Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/05BE51Andjelovski

OBJEKTNA PARADIGMA RUBY PROGRAMSKOG JEZIKA SA KOMPARATIVNOM ANALIZOM RUBY ON RAILS I JAVA SPRING OKRUŽENJA

THE OBJECTIVE PARADIGM OF THE RUBY PROGRAMMING LANGUAGE WITH COMPARATIVE ANALYSIS OF RUBY ON RAILS AND JAVA SPRING ENVIRONMENT

Olja Anđelovski, Srđan Popov, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je analizirana objektno-orijentisana paradigma Ruby programskog jezika uz komparativnu analizu Ruby on Rails i Java Spring okruženja za kreiranje web aplikacija. Implementirana aplikacija Travel je standardna web aplikacija za upravljanje šifrarnicima, pregled, kreiranje i rezervisanje smeštaja.*

Ključne reči: *Ruby, Objektno-orijentisana paradigma, Ruby on Rails, Java, Spring*

Abstract – *The paper analyzes the object-oriented paradigm of the Ruby programming language with a comparative analysis of the Ruby on Rails and Java Spring environment for creating web applications. Implemented Travel application is a standard web application for managing codebooks, viewing, creating and booking accommodations.*

Keywords: *Ruby, Object-oriented paradigm, Ruby on Rails, Java, Spring*

1. UVOD

Objektno-orijentisana paradigma se bavi modelovanjem entiteta koji se zovu objekti. Omogućava upotrebu funkcionalnosti bez poznavanja specifičnosti ponašanja ili podataka unutar objekata, dozvoljava definisanje apstraktnih tipova gde je fukcionalnost samo delimično definisana kao i mnoge druge pogodnosti.

Cilj rada je upoznavanje koncepata objektno-orijentisane paradigme i implementacija istih u okviru programskog jezika Ruby. Na studiji slučaja je izvršena komparativna analiza Ruby on Rails, kao jednog od najpopularnijih okruženja za razvoj klijent-server baziranih aplikacija i Java Spring okruženja.

2. POJMOVI I KONCEPTI OBJEKTNO-ORIJENTISANOG PROGRAMIRANJA U PROGRAMSKOM JEZIKU RUBY

2.1. Klase i objekti

Osnovna uloga klase je kreiranje objekata, inicijalizacija početnih vrednosti atributa, kao i implementacija ponašanja u okviru metoda.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Srđan Popov, vanredni profesor.

Klase se definišu uz pomoć ključne reči ***class*** iza koje ide naziv klase koji uvek započinje velikim slovom, nakon toga se pišu atributi i metode i obavezno se završava sa ***end***.

Objekat predstavlja komponentu koja sadrži atribute i metode i uz pomoć metoda može da izvršava određene zadatke i utiče na ostale objekte. Kreira se pozivanjem metode ***new***, a potom se u pozadini poziva metoda ***initialize*** kojom se inicijalizuju vrednosti atributa objekata.

2.1.1. Klasne metode

Klasne metode se pozivaju nad samim klasama. Kreiraju se dodavanjem prefiksa ***self.*** ispred naziva metode.

2.1.2. Pristupne metode

Pristupnim metodama pristupamo atributima unutar klase. Pristupne metode u Ruby programskom jeziku su: ***attr***, ***attr_reader***, ***attr_writer*** i ***attr_accessor***.

2.2. Konstruktor

Konstruktor je metoda kojom se kreira objekat i vrši njegova inicijalizacija. Konstruktor se u Ruby programskom jeziku kreira definisanjem metode ***initialize***, koja ima ***private*** pravo pristupa. Ukoliko postoji konstruktor, pozivom klasne metode ***new*** pozvaće se metoda *initialize* i kreiraće se inicijalizovani objekat, u suprotnom biće pozvan podrazumevani konstruktor i kreiraće se neinicijalizovan objekat.

2.3. Destruktor

U Ruby programskom jeziku brisanje objekata i oslobađanje memorije vrši sakupljač smeća. Programer ne upravlja memorijom, već Ruby inicira dodeljivanje memorije. Prilikom kreiranja objekta Ruby zahteva od kernela memoriju. Ukoliko nema slobodnih slotova sakupljač smeća proverava objekte koji više nisu referencirani od strane drugih objekata i oslobađa ih.

2.4. Moduli

Moduli predstavljaju skup konstanti, klasnih varijabli i metoda instanci. Modul se definiše uz pomoć ključne reči ***module*** iza koje ide ime modula sa velikim početnim slovom, nakon toga ide struktura i završava se sa ***end***. Najvažnije uloge modula su: *kreiranje prostora imena i implementiranje mixin mehanizma*.

2.4.1. Prostori imena

Moduli nam daju mogućnost da definišemo prostor imena, izolovani skup metoda i konstanti gde ne mora da se vodi računa o načinu na koji će biti uključene u druge module ili klase.

2.4.2. Mixin mehanizam

Uključivanje modula u definiciju klase predstavlja ***mixin*** mehanizam. Uz pomoć njega je moguće implementirati višestruko nasleđivanje. Moduli se u klase uključuju korišćenjem ključne reči ***include*** ili ***extend***.

2.5. Nasleđivanje

Nasleđivanje je veza između klasa koja podrazumeva preuzimanje sadržaja nekih klasa, odnosno klasa-predaka i na taj način, uz mogućnost modifikacije preuzetog sadržaja i dodavanjem novog dobija se klasa-potomak [1]. U Ruby programskom jeziku je moguće naslediti isključivo jednu klasu dodavanjem operatora < nakon imena potklase, a pre imena natkalase koju želimo da naledimo. Višestruko nasleđivanje se implementira uz pomoć *mixin* mehanizama na taj način što se moduli uključuju u definiciju drugog modula ili klase.

2.6. Polimorfizam

Polimorfizam je koncept programiranja koji se odnosi na sposobnost varijabli, funkcija ili objekata da prihvate više oblika. Polimorfizam pruža jedan interfejs entitetima različitog tipa. U Ruby-u se ostvaruje upotrebom: *dinamičkog tipiziranja, nasleđivanja i modula.*

2.7. Enkapsulacija

Enkapsulacija je pojam koji se odnosi na sakrivanje informacija. Koristi se kako bi se sakrila unutrašnja reprezentacija objekta i njegovih stanja od strane spoljašnosti. Podaci i funkcije za izvršavanje akcija nad podacima se grupišu u jedinstvenu jedinicu – klasu.

Osnovna svrha enkapsulacije je da obezbedi sigurnost podataka u klasi. Da bi podaci unutar klase bili sigurni i zaštićeni od spoljašnjeg uticaja, neophodno je da se atributi definišu sa *private* pravom pristupa koji se kombinuju sa *public* metodama koje omogućavaju pristup atributima i izvršavanje zadataka. Još jedan način za enkapsulaciju podataka u Ruby-u jeste mehanizam za ugnježdavanje klasa i modula. Prednosti korišćenja enkapsulacije su: *sakrivanje informacija, ponovna upotrebljivost koda i jednostavno testiranje.*

2.8. Modularnost

Modularnost se odnosi na koncept kreiranja više različitih, relativno nezavisnih modula koji se povezuju i kombinuju u cilju formiranja kompletnog sistema. Modul predstavlja zasebnu softversku komponentu, koja ima određenu funkciju. Modularnost se u Ruby programskom jeziku postiže na više načina.

Jedna od mogućnosti je definisanje klasa koje imaju privatne atribute i kojima je ograničeno pravo pristupa, klasa ne sadrži spoljašnje članove i predstavlja izolovanu celinu. Takođe modularnost se može postići korišćenjem modula, kao i definisanjem klasa koje sadrže ugnježdene klase, odnosno module.

Prednosti modularnog programiranja su: *skraćivanje vremena pisanja koda, ponovna upotrebljivost, olakšano dizajniranje programa, olakšana čitljivost programa, olakšano otklanjanje grešaka.*

2.9. Konverzija

Da bismo omogućili rad sa podacima koji dolaze iz spoljnih izvora kao što su tastatura, baza podataka, odgovor api-a neophodno je izvršiti konverziju. U Ruby programskom jeziku je moguće izvršiti *implicitnu i eksplicitnu* konverziju.

Eksplicitna konverzija se vrši na izričit način, pozivanjem metoda za konverziju nad podacima koje želimo da konvertujemo u ciljani tip. Neke od metoda su: ***to_s***, ***to_i***, ***to_a***, itd.

Implicitna konverzija se vrši prilikom izračunavanja izraza koji sadrže operande različitog tipa i Ruby je vrši automatski. Konverzija se vrši na taj način što se manje složeni operand konvertuje u složeniji. Metode koje treba implementirati da bi se obavila implicitna konverzija u slučaju da nije podržana su: ***to_str***, ***to_int***, ***to_ary***, itd.

2.10. Preklapanje operatora

Preklapanje operatora predstavlja pridruživanje istog simbola različitim rutinama. Dva ili više operatora su preklopljena ukoliko za isti simbol imaju pridružen različit kod [2].

U programskom jeziku Ruby preklapanje operatora se vrši na taj način što se u korisnički definisanim tipovima definišu operatori sa posebnom logikom koja je prilagođena prosleđenim argumentima, odnosno njihovom tipu. Operatori u Ruby programskom jeziku predstavljaju poziv odgovarajućih metoda u pozadini.

2.11. Dinamičko povezivanje

Povezivanje predstavlja vezu između poziva metode i koda koji se izvršava. Postoje dva tipa povezivanja: *statičko i dinamičko*.

Statičko povezivanje kompajler može da reši u toku kompajliranja, dok se kod dinamičkog povezivanja na osnovu tipa objekta nad kojim se poziva metoda odlučuje koja će metoda izvršiti. Ruby podržava samo dinamičko povezivanje.

3. STUDIJA SLUČAJA

Za studiju slučaja je odabrana implementacija web aplikacije *Travel* koja je napisana u programskom jeziku Ruby u okviru Ruby on Rails okruženja, koristeći objektno-orijentisanu paradigmu. Izvršeno je poređenje sa aplikacijom napisanom u Javi koristeći Spring okruženje.

3.1. Specifikacija projekta

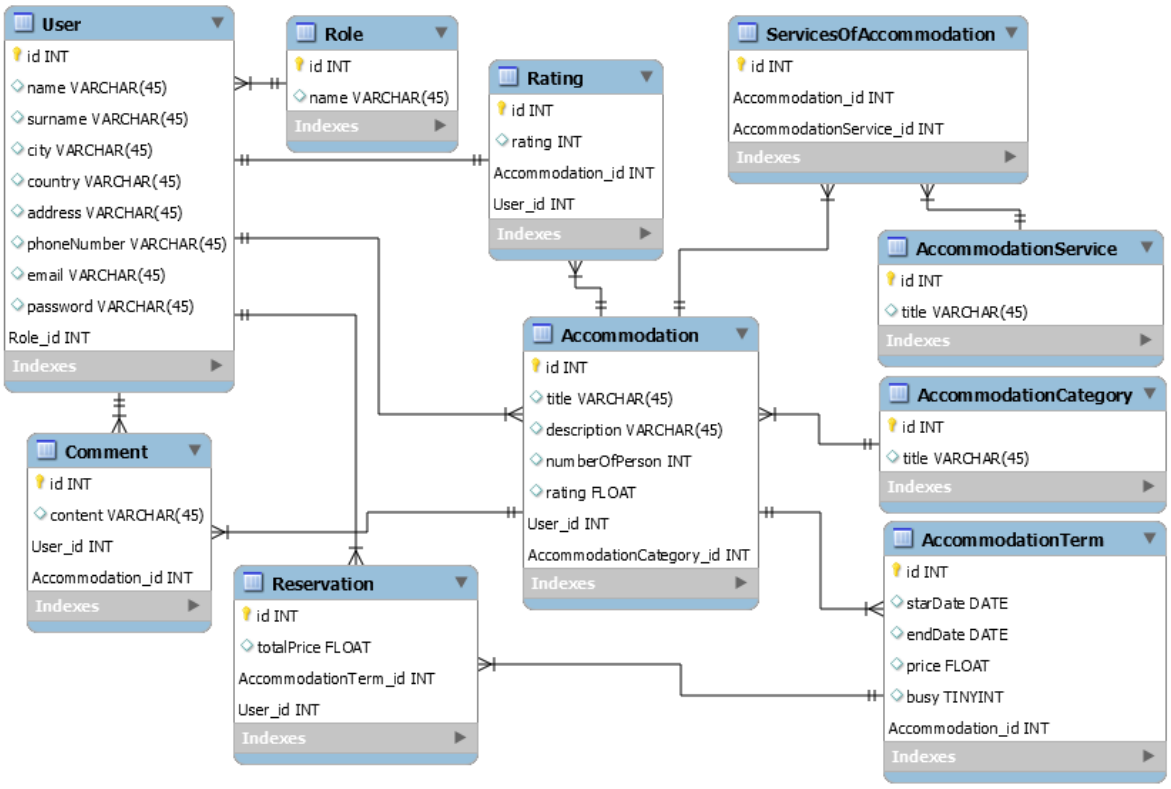
Aplikacija Travel je web aplikacija čija je osnovna funkcionalnost pregled i rezervisanje smeštaja, kao i obavljanje osnovnih crud operacija nad definisanim entitetima.

Za izvršavanje glavnih funkcionalnosti u okviru aplikacije su identifikovani sledeći korisnici: *administrator, agent i kupac*.

Administrator predstavlja predefinisanog korisnika koji ima pristup bazi podataka i čiji je glavni zadatak dinamičko upravljanje entitetima u vidu brisanja, dodavanja i izmene kategorija smeštaja, servisa koje pruža smeštaj, kao i upravljanje registrovanim korisnicima.

Agent je zadužen za kreiranje smeštaja, kao i dodavanje termina u okviru smeštaja. Ima mogućnost brisanja i izmene smeštaja, kao i pregled rezervisanih smeštaja.

Kupac predstavlja krajnjeg korisnika, koji može da pregleda i rezerviše termine u okviru smeštaja, nakon rezervisanja ima mogućnost ocenjivanja i ostavljanja komentara za smeštaj u kome je boravio. Na slici 1. je prikazan model podataka.



Slika 1. *Model podataka*

3.2. Ruby on Rails

Ruby on Rails je jedan od najpopularnijih okruženja za razvoj klijent-server baziranih web aplikacija napisan u Ruby programskom jeziku. Rails koristi *MVC (Model-View-Controller)* arhitekturu čija je osnovna ideja ponovna upotreba postojaćeg koda, olakšavanje razvoja koda, kao i razdvajanje koda na odvojene komponente. Pored MVC-a Rails podržava i ostale poznate šablone i paradigme, sa akcentom na principe *DRY (Don't Repeat Yourself)* i *COC (Convention Over Configuration)*. Korišćenjem konvencija prilikom kreiranja aplikacija, uvođenjem migracija i mnogih drugih koncepata Rails je imao veliki uticaj na razvoj web aplikacija.

3.3. MySQL

Za konfigurisanje MySQL baze podataka bilo je potrebno izmeniti konfiguracije u okviru *database.yml* datoteke koja je smeštena unutar *config* direktorijuma i instalirati *mysql2 adapter*.

U poređenju sa aplikacijom napisanom u Java programskom jeziku, koristeći Spring okruženje, u okviru koje se koristi H2 baza podataka možemo da primetimo da je jedina razlika u konfiguraciji u tome što prilikom dodavanja novih zavisnosti Spring sam vrši instalaciju.

3.4. Model

Za kreiranje prethodno definisanog modela u okviru Ruby on Rails platforme je korišćen *ActiveRecord* koji predstavlja model u MVC arhitekturi i koji je zadužen za predstavljanje poslovnih podataka i logike. Veze između entiteta se definišu asocijacijama.

Kreiranje modela i mapiranje u Spring-u, za razliku od Rails-a predstavlja teži posao i zahteva dobro poznavanje anotacija i njihovih mogućnosti. U Spring-u je neophodno ručno kreirati svaku od klasa, definisati atribute, konstruktore i metode koje dodatno treba anotirati kako bi se uspešno izvršila konfiguracija. Model u Rails-u je dosta pregledniji, veliki deo koda se automatski generiše i ukoliko se ispoštuju konvencije prilikom imenovanja Rails oslobađa programera od dodatnog posla.

3.5. Controller

Za implementaciju sloja poslovne logike u Ruby on Rails-u je korišćen *ActionController* koji predstavlja kontroler u MVC arhitekturi. Kontroler je zadužen za proizvodnju odgovarajućeg izlaza nakon što ruter utvrdi koji od kontrolera treba da obradi zahtev.

Implementacija poslovne logike u Spring-u zahteva kreiranje klasa koje vrše uloge kontrolera i servisa.

Anotiranjem smo morali da izvršimo konfiguraciju ponaosob za svaki od kontrolera i servisa. Kontroler kao i u Rails-u sadrži metode koje moraju da se anotiraju vrstom zahteva koji se mapira na datu metodu. Za razliku od Rails-a koji unutar metoda kontrolera ima implementiranu funkcionalnost, Spring odvaja zahtev od implementacije korišćenjem servisa u okviru kojih se implementiraju metode koje komuniciraju sa bazom podataka i vraćaju rezultat kontroleru. Implementacija koda u Ruby on Rails-u je brža od implementacije u Spring-u, potencijalne greške se na samom početku uklanjaju ukoliko se ispoštuje konvencija imenovanja asocijacija, kontrolera i modela.

3.6. View

Rails sam generiše odgovarajući direktorijum koji treba da sadrži stranice koje će biti prikazane korisniku. Interfejs koji je kreiran kako bi korisnik mogao da manipuliše web aplikacijom je napisan koristeći jQuery u okviru html stranica. Za kreiranje interfejsa je korišćeno i Bootstrap okruženje.

U aplikaciji napisanoj u Spring-u smo za korisnički interfejs koristili AngularJS. Mapiranje rezultata dobijenih od

strane kontrolera je jednostavnije u Rails-u. Metode ne sadrže povratne vrednosti već se vrši direktno mapiranje podataka sa stranice koja poziva kontroler na kontroler i obrnuto.

3.7. Autentifikacija

Autentifikacija i autorizacija predstavljaju ključne pojmove koje je neophodno implementirati tako da se postigne što veća bezbednost prilikom korišćenja web aplikacija koje sadrže osetljive podatke. U aplikaciji Travel u Rails-u je odabran *Devise* kao jedan od brzih i fleksibilnih rešenja za autentifikaciju. Devise nudi mehanizme heširanja lozinke, sadrži već kreirane forme za registraciju i logovanje, čuva i briše korisničke sesije itd. Pored sloja na serveru, da bi omogućili autentifikaciju potrebno je definisati odgovarajuće forme na klijentskoj strani. Za razliku od Devise-a koji u pozadini vodi računa o heširanju i čuvanju lozinke, u Spring-u sve stavke moraju da se implementiraju ručno u kodu uz pomoć *Spring Security* modula.

4. ZAKLJUČAK

U radu je pokazano da se u Ruby programskom jeziku mogu u potpunosti implementirati koncepti objektno-orijentisane paradigme. Ruby nudi brojne mogućnosti u vidu metoda koje olakšavaju i ubrzavaju pisanje koda. Mnogi koraci u implementaciji koncepata se već nalaze kao podrazumevani u okviru programskog jezika.

Ruby on Rails okruženje omogućava kreiranje web aplikacija u Ruby programskom jeziku. Rails okruženje prati trendove u razvoju web aplikacija i kao takav zbog brzog razvoja aplikacija, konvencija koje pruža okruženje uživa popularnost. Posmatrano sa aspekta programera koji se prvi put susteće sa konceptima objektno-orijentisanog programiranja i web programiranja, Spring ima prednost zbog sličnosti sa standardnim Java aplikacijama. Sa aspekta programera koji ima iskustva u pisanju web aplikacija, Ruby on Rails je okruženje koje predstavlja moćan alat za brz razvoj aplikacija i uz puno fleksibilnosti predstavlja pravi izbor.

5. LITERATURA

[1] Malbaški Dušan, Kupusinac Aleksandar, „Praktikum za vežbe iz objektno orijentisanog programiranja”, Novi Sad, Fakultet tehničkih nauka u Novom Sadu, 2010.

[2] Malbaški Dušan, „Objektno orijentisano programiranje programski jezik java”, Novi Sad – Sremska Kamenica, Univerzitet Educons – Fakultet informacionih tehnologija, 2015.

Kratka biografija:

Olja Anđelovski, rođena je 03.11.1995. u Beogradu. Završila je Osnovnu školu „Sava Žebeljan” u Crepaji. Nakon završene Osnovne škole upisuje srednju školu, Gimnaziju „Mihajlo Pupin” u Kovačici. 2014. godine upisuje „Fakultet tehničkih nauka” u Novom Sadu, smer „Računarstvo i automatika”. 2016/17. godine upisuje smer „Primenjene računarske nauke i informatika”. 2017/18. godine upisuje smer „Softversko inženjerstvo”. 2018. godine dobija zvanje Diplomirani inženjer elektrotehnike i računarstva. 2018/19. godine upisuje master akademske studije, smer „Primenjene računarske nauke i informatika – Elektronsko poslovanje”. 09.11.2018. godine dobija zvanje Saradnika u nastavi na Departmanu za računarstvo i automatiku na Fakultetu tehničkih nauka u Novom Sadu.
kontakt: andjelovskiolja@gmail.com