_drugi_primjer;

public class Proba {
	public static void main(String[] args) {
		Ispisivanje.pisem("Pozdrav!");
		Ispisivanje.pisem("Bruce", "gdje je Batman?");
	}
}
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 38.

Nakon prevođenja i izvršavanja programa vidimo sljedeći izlaz:

```
Pozdrav!
Bruce, gdje je Batman?
```

Bez promjena u klasi **Ispisivanje**, njezinu funkcionalnost možemo izmijeniti ako dodamo aspekt u sustav – dodati ćemo implementaciju presijecajuće dužnosti – nakon ispisivanja poruke želimo da se ispiše „Laku noć!“.

Primjer 21. ProbniAspekt.java.

```
package deo_drugi_primjer;

public aspect ProbniAspekt {                    → Deklaracija aspekta
   pointcut isporukaPoruke()                    → Deklaracija točke presjeka
      : call(* Ispisivanje.pisem(..));

   after() : isporukaPoruke() {
      System.out.println("Laku noć!");         → Savjet
   }
}
```

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 38.

AJC (***AspectJ Compiler***) zahtjeva da su sve ulazne datoteke dostupne zajedno kako bi mogao proizvesti klasne datoteke sa utkanim aspektima. Pokretanjem programa dobivamo izlaz:

```
Pozdrav!
Laku noć!
Bruce, gdje je Batman?
Laku noć!
```

Objašnjenje: datoteka **ProbniAspekt.java** deklarira aspekt **ProbniAspekt**. U tom aspektu definirana je točka presjeka **isporukaPoruke()**. Ona dohvaća pozive svih metoda sa imenom **pisem()** u klasi **Ispisivanje**. Znak "*****" označava da nije bitan tip povrata (***return type***), a dvije točke (" (..) ") unutar zagrada nakon metode **pisem** označava da nije bitan broj niti tip argumenata. U primjeru, točka presjeka će dohvatiti pozive obje verzije (normalnu i *overloaded*) metode **pisem()** u klasi **Ispisivanje**. Aspekt također definira i savjet koji će se

izvršiti nakon (**after()**) izvršenja točke presjeka **isporukaPoruke()**. U primjeru, savjet će se izvršiti nakon poziva metode **Ispisivanje.pisem()**. U savjetu će program ispisati „Laku noć!“.

Dodati ćemo još jedan aspekt – u znak poštovanja, želimo na ime osobe dodati prefiks „g.“. Na primjeru broj 22 napraviti ćemo izmjenu tako da se prefiks dodaje na ime osobe svaki put kad se ispisuje poruka toj osobi.

Primjer 22. AspektPostovanje.java.

```
public aspect AspektPostovanje {
   pointcut isporukaOsobi(String osoba)
      : call (* Ispisivanje.pisem(String, String))
      && args(osoba, String);                                  1

   void around (String osoba) : isporukaOsobi(osoba) {         2
      proceed("g." + osoba);   <-- 3
   }
}
```

1. Točka presjeka za dohvat metode deliver()
2. Savjet za točku presjeka
3. Tijelo savjeta

Izvor: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 39.

Nakon pokretanja rezultat je sljedeći:

```
Pozdrav!
Laku noć!
g.Bruce, gdje je Batman?
Laku noć!
```

Aspekt **AspektPostovanje** možemo objasniti ovako[13]:

- Točka presjeka dohvaća sve točke spajanja koje pozivaju metodu **Ispisivanje.pisem()** koja prima dva argumenta. Ako želimo dodati „g.“ na ime osobe, moramo dohvatiti argument **osoba**. Dio **args()** radi baš to – prvi njegov

[13] Prema: Laddad, R. (2003.) AspectJ In Action – Practial Aspect-Oriented Programming. Greenwich: Manning, str. 40.

parametar specificira da će prvi argument metode biti dostupan kao varijabla **osoba**. Drugi parametar specificira da drugi argument (poruka) ne mora biti dohvaćen, ali mora biti tipa **String** – to je određeno specificiranjem tipa **String** za drugi argument, umjesto argumenta **osoba**.

- Kako bi izmijenili izlazno ime osobe (sa dodatkom „-ji“), moramo izvršiti originalnu akciju sa izmijenjenim argumentom. To ne možemo postići tako što ćemo koristiti **before()** savjet jer bi to onda izvršilo kôd prije savjetovane operacije. Umjesto toga, moramo izmijeniti argument savjetovane operacije. Dakle, potreban nam je **around** savjet na toj točki spajanja, budući da on može izvršiti originalnu operaciju uz izmijenjeni kontekst.
- U tijelu savjeta se nalazi ključna riječ **proceed()** – ključna riječ koja govori dohvaćenoj točki spajanja da se izvrši. Dohvaćamo originalni argument, dodajemo sufiks „g.“, i prosljeđujemo ga u **proceed()**. Rezultat je prizivanje metode **Ispisivanje.pisem()** sa izmijenjenim argumentom.

8. Nekoliko pitanja za kraj

Da li AOP zamjenjuje OOP? Ne. AOP je zapravo samo nadogradnja na OOP. Osnovni dio programa se mora napisati u OOP-u budući da OOP jako dobro modelira osnovne karakteristike sustava, a nakon toga AOP služi za dodavanje novih funkcionalnosti.

Da li AOP rješava neke nove probleme? – Ne. AOP je samo elegantniji način za rješavanje problema koji se već mogu riješiti OOP-om. Čar AOP-a je u tome da pruža bolji i jednostavniji način za rješavanje problema. Obzirom da su zahtjevi sve složeniji, potreba za takvim lakšim načinom je očita.

Da li je AOP opasan? Da. Sa AOP-om se mogu neke stvari koje se OOP ne može. Mogu se presresti pozivi metoda, pročitati i izmijeniti proslijeđene parametre, i onda pozvati tu metodu. AOP omogućava da se u klase uvedu nove metode i polja koja prije nisu postojala. To su radnje koje mogu biti opasne ako se posao ne odradi pravilno.

Iako postoje razlozi za neprihvaćanje, AOP sve više dobiva na popularnosti. Dio otpora leži u činjenici da AOP ne rješava ništa što OOP već ne može riješiti, međutim, AOP rješava probleme na bolji način nego što je slučaj kod konvencionalnih tehnika. Dio otpora pak leži u težini praćenja programskog toka jednog AOP sustava. Teško je pratiti programski tok u kojem ne znamo kako i kada će se neki aspekt ubaciti. To je zapravo dobra stvar, jer se onda programer može posvetiti tome koliko kvalitetno će implementirati funkcionalnost a ne kojim redoslijedom se izvršava kôd. Neki također tvrde da AOP promiče neuredan dizajn, dok je istina sasvim drukčija – takav pristup zahtjeva dobro oblikovanje centralnih dužnosti kako bi se dobio jasan dizajn cijelog sustava.

9. Zaključak

Temeljna postavka softverskog inženjerstva je da razdvajanje dužnosti dovodi do sustava kojeg je lakše održavati i jednostavnije razumjeti. Kod OOP-a se takva modularizacije osnovnih dužnosti postiže razdvajanjem sučelja od njihovih implementacija, ali kod tog pristupa nije moguće razdvojiti kôd presijecajućih dužnosti iz kôda osnovnih modula. Tu se kao rješenje pojavljuje AOP – pristup koji proširuje metodologiju OOP-a time što pruža nove programske konstrukte, tzv. aspekte, koji zasebno enkapsuliraju presijecajuće dužnosti u odvojene module. Korištenje takvog pristupa omogućava jednostavne izmjene, lakšu razumljivost i bolju ponovnu iskoristivog kôda.

Kod aspektno-orijentiranog razvoja softvera potrebno je ispravno definirati točke presjeka kako se savjet (koji predstavlja novo ponašanje) ne bi utkao na neko neželjeno mjesto u izvornom kôdu i tako doveo do neželjenog i neočekivanog ponašanja sustava. Treba se pobrinuti za potpunu neovisnost aspekata (ponašanje programa ne smije ovisiti o redoslijedu kojim su aspekti utkani u program) kako bi se osiguralo da neće biti miješanja aspekata ako se više aspekta utkaju na istu točku spajanja.

S obzirom na relativnu novost takvog pristupa, AOP nailazi na određen otpor u prihvaćanju iz razloga što zahtjeva novi način razmišljanja o sustavu. Također, radi složenosti odvijanja AOP programa, problemi se pojavljuju i prilikom verifikacije i validacije, pa se stoga provode istraživanja o tome kako najbolje testirati jedan takav AOP sustav. Iz tih i još nekih drugih razloga, AOP još uvijek nije opće prihvaćena metoda u softverskom inženjerstvu, ali sve više se uviđa važnost razdvajanja funkcionalnosti na kojem se temelji aspektno-orijentiran razvoj softvera.

Time je dan kvalitetan alat za prikazivanje sustava kao skup međusobno potpuno neovisnih dužnosti. Nadogradnjom na postojeće metodologije programiranja, AOP čuva investiciju u znanje koje je stečeno u posljednjih par desetljeća.

10. Literatura

KNJIGE:

1. Sommerville, I. (2010.) *Software Engineering*. Ninth Edition. Boston: Addison-Wesley.
2. Jacobson, I. i Ng, P.-W. (2004.) *Aspect-Oriented Software Development with Use Cases*. Boston: Addison-Wesley.
3. D. Gradecki, J. i Lesiecki, N. (2003) *Mastering AspectJ – Aspect-Oritented Programming in Java*. Indianapolis: Wiley Publishing.
4. Laddad, R. (2003.) *AspectJ In Action – Practial Aspect-Oriented Programming*. Greenwich: Manning.
5. Kiselev, I. (2003.) *Aspect-Oriented Programming with AspectJ*. Indianapolis: SAMS.
6. Colyer, A., Clement, A., Harley, G. i Webster, M. (2004.) *Eclipse AspectJ: Aspect-Oriented Programming with AspectJ and the Eclipse AspectJ Development Tools*. Boston: Addison-Wesley.

RADOVI:

7. Kostić, N. (2004.) Aspektno Programiranje u Javi. Na *1. sastanku JavaSvet Zajednice*. 14.08.2004. Dostupno na: http://javasvet.rs/ [Pristupano: 29.12.2014.]
8. Lee, K.W.K. (2002.) *An Introduction to Aspect-Oriented Programming*. Hong Kong University of Science and Technology.

11. Popis slika, tablica i primjera

11.1 Slike

11.2 Tablice

11.3 Primjeri

11.4 Rječnik

- Encapsulate – Enkapsulacija
- Crossccutting – Presijecajuće, isprepletene, rasprostranjene
- Concern – Dužnost, funkcionalnost
- Core – Centralne, osnovne, temeljne, središnje
- Code scattering – Rasipanje / raspršivanje kôda
- Reflect – Odražavati
- Feature – Značajka
- Extension – Proširenje

- Authentication – Autentifikacija / Autentikacija (utvrđivanje identiteta korisnika)
- Authorization – Autorizacija (provjera ovlaštenja za pristup resursu)
- Capture – Dohvatiti, označiti

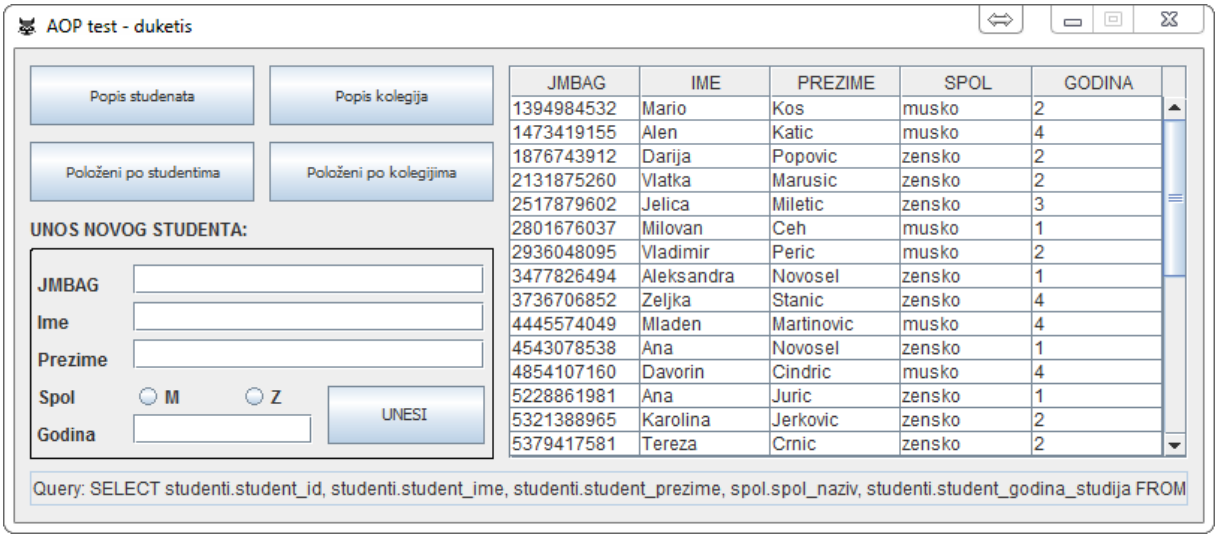
12. Prilog – primjer aplikacije za pristup bazi podataka i izvoz podataka u HTML datoteku

Aplikacija koja je napisana kao prilog radu omogućava pristup bazi podataka, prikaz podataka iz baze podataka te unos zapisa u bazu podataka.

Omogućen je prikaz rezultata sljedećih upita:

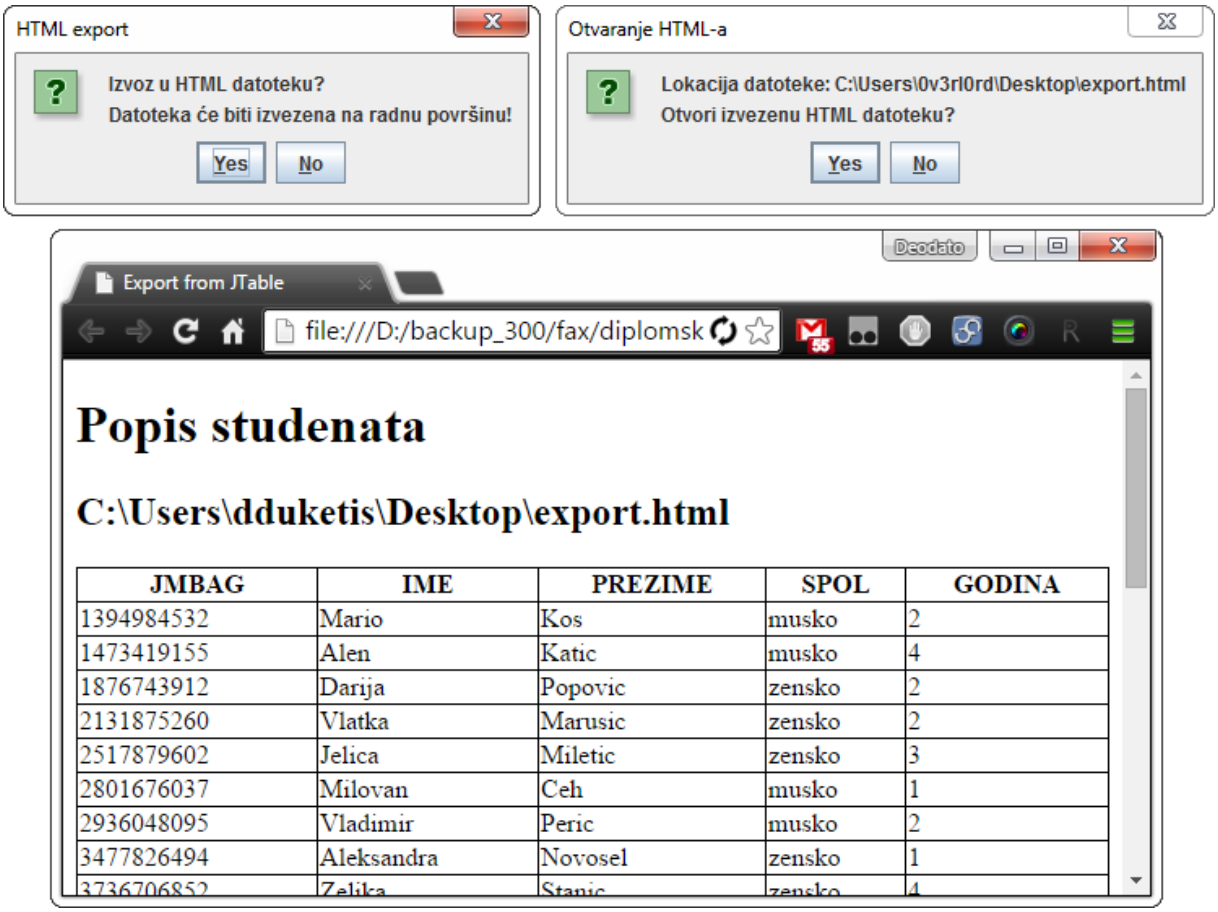
- Popis studenata
- Popis kolegija
- Položeni kolegiji poredani po studentima
- Položeni kolegiji poredani po nazivu kolegija.

Slika 18. Glavni prozor aplikacije.



Također, nakon prikaza rezultata upita unutar prozora aplikacije, omogućen je izvoz rezultata u vanjsku HTML datoteku. Slika 19 prikazuje prozor upita za izvoz, prozor za otvaranje stvorene HTML datoteke, te sadržaj HTML datoteke.

Slika 19. Izvoz u vanjsku HTML datoteku.



Projekt se sastoji iz dva primjera:

- Aplikacija bez korištenja aspektnih elemenata
- Aplikacija uz korištenje aspekata.

12.1 Pisanje aplikacije

Za stvaranje aplikacije korišten je alat Eclipse (Luna, v4.4.1). Kako bi se omogučio razvoj aspektno-orijentiranog softvera, potrebno je Eclipse alatu instalirati nadogradnju naziva AspectJ[14]. AspectJ je aspektno-orijentirana nadogradnja koja se lako uči i jednostavno koristi zahvaljujući sintaksi koja je slična sintaksi Jave.

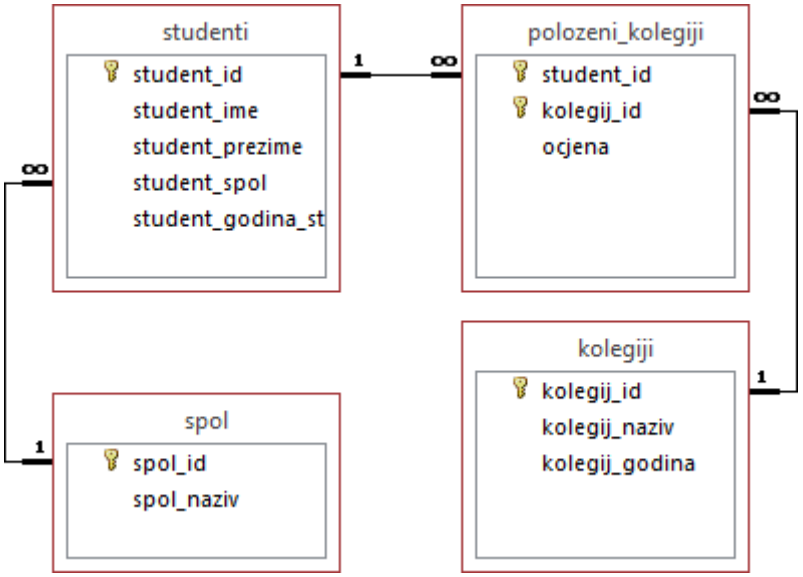
12.2 Stvaranje baze podataka

Za stvaranje baze podataka iz koje će se uzimati podaci za prikaz u prozoru aplikacije potrebno je instalirati XAMPP alat koji sadrži MySQL, Apache server i phpMyAdmin sučelje.

[14] AspectJ: https://eclipse.org/aspectj/

phpMyAdmin sučelje je alat kojim administriramo MySQL bazu podataka putem web pretraživača. Pomoću phpMyAdmin alata stvorena je baza podataka pod imenom „`testbaza`“. Baza sadrži 4 tablice, povezane kako je to prikazano na donjoj slici.

Slika 20. Shema baze podataka.



Opis tablica:

- Studenti – sadrži popis studenata.
- Kolegiji – sadrži popis kolegija.
- Spol – sadrži popis spolova.
- Položeni kolegiji – tablica veze „*više-prema-više*“. Sadrži popis studenata i položenih kolegija.

12.3 Spajanje na bazu podataka

Za omogućavanje pristupa bazi podataka potrebno je naknadno instalirati JDBC Driver za MySQL, pod nazivom MySQL Connector/J[15]. To je JDBC[16] Type 4 driver – tip 4 predstavlja čistu Java implementaciju MySQL protokola i ne oslanja se na MySQL klijentske biblioteke. Nakon skidanja *.jar datoteke sa službenih MySQL Internet stranica, potrebno je dodati MySQL Connector/J u putanju za izgradnju (eng. *build path*) razvijane java aplikacije (Build path → Add external archives).

[15] Preuzeto sa: https://dev.mysql.com/downloads/connector/j/5.1.html

[16] JDBC – Java Database Connectivity

12.4 Primjer bez korištenja aspekata

Primjer aplikacije bez korištenja aspektnih elemenata sastoji se od tri datoteke:

- databaseConnection.java
- printToHtml.java
- prozor.java

Datoteka `databaseConnection.java` predstavlja osnovnu klasu koja omogućava čitanje podataka iz baze podataka. Klasa definira potrebne parametre (url, user, password) za pristup bazi. Također, definirane su i sljedeće metode:

- `database_connect ()` – metoda za otvaranje konekcije na bazu podataka.
- `execute_update (String upit_arg)` – metoda za izvršavanje izmjena u bazi.
- `execute_query (String upit_arg)` – metoda za izvršavanje upita nad bazom.
- `database_disconnect ()` – metoda za prekidanje konekcije sa bazom.
- `ispis_rezultata ()` – metoda za ispis rezultata upita u konzolu.
- `upit ()` – metoda koja redom poziva metode: database_connect, execute_query, database_disconnect.
- `izmjena ()` – metoda koja redom poziva metode: database_connect, execute_update, database_disconnect.

Datoteka `printToHtml.java` predstavlja klasu koja omogućava ispis upita nad bazom u vanjsku HTML datoteku. Klasa definira potrebne objekte (`PrintWriter`) za ispis u vanjsku datoteku.

Datoteka `prozor.java` predstavlja glavni prozor aplikacije koji sadrži jednostavne kontrole za upravljanje bazom. Klasa `prozor` sadrži metodu `main ()` koja se pokreće prilikom pokretanja aplikacije. Metoda stvara instancu klase prozor te inicijalizira sav potreban sadržaj. Stvara se tzv. `jframe` unutar kojeg su smještene sve kontrole aplikacije. U okviru se nalaze 4 gumba za ispis podataka iz baze: popis studenata, popis kolegija, položeni kolegiji poredani po studentima, te položeni kolegiji poredani abecedno po kolegijima. Rezultat upita nad bazom prikazuje se pomoću elementa `jTable` – dinamički stvorene tablice. U klasi `prozor` definirane su i dvije metode koje se pozivaju pritiskom na jedan od navedena 4 gumba:

- `stvaranjeTablice (String header[], String upit)` – izvršava upit nad bazom i osvježava sadržaj tablice.

- `izvozHtml (String header[], String upit, String naziv)` – nakon osvježavanja tablice, otvara se dijalog sa upitnikom o izvozu u HTML datoteku. Ukoliko korisnik odluči tako, stvara se instanca klase `printToHtml` te se sadržaj tablice izveze u datoteku `export.html`.

Također, u klasi `prozor` nalaze se i kontrole za unos novog studenta u bazu podataka. Pritiskom na gumb „unesi“ čitaju se podaci iz polja (`txt_jmbag, txt_ime, txt_prezime, rdbtn_m, rdbtn_z, txt_godina`) te se unose u bazu korištenjem metode `izmjena (String upit)`.

12.4.1 databaseConnection.java

```
001    package dduketis;
002    import java.sql.Connection;
003    import java.sql.DriverManager;
004    import java.sql.ResultSet;
005    import java.sql.SQLException;
006    import java.sql.Statement;
007    import java.sql.ResultSetMetaData;
008    import java.util.ArrayList;
009    import java.util.List;
010    class databaseConnection {
011            Connection conn = null;
012            Statement statement = null;
013            ResultSet resultSet = null;
014            String url = null;
015            String user = null;
016            String password = null;
017            List<List<String>> testna_tablica = new ArrayList<List<String>>();
018            void database_connect () {
019                    url = "jdbc:mysql://localhost/testbaza";
020                    user = "testkorisnik";
021                    password = "testlozinka";
022                    try {
023                            conn = DriverManager.getConnection(url, user, password);
024                    } catch (SQLException e) {
025                            System.out.println(e.getMessage());
026                    }
027                    if (conn != null) {
028                            try {
029                                    statement = conn.createStatement();
030                            } catch (SQLException e) {
031                                    System.out.println(e.getMessage());
032                            }
033                    }
034            }
035            void execute_update (String upit_arg) {
036                    if(statement != null) {
037                            try {
038                                    statement.executeUpdate(upit_arg);
039                            } catch (SQLException e) {
040                                    System.out.println(e.getMessage());
041                            }
042                    }
043            }
```

```
044     void execute_query (String upit_arg) {
045             if(statement != null) {
046                     try {
047                             resultSet = statement.executeQuery(upit_arg);
048                     } catch (SQLException e) {
049                             System.out.println(e.getMessage());
050                     }
051             }
052             try {
053                     if (!resultSet.isBeforeFirst()) {
054                             System.out.println("resultSet prazan - no data");
055                     }
056             } catch (SQLException e) {
057                     System.out.println(e.getMessage());
058             }
059             int columnCount=0;
060             try {
061                     testna_tablica.clear();
062                     ResultSetMetaData metadata = this.resultSet.getMetaData();
063                     columnCount = metadata.getColumnCount();
064             } catch (SQLException e) {
065                     System.out.println(e.getMessage());
066             }
067             if (this.resultSet != null) {
068                     try {
069                             int i = 0; // brojac za redove
070                             while (this.resultSet.next()) {
071                                     this.testna_tablica.add(new ArrayList<String>());
072                                     for (int j=1; j<columnCount+1; j++) {
073                             this.testna_tablica.get(i).add(this.resultSet.getString(j));
074                                     }
075                                     i++;
076                             }
077                     } catch (SQLException e) {
078                             System.out.println(e.getMessage());
079                     }
080             }
081     }


082     void database_disconnect () {
083             try {
084                     conn.close();
085             } catch (SQLException e) {
086                     System.out.println(e.getMessage());
087             }
088     }
089     void ispis_rezultata () {
090             for (int i=0; i<this.testna_tablica.size(); i++) { // od nule do onoliko
koliko ima redaka (zapisa) u tablici
091                     for (int j=0; j<this.testna_tablica.get(i).size(); j++) { // od nule
do onoliko koliko ima stupaca u tablici
092                             System.out.print(this.testna_tablica.get(i).get(j) + " ");
093                     }
094                     System.out.println("");
095             }
096     }
097     void upit (String upit) {
098             this.database_connect();
099             this.execute_query(upit);
100             this.database_disconnect();
101     }
102     void izmjena (String upit) {
103             this.database_connect();
104             this.execute_update(upit);
105             this.database_disconnect();
106     }
107     }
```

12.4.2 printToHtml.java

```
01      package dduketis;
02      import java.io.*;
03      import java.awt.Desktop;
04      import javax.swing.JOptionPane;
05      import javax.swing.filechooser.FileSystemView;
06
07      public class printToHtml {
08      void export (String header[], String upit, String naziv) throws Exception {
09              FileSystemView filesys = FileSystemView.getFileSystemView();
10              String filepath = filesys.getHomeDirectory() + "\\export.html";
11
12              databaseConnection spajanje = new databaseConnection();
13              spajanje.upit(upit); //spajanje.execute_query(upit);
14
15              PrintWriter writer = new PrintWriter(filepath, "UTF-8");
16
17              writer.println("<!DOCTYPE html>");
18              writer.println("<HTML>");
19              writer.println("<HEAD>");
20              writer.println("<TITLE>Export from JTable</TITLE>");
21              writer.println("<STYLE>");
22              writer.println("table, th, td {");
23              writer.println("border: 1px solid black;");
24              writer.println("border-collapse: collapse;");
25              writer.println("</STYLE>");
26              writer.println("</HEAD>");
27              writer.println("<BODY>");
28
29              writer.println("<h1>" + naziv + "</h1>");
30              writer.println("<h2>" + filepath + "</h2>");
31
32              writer.println("<table style='width:100%'>");
33              writer.println("<tr>");
34              for (int i=0; i<header.length; i++) {
35                      writer.println("<th>");
36                      writer.println(header[i]);
37                      writer.println("</th>");
38              }
39              writer.println("</tr>");
40
41              for (int i=0; i<spajanje.testna_tablica.size(); i++) {
42                      writer.println("<tr>");
43                      for (int j=0; j<spajanje.testna_tablica.get(i).size(); j++) {
44                              writer.println("<td>");
45                              writer.println(spajanje.testna_tablica.get(i).get(j));
46                              writer.println("</td>");
47                      }
48                      writer.println("</tr>");
49              }
50              writer.println("</table>");
51
52              writer.println("</BODY>");
53              writer.println("</HTML>");
54              writer.close();
55
56              int dijalogGumb = JOptionPane.YES_NO_OPTION;
57              int dijalogRezultat = JOptionPane.showConfirmDialog(null, "Lokacija
datoteke: " + filepath + "\nOtvori izvezenu HTML datoteku?", "Otvaranje HTML-a",
dijalogGumb);
58              if(dijalogRezultat == 0) {
59                      File f = new File (filepath);
60                      if (f.exists()) {
61                              if (Desktop.isDesktopSupported()) {
62                                      Desktop.getDesktop().open(f);
63                              }
64                              else {
65                                      System.out.println("Datoteka ne postoji");
66                              }
67                      }
68              }
69      }
70      }
```

12.4.3 prozor.java

```
001  package dduketis;
002
003  import javax.swing.JFrame;
004  import javax.swing.JTable;
005  import javax.swing.JButton;
006  import javax.swing.table.DefaultTableModel;
007  import javax.swing.ButtonGroup;
008  import javax.swing.JOptionPane;
009  import javax.swing.JScrollPane;
010  import javax.swing.JLabel;
011  import javax.swing.JTextField;
012  import javax.swing.JRadioButton;
013  import javax.swing.JPanel;
014  import java.awt.EventQueue;
015  import java.awt.Font;
016  import java.awt.event.ActionListener;
017  import java.awt.event.ActionEvent;
018  import javax.swing.border.LineBorder;
019  import java.awt.Color;
020
021  public class prozor {
022  private JFrame prvi_frame;
023  private JTable prva_tablica;
024  private JScrollPane prvi_scrollpane;
025  private JButton btn_popis_studenata;
026  private JButton btn_popis_kolegija;
027  private JButton btn_polozeni_po_studentima;
028  private JButton btn_polozeni_po_kolegijima;
029  private JButton btn_unesi;
030  private JTextField txt_jmbag;
031  private JTextField txt_ime;
032  private JTextField txt_prezime;
033  private JTextField txt_godina;
034  private JTextField txt_greska;
035
036  private JLabel lbl_unos_novog_studenta;
037
038  // pokretanje aplikacije
039  public static void main(String[] args) {
040          EventQueue.invokeLater(new Runnable() {
041                  public void run() {
042                          try {
043                                  prozor window = new prozor();
044                                  window.prvi_frame.setVisible(true);
045                          } catch (Exception e) {
046                                  e.printStackTrace();
047                          }
048                  }
049          });
050  }
051  // stvaranje aplikacije
052  public prozor() {
053          initialize();
054  }
056  private void initialize() {
057          prvi_frame = new JFrame();
058          prvi_frame.setBounds(100, 100, 800, 344);
059          prvi_frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
060          prvi_frame.getContentPane().setLayout(null);
061          prvi_frame.setLocationRelativeTo(null);
062          prvi_frame.setResizable(false);
```

```
064             btn_popis_studenata = new JButton("Popis studenata");
065             btn_popis_studenata.setFont(new Font("Tahoma", Font.PLAIN, 10));
066             btn_popis_studenata.addActionListener(new ActionListener() {
067                     public void actionPerformed(ActionEvent e) {
068                             String header[] = new String[] {"JMBAG", "IME", "PREZIME",
"SPOL", "GODINA"};
069                             String upit = "SELECT studenti.student_id,
studenti.student_ime, studenti.student_prezime, spol.spol_naziv,
studenti.student_godina_studija FROM studenti JOIN spol ON
studenti.student_spol=spol.spol_id ";
070                             stvaranjeTablice(header, upit);
071                             izvozHtml(header, upit, btn_popis_studenata.getText());
072                             txt_greska.setText("Query: " + upit);
073                     }
074             });
075             btn_popis_studenata.setBounds(10, 11, 150, 40);
076             prvi_frame.getContentPane().add(btn_popis_studenata);
077
078             btn_popis_kolegija = new JButton("Popis kolegija");
079             btn_popis_kolegija.setFont(new Font("Tahoma", Font.PLAIN, 10));
080             btn_popis_kolegija.addActionListener(new ActionListener() {
081                     public void actionPerformed(ActionEvent e) {
082                             String header[] = new String[] {"ID", "NAZIV", "GODINA"};
083                             String upit = "SELECT * FROM kolegiji";
084                             stvaranjeTablice(header, upit);
085                             izvozHtml(header, upit, btn_popis_kolegija.getText());
086                             txt_greska.setText("Query: " + upit);
087                     }
088             });
089             btn_popis_kolegija.setBounds(170, 11, 150, 40);
090             prvi_frame.getContentPane().add(btn_popis_kolegija);
091
092             btn_polozeni_po_studentima = new JButton("Položeni po studentima");
093             btn_polozeni_po_studentima.setFont(new Font("Tahoma", Font.PLAIN, 10));
094             btn_polozeni_po_studentima.addActionListener(new ActionListener() {
095                     public void actionPerformed(ActionEvent e) {
096                             String header[] = new String[] {"JMBAG", "IME", "PREZIME",
"ID KOLEGIJA", "NAZIV", "OCJENA"};
097                             String upit = "SELECT studenti.student_id,
studenti.student_ime, studenti.student_prezime, kolegiji.kolegij_id,
kolegiji.kolegij_naziv, polozeni_kolegiji.ocjena FROM studenti JOIN polozeni_kolegiji ON
studenti.student_id=polozeni_kolegiji.student_id JOIN kolegiji ON
polozeni_kolegiji.kolegij_id=kolegiji.kolegij_id;";
098                             stvaranjeTablice(header, upit);
099                             izvozHtml(header, upit,
btn_polozeni_po_studentima.getText());
100                             txt_greska.setText("Query: " + upit);
101                     }
102             });
103             btn_polozeni_po_studentima.setBounds(10, 62, 150, 40);
104             prvi_frame.getContentPane().add(btn_polozeni_po_studentima);
105
106             btn_polozeni_po_kolegijima = new JButton("Položeni po kolegijima");
107             btn_polozeni_po_kolegijima.setFont(new Font("Tahoma", Font.PLAIN, 10));
108             btn_polozeni_po_kolegijima.addActionListener(new ActionListener() {
109                     public void actionPerformed(ActionEvent e) {
110                             String header[] = new String[] {"JMBAG", "IME", "PREZIME",
"ID KOLEGIJA", "NAZIV", "OCJENA"};
111                             String upit = "SELECT studenti.student_id,
studenti.student_ime, studenti.student_prezime, kolegiji.kolegij_id,
kolegiji.kolegij_naziv, polozeni_kolegiji.ocjena FROM studenti JOIN polozeni_kolegiji ON
studenti.student_id=polozeni_kolegiji.student_id JOIN kolegiji ON
polozeni_kolegiji.kolegij_id=kolegiji.kolegij_id ORDER BY kolegiji.kolegij_id;";
112                             stvaranjeTablice(header, upit);
113                             izvozHtml(header, upit,
btn_polozeni_po_kolegijima.getText());
114                             txt_greska.setText("Query: " + upit);
115                     }
116             });
```

```
117             btn_polozeni_po_kolegijima.setBounds(170, 62, 150, 40);
118             prvi_frame.getContentPane().add(btn_polozeni_po_kolegijima);
119 
120             prvi_scrollpane = new JScrollPane();
121             prvi_scrollpane.setBounds(330, 11, 454, 263);
122             prvi_frame.getContentPane().add(prvi_scrollpane);
123 
124             prva_tablica = new JTable();
125             DefaultTableModel model = new DefaultTableModel(0, 0);
126             String header[] = new String[] {"POPIS OVDJE"};
127             model.setColumnIdentifiers(header);
128             prva_tablica.setModel(model);
129 
130             prvi_scrollpane.setViewportView(prva_tablica);
131 
132             JPanel panel = new JPanel();
133             panel.setBorder(new LineBorder(new Color(0, 0, 0), 1, true));
134             panel.setBounds(10, 133, 310, 141);
135             prvi_frame.getContentPane().add(panel);
136             panel.setLayout(null);
137 
138             JLabel lbl_jmbag = new JLabel("JMBAG");
139             lbl_jmbag.setBounds(5, 17, 60, 14);
140             panel.add(lbl_jmbag);
141 
142             JLabel lbl_ime = new JLabel("Ime");
143             lbl_ime.setBounds(5, 42, 60, 14);
144             panel.add(lbl_ime);
145 
146             JLabel lbl_prezime = new JLabel("Prezime");
147             lbl_prezime.setBounds(5, 67, 60, 14);
148             panel.add(lbl_prezime);
149 
150             JLabel lbl_spol = new JLabel("Spol");
151             lbl_spol.setBounds(5, 92, 60, 14);
152             panel.add(lbl_spol);
153 
154             JLabel lbl_godina = new JLabel("Godina");
155             lbl_godina.setBounds(5, 117, 60, 14);
156             panel.add(lbl_godina);
157 
158             txt_jmbag = new JTextField();
159             txt_jmbag.setBounds(69, 11, 235, 20);
160             panel.add(txt_jmbag);
161             txt_jmbag.setColumns(10);
162 
163             txt_ime = new JTextField();
164             txt_ime.setBounds(69, 36, 235, 20);
165             panel.add(txt_ime);
166             txt_ime.setColumns(10);
167 
168             txt_prezime = new JTextField();
169             txt_prezime.setBounds(69, 61, 235, 20);
170             panel.add(txt_prezime);
171             txt_prezime.setColumns(10);
172 
173             ButtonGroup spolovi = new ButtonGroup();
174             JRadioButton rdbtn_m = new JRadioButton("M");
175             rdbtn_m.setBounds(69, 88, 50, 23);
176             panel.add(rdbtn_m);
177             spolovi.add(rdbtn_m);
178             JRadioButton rdbtn_z = new JRadioButton("Z");
179             rdbtn_z.setBounds(140, 88, 50, 23);
180             panel.add(rdbtn_z);
181             spolovi.add(rdbtn_z);
182 
183             txt_godina = new JTextField();
184             txt_godina.setBounds(69, 111, 120, 20);
185             panel.add(txt_godina);
186             txt_godina.setColumns(10);
```

```
188             btn_unesi = new JButton("UNESI");
189             btn_unesi.setFont(new Font("Tahoma", Font.PLAIN, 11));
190             btn_unesi.setBounds(199, 92, 105, 39);
191             panel.add(btn_unesi);
192 
193             btn_unesi.addActionListener(new ActionListener() {
194                     public void actionPerformed(ActionEvent e) {
195                             try {
196                                     int jmbag = Integer.parseInt(txt_jmbag.getText());
197                                     String ime = txt_ime.getText();
198                                     String prezime = txt_prezime.getText();
199                                     int godina = Integer.parseInt(txt_godina.getText());
200                                     int spol = 0;
201                                     if (rdbtn_m.isSelected()) {
202                                             spol = 1;
203                                     }
204                                     else if (rdbtn_z.isSelected()) {
205                                             spol = 2;
206                                     }
207                                     databaseConnection spajanje = new
databaseConnection();
208                                     String upit = "INSERT INTO studenti VALUES ('" +
jmbag + "', '" + ime + "', '" + prezime + "', '" + spol + "', '" + godina + "');";
209                                     spajanje.izmjena(upit);
210 
211                             } catch (Exception gre) {
212                                     txt_greska.setText("Greška: " + gre.getMessage());
213                             }
214                     }
215             });
216 
217             txt_greska = new JTextField();
218             txt_greska.setBounds(10, 282, 774, 23);
219             prvi_frame.getContentPane().add(txt_greska);
220             txt_greska.setColumns(10);
221             txt_greska.setVisible(true);
222             txt_greska.setEditable(false);
223 
224             lbl_unos_novog_studenta = new JLabel("UNOS NOVOG STUDENTA:");
225             lbl_unos_novog_studenta.setBounds(10, 113, 310, 14);
226             prvi_frame.getContentPane().add(lbl_unos_novog_studenta);
227     }
228 
229     public void izvozHtml (String header[], String upit, String naziv) {
230             int dijalogGumb = JOptionPane.YES_NO_OPTION;
231             int dijalogRezultat = JOptionPane.showConfirmDialog(null, "Izvoz u HTML
datoteku? \nDatoteka će biti izvezena na radnu površinu!", "HTML export", dijalogGumb);
232             if(dijalogRezultat == 0) {
233                     printToHtml izvoz = new printToHtml();
234                     try {
235                             izvoz.export(header, upit, naziv);
236                     } catch (Exception e) {
237                             txt_greska.setText("Greška: " + e.getMessage());
238                     }
239             }
240     }
241 
242     public void stvaranjeTablice (String header[], String upit) {
243             databaseConnection spajanje = new databaseConnection();
244 
245             DefaultTableModel model = new DefaultTableModel(0, 0);
246             model.setColumnIdentifiers(header);
247             prva_tablica.setModel(model);
248             spajanje.upit(upit);
249             for (int i=0; i<spajanje.testna_tablica.size(); i++) {
250                     model.addRow(new Object[]{});
251                     for (int j=0; j<spajanje.testna_tablica.get(i).size(); j++) {
252                             model.setValueAt(spajanje.testna_tablica.get(i).get(j), i,
j);
253                     }
254             }
255     }
256     }
```

12.5 Primjer uz korištenje aspekata

Primjer aplikacije uz korištenje aspekata sastoji se od četiri datoteke:

- databaseConnectionAs.java
- printToHtmlAs.java
- prozorAs.java
- aspekti.aj

Datoteka `databaseConnectionAs.java` predstavlja osnovnu klasu koja omogućava čitanje podataka iz baze podataka. Za razliku od verzije bez korištenja aspekata, ova klasa definira sljedeće metode:

- `execute_update (String upit_arg)` – metoda za izvršavanje izmjena u bazi.
- `execute_query (String upit_arg)` – metoda za izvršavanje upita nad bazom.
- `ispis_rezultata ()` – metoda za ispis rezultata upita u konzolu.

Datoteka `printToHtml.java` predstavlja klasu koja omogućava ispis upita nad bazom u vanjsku HTML datoteku. Klasa definira potrebne objekte (`PrintWriter`, `FileWriter`) za ispis u vanjsku datoteku. `FileWriter` omogućuje dodavanje sadržaja (*eng.* ***append***) na postojeću datoteku. U klasi `printToHtmlAs.java` se u postojeću datoteku dodaje samo sadržaj tablice – bez ostalih HTML tagova (DOCTYPE, HTML, HEAD, BODY, i dr.).

Datoteka `prozorAs.java` je skoro identična datoteci `prozor.java`. Razlika je metodi `stvaranjeTablice (String header[], String upit)` koja u verziji bez aspekata poziva metodu `databaseConnection.upit(upit)`, a u aspektnoj verziji direktno poziva metodu `databaseConnectionAs.execute_query(upit)`.

Datoteka `aspekti.aj` predstavlja aspektni dio aplikacije. Definirana je točka presjeka: konekcija ().

```
pointcut konekcija() : execution(* execute_query(..)) || execution(*
execute_update(..));
```

Točka presjeka pod imenom `konekcija` označava točke spajanja koje su zapravo izvršenja (*eng.* ***execution***) svih metoda `execute_query()` i metoda `execute_update()`.

Prije izvršenja bilo kakvog upita ili izmjene nad bazom podataka, potrebno je otvoriti konekciju prema toj bazi podataka. To je riješeno tijelom savjeta `before() : konekcija()` u

kojem se definiraju potrebni atributi za spajanje na bazu (url, user, password), te se otvara konekcija prema bazi.

Nakon izvršavanja određenog upita ili izmjene nad bazom, potrebno je tu konekciju zatvoriti. To je zadaća tijela savjeta `after() : konekcija()` koji definira zatvaranje korištene konekcije.

Također, u istoj datoteci (`aspekti.aj`) nalazi se i točka presjeka htmlExport:

```
pointcut htmlExport() : execution (* export(..));
```

Datoteka `printToHtml.java` dodaje sadržaj tablice u postojeću datoteku (bez zaglavlja HTML-a). Međutim, prije toga potrebno je tu datoteku stvoriti te ubaciti zaglavlja kako bi se ona mogla ispravno prikazati u web pretraživaču. Taj problem rješava tijelo savjeta `before() : htmlExport()`. Na radnoj površini se stvara datoteka `export.html` te se, dodavanjem potrebnih zaglavlja, datoteka priprema za ubacivanje ostalog sadržaja. Zatim, izvršavanjem kôda iz klase `printToHtml.java`, dodaje se potreban sadržaj (tablica).

Nakon dodavanja glavnog sadržaja u HTML datoteku, potrebno je zatvoriti zaglavlja HTML datoteke. Za to se brine tijelo savjeta `after() : htmlExport()` koje u datoteku dodaje potrebna završna zaglavlja te pita korisnika da li želi otvoriti stvorenu datoteku.

12.5.1 databaseConnectionAs.java

```
01  package dduketis;
02  import java.sql.Connection;
03  import java.sql.ResultSet;
04  import java.sql.SQLException;
05  import java.sql.Statement;
06  import java.sql.ResultSetMetaData;
07  import java.util.ArrayList;
08  import java.util.List;
09
10  class databaseConnectionAs {
11  static Connection conn = null;
12  static Statement statement = null;
13  ResultSet resultSet = null;
14  String url = null;
15  String user = null;
16  String password = null;
17
18  List<List<String>> testna_tablica = new ArrayList<List<String>>();
19  void execute_update (String upit_arg) {
20          if(statement != null) {
21                  try {
22                          statement.executeUpdate(upit_arg);
23                  } catch (SQLException e) {
24                          System.out.println(e.getMessage());
25                  }
26          }
27  }
```

```
28      void execute_query (String upit_arg) {
29              if(statement != null) {
30                      try {
31                              resultSet = statement.executeQuery(upit_arg);
32                      } catch (SQLException e) {
33                              System.out.println(e.getMessage());
34                      }
35              }
36              try {
37                      if (!resultSet.isBeforeFirst()) {
38                              System.out.println("resultSet prazan - no data");
39                      }
40              } catch (SQLException e) {
41                      System.out.println("Error: " + e.getMessage());
42              }
43              int columnCount=0;
44              try {
45                      testna_tablica.clear();
46                      ResultSetMetaData metadata = this.resultSet.getMetaData();
47                      columnCount = metadata.getColumnCount();
48              } catch (SQLException e) {
49                      System.out.println("Error: " + e.getMessage());
50              }


51              if (this.resultSet != null) {
52                      try {
53                              int i = 0;
54                              while (this.resultSet.next()) {
55                                      this.testna_tablica.add(new ArrayList<String>());
56                                      for (int j=1; j<columnCount+1; j++) {
57                              this.testna_tablica.get(i).add(this.resultSet.getString(j));
58                                      }
59                                      i++;
60                              }
61                      } catch (SQLException e) {
62                              System.out.println(e.getMessage());
63                      }
64              }
65      }
66      void ispis_rezultata () {
67              for (int i=0; i<this.testna_tablica.size(); i++) {
68                      for (int j=0; j<this.testna_tablica.get(i).size(); j++) {
69                              System.out.print(this.testna_tablica.get(i).get(j) + " ");
70                      }
71                      System.out.println("");
72              }
73      }
74      }
```

12.5.2 PrintToHtmlAs.java

```
01      package dduketis;
02      import java.io.*;
03
04      public class printToHtmlAs {
05
06      static String filepath;
07      FileWriter writerDodaj;
08      databaseConnectionAs spajanje = new databaseConnectionAs();
09
10      void export (String header[], String upit, String naziv) throws Exception {
11
12              spajanje.execute_query(upit);
13
14              try {
15                      writerDodaj = new FileWriter (filepath, true);
16
17                      writerDodaj.write("<h1>" + naziv + "</h1>\n");
18                      writerDodaj.write("<h2>" + filepath + "</h2>\n");
19                      writerDodaj.write("<table style='width:100%'>\n");
20                      writerDodaj.write("<tr>\n");
21                      for (int i=0; i<header.length; i++) {
22                              writerDodaj.write("<th>" + header[i] + "</th>\n");
23                      }
24                      writerDodaj.write("</tr>\n");
25
26                      for (int i=0; i<spajanje.testna_tablica.size(); i++) {
27                              writerDodaj.write("<tr>\n");
28                              for (int j=0; j<spajanje.testna_tablica.get(i).size(); j++) {
29                                      writerDodaj.write("<td>" +
spajanje.testna_tablica.get(i).get(j) + "</td>\n");
30                              }
31                              writerDodaj.write("</tr>\n");
32                      }
33                      writerDodaj.write("</table>\n");
34
35                      writerDodaj.close();
36              } catch (IOException e) {
37                      writerDodaj.close();
38                      System.err.println(e.getMessage());
39              }
40      }
41      }
```

12.5.3 prozor.java

```
001  package dduketis;
002
003  import javax.swing.JFrame;
004  import javax.swing.JTable;
005  import javax.swing.JButton;
006  import javax.swing.table.DefaultTableModel;
007  import javax.swing.ButtonGroup;
008  import javax.swing.JOptionPane;
009  import javax.swing.JScrollPane;
010  import javax.swing.JLabel;
011  import javax.swing.JTextField;
012  import javax.swing.JRadioButton;
013  import javax.swing.JPanel;
014  import javax.swing.border.LineBorder;
015  import java.awt.Color;
016  import java.awt.EventQueue;
017  import java.awt.Font;
018  import java.awt.event.ActionListener;
019  import java.awt.event.ActionEvent;
020
021  public class prozorAs {
022
023  private JFrame prvi_frame;
024  private JTable prva_tablica;
025  private JScrollPane prvi_scrollpane;
026  private JButton btn_popis_studenata;
027  private JButton btn_popis_kolegija;
028  private JButton btn_polozeni_po_studentima;
029  private JButton btn_polozeni_po_kolegijima;
030  private JButton btn_unesi;
031  private JTextField txt_jmbag;
032  private JTextField txt_ime;
033  private JTextField txt_prezime;
034  private JTextField txt_godina;
035  private JTextField txt_greska;
036
037  private JLabel lbl_unos_novog_studenta;
038
039  // pokretanje aplikacije
040  public static void main(String[] args) {
041          EventQueue.invokeLater(new Runnable() {
042                  public void run() {
043                          try {
044                                  prozorAs window = new prozorAs();
045                                  window.prvi_frame.setVisible(true);
046                          } catch (Exception e) {
047                                  e.printStackTrace();
048                          }
049                  }
050          });
051  }
052  // stvaranje aplikacije
053  public prozorAs() {
054          initialize();
055  }
056  private void initialize() {
057          prvi_frame = new JFrame();
058          prvi_frame.setBounds(100, 100, 800, 344);
059          prvi_frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
060          prvi_frame.getContentPane().setLayout(null);
061          prvi_frame.setLocationRelativeTo(null);
062          prvi_frame.setResizable(false);
```

```
064             btn_popis_studenata = new JButton("Popis studenata");
065             btn_popis_studenata.setFont(new Font("Tahoma", Font.PLAIN, 10));
066             btn_popis_studenata.addActionListener(new ActionListener() {
067                     public void actionPerformed(ActionEvent e) {
068                             String header[] = new String[] {"JMBAG", "IME", "PREZIME",
"SPOL", "GODINA"};
069                             String upit = "SELECT studenti.student_id,
studenti.student_ime, studenti.student_prezime, spol.spol_naziv,
studenti.student_godina_studija FROM studenti JOIN spol ON
studenti.student_spol=spol.spol_id ";
070                             stvaranjeTablice(header, upit);
071                             izvozHtml(header, upit, btn_popis_studenata.getText());
072                             txt_greska.setText("Query: " + upit);
073                     }
074             });
075             btn_popis_studenata.setBounds(10, 11, 150, 40);
076             prvi_frame.getContentPane().add(btn_popis_studenata);
077
078             btn_popis_kolegija = new JButton("Popis kolegija");
079             btn_popis_kolegija.setFont(new Font("Tahoma", Font.PLAIN, 10));
080             btn_popis_kolegija.addActionListener(new ActionListener() {
081                     public void actionPerformed(ActionEvent e) {
082                             String header[] = new String[] {"ID", "NAZIV", "GODINA"};
083                             String upit = "SELECT * FROM kolegiji";
084                             stvaranjeTablice(header, upit);
085                             izvozHtml(header, upit, btn_popis_kolegija.getText());
086                             txt_greska.setText("Query: " + upit);
087                     }
088             });
089             btn_popis_kolegija.setBounds(170, 11, 150, 40);
090             prvi_frame.getContentPane().add(btn_popis_kolegija);
091
092             btn_polozeni_po_studentima = new JButton("Položeni po studentima");
093             btn_polozeni_po_studentima.setFont(new Font("Tahoma", Font.PLAIN, 10));
094             btn_polozeni_po_studentima.addActionListener(new ActionListener() {
095                     public void actionPerformed(ActionEvent e) {
096                             String header[] = new String[] {"JMBAG", "IME", "PREZIME",
"ID KOLEGIJA", "NAZIV", "OCJENA"};
097                             String upit = "SELECT studenti.student_id,
studenti.student_ime, studenti.student_prezime, kolegiji.kolegij_id,
kolegiji.kolegij_naziv, polozeni_kolegiji.ocjena FROM studenti JOIN polozeni_kolegiji ON
studenti.student_id=polozeni_kolegiji.student_id JOIN kolegiji ON
polozeni_kolegiji.kolegij_id=kolegiji.kolegij_id;";
098                             stvaranjeTablice(header, upit);
099                             izvozHtml(header, upit,
btn_polozeni_po_studentima.getText());
100                             txt_greska.setText("Query: " + upit);
101                     }
102             });
103             btn_polozeni_po_studentima.setBounds(10, 62, 150, 40);
104             prvi_frame.getContentPane().add(btn_polozeni_po_studentima);
105
106             btn_polozeni_po_kolegijima = new JButton("Položeni po kolegijima");
107             btn_polozeni_po_kolegijima.setFont(new Font("Tahoma", Font.PLAIN, 10));
108             btn_polozeni_po_kolegijima.addActionListener(new ActionListener() {
109                     public void actionPerformed(ActionEvent e) {
110                             String header[] = new String[] {"JMBAG", "IME", "PREZIME",
"ID KOLEGIJA", "NAZIV", "OCJENA"};
111                             String upit = "SELECT studenti.student_id,
studenti.student_ime, studenti.student_prezime, kolegiji.kolegij_id,
kolegiji.kolegij_naziv, polozeni_kolegiji.ocjena FROM studenti JOIN polozeni_kolegiji ON
studenti.student_id=polozeni_kolegiji.student_id JOIN kolegiji ON
polozeni_kolegiji.kolegij_id=kolegiji.kolegij_id ORDER BY kolegiji.kolegij_id;";
112                             stvaranjeTablice(header, upit);
113                             izvozHtml(header, upit,
btn_polozeni_po_kolegijima.getText());
114                             txt_greska.setText("Query: " + upit);
115                     }
116             });
117             btn_polozeni_po_kolegijima.setBounds(170, 62, 150, 40);
118             prvi_frame.getContentPane().add(btn_polozeni_po_kolegijima);
119
```

```
120             prvi_scrollpane = new JScrollPane();
121             prvi_scrollpane.setBounds(330, 11, 454, 263);
122             prvi_frame.getContentPane().add(prvi_scrollpane);
123 
124             prva_tablica = new JTable();
125             DefaultTableModel model = new DefaultTableModel(0, 0);
126             String header[] = new String[] {"POPIS OVDJE"};
127             model.setColumnIdentifiers(header);
128             prva_tablica.setModel(model);
129 
130             prvi_scrollpane.setViewportView(prva_tablica);
131 
132             JPanel panel = new JPanel();
133             panel.setBorder(new LineBorder(new Color(0, 0, 0), 1, true));
134             panel.setBounds(10, 133, 310, 141);
135             prvi_frame.getContentPane().add(panel);
136             panel.setLayout(null);
137 
138             JLabel lbl_jmbag = new JLabel("JMBAG");
139             lbl_jmbag.setBounds(5, 17, 60, 14);
140             panel.add(lbl_jmbag);
141 
142             JLabel lbl_ime = new JLabel("Ime");
143             lbl_ime.setBounds(5, 42, 60, 14);
144             panel.add(lbl_ime);
145 
146             JLabel lbl_prezime = new JLabel("Prezime");
147             lbl_prezime.setBounds(5, 67, 60, 14);
148             panel.add(lbl_prezime);
149 
150             JLabel lbl_spol = new JLabel("Spol");
151             lbl_spol.setBounds(5, 92, 60, 14);
152             panel.add(lbl_spol);
153 
154             JLabel lbl_godina = new JLabel("Godina");
155             lbl_godina.setBounds(5, 117, 60, 14);
156             panel.add(lbl_godina);
157 
158             txt_jmbag = new JTextField();
159             txt_jmbag.setBounds(69, 11, 235, 20);
160             panel.add(txt_jmbag);
161             txt_jmbag.setColumns(10);
162 
163             txt_ime = new JTextField();
164             txt_ime.setBounds(69, 36, 235, 20);
165             panel.add(txt_ime);
166             txt_ime.setColumns(10);
167 
168             txt_prezime = new JTextField();
169             txt_prezime.setBounds(69, 61, 235, 20);
170             panel.add(txt_prezime);
171             txt_prezime.setColumns(10);
172 
173             ButtonGroup spolovi = new ButtonGroup();
174             JRadioButton rdbtn_m = new JRadioButton("M");
175             rdbtn_m.setBounds(69, 88, 50, 23);
176             panel.add(rdbtn_m);
177             spolovi.add(rdbtn_m);
178             JRadioButton rdbtn_z = new JRadioButton("Z");
179             rdbtn_z.setBounds(140, 88, 50, 23);
180             panel.add(rdbtn_z);
181             spolovi.add(rdbtn_z);
182 
183             txt_godina = new JTextField();
184             txt_godina.setBounds(69, 111, 120, 20);
185             panel.add(txt_godina);
186             txt_godina.setColumns(10);
188             btn_unesi = new JButton("UNESI");
189             btn_unesi.setFont(new Font("Tahoma", Font.PLAIN, 11));
190             btn_unesi.setBounds(199, 92, 105, 39);
191             panel.add(btn_unesi);
192 
```

```
193             btn_unesi.addActionListener(new ActionListener() {
194                     public void actionPerformed(ActionEvent e) {
195                             try {
196                                     int jmbag = Integer.parseInt(txt_jmbag.getText());
197                                     String ime = txt_ime.getText();
198                                     String prezime = txt_prezime.getText();
199                                     int godina = Integer.parseInt(txt_godina.getText());
200                                     int spol = 0;
201                                     if (rdbtn_m.isSelected()) {
202                                             spol = 1;
203                                     }
204                                     else if (rdbtn_z.isSelected()) {
205                                             spol = 2;
206                                     }
207                                     databaseConnectionAs spajanje = new
databaseConnectionAs();
208                                     String upit = "INSERT INTO studenti VALUES ('" +
jmbag + "', '" + ime + "', '" + prezime + "', '" + spol + "', '" + godina + "');";
209                                     spajanje.execute_update(upit);
210
211                             } catch (Exception gre) {
212                                     txt_greska.setText("Greška: " + gre.getMessage());
213                             }
214                     }
215             });
216
217             txt_greska = new JTextField();
218             txt_greska.setBounds(10, 282, 774, 23);
219             prvi_frame.getContentPane().add(txt_greska);
220             txt_greska.setColumns(10);
221             txt_greska.setVisible(true);
222             txt_greska.setEditable(false);
223
224             lbl_unos_novog_studenta = new JLabel("UNOS NOVOG STUDENTA:");
225             lbl_unos_novog_studenta.setBounds(10, 113, 310, 14);
226             prvi_frame.getContentPane().add(lbl_unos_novog_studenta);
227     }


229     public void izvozHtml (String header[], String upit, String naziv) {
230             int dijalogGumb = JOptionPane.YES_NO_OPTION;
231             int dijalogRezultat = JOptionPane.showConfirmDialog(null, "Izvoz u HTML
datoteku? \nDatoteka će biti izvezena na radnu površinu!", "HTML export", dijalogGumb);
232             if(dijalogRezultat == 0) {
233                     printToHtmlAs izvoz = new printToHtmlAs();
234                     try {
235                             izvoz.export(header, upit, naziv);
236                     } catch (Exception e) {
237                             txt_greska.setText("Greška: " + e.getMessage());
238                     }
239             }
240     }
241
242     public void stvaranjeTablice (String header[], String upit) {
243             databaseConnectionAs spajanje = new databaseConnectionAs();
244
245             DefaultTableModel model = new DefaultTableModel(0, 0);
246             model.setColumnIdentifiers(header);
247             prva_tablica.setModel(model);
248
249             spajanje.execute_query(upit);
250             for (int i=0; i<spajanje.testna_tablica.size(); i++) {
251                     model.addRow(new Object[]{});
252                     for (int j=0; j<spajanje.testna_tablica.get(i).size(); j++) {
253                             model.setValueAt(spajanje.testna_tablica.get(i).get(j), i,
j);
254                     }
255             }
256     }
257     }
```

12.5.4 aspekti.aj

```
001     package dduketis;
002
003     import java.io.FileWriter;
004     import java.io.IOException;
005     import java.io.PrintWriter;
006     import java.io.*;
007     import javax.swing.JOptionPane;
008     import java.awt.Desktop;
009     import java.sql.DriverManager;
010     import java.sql.SQLException;
011
012     import javax.swing.filechooser.FileSystemView;
013
014     public aspect aspekti {
015
016             pointcut konekcija() : execution(* execute_query(..)) || execution(*
execute_update(..));
017
018             //savjet za otvaranje konekcije i stvaranje statementa prije pristupa bazi
podataka
019             before() : konekcija() {
020                     String url = "jdbc:mysql://localhost/testbaza";
021                 String user = "testkorisnik";
022                 String password = "testlozinka";
023
024                 try {
025                     databaseConnectionAs.conn = DriverManager.getConnection(url, user,
password);
026                 } catch (SQLException e) {
027                     System.out.println("\nUnable to connect to database: " + e.getMessage());
028                 }
029
030                 if (databaseConnectionAs.conn != null) {
031                     try {
032                             databaseConnectionAs.statement =
databaseConnectionAs.conn.createStatement();
033                     } catch (SQLException e) {
034                             System.out.println("Unable to create statement: " +
e.getMessage());
035                     }
036                 }
037             }
038
039             //savjet za zatvaranje konekcije nakon obradenog upita nad bazom podataka
040             after() : konekcija() {
041                 try {
042                     databaseConnectionAs.conn.close();
043                 } catch (SQLException e) {
044                     System.out.println("\nError while disconnecting the database: " +
e.getMessage());
045                 }
046             }
```

```
047             pointcut htmlExport() : execution (* export(..));
048
049             before() : htmlExport() {
050                     FileSystemView filesys = FileSystemView.getFileSystemView();
//filesys.getHomeDirectory();
051                     printToHtmlAs.filepath = filesys.getHomeDirectory() + "\\export.html";
052
053                     PrintWriter writerPocetak;
054                     try {
055                             writerPocetak = new PrintWriter(printToHtmlAs.filepath, "UTF-8");
056                             writerPocetak.println("<!DOCTYPE html>");
057                             writerPocetak.println("<HTML>");
058                             writerPocetak.println("<HEAD>");
059                             writerPocetak.println("<TITLE>Export from JTable</TITLE>");
060                             writerPocetak.println("<STYLE>");
061                             writerPocetak.println("table, th, td {");
062                             writerPocetak.println("border: 1px solid black;");
063                             writerPocetak.println("border-collapse: collapse;");
064                             writerPocetak.println("</STYLE>");
065                             writerPocetak.println("</HEAD>");
066                             writerPocetak.println("<BODY>");
067                             writerPocetak.close();
068                     } catch (IOException e) {
069                             e.getMessage();
070                     }
071             }
072
073             after() : htmlExport() {
074                     FileSystemView filesys = FileSystemView.getFileSystemView();
//filesys.getHomeDirectory();
075                     printToHtmlAs.filepath = filesys.getHomeDirectory() + "\\export.html";
076
077                     FileWriter writerKraj;
078                     try {
079                             writerKraj = new FileWriter (printToHtmlAs.filepath, true);
080                             writerKraj.write("</BODY>\n");
081                             writerKraj.write("</HTML>");
082                             writerKraj.close();
083                     } catch (IOException e) {
084                             e.getMessage();
085                     }
086
087                     int dijalogGumb = JOptionPane.YES_NO_OPTION;
088                     int dijalogRezultat = JOptionPane.showConfirmDialog(null, "Lokacija
datoteke: " + printToHtmlAs.filepath + "\nOtvori izvezenu HTML datoteku?", "Otvaranje HTML-a",
dijalogGumb);
089                     if(dijalogRezultat == 0) {
090                             File f = new File (printToHtmlAs.filepath);
091                             if (f.exists()) {
092                                     if (Desktop.isDesktopSupported()) {
093                                             try {
094                                                     Desktop.getDesktop().open(f);
095                                             } catch (IOException e) {
096                                                     System.out.println(e.getMessage());
097                                             }
098                                     }
099                                     else {
100                                             System.out.println("Datoteka ne postoji");
101                                     }
102                             }
103                     }
104             }
105     }
```

12.6 Kôd za stvaranje baze podataka

```
001    -- phpMyAdmin SQL Dump
002    -- version 4.2.11
003    -- http://www.phpmyadmin.net
004    --
005    -- Host: 127.0.0.1
006    -- Generation Time: Aug 04, 2015 at 10:59 AM
007    -- Server version: 5.6.21
008    -- PHP Version: 5.6.3
009
010    SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
011    SET time_zone = "+00:00";
012
013
014    /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
015    /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
016    /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
017    /*!40101 SET NAMES utf8 */;
018
019    --
020    -- Database: `testbaza`
021    --
022
023    -- --------------------------------------------------------
024
025    --
026    -- Table structure for table `kolegiji`
027    --
028
029    CREATE TABLE IF NOT EXISTS `kolegiji` (
030    `kolegij_id` int(5) NOT NULL,
031      `kolegij_naziv` varchar(50) NOT NULL,
032      `kolegij_godina` int(1) NOT NULL
033    ) ENGINE=InnoDB AUTO_INCREMENT=53 DEFAULT CHARSET=latin1;
034
035    --
036    -- Dumping data for table `kolegiji`
037    --
038
039    INSERT INTO `kolegiji` (`kolegij_id`, `kolegij_naziv`, `kolegij_godina`) VALUES
040    (1, 'Financijska Matematika', 1),
041    (2, 'Marketing', 1),
042    (3, 'Marketing', 1),
043    (4, 'Statistika', 1),
044    (5, 'Trgovacko Pravo', 1),
045    (6, 'Informaticko Pravo', 1),
046    (7, 'Ured I Uredsko Poslovanje', 1),
047    (8, 'Baze Podataka', 2),
048    (9, 'Informatika u Primjeni', 2),
049    (10, 'Kvantitativni Menadzment', 2),
050    (11, 'Matematicke Metode za Poslovne Analize', 2),
051    (12, 'Poslovni Informacijski Sustavi Drzvane Uprave', 2),
052    (13, 'Elektronicko Poslovanje', 2),
053    (14, 'Financije Malih i Srednjih Poduzeca', 1),
054    (15, 'IS Malih i Srednjih Poduzeca', 1),
055    (16, 'Kontroling i Politika Kvalitete', 2),
056    (17, 'Medunarodna Trgovina', 1),
057    (18, 'Transport Spedicija i Osiguranje', 1),
058    (19, 'Izgradnja Web Aplikacija', 3),
059    (20, 'Modeliranje Poslovnih Pravila', 3),
060    (21, 'Multimedijski Sustavi', 3),
061    (22, 'Napredno Programiranje', 3),
062    (23, 'Poslovno Planiranje', 3),
063    (24, 'Primjena Mreznih Servisa', 3),
064    (25, 'Programiranje', 3),
065    (26, 'Sigurnost Informacijskih Sustava', 3),
066    (27, 'Upravljanje Informatickim Uslugama', 3),
067    (28, 'Engleski 1', 1),
068    (29, 'Informatika 1', 1),
069    (30, 'Engleski 2', 2),
070    (31, 'Informatika 2', 2),
071    (32, 'Obrada Teksta i Slike', 1),
072    (33, 'Organizacija', 1),
```

```
(34, 'Osnove Ekonomije', 1),
(35, 'Poslovno Komuniciranje', 1),
(36, 'Statistika', 1),
(37, 'Matematika 1', 1),
(38, 'Racunovodstvo', 2),
(39, 'Teorija Sustava', 2),
(40, 'Marketing', 3),
(41, 'Ergonomija', 4),
(42, 'IS u Potpori Upravljanju i Odlucivanju', 4),
(43, 'Razvoj IS', 4),
(44, 'Ekonomika', 4),
(45, 'Dinamicke Web Aplikacije', 4),
(46, 'Management Kvalitete', 4),
(47, 'Softversko Inzenjerstvo', 4),
(48, 'Teorija Informacija', 4),
(49, 'ICT i Drustvo', 5),
(50, 'Informacijski Management', 5),
(51, 'Modeliranje i Simulacija', 5),
(52, 'Poduzetnistvo i Gospodarstvo', 5);

-- --------------------------------------------------------

--
-- Table structure for table `polozeni_kolegiji`
--

CREATE TABLE IF NOT EXISTS `polozeni_kolegiji` (
  `student_id` bigint(10) NOT NULL,
  `kolegij_id` int(5) NOT NULL,
  `ocjena` int(1) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `polozeni_kolegiji`
--

INSERT INTO `polozeni_kolegiji` (`student_id`, `kolegij_id`, `ocjena`) VALUES

(1394984532, 1, 3),
(1394984532, 2, 5),
(1394984532, 3, 2),
(1394984532, 4, 2),
(1394984532, 5, 5),
(1394984532, 6, 3),
(1394984532, 7, 3),
(1394984532, 14, 5),
(1394984532, 15, 4),
(1394984532, 17, 4),
(1394984532, 18, 2),
(1473419155, 1, 2),
(1473419155, 2, 4),
(1473419155, 3, 3),
(1473419155, 4, 4),
(1473419155, 5, 2),
(1473419155, 6, 3),
(1473419155, 7, 3),
(1473419155, 8, 4),
(1473419155, 9, 2),
(1473419155, 10, 3),
(1473419155, 11, 3),
(1473419155, 12, 5),
(1473419155, 13, 5),
(1473419155, 14, 3),
(1473419155, 15, 3),
(1473419155, 16, 4),
(1473419155, 17, 3),
(1473419155, 18, 5),
(1473419155, 19, 5),
(1473419155, 20, 2),
(1473419155, 21, 4),
(1473419155, 22, 2),
(1473419155, 23, 4),
(1473419155, 24, 3),
(1473419155, 25, 5),
(1473419155, 26, 5),
(1473419155, 27, 4),
(1473419155, 28, 3),
(1473419155, 29, 2),
(1473419155, 30, 2),
(1473419155, 31, 2),
(1473419155, 32, 2),
(1473419155, 33, 5),
(1473419155, 34, 3),
(1473419155, 35, 5),
(1473419155, 36, 5),
(1473419155, 37, 4),
(1473419155, 38, 4),
(1473419155, 39, 5),
(1473419155, 40, 2),
(1876743912, 1, 5),
(1876743912, 2, 4),
(1876743912, 3, 4),
(1876743912, 4, 4),
(1876743912, 5, 2),
(1876743912, 6, 5),
(1876743912, 7, 2),
(1876743912, 14, 4),
(1876743912, 15, 5),
(1876743912, 17, 5),
(1876743912, 18, 5),
(1876743912, 28, 5),
(1876743912, 29, 5),
(1876743912, 32, 5),
(1876743912, 33, 4),
(1876743912, 34, 5),
(1876743912, 35, 5),
(1876743912, 36, 4),
(1876743912, 37, 5),
(2131875260, 1, 5),
(2131875260, 2, 2),
(2131875260, 3, 3),
(2131875260, 4, 2),
(2131875260, 5, 4),
(2131875260, 6, 3),
(2131875260, 7, 4),
(2131875260, 14, 5),
(2131875260, 15, 4),
(2131875260, 17, 2),
(2131875260, 18, 5),
(2131875260, 28, 4),
(2131875260, 29, 4),
(2131875260, 32, 5),
(2131875260, 33, 4),
(2131875260, 34, 3),
(2131875260, 35, 3),
(2131875260, 36, 4),
(2131875260, 37, 3),
(2517879602, 1, 2),
(2517879602, 2, 3),
(2517879602, 3, 3),
(2517879602, 4, 3),
(2517879602, 5, 4),
(2517879602, 6, 2),
(2517879602, 7, 2),
(2517879602, 8, 2),
(2517879602, 9, 2),
(2517879602, 10, 2),
(2517879602, 11, 4),
(2517879602, 12, 5),
(2517879602, 13, 4),
(2517879602, 14, 3),
(2517879602, 15, 5),
(2517879602, 16, 5),
(2517879602, 17, 2),
```

```
(2517879602, 18, 4),
(2517879602, 28, 3),
(2517879602, 29, 2),
(2517879602, 30, 5),
(2517879602, 31, 5),
(2517879602, 32, 2),
(2517879602, 33, 4),
(2517879602, 34, 5),
(2517879602, 35, 4),
(2517879602, 36, 2),
(2517879602, 37, 3),
(2517879602, 38, 5),
(2517879602, 39, 2),
(2936048095, 1, 2),
(2936048095, 2, 5),
(2936048095, 3, 3),
(2936048095, 4, 5),
(2936048095, 5, 4),
(2936048095, 6, 4),
(2936048095, 7, 2),
(2936048095, 14, 5),
(2936048095, 15, 5),
(2936048095, 17, 3),
(2936048095, 18, 3),
(2936048095, 28, 5),
(2936048095, 29, 2),
(2936048095, 32, 5),
(2936048095, 33, 5),
(2936048095, 34, 3),
(2936048095, 35, 4),
(2936048095, 36, 5),
(2936048095, 37, 3),
(3736706852, 1, 5),
(3736706852, 2, 4),
(3736706852, 3, 2),
(3736706852, 4, 2),
(3736706852, 5, 4),
(3736706852, 6, 3),
(3736706852, 7, 4),
(3736706852, 8, 4),
(3736706852, 9, 5),
(3736706852, 10, 2),
(3736706852, 11, 2),
(3736706852, 12, 4),
(3736706852, 13, 2),
(3736706852, 14, 3),
(3736706852, 15, 4),
(3736706852, 16, 4),
(3736706852, 17, 3),
(3736706852, 18, 5),
(3736706852, 19, 2),
(3736706852, 20, 5),
(3736706852, 21, 3),
(3736706852, 22, 5),
(3736706852, 23, 4),
(3736706852, 24, 4),
(3736706852, 25, 3),
(3736706852, 26, 5),
(3736706852, 27, 5),
(3736706852, 28, 3),
(3736706852, 29, 2),
(3736706852, 30, 4),
(3736706852, 31, 3),
(3736706852, 32, 3),
(3736706852, 33, 4),
(3736706852, 34, 3),
(3736706852, 35, 5),
(3736706852, 36, 3),
(3736706852, 37, 3),
(3736706852, 38, 5),
(3736706852, 39, 2),
(3736706852, 40, 4),
(4445574049, 1, 2),
(4445574049, 2, 3),
(4445574049, 3, 3),
(4445574049, 4, 3),
(4445574049, 5, 2),
(4445574049, 6, 4),
(4445574049, 7, 4),
(4445574049, 8, 3),
(4445574049, 9, 2),
(4445574049, 10, 3),
(4445574049, 11, 3),
(4445574049, 12, 2),
(4445574049, 13, 3),
(4445574049, 14, 4),
(4445574049, 15, 4),
(4445574049, 16, 3),
(4445574049, 17, 3),
(4445574049, 18, 3),
(4445574049, 19, 5),
(4445574049, 20, 4),
(4445574049, 21, 2),
(4445574049, 22, 3),
(4445574049, 23, 5),
(4445574049, 24, 3),
(4445574049, 25, 3),
(4445574049, 26, 2),
(4445574049, 27, 3),
(4445574049, 28, 3),
(4445574049, 29, 3),
(4445574049, 30, 3),
(4445574049, 31, 3),
(4445574049, 32, 4),
(4445574049, 33, 3),
(4445574049, 34, 2),
(4445574049, 35, 3),
(4445574049, 36, 4),
(4445574049, 37, 4),
(4445574049, 38, 3),
(4445574049, 39, 4),
(4445574049, 40, 2),
(4854107160, 1, 3),
(4854107160, 2, 2),
(4854107160, 3, 4),
(4854107160, 4, 2),
(4854107160, 5, 5),
(4854107160, 6, 4),
(4854107160, 7, 3),
(4854107160, 8, 2),
(4854107160, 9, 3),
(4854107160, 10, 3),
(4854107160, 11, 5),
(4854107160, 12, 3),
(4854107160, 13, 5),
(4854107160, 14, 4),
(4854107160, 15, 4),
(4854107160, 16, 2),
(4854107160, 17, 3),
(4854107160, 18, 5),
(4854107160, 19, 2),
(4854107160, 20, 2),
(4854107160, 21, 3),
(4854107160, 22, 4),
(4854107160, 23, 3),
(4854107160, 24, 4),
(4854107160, 25, 5),
(4854107160, 26, 4),
(4854107160, 27, 5),
(4854107160, 28, 4),
(4854107160, 29, 5),
(4854107160, 30, 3),
(4854107160, 31, 5),
(4854107160, 32, 2),
(4854107160, 33, 3),
(4854107160, 34, 2),
(4854107160, 35, 3),
(4854107160, 36, 5),
(4854107160, 37, 5),
(4854107160, 38, 5),
(4854107160, 39, 2),
(4854107160, 40, 4),
(5321388965, 1, 2),
(5321388965, 2, 2),
(5321388965, 3, 5),
(5321388965, 4, 4),
(5321388965, 5, 2),
(5321388965, 6, 4),
(5321388965, 7, 5),
(5321388965, 14, 5),
(5321388965, 15, 2),
(5321388965, 17, 2),
(5321388965, 18, 2),
(5321388965, 28, 3),
(5321388965, 29, 2),
(5321388965, 32, 2),
(5321388965, 33, 4),
(5321388965, 34, 3),
(5321388965, 35, 3),
(5321388965, 36, 5),
(5321388965, 37, 2),
(5379417581, 1, 2),
(5379417581, 2, 3),
(5379417581, 3, 4),
(5379417581, 4, 4),
(5379417581, 5, 5),
(5379417581, 6, 2),
(5379417581, 7, 5),
(5379417581, 14, 5),
(5379417581, 15, 3),
(5379417581, 17, 3),
(5379417581, 18, 2),
(5379417581, 28, 4),
(5379417581, 29, 4),
(5379417581, 32, 5),
(5379417581, 33, 2),
(5379417581, 34, 3),
(5379417581, 35, 5),
(5379417581, 36, 5),
(5379417581, 37, 2),
(6077658506, 1, 3),
(6077658506, 2, 5),
(6077658506, 3, 5),
(6077658506, 4, 4),
(6077658506, 5, 3),
(6077658506, 6, 4),
(6077658506, 7, 4),
(6077658506, 8, 4),
(6077658506, 9, 5),
(6077658506, 10, 4),
(6077658506, 11, 5),
(6077658506, 12, 5),
(6077658506, 13, 4),
(6077658506, 14, 2),
(6077658506, 15, 2),
(6077658506, 16, 3),
(6077658506, 17, 3),
(6077658506, 18, 4),
(6077658506, 28, 2),
(6077658506, 29, 2),
(6077658506, 30, 5),
(6077658506, 31, 4),
(6077658506, 32, 3),
(6077658506, 33, 3),
(6077658506, 34, 5),
(6077658506, 35, 3),
(6077658506, 36, 3),
(6077658506, 37, 3),
(6077658506, 38, 3),
(6077658506, 39, 5),
(6679470818, 1, 2),
(6679470818, 2, 4),
```

```
438     (6679470818, 3, 2),     512     (7242099995, 17, 5),    586     (7429859039, 25, 3),
439     (6679470818, 4, 4),     513     (7242099995, 18, 4),    587     (7429859039, 26, 4),
440     (6679470818, 5, 2),     514     (7242099995, 28, 4),    588     (7429859039, 27, 4),
441     (6679470818, 6, 4),     515     (7242099995, 29, 3),    589     (7429859039, 28, 5),
442     (6679470818, 7, 5),     516     (7242099995, 32, 5),    590     (7429859039, 29, 5),
443     (6679470818, 8, 3),     517     (7242099995, 33, 2),    591     (7429859039, 30, 2),
444     (6679470818, 9, 2),     518     (7242099995, 34, 4),    592     (7429859039, 31, 2),
445     (6679470818, 10, 5),    519     (7242099995, 35, 3),    593     (7429859039, 32, 3),
446     (6679470818, 11, 2),    520     (7242099995, 36, 2),    594     (7429859039, 33, 4),
447     (6679470818, 12, 3),    521     (7242099995, 37, 5),    595     (7429859039, 34, 4),
448     (6679470818, 13, 2),    522     (7419952825, 1, 3),     596     (7429859039, 35, 4),
449     (6679470818, 14, 4),    523     (7419952825, 2, 4),     597     (7429859039, 36, 3),
450     (6679470818, 15, 3),    524     (7419952825, 3, 4),     598     (7429859039, 37, 5),
451     (6679470818, 16, 5),    525     (7419952825, 4, 5),     599     (7429859039, 38, 2),
452     (6679470818, 17, 2),    526     (7419952825, 5, 2),     600     (7429859039, 39, 2),
453     (6679470818, 18, 3),    527     (7419952825, 6, 3),     601     (7429859039, 40, 5),
454     (6679470818, 19, 5),    528     (7419952825, 7, 2),     602     (7764690558, 1, 4),
455     (6679470818, 20, 2),    529     (7419952825, 8, 3),     603     (7764690558, 2, 2),
456     (6679470818, 21, 2),    530     (7419952825, 9, 3),     604     (7764690558, 3, 4),
457     (6679470818, 22, 5),    531     (7419952825, 10, 4),    605     (7764690558, 4, 2),
458     (6679470818, 23, 4),    532     (7419952825, 11, 3),    606     (7764690558, 5, 4),
459     (6679470818, 24, 2),    533     (7419952825, 12, 4),    607     (7764690558, 6, 2),
460     (6679470818, 25, 2),    534     (7419952825, 13, 3),    608     (7764690558, 7, 3),
461     (6679470818, 26, 3),    535     (7419952825, 14, 5),    609     (7764690558, 8, 4),
462     (6679470818, 27, 4),    536     (7419952825, 15, 3),    610     (7764690558, 9, 2),
463     (6679470818, 28, 5),    537     (7419952825, 16, 5),    611     (7764690558, 10, 5),
464     (6679470818, 29, 4),    538     (7419952825, 17, 4),    612     (7764690558, 11, 2),
465     (6679470818, 30, 2),    539     (7419952825, 18, 4),    613     (7764690558, 12, 4),
466     (6679470818, 31, 3),    540     (7419952825, 19, 3),    614     (7764690558, 13, 3),
467     (6679470818, 32, 3),    541     (7419952825, 20, 3),    615     (7764690558, 14, 4),
468     (6679470818, 33, 2),    542     (7419952825, 21, 4),    616     (7764690558, 15, 2),
469     (6679470818, 34, 5),    543     (7419952825, 22, 5),    617     (7764690558, 16, 2),
470     (6679470818, 35, 3),    544     (7419952825, 23, 3),    618     (7764690558, 17, 5),
471     (6679470818, 36, 3),    545     (7419952825, 24, 4),    619     (7764690558, 18, 3),
472     (6679470818, 37, 4),    546     (7419952825, 25, 3),    620     (7764690558, 19, 4),
473     (6679470818, 38, 2),    547     (7419952825, 26, 3),    621     (7764690558, 20, 3),
474     (6679470818, 39, 2),    548     (7419952825, 27, 2),    622     (7764690558, 21, 3),
475     (6679470818, 40, 2),    549     (7419952825, 28, 2),    623     (7764690558, 22, 4),
476     (6679470818, 41, 5),    550     (7419952825, 29, 3),    624     (7764690558, 23, 4),
477     (6679470818, 42, 4),    551     (7419952825, 30, 5),    625     (7764690558, 24, 3),
478     (6679470818, 43, 2),    552     (7419952825, 31, 5),    626     (7764690558, 25, 4),
479     (6679470818, 44, 5),    553     (7419952825, 32, 2),    627     (7764690558, 26, 3),
480     (6679470818, 45, 5),    554     (7419952825, 33, 3),    628     (7764690558, 27, 5),
481     (6679470818, 46, 2),    555     (7419952825, 34, 4),    629     (7764690558, 28, 2),
482     (6679470818, 47, 5),    556     (7419952825, 35, 2),    630     (7764690558, 29, 4),
483     (6679470818, 48, 4),    557     (7419952825, 36, 3),    631     (7764690558, 30, 5),
484     (7240700128, 1, 5),     558     (7419952825, 37, 4),    632     (7764690558, 31, 5),
485     (7240700128, 2, 4),     559     (7419952825, 38, 5),    633     (7764690558, 32, 5),
486     (7240700128, 3, 2),     560     (7419952825, 39, 4),    634     (7764690558, 33, 3),
487     (7240700128, 4, 2),     561     (7419952825, 40, 3),    635     (7764690558, 34, 4),
488     (7240700128, 5, 2),     562     (7429859039, 1, 3),     636     (7764690558, 35, 4),
489     (7240700128, 6, 5),     563     (7429859039, 2, 5),     637     (7764690558, 36, 4),
490     (7240700128, 7, 4),     564     (7429859039, 3, 4),     638     (7764690558, 37, 2),
491     (7240700128, 14, 4),    565     (7429859039, 4, 3),     639     (7764690558, 38, 2),
492     (7240700128, 15, 5),    566     (7429859039, 5, 4),     640     (7764690558, 39, 2),
493     (7240700128, 17, 3),    567     (7429859039, 6, 3),     641     (7764690558, 40, 2),
494     (7240700128, 18, 2),    568     (7429859039, 7, 5),     642     (7764690558, 41, 2),
495     (7240700128, 28, 3),    569     (7429859039, 8, 4),     643     (7764690558, 42, 4),
496     (7240700128, 29, 4),    570     (7429859039, 9, 4),     644     (7764690558, 43, 3),
497     (7240700128, 32, 2),    571     (7429859039, 10, 4),    645     (7764690558, 44, 3),
498     (7240700128, 33, 3),    572     (7429859039, 11, 4),    646     (7764690558, 45, 4),
499     (7240700128, 34, 3),    573     (7429859039, 12, 4),    647     (7764690558, 46, 2),
500     (7240700128, 35, 2),    574     (7429859039, 13, 3),    648     (7764690558, 47, 5),
501     (7240700128, 36, 5),    575     (7429859039, 14, 4),    649     (7764690558, 48, 5),
502     (7240700128, 37, 3),    576     (7429859039, 15, 4),    650     (7816359400, 1, 2),
503     (7242099995, 1, 4),     577     (7429859039, 16, 4),    651     (7816359400, 2, 3),
504     (7242099995, 2, 3),     578     (7429859039, 17, 4),    652     (7816359400, 3, 5),
505     (7242099995, 3, 4),     579     (7429859039, 18, 5),    653     (7816359400, 4, 3),
506     (7242099995, 4, 5),     580     (7429859039, 19, 4),    654     (7816359400, 5, 2),
507     (7242099995, 5, 3),     581     (7429859039, 20, 4),    655     (7816359400, 6, 3),
508     (7242099995, 6, 2),     582     (7429859039, 21, 4),    656     (7816359400, 7, 3),
509     (7242099995, 7, 5),     583     (7429859039, 22, 5),    657     (7816359400, 8, 3),
510     (7242099995, 14, 4),    584     (7429859039, 23, 4),    658     (7816359400, 9, 4),
511     (7242099995, 15, 5),    585     (7429859039, 24, 5),    659     (7816359400, 10, 3),
```

```
660     (7816359400, 11, 2),    675     (7816359400, 26, 5),    689     (7816359400, 40, 5),
661     (7816359400, 12, 4),    676     (7816359400, 27, 5),    690     (7816359400, 41, 3),
662     (7816359400, 13, 3),    677     (7816359400, 28, 3),    691     (7816359400, 42, 3),
663     (7816359400, 14, 2),    678     (7816359400, 29, 2),    692     (7816359400, 43, 3),
664     (7816359400, 15, 5),    679     (7816359400, 30, 5),    693     (7816359400, 44, 5),
665     (7816359400, 16, 3),    680     (7816359400, 31, 2),    694     (7816359400, 45, 2),
666     (7816359400, 17, 2),    681     (7816359400, 32, 3),    695     (7816359400, 46, 5),
667     (7816359400, 18, 4),    682     (7816359400, 33, 4),    696     (7816359400, 47, 3),
668     (7816359400, 19, 4),    683     (7816359400, 34, 2),    697     (7816359400, 48, 2),
669     (7816359400, 20, 2),    684     (7816359400, 35, 2),    698     (7816359400, 49, 2),
670     (7816359400, 21, 3),    685     (7816359400, 36, 3),    699     (7816359400, 50, 5),
671     (7816359400, 22, 4),    686     (7816359400, 37, 3),    700     (7816359400, 51, 2),
672     (7816359400, 23, 3),    687     (7816359400, 38, 4),    701     (7816359400, 52, 2);
673     (7816359400, 24, 3),    688     (7816359400, 39, 4),    702
674     (7816359400, 25, 2),    688     (7816359400, 39, 4),


703     -- --------------------------------------------------------
704
705     --
706     -- Table structure for table `spol`
707     --
708
709     CREATE TABLE IF NOT EXISTS `spol` (
710       `spol_id` int(1) DEFAULT NULL,
711       `spol_naziv` varchar(6) DEFAULT NULL
712     ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
713
714     --
715     -- Dumping data for table `spol`
716     --
717
718     INSERT INTO `spol` (`spol_id`, `spol_naziv`) VALUES
719     (1, 'musko'),
720     (2, 'zensko');
721
722     -- --------------------------------------------------------
723
724     --
725     -- Table structure for table `studenti`
726     --
727
728     CREATE TABLE IF NOT EXISTS `studenti` (
729       `student_id` bigint(10) unsigned NOT NULL,
730       `student_ime` varchar(25) NOT NULL,
731       `student_prezime` varchar(25) NOT NULL,
732       `student_spol` int(1) unsigned NOT NULL,
733       `student_godina_studija` int(1) NOT NULL
734     ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
735
736     --
737     -- Dumping data for table `studenti`
738     --
739
740     INSERT INTO `studenti` (`student_id`, `student_ime`, `student_prezime`,
`student_spol`, `student_godina_studija`) VALUES
```

```
741     (1394984532, 'Mario', 'Kos', 1, 2),
742     (1473419155, 'Alen', 'Katic', 1, 4),
743     (1876743912, 'Darija', 'Popovic', 2, 2),
744     (2131875260, 'Vlatka', 'Marusic', 2, 2),
745     (2517879602, 'Jelica', 'Miletic', 2, 3),
746     (2801676037, 'Milovan', 'Ceh', 1, 1),
747     (2936048095, 'Vladimir', 'Peric', 1, 2),
748     (3477826494, 'Aleksandra', 'Novosel', 2, 1),
749     (3736706852, 'Zeljka', 'Stanic', 2, 4),
750     (4445574049, 'Mladen', 'Martinovic', 1, 4),
751     (4543078538, 'Ana', 'Novosel', 2, 1),
752     (4854107160, 'Davorin', 'Cindric', 1, 4),
753     (5228861981, 'Ana', 'Juric', 2, 1),
754     (5321388965, 'Karolina', 'Jerkovic', 2, 2),
755     (5379417581, 'Tereza', 'Crnic', 2, 2),
756     (5555260556, 'Gabriela', 'Vidakovic', 2, 1),
757     (6077658506, 'Vjenceslav', 'Knezevic', 1, 3),
758     (6679470818, 'Ana', 'Loncar', 2, 5),
759     (7240700128, 'Zvjezdana', 'Jerkovic', 2, 2),
760     (7242099995, 'Oliver', 'Baric', 1, 2),
761     (7419952825, 'Sabina', 'Novosel', 2, 4),
762     (7429859039, 'Elena', 'Ivankovic', 2, 4),
763     (7764690558, 'Vidoslav', 'Milic', 1, 5),
764     (7816359400, 'Dragica', 'Stanic', 2, 2),
765     (7895435912, 'Jadranko', 'Stolar', 1, 6),
766     (8060373315, 'Bojana', 'Baric', 2, 4),
767     (8270380323, 'Gojislav', 'Juric', 1, 1),
768     (8346758067, 'Grubisa', 'Baric', 1, 1),
769     (8488896760, 'Bogomil', 'Simunovic', 1, 4),
770     (9297783261, 'Marko', 'Dujmovic', 1, 3);
771
772     --
773     -- Indexes for dumped tables
774     --
775
776     --
777     -- Indexes for table `kolegiji`
778     --
779     ALTER TABLE `kolegiji`
780      ADD PRIMARY KEY (`kolegij_id`);
781
782     --
783     -- Indexes for table `polozeni_kolegiji`
784     --
785     ALTER TABLE `polozeni_kolegiji`
786      ADD PRIMARY KEY (`student_id`,`kolegij_id`);
787
788     --
789     -- Indexes for table `studenti`
790     --
791     ALTER TABLE `studenti`
792      ADD PRIMARY KEY (`student_id`);
793
794     --
795     -- AUTO_INCREMENT for dumped tables
796     --
797
798     --
799     -- AUTO_INCREMENT for table `kolegiji`
800     --
801     ALTER TABLE `kolegiji`
802     MODIFY `kolegij_id` int(5) NOT NULL AUTO_INCREMENT,AUTO_INCREMENT=53;
803     /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
804     /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
805     /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
806
```

13. Sažetak

Cilj ovog rada je pokazati potrebu za prihvaćanjem aspektno-orijentiranog pristupa razvoju softvera kao novu, naprednu metodologiju programiranja koja pojednostavljuje razvoj i održavanje poslovnih informacijskih sustava.

Prilikom korištenja konvencionalnih tehnika razvoja softvera postoje problemi koji nisu u potpunosti rješivi: zaplitanje kôda koje se pojavljuje kada jedan modul sadrži kôd za obavljanje više dužnosti istovremeno, te raspršivanje kôda koje se pojavljuje kada više modula sadrži dijelove za implementaciju jedne funkcionalnosti. Te probleme rješava AOP pristup uvođenjem apstrakcije aspekta.

Kako bi se teorija prevela u praksu, potrebno ju je implementirati, pa se tako implementacija AOP-a sastoji od specifikacije jezika (opisuje sintaksu) i implementacije jezika (provjerava pridržavanje sintakse i prevađa kôd u izvršivi oblik). Implementaciju obavlja novi entitet nazvan aspektni tkalac – korištenjem pravila tkanja on ubacuje aspektni kôd u osnovni kôd i tako stvara završni program.

Aspekt je ključna jedinica (modul) koja obavlja zadaću isprepletene dužnosti, a sadrži specifikaciju točke presjeka, uvode, savjete i metode. Aspekt mijenja odvijanje programa tako da na potrebnim mjestima koji su definirani u točkama presjeka ubaci izmijenjeni kôd koji se naziva savjet.

Za stvaranje kvalitetnog aspektno-orijentiranog sustava potrebno je i inženjering zahtjeva provesti na takav, AOP-u prilagođen način – potrebno je na sustav gledati kao na osnovni sustav sa proširenjima. Nakon toga potrebno je napisati osnovni kôd, te takav osnovni dio nadograditi pisanjem aspektnog dijela programa. U stvaranju aspekta potrebno je ispravno i detaljno definirati točke presjeka kako ne bi došlo do slučajnog miješanja aspekata koji imaju istu definiciju točke presjeka.

U radu je takav aspektno-orijentiran pristup odrađen na primjeru „*Hello World!*“ te na primjeru korištenja „*around*“ savjeta koji omogućuje kontrolu toka izvršavanja programa.

14. Summary

This thesis aims to show the need to accept the aspect-oriented approach to software development as a new, advanced programming methodology which simplifies the development and maintenance of business information systems.

There are some difficulties in using the conventional techniques of software development which are not completely solvable: code tangling which happens when one module contains the code for implementation of several functionalities at once, and code scattering which happens when multiple modules contain pieces of code to implement a single concern. Those problems are solved with AOP approach with introduction of aspects.

To make the theory useful, it needs to be implemented. An AOP implementation consists of two parts: language specification (describes the syntax) and the language implementation (verifies the compliance with the syntax and translates the code in an executable form). The implementation is done by a new entity called the aspect weaver – using the weaving rules, it injects the aspect code into the basic code creating the final program.

The aspect is the key unit (module) which performs the task of crosscutting concern, and it contains the pointcut specification, introductions, advices and methods. The aspect makes the actual changes to the program execution in such way that on the needed places, which are defined in the pointcuts, injects the changed code which is called the advice.

To create a high quality aspect-oriented system, an adapted-to-AOP-approach requirements engineering is needed – a system must be looked at like a base system with extensions. Afterwards, the basic code is to be written, and later, that basic part is upgraded writing the aspect code. While creating the aspect, it is needed to correctly specify the pointcuts to assure that the aspects don't interfere with each other because of the accidentally similarly defined pointcuts.

In the thesis, the aspect-oriented approach is explained on a "Hello World!" example and on the second example which uses the "around" advice to control the execution of the program.