**UNIVERZITET U BEOGRADU**

**FAKULTET ORGANIZACIONIH NAUKA**

**ZAVRŠNI RAD**

**naslov…..**

**Mentor Student**

**Dr… Ilija Košanin 205/2015**

**Beograd, 2022. godine**

**Apstrakt**

Apstrakt opisuje ceo rad u desetak rečenica (oko 200 reči). Struktura apstrakta obuhvata 1-3 uvodne rečenice koje opisuju temu rada, odnosno oblast na koju se rad odnosi. Nakon toga sledi jedan rečenica koja opisuje cilј rada. Zatim, jedna do tri rečenice o metodologiji koja je primenjena u radu. Slede jedan do tri rečenice o ključnim rezultatima koji su predstavljeni u radu. Na kraju, jedna do dve rečenice o zaključku rada.

**Sadržaj**

Contents

**No table of contents entries found.**

# Uvod

Pored uvoda u oblast kojoj pripada tema završnog rada, na kraju uvoda, u poslednjem pasusu, potrebno je opisno predstaviti sadržaj rada po poglavlјima. Nakon uvoda slede ostala poglavlјa rada.

# Pregled korišćenih tehnologija

## Јава програмски језик

Java je objektno-orijentisani, višeplatformski jezik opšte namene. Razvoj ovog programskoj jezika započet je 1991. godine u kompaniji Sun Microsystems. Razvoj je vodio Džejm Gosling zajedno sa Majkom Šeridanom i Patrikom Notonom. Danas, Java je jedan od najpopularnijih programskih jezika i koristi se u pokretanju velikog broja aplikacija.

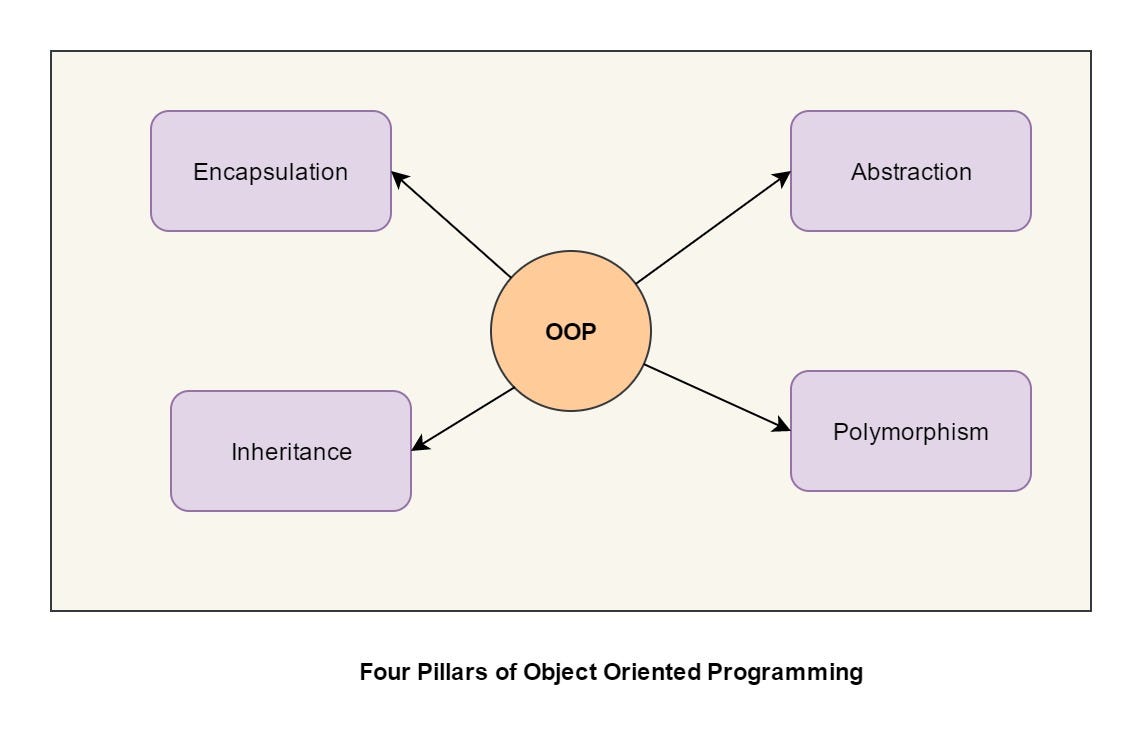
## Objektno orijentisan jezik

Ovaj princip se zasniva na kombinaciji različitih tipova podataka, nad kojima se izvršavaju različite operacije. Ovo praktično znači da su kod i podaci spojeni u jedan koncept koji se naziva objekat.

* Prvenstveno ideja je odvajanje dela koda koji se menja od dela koji uvek ostaje isti. Ovo omogućava pouzdaniju osnovu za projektovanje softverskog sistema. Takođe, cilj je promovisanje upravljanja velikim softverskim projektima, povećanje kvaliteta i smanjenje broja neuspešnih projekata.
* Drugi važan deo je povećanje moguće ponovne upotrebe koncepta iz jednog projekta u drugom. Na primer, korisnik iz jednog softverskog sistema ne treba da se razlikuje od korisnika u drugom projektu.

Četiri osnovna stuba objektno orijentisanog programiranja su:

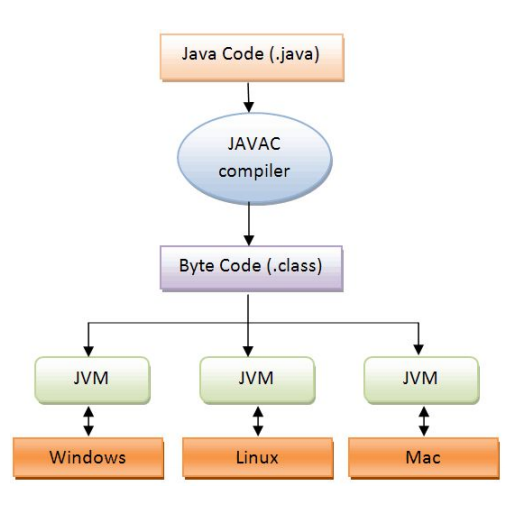
* Enkapsulacija – skrivanje implementacije podataka pomoću privatnih I javnih metoda.
* Apstrakcija – implementira se korišćenjem apstraktnih klasa i interfejsa I podrazumeva ideju da znamo šta će neka klasa uradi bez da znamo kako će to da uradi.
* Nasledjivanje – Koristi se kada jedan objekat ima vezu sa drugim. Na primer, prica i patka. Patka će da bude objekat koji će da nasledi objekat ptica i koristiće osnovne funkcionalnsti iz tog objekta. Ono što je bitno jeste da sam objekat patka moze da se dopuni novim funcionalnostima koje neće da poremete objekat pritca na bilo koji nacin. Ovo praktično znaci da možemo da implementiramo veliki broj objekata koji će da koristi osnovne funkcionalnosti objekta koji nasleđuju i u isto vreme implementiraju svoje specifične funkcionalnosti.
* Polimorfizam – Podrazumeva koncent “Jedno ime više oblika”, Deli se na statični i dinamični. Usko je povezan sa nasleđivanjem i omogućava korišćenje različitih funkcionalnosti iz klase koja se nasleđuje i daje mogućnost promene tih funkcionalnsti ukoliko postoji potreba za tim.

****

**Slika 1**- Naslov (autor od kog si preuzeo i godina)

## Nezavisnost od platforme

Nezavisnost od platforme je koncept na osnovu kojeg je projektovan Java programski jezik. On predstavlja jednu vrstu filozofije i principa dizajniranja programskih jezika. Podrazumeva pravilo da kada se jednom napiše kod, taj isti kod može da se pokrene na bilo kojoj mašini, bez obzira na kojoj platformi je ta mašina. Kako Java programski jezik postiže ovo?

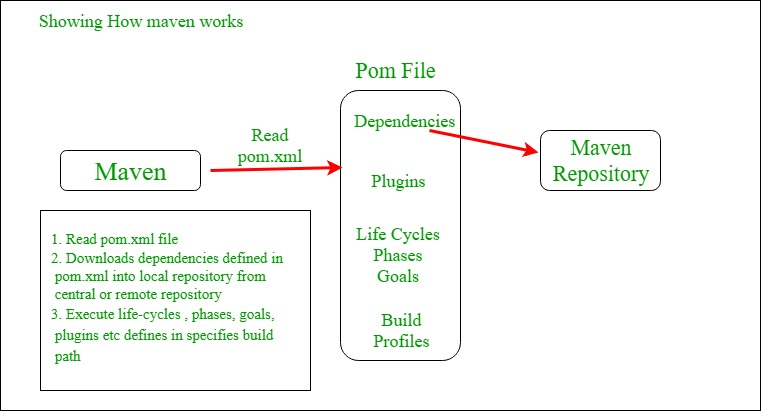


**Slika 2**- Platformska nezavisnost programskog jezika Java

Postiže ga na dva načina. Prvi način je preko Java kompajlera koji prevodi Java kod u bytecode. Bytecode je visokooptimizovan skup pojednostavljenih mašinskih instrukcija specifičnih za Java platformu. Kada se ovaj prvi deo završi, na scenu stupa Java virtuelna mašina koja tumači i izvršava generički Java bajtkod u izvorni mašinski kod na hardveru domaćina. Takođe, uz ova dva mehanizma, dostupne su i standardizovane biblioteke koje omogućavaju jedinstven pristup domaćinovim hardverskim funkcijama, kao što su grafika i niti.

Apache Maven

Apache Maven je alat za upravljanje projektima koji se bazira na POM (project object model) modelu. POM fajlovi su konfiguracioni fajlovi koji sadrže informacije o projektu kao što su: zavisnosti, plugin-ovi itd… Maven koristi ove fajlove za izgradnju (build) projekta.

Osnovni koncepti Maven-a su:

POM Fajlovi

Zavisnosti i repozitorijumi - Zavisnosti su Java biblioteke koje su eksterno uključene u projekat, dok su repozitorijumi mesta gde se nalaze JAR (Java ARchive) fajlovi. Lokalni repozitorijum je samo direktorijum na hard disku vašeg računara. Ako Maven ne pronađe zavisnosti u lokalnom repozitorijumu, on ih preuzima sa centralnog Maven repozitorijuma i smešta u lokalni repozitorijum.

Ciklusi izgradnje, faze i ciljevi – Izgradnja projekta se sastoji od niza faza, a svaka faza izgradnje se sastoji od skupa ciljeva. Maven komanda može biti ime ciklusa izgradnje, faze ili cilja. Ako se zahteva izvršavanje određenog ciklusa izgradnje, izvršavaju se sve faze izgradnje u tom ciklusu. Ako se zahteva izvršavanje određene faze izgradnje, izvršavaju se sve faze izgradnje koje prethode toj fazi u definisanoj sekvenci.

Build profili: Build profili predstavljaju skup konfiguracionih vrednosti koje vam omogućavaju da izgradite projekat koristeći različite konfiguracije. Na primer, možda vam je potrebno da izgradite projekat za lokalni računar, za razvoj i testiranje. Da biste omogućili različite izgradnje, možete dodati različite profile izgradnje u POM fajlove koristeći elemente profila koji se aktiviraju na različite načine.

Build pluginovi: Build pluginovi se koriste za izvršavanje specifičnih ciljeva. Možete dodati plugin u POM fajl. Maven ima nekoliko standardnih pluginova koje možete koristiti, a takođe možete implementirati i sopstvene pluginove

Spring, Spring Boot

Spring Framework je Јава платформа која пружа опширну инфраструктуру за развијање Јава апликација. Razvijen je od strane dva konsultanta Rod-a Johnsona I Jurgen Haller. Oni su 2000-tih godina radili kao eksperti za Java programski jezik. Zbog prirode posla imali su kontakt sa velikim brojem različitih klijenata I softvera koji su bazirani na ovom programskim jeziku. Ubrzo su uvideli niz konzistentnih problema koji su se ponavljali iz projekta u projekt. Kao odgovor na su dizajnirali okvir koji je za cilj ima poboljšanje procesa razvoja aplikacija. Taj okvir je nazvan Spring. Spring je brzo dobio na popularnosti I Rod Jonson izbacuje knjigu sa naslovom J2EE Design and Development.

Pre nego što nastavim sa objašnjenjem Spring okvira potrebno je objasniti koncept koji nije potekao direktno iz Springa, ali je dosta korišićen. Koncept o kome govorim je je Најважнији концепт у Spring-у представља концепт Bean-а. Bean је, у најапстрактнијем

погледу, било која Јава класа која садржи специфичну пословну логику и има следеће

особине:

-Конструктор без аргумената

-Може се серијализовати

-Има отворене методе за постављање и добављање свих својих атрибута.

Upravo su odgovori na konzistentne probleme koji su uočili postali fundamentalni delovi razvoja I filozofije Spring okvira. Osnovni I najbitniji su:

* Inverzija kontrole (Inversion of Control, IoC)
* Umetanje zavisnosti (Dependancy injection, DI)
* Utils, Testing, DB Tools (Utils, Testing, DB Tools)
* MVC

Inversion of control – Problem sa kojim su se susretali 2000-tih godina jeste da u to vreme programeri su morali da napišu celu aplikaciju od početka do kraja, što je podrazumevalo pisanje toka izvršavanja aplikacije (control flow), odnosno kako će ta aplikacija da funkcioniše, kako će da uhvati sve događaje (events) I kako će da odgovori na njih. Ideja dizajnera Springa je da pomeri odgovornost kontrole toka izvršavanja sa programera na okvir. Što suštniski znači da programeri trebaju samo da brinu o biznis problemu koji rešavaju, a da će Spring pozvati njihove Beanove u pravo vreme. Pošto ovaj koncept podrazumeva promenu odgovornosti sa programera na sistem, tako je I dobio svoj naziv inverzija kontrole.

Dependancy Injection – U aplikaciji retko se dešava da jedan Bean postoji samostalno, uglavnom je scenario da mu je potrebna druga komponenta kao što je izvor podataka. Problem koji nastaje iz toga je kako da umetnemo tom Beanu zavisnost ka izvoru podataka koji mu je potreban. Pre Spring okvira svaki programer je kreirao svoje objekte I morao je da brine o zavisnostima koje su potrebne da bi se objekat kreirao. Ovakvo programiranje je u velikoj meri povećavalo zavisnost između komponenti, odnosno otežavalo je realizaciju pravila da komponente treba da budu slabo kuplovane. Upravo je ovo osnovna ideja umetanja zavisnosti, da se pređe sa samostalnog kreiranja objekata na kreiranje pomoću konstruktora, setera ili metoda iz eksternog izvora I na taj način smanji zavisnost između komponenata.