## Šta se desi kada se ukuca neki URL i pritisne enter?

* zasnovana na zahtev/odgovor principu
* svaki par zahtev/odgovor se smatra nezavisnim od ostalih
* ne omogućava praćenje korisničke sesije, tj. niza zahteva upućenih od strane istog klijenta
* U verziji 1.0 po završetku isporuke odgovora klijentu konekcija se zatvara (za novu komunikaciju klijenta sa serverom opet treba da se uspostavi konekcija).
* U verziji 1.1 konekcija se ne zatvara tj. konekcija ostaje otvorena (keep-alive). Klijent će istu konekciju da koristi pri slanju novog zahteva ka serveru. Konekcija ostaje otvorena sve dok neko od stana u komunikaciji (klijent ili server) ne odluči da je neophodno za završi komunikaciju sa drugom stranom, što će uraditi tako što će zatvoriti konekciju.
* [..\Documents\OWP\01 - Servleti Osnove\01 - HTTP Protokol.ppt](../Documents/OWP/01%20-%20Servleti%20Osnove/01%20-%20HTTP%20Protokol.ppt)

## Šta je CORS?

**Preuzimanje resursa sa različitih izvora** - u daljem tekstu PRRI ([енгл.](https://sr.wikipedia.org/wiki/%D0%95%D0%BD%D0%B3%D0%BB%D0%B5%D1%81%D0%BA%D0%B8_%D1%98%D0%B5%D0%B7%D0%B8%D0%BA) *Cross-origin resource sharing - CORS*) je mehanizam koji dopušta da se resursi koji nisu dostupni (kao što su fontovi, [JavaScript](https://sr.wikipedia.org/wiki/%D0%88%D0%B0%D0%B2%D0%B0%D1%81%D0%BA%D1%80%D0%B8%D0%BF%D1%82) kod, i slično), a koji se nalaze na [veb stranici](https://sr.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0), zahtevaju od drugog domena koji je van onog iz kog je zahtev za resursom potekao.

Veb stranica može slobodno da ugrađuje slike, stil, [skriptove](https://sr.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BD%D0%B8_%D1%98%D0%B5%D0%B7%D0%B8%D0%BA), video i ostale potrebne sadržaje (kao što je [Adobe Flash](https://sr.wikipedia.org/wiki/Adobe_Flash)) sa bilo kog drugog domena. Međutim, ugrađeni veb fontovi i [AJAX](https://sr.wikipedia.org/wiki/Ajaks_(programiranje)) ([XMLHttpRequest](https://sr.wikipedia.org/wiki/XMLHttpRequest)) zahtevi su tradicionalno bili ograničeni na pristup sa istog domena kao roditelja veb stranice (zbog bezbednosti). Zahtevi za AJAX-om sa različitih domena su podrazumevano zabranjeni zbog mogućnosti da se vrše napredni zahtevi (POST, PUT i DELETE, kao i druge vrste [HTTP](https://sr.wikipedia.org/wiki/%D0%A5%D0%A2%D0%A2%D0%9F) zahteva, uključujući i specifična HTTP zaglavlja) što može dovesti do raznih bezbednosnih problema.

<https://sr.wikipedia.org/wiki/Preuzimanje_resursa_sa_razli%C4%8Ditih_izvora>

## HTTPS?

**HTTPS** ([енгл.](https://sr.wikipedia.org/wiki/%D0%95%D0%BD%D0%B3%D0%BB%D0%B5%D1%81%D0%BA%D0%B8_%D1%98%D0%B5%D0%B7%D0%B8%D0%BA) ***H****yper****t****ext****T****ransfer****P****rotocol****S****ecure*) је комбинација [Hypertext Transfer Protocol](https://sr.wikipedia.org/wiki/%D0%A5%D0%A2%D0%A2%D0%9F)-a са SSL/TSL протоколом да би се обезбедила [енкрипција](https://sr.wikipedia.org/wiki/Enkripcija) и сигурна идентификација [сервера](https://sr.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80). Користи порт 443. HTTPS конекција се често користи за новчане трансакције преко [Интернета](https://sr.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82) и за пренос осетљивих информација. HTTPS не треба мешати са Secure HTTP (S-HTTP).

<https://sr.wikipedia.org/wiki/HTTPS>

## Cookies, šta su, koji postoje, ko ih postavlja?

**HTTP cookie**, takođe i **cookie** ili **kolačić** je jednostavna [tekstualna datoteka](https://sr.wikipedia.org/wiki/Tekstualna_datoteka) koja se čuva u [veb-pregledaču](https://sr.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B5%D0%B3%D0%BB%D0%B5%D0%B4%D0%B0%D1%87) dok korisnik pregleda neki [veb-sajt](https://sr.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B0%D1%98%D1%82). Kada korisnik u budućnosti pregleda taj isti sajt, sajt može "izvući" ili dohvatiti podatke koji su sačuvani u kolačiću, kako bi bio obavešten o prethodnoj korisničkoj aktivnosti. Kolačići mogu sadržati informacije o tome koje je stranice korisnik posetio, podatke o prijavi, pa i koju dugmad je korisnik kliknuo. Ovi podaci mogu ostati u kolačiću mesecima, čak i godinama.

Kolačići se ponekad mešaju s [programima](https://sr.wikipedia.org/wiki/%D0%A0%D0%B0%D1%87%D1%83%D0%BD%D0%B0%D1%80%D1%81%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC) - međutim, oni su samo komadić [podataka](https://sr.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%B0%D1%82%D0%B0%D0%BA) - i sami ne mogu ništa uraditi. Ne mogu sadržati [viruse](https://sr.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%98%D1%83%D1%82%D0%B5%D1%80%D1%81%D0%BA%D0%B8_%D0%B2%D0%B8%D1%80%D1%83%D1%81) niti instalirati maliciozni softver na računaru na kojem se koriste. Međutim, kolačići za praćenje (*tracking cookies*) i posebno kolačići trećih strana često se koriste kao način da se načine dugoročni zapisi o korisnikovoj istoriji pregledanja.

Ostale vrste kolačića vrše funkcije nužne za moderan [Veb](https://sr.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1). Možda su najvažniji [autentifikacijski](https://sr.wikipedia.org/w/index.php?title=Autentifikacija&action=edit&redlink=1) kolačići, pomoću kojih server zna da li je korisnik prijavljen ili ne. Takođe zna i pod kojim je računarom korisnik prijavljen. [Sigurnost](https://sr.wikipedia.org/wiki/%D0%A1%D0%B8%D0%B3%D1%83%D1%80%D0%BD%D0%BE%D1%81%D1%82) ove vrste kolačića uglavnom zavisi od sigurnosti veb-sajta koji šalje kolačić, pregledača korisnika, i od toga jesu li podaci u kolačiću [šifrovani](https://sr.wikipedia.org/wiki/Enkripcija). Ako je sigurnost niska, [haker](https://sr.wikipedia.org/wiki/%D0%A5%D0%B0%D0%BA%D0%B5%D1%80) može pročitati podatke u kolačiću, pa ih može iskoristiti tako da dobije pristup podacima korisnika, ili se može prijaviti na veb-sajt pod korisničkim imenom i lozinkom.

<https://sr.wikipedia.org/wiki/HTTP_kola%C4%8Di%C4%87>

## Za koji vremenski interval bih podesio da cookie istekne?

Sve zavisi od sistema za koji se koristi cookie. Ako je u pitanju neki system koji se koristi za transakcije novca, coockie bih menja cesce, na tri ili sest meseci, a ako je u pitanju neki manje bitniji system, npr neka drustvena mreza, menjao bih ga redje.

## Da li lozinke treba da se čuvaju u bazi i na koji način? Hesh algoritmi za enkripciju, static i dinamic salt da li se čuvaju, gdje?

Šifre mogu da se čuvaju u bazi, ali ne kao otvoreni tekst, već kao hesh tog teksta.

Algoritmi za enkripciju:

**MD5 algoritam**- MD5 se često koristi za provjeru integriteta datoteka pomoću hesh-a datoteke – prilikom primanja datoteke izračunava se njen MD5 hesh i uspoređuje se sa hesh koji je dobiven uz datoteku pa ako su identični, onda je datoteka nepromijenjena. Druga najčešća primjena MD5 algoritma je za enkripciju kod pohranjivanja lozinki, no postoji velik broj MD5 reverznih baza podataka kojima se tako zaštićene zaporke mogu dekriptirati. Kao zaštita od takvih napada preporučljivo je prije kriptiranja u tekst zaporke dodati poznati slučajni niz bitova što će znatno otežati napad. Isto tako preporučljivo je MD5 enkripciju u takvim situacijama koristiti u više iteracija što također povećava nivo sigurnosti.

**SHA-1 algoritam-** SHA-1 algoritam uzima kao ulaz poruku maksimalne dužine 264 bita i iz nje izračunava sažetak dužine 160 bita. SHA-1 algoritam je također dokazano probijen, ali je za ostvarenje proboja potrebno korištenje kompleksnih analitičkih funkcija za koje se sumnja da su dostupne Internet korisnicima. Unatoč toj relativnoj sigurnosti, upotreba SHA-1 algoritma u budućim aplikacijama je upitna. Kao poboljšani nasljednik SHA-1 algoritma predstavljen je SHA-2 algoritam koji sadrži neke izmjene u odnosu na SHA-1 i kao rezultat daje duži sažetak i to u nekoliko varijanti SHA-224, SHA-256, SHA-384 i SHA512, gdje brojke u nazivu označavaju dužinu sažetka. Ovi algoritmi pošto nisu u široj upotrebi nisu bili predmet detaljnijih ispitivanja, ali zasad za njih ne postoje dokazani proboji.

**RIPEMD algoritam-** RIPEMD (eng. RACE Integrity Primitives Evaluation Message Digest) algoritam je kriptografski algoritam za izračunavanje sažetka objavljen 1996. godine od strane tri autora iz Evrope: Hans Dobbertin, Antoon Bosselaers i Bart Preneel. RIPEMD algoritam bazira se na istim principima kao i MD4 algoritam, a po sigurnosnim svojstvima je sličan popularnijem SHA-1 algoritmu. Postoje 4 varijante RIPEMD algoritma: RIPEMD-128, RIPEMD-160, RIPEMD-256 i RIPEMD-320 pri čemu brojke u nazivu odgovaraju dužinama sažetka koje se algoritmima dobivaju. RIPEMD-128 algoritam je upitne sigurnosti dok algoritmi s dužim sažetkom nemaju povećan nivo sigurnosti već samo smanjenu mogućnost kolizije. Osnovna razlika između SHA-1 i RIPEMD algoritma je u tome što je RIPEMD algoritam razvijen od strane akademske zajednice pa prema tome njegova upotreba nije ograničena patentima. RIPEMD zbog svoje manje popularnosti nije toliko proučavan kao SHA-1 algoritam, ali je i za njega, tj. za originalni RIPEMD dizajn, pronađen proboj 2004. godine.

**MAC algoritam-** MAC (eng. Message Authentication Code) je algoritam koji se upotrebljava za autentikaciju i provjeru integriteta poruka tako što se uz poruku šalje i dodatni podatak koji se naziva MAC sažetak i koji se dobiva upotrebom MAC algoritma i tajnog ključa. Primatelj poruke može uz posjed identičnog tajnog ključa upotrebom istog algoritma provjeriti ukoliko MAC sažetak odgovara primljenoj poruci te time može verificirati integritet i autentičnost poruke. Treba napomenuti kako MAC algoritam nije identičan digitalnom potpisu jer se za verifikaciju koristi isti ključ kojim se i izračunava sažetak. Zbog istog razloga MAC sažetak se ne može koristiti kao dokaz neporecivosti poruke jer svaki primatelj poruke posjeduje tajni ključ kojim može generirati novu poruku i novi MAC sažetak. MAC algoritmi imaju striktne sigurnosne zahtjeve koji specificiraju da takav algoritam mora biti siguran od krivotvorenja, tj. da nitko ne smije biti u stanju generirati važeći MAC sažetak za neku poruku M bez obzira na sadržaj poruke. To znači da ako netko uspije generirati važeći MAC sažetak za bilo kakvu poruku (pa makar njen sadržaj nema nikakvog smisla) korištenjem nekog ključa, onda algoritam nije siguran. Da bi se povećala sigurnost MAC algoritama oni se nadograđuju upotrebom drugih kriptografskih algoritama kao što su na primjer MD5 ili SHA-1 – takvi algoritmi se onda nazivaju prema funkciji koja se koristi, npr. HMAC-MD5 ili HMAC-SHA-1.

<https://www.cert.hr/wp-content/uploads/2006/08/CCERT-PUBDOC-2006-08-166.pdf>

Kod statičkog salta se koristi stalno isti salt I salt se čuva u fajlu sa hesh-om od lozinke, dok bi za dinamički salt trebalo generisati salt iz metapodataka korisničkih zapisa ili nekih drugih atributa i on se nigde ne čuva, već se generiše novi, jedinstveni salt za svakog korisnika.

## Aplikacija usporila jer upit nad bazom dugo traje, šta da uradimo kako bi ubrzali rad aplikacije bez da mijenjamo aplikativni sloj?

Moraš nać odgovor!!!

## Dva korisnika mijenjaju isti resurs (kolonu u redu tabele). Jedan je izmijenio i sačuvao promjene, drugi hoće da sačuva svoje međutim njegovi podaci više nisu validni jer ima stare podatke. Kako da riješim taj problem?

Problem ćemo rešiti zaključavanjem n-torke ili cele tabele nad kojom se vrši izmena.

Generalno, postoje dva osnovna stila zaključavanja: optimistično I pesimistično.

**Kod**optimističnog zaključavanja**, pretpostavka je da se konflikt neće dogoditi.** **Podaci se čitaju, transakcije se obrađuju, vrše se ažuriranja i tada se proverava da li je došlo do konflikta. Ako nije, transakcija se završava. Ako se koflikt pojavio, transakcija se ponavlja sve dok se ona ne obradi bez konflikta.**

**Kod** pesimističnog zaključavanja, **pretpostavka je da će se konflikt pojaviti. Najpre se vrši zaključavanje, zatim obrađuje transakcija, a zatim se vrši otključavanje.**

Prednost optimističnog zaključavanja je da zaključavanje traje mnogo kraće nego u slučaju pesimističnog zaključavanja. Ova prednost je naročito značajna ako je granularnost zaključavanja velika - na nivou cele tabele (u ovom slučaju tabele PRODUCT).

Primenom mehanizma zaključavanja uz dvofaznost transakcija, mogu se rešiti sva tri problema konkurentnosti.

## Gdje bih čuvao flag (timestamp ili broj verzije) a da ne koristim cookies ili js, imam samo HTML?

Moraš nać odgovor!!!

## GET metoda za šta se koristi, da li mora da se koristi samo za dobavljanje resursa?

GET metoda sluzi za dobavljanje resursa sa server. Podaci za dobavljanje resursa sa server se smeštaju u uri. **GET metoda** se koristi za traženje URL-a od web poslužitelja za dohvaćanje HTML dokumenata. To je uobičajena metoda za preglednike da dostave informacije koje se računaju kao dio HTTP protokola. GET metoda predstavljena u obliku URL-a, tako da se može označiti. GET se u velikoj mjeri koristi u tražilicama. Nakon što korisnik uputi upit tražilici, motor izvršava upit i daje dobivenu stranicu. Rezultati upita mogu se postaviti kao veza (označena).

GET metoda omogućuje generiranje sidara, što pomaže u pristupu CGI programu s upitom bez uporabe forme. Upit je konstruiran u vezu, tako da kada se veza posjeti, program CGI dohvatit će prikladne informacije iz baze podataka.

GET metoda ima nekih sigurnosnih problema jer su umetnuti podaci vidljivi u URL-u. Samo ograničena količina podataka može se prenijeti metodom GET, jer duljina URL-a koji preglednik može preći može biti tisuću znakova.

Drugo pitanje koje se tiče GET metode je da se ne može baviti stranim jezicima. GET metoda se ne predlaže koristiti, ali ipak kada se ne definiraju atributi metode, GET metoda se koristi kao zadana.

Mora da se koristi samo za dobavljanje resursa, inace moze nastati haos.

## Da li može da se pozove privatna metoda?

Programski jezik Java, objavljen 1996. godine od strane Sun Microsystems, je najkoriˇs´cenijih programski jezik objekno-orijentisane paradigme.

Refleksija u Javi sluˇzi za ispitivanje i modifikaciju ponaˇsanja metoda, klasa i interfejsa u toku izvrˇsavanja programa. Potrebne klase za koriˇs´cenje refleksije se nalaze u java.lang.reflect [10] paketu i omogu´cavaju dobijanje informacija o klasi kojoj pripada prosledeni objekat i metodama koje korisnik moˇze da izvrˇsi nad istim. Kroz refleksiju je takode omogu´ceno pozivanje metoda klase bez obzira na njihovu vidljivost. Refleksija omogu´cava dobijanje informacija o:

• Klasama - getClass() metoda vra´ca ime klase datog objekta 4

• Konstruktorima - getConstructors() metoda se koristi za dobijanje informacija o javnim konstruktorima klase objekta

• Metodama - getMethods() metod sluˇzi za dobijanje informacija o javnim metodama klase objekta.

За сваки претходно побројани елемент тј. тип, Јава виртуална машина формира немутирајући примерак класе java.lang.Class

• Он обезбеђује методе за истраживање run-time особина објекта, укључујући информације о пољима, методама и типовима.

• Примерак класе Class такође обезбеђује и могућност да се „у лету“ креирају нове класе и објекти.

Током извршавања Јава програма, JVM учитава бајт-код тј. class датотеке и креира објекте који представљају те класе. Објекат који представља класу садржи име (поље типа String), листу поља (свако је типа Field), листу метода…

Рефлексија је искључиво самоиспитивање

• Није могуће додати/модификовати поља (тј. мењати структуру класе)

• Није могуће додати/модификовати методе (тј. мењати понашање)

• Преко рефлексије није доступна имплементација

• Рефлексијом се не одсликава програмерска логика

• Велики утицај на перформансе

• Код је много спорији него у случају када се иста операција реализује директним путем…

• Резултујући код је веома комплексан.

## Modifikatori pristupa u javi?

* Ponekad je potrebno obezbediti kontrolisan pristup atributima, kako za čitanje, tako i za pisanje. Tada se koriste se modifikatori pristupa
  + **public** – vidljiv za sve klase
  + **protected** – vidljiv samo za klase naslednice i klase iz istog paketa
  + **private** – vidljiv samo unutar svoje klase
  + **nespecificiran** (package-private) – vidljiv samo za klase iz istog paketa (direktorijuma, foldera)
* Modifikatori pristupa se navode ispred definicija klasa, metoda i atributa.
* Kada atributima i metodama odredimo i napišemo modifikatore pristupa, dobija se klasa kojoj je omogućen **kontrolisani** pristup iz dugih klasa i programa.
* Klase mogu međusobno da komuniciraju bez znanja o međusobnoj implementaciji njihovih metoda, npr. da li pozvana javna metoda u svom telu poziva i neke druge zaštićene metode ili da ona koristi neke zaštićene atribute, itd.
* **Detalji implementacije su skriveni**, tj. **enkapsulirani** unutar klase. Druga klasa vidi prvu klasu samo kroz metode kojima ona može da pristupi!!!

## Apstraktna klasa vs Interfejs?

Apstraktne klase:

* Osnovna klasa koja nema nijedan konkretan (realan) objekat, već samo predstavlja generalizaciju izvedenih klasa, naziva se apstraktnom klasom.
* Apstraktna klasa može da sadrži apstraktne funkcije, koje su u ovoj klasi samo deklarisane, a nije implementirane
* Klase koje ne mogu imati svoje objekte, već samo njene klase naslednice mogu da imaju objekte (ako i one nisu apstraktne)
* Ako klasa ima makar jednu apstraktnu metodu, mora da se deklariše kao apstraktna.
* Apstraktna klasa ne mora da ima apstraktne metode!

Interfejsi:

* Omogućavaju definisanje samo apstraktnih metoda, konstanti i statičkih atributa
* Ključna reč **implements**
* **Interfejs nije klasa**! On je spisak metoda i atributa koje klasa koja implementira interfejs mora da poseduje.
* Interfejsi se ne nasleđuju, već implementiraju
* Da bi klasa implementirala interfejs, mora da redefiniše sve njegove metode
* Jedan interfejs može da nasledi drugog
* Sve metode su implicitno **public**, a svi atributi su implicitno **public static final**

## U javaScript razlika izmedju "==" i "==="?

• Operatori == i != obavljaju potrebnu konverziju podataka pre poređenja, ukoliko su operandi različitog tipa. Znači za ove operatore vrednosti 5 (integer) i "5" (string) su iste, pa će posle njihovog poređenja rezultat sa operatorom == biti TRUE, a sa operatorom != FALSE.

• S druge strane operatori === i !== ne obavljaju potrebnu konverziju podataka pre poređenja, ukoliko su operandi različitog tipa. Znači za ove operatore vrednosti 5 (ceo broj) i "5" (string) su različite, pa će posle njihovog poređenja rezultat sa operatorom === biti FALSE, a sa operatorom !== TRUE.

## Dal moze da se menja index u bazi?

Index u bazi ne moze da se menja.

<http://poincare.matf.bg.ac.rs/~jelenagr/prbp/crud5.pdf>

## Koja je razlika izmedju fizickog i logickog brisanja?

Kod fizickog brisanja, podatak se uklanja sa fajl sistema, dok se kod logickog brisanja koristi polje podatka, koje je cesto tipa Boolean i vrednost tog polja se menja kada se vrsi logicko brisanje.

## Sta je polimorfizam?

* Opisuje koncept u kome se određena akcija može izvršiti na više načina. Polymorphism je nastao kombinacijom grčkih reči *poly* (više) i *morphs* (izgled/forma)
* Može biti:
  + *compile time polymorphism* (implementira sa *method overloading*). Za vreme kompajliranja zna se koji se metod poziva na osnovu argumenata
  + *runtime polymorphism* (implementira sa *method overriding*). Kompajler ne određuje koja konkretna implementacija metoda će biti pozvana. Implementacija se određuje za vreme izvršavanja u zavisnosti od toga koji je konkretan objekat instanciran.
* Naglasak na *Runtime polymorphism* koji se još zove *Dynamic Method Dispatch.*
* Situacija kada se poziva metoda nekog objekta, a ne zna se unapred kakav je to konkretan objekat
  + ono što se zna je koja mu je bazna klasa
* Tada je moguće u programu pozivati metode bazne klase, a da se zapravo pozivaju metode konkretne klase koja nasleđuje baznu klasu
* Prednost korišćenja polimorfizma ogleda se u toma da nam on omogućava kreiranje uniformnog pristupa/kontrole ka različitim objektima koji imaju zajednički podskup operacija.
* Rezultat polimorfizma je kod koji je više koncizan i lakši za održavanje.

## Nabroj solid principe i objasni jedan?

[S.O.L.I.D.](https://en.wikipedia.org/wiki/SOLID) dizajn principi predstavljaju kolekciju najbolje prakse za objektno-orjentisan dizajn. SOLID principi su prvi put opisani u knjizi Design Patterns koju su napisali čuveni Gang of Four ( Richard Helm, Erih Gama, Ralf Džonson, Džon Vilisids). Naziv SOLID principi dolazi iz prvih slova svakog principa: S – Single Princip, O – Open-Close Princip, L- Liskov Substitution Princip, I – Interface Segregation Princip, D – Dependancy Inversion Princip.

**Single Responsibility Princip:** Single Responsibility Princip govori da svaki objekat treba da ima jedan fokus. Pridržavanjem ovog principa, izbegava se problem monolitnih klasa. Ako su objekti koncizni, povećava se čitljivost i omogućava lakše održavanje sistema. Primena ovog principa omogućava veliku prednost u fleksibilnosti softvera.

**Open-close Princip:** Open-close Princip  govori da klasa treba da bude otvorena za proširenja a zatvorena za izmene. Dalje, treba omogućiti dodavanje novih karakteristika i proširenje klase bez promene unutrašnjeg ponašanja postojećih metoda. Princip teži da se izbegne menjanje postojeće klase i drugih klasa koje od nje zavise, jer će to dovesti do pojavljivanja velikog broja grešaka u samoj aplikaciji.

**Liskov Substitution Princip:** Liskov Substitution Princip govori da bi trebalo omogućiti korišćenje bilo koje izvedene klase na mestu klase roditelja i da bi ta klasa trebala da se ponaša na isti način bez izmena.

**Interface Segregation Princip:** Interface Segregation Princip ima za cilj dodeljivanje novog interfejsa grupama metoda koje imaju isti fokus kako bi se izbeglo da klijent mora da implementira jedan veliki interfejs i veliki broj metoda koje mu nisu potrebne.

**Dependency Inversion Princip:** Dependency Inversion Princip se bazira na izolovanju klasa od konkretne implementacije kako bi zavisile od apstraktnih klasa i interfejsa. On promoviše kodiranje bazirano na interfejsima, a ne na implementaciji.

Najvažnijim principom smatra se DRY, koji kaže da ne ponavljamo kod. Pored toga DRY princip kaže da se osim koda ni informacija ne sme ponavljati. Informacija se u arhitekturi sitema ili bazi sme nalaziti samo na jednom mestu.

Ukoliko želite da saznate više o SOLID principima kliknite na link ispod teksta.

## Multithreding prednosti i mane?

**Mane:** Konflikt medju threadovima ne postoji ukoliko rade nad ¯ razlicitim podacima ili nad disjunktnim celinama istih ˇ podataka/resursa, Konflikt nastaje kada više threadova želi da vrši operacije nad istim podskupom resursa

U nastavku cemo smatrati da su programi koje koristimo ´ tacni

ˇ I To ne znaci da oni ne ˇ ce izazvati konflikte kada se ´ izvršavaju u više threadova I

Zato treba da obezbedimo da oni budu thread-safe

<http://ns2.math.rs/~jelenagr/paralelniAlgoritmi/UvodSuperKompjuteriMultuthreadParalenoProcesiranje.pdf>

[https://sr.wikipedia.org/wiki/%D0%92 %D0%B8%D1%88%D0%B5%D0%BD%D0%B8%D1%82%D0%BD%D0%B0\_%D0%BE%D0%B1%D1%80%D0%B0%D0%B4%D0%B0](https://sr.wikipedia.org/wiki/%D0%92%D0%B8%D1%88%D0%B5%D0%BD%D0%B8%D1%82%D0%BD%D0%B0_%D0%BE%D0%B1%D1%80%D0%B0%D0%B4%D0%B0)

Предности

Неке од предности су:

Ако нит направи пуно промашаја кеша, друге нити могу наставити, користећи предност неискоришћених рачунарских ресурса, што може довести до бржег извршавања, с обзиром да би ти ресурси били неискоришћени да се само једна нит извршава.

Ако нит не може искористити све рачунарске ресурсе процесора (јер инструкције зависе од међусобних резултата), покретање друге нити их може искористити.

Ако неколико нити обрађује исти скуп података, оне могу делити кеш, што доводи то боље искоришћености кеша или синхронизације њихових вредности.

Мане

Неке од критика вишенитне обраде су:

Вишеструке нити могу међусобно сметати једна другој при дељењу хардвера као што су кешеви или [бафер асоцијативног превођења](https://sr.wikipedia.org/wiki/%D0%91%D0%B0%D1%84%D0%B5%D1%80_%D0%B0%D1%81%D0%BE%D1%86%D0%B8%D1%98%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B3_%D0%BF%D1%80%D0%B5%D0%B2%D0%BE%D1%92%D0%B5%D1%9A%D0%B0) (TLBs).

Извршно време једне нити није побољшано већ може бити смањено, чак и када се само једна нит извршава. Ово је због ниже фреквенције и/или додатних фаза проточне обраде које су неопходне да се реализује хардвер смене нити.

Хардверска подршка за вишенитну обраду је видљивија софтверу, стога захтева више промена на нивоу програма и оперативног система, за разлику од вишепроцесорских система

## Sta je konstruktor?

* Ako ne napravimo konstruktor, kompajler će sam napraviti default konstruktor, koji ništa ne radi.
* Taj konstruktor se neće kreirati na nivou izvornog koda, već na nivou bajt-koda (prevedenog koda).
* U konstruktoru inicijalizujemo atribute koji bi trebalo da su inicijalizovani.
* Konstruktor može primati i parametre.
* Kako je konstruktor metoda klase možemo da napravimo proizvoljan broj konstruktora sve dok se oni razlikuju po broju i tipu parametara.
* Ukoliko se u klasi napiše barem jedan konstruktor, tada podrazumevani konstruktor više ne postoji.

## Referencijalni integritet?

U oblasti [baza podataka](https://sr.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%BF%D0%BE%D0%B4%D0%B0%D1%82%D0%B0%D0%BA%D0%B0), **referencijalni integritet** je svojstvo koje osigurava da su veze (reference) između podataka ispravne.[[1]](https://sr.wikipedia.org/sr-el/Referencijalni_integritet" \l "cite_note-blaha-1)

Preciznije, kako bi referencijalni integritet važio, svako polje u nekoj [tabeli](https://sr.wikipedia.org/w/index.php?title=%D0%A2%D0%B0%D0%B1%D0%B5%D0%BB%D0%B0_(%D0%B1%D0%B0%D0%B7%D0%B5_%D0%BF%D0%BE%D0%B4%D0%B0%D1%82%D0%B0%D0%BA%D0%B0)&action=edit&redlink=1) koje je deklarisano kao [strani ključ](https://sr.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D0%BD%D0%B8_%D0%BA%D1%99%D1%83%D1%87) mora da sadrži samo one vrednosti koje se javljaju u određenom[[2]](https://sr.wikipedia.org/sr-el/Referencijalni_integritet#cite_note-2) u [primarnom ključu](https://sr.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BC%D0%B0%D1%80%D0%BD%D0%B8_%D0%BA%D1%99%D1%83%D1%87) ili [kandidatu za ključ](https://sr.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B4%D0%B8%D0%B4%D0%B0%D1%82_%D0%B7%D0%B0_%D0%BA%D1%99%D1%83%D1%87) tabele na koju pokazuje strani ključ. Na primer, brisanje unosa koji sadrži vrednost na koju referiše strani ključ iz neke druge tabele bi prekršilo referencijalni integritet. Neki [sistemi za upravljanje relacionim bazama podataka](https://sr.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC_%D0%B7%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D1%99%D0%B0%D1%9A%D0%B5_%D1%80%D0%B5%D0%BB%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%B8%D0%BC_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B0_%D0%BF%D0%BE%D0%B4%D0%B0%D1%82%D0%B0%D0%BA%D0%B0) (SURBP) mogu da zahtevaju da referencijalni integritet bude očuvan: obično brisanjem i reda u kome se nalazi strani ključ koji pokazuje na podatak koji treba da bude obrisan, ili prijavljivanjem greške i nebrisanjem podatka. Koji metod će biti korišćen se određuje prilikom definisanja stranih ključeva.

198. strana u knjizi

## Kompozitni kljuc?

o Kompozitni ključ je ključ koji je sastavljen iz vrednsoti koje se nalaze u različitim kolonama o Koristi se u situaciji kada na osnovu vrednosti u jednoj koloni ne možemo da identifikujemo svaki zapis o Često se umesto kompozitnog ključa koristi generički ključ.

## HTTP vs HTTPS?

ХТТП је скраћеница од Хипертект Трансфер Протоцол. У основи је основно, омогућава комуникацију између различитих система. Најчешће се користи за пренос података са веб сервера у прегледач како би се корисницима омогућило прегледање веб страница. То је протокол који се користио у основи за све ране веб локације.

ХТТПС је скраћеница од Хипертект Трансфер Протоцол Сецуре. Проблем са уобичајеним ХТТП протоколом је тај што информације које прелазе са сервера на прегледач нису шифроване, што значи да се лако могу украсти. ХТТПС протоколи то исправљају коришћењем ССЛ (слој сигурних утичница) сертификата, који помаже у стварању сигурне шифроване везе између сервера и прегледача, штитећи тако потенцијално осетљиве информације од крађе током њиховог преноса између сервера и прегледача.

Најважнија разлика између ова два протокола је ССЛ сертификат. У ствари, ХТТПС је у основи ХТТП протокол са додатном сигурношћу. Међутим, ова додатна сигурност може бити изузетно важна, посебно за веб локације које од својих корисника узимају осетљиве податке, као што су подаци о кредитној картици и лозинке.

Како функционише ХТТПС? ССЛ сертификат шифрира информације које корисници достављају на веб локацију, што у основи преводи податке у код. Чак и ако неко успе да украде податке који се преносе између пошиљаоца и примаоца, због ове енкрипције не би могао да их разуме.

Али поред додавања додатног нивоа заштите, ХТТПС је такође заштићен и путем ТЛС (Транспорт Лаиер Сецурити) протокола. ТЛС помаже у обезбеђивању интегритета података, што помаже у спречавању модификовања или оштећења преноса података и аутентификације, што доказује вашим корисницима да комуницирају са предвиђеном веб локацијом.

Корисници могу да идентификују да ли нека веб локација користи ХТТПС протокол по веб адреси. Први део веб адресе (пре „ввв“) показује да ли веб локација користи ХТТП или ХТТПС протоколе.

Дакле, да резимирамо, разлика између ХТТП-а и ХТТПС-а је једноставно присуство ССЛ сертификата. ХТТП нема ССЛ, а ХТТПС има ССЛ, који шифрује ваше податке тако да су ваше везе заштићене. ХТТПС такође има ТЛС (Транспорт Лаиер Сецурити) протокол који недостаје ХТТП-у. ХТТПС је сигурнији од ХТТП-а.

Ако гледате главну разлику између ХТТП-а и ХТТПС-а, ХТТПС очигледно има велику предност. На крају, не бисте ли желели да ваша веб локација буде што сигурнија? Ствар је у томе што ако немате страницу за е-трговину и не прихватате потенцијално осетљиве информације од посетилаца веб локације, можда мислите да прелазак на ХТТПС локацију није толико неопходан и да то чини је већа гњаважа него што вреди.

Међутим, безбедносна предност није једина предност коришћења ХТТПС-а. У ствари, прелазак на ХТТПС може на крају подстаћи и ваше СЕО напоре. Следи неколико начина на које ХТТПС може да помогне да се побољша ваш СЕО:

**Повећајте рангирање своје веб странице**

Ако се питате, да ли је ХТТПС добар за СЕО? Доврага, ХТТПС је важан за СЕО! Јер поред чињенице да је сам Гоогле најавио да ће веб локације пребачене на ХТТПС примити мали ударац у рангирању, то у сваком случају може довести до повећања рангирања веб сајта у сваком случају, јер ће посетиоци вероватније прегледати странице за које знају да јесу сигуран.

**Сачувај податке о препоруци**

Такође, употреба ХТТПС странице чини Гоогле аналитику ефикаснијом. То је зато што се безбедносни подаци веб локације која вас је упутила чувају помоћу ХТТПС-а - нису на ХТТП веб локацијама. Код ХТТП веб локација, реферални извори ће се појавити само као „директни саобраћај“. То ХТТПС-у само по себи даје велику предност за СЕО.

**Изградите поверење са посетиоцима**

Будући да ХТТПС веб локација шифрира сву комуникацију, посетиоци ће имати заштиту не само на својим осетљивим информацијама, попут лозинки и података о кредитним картицама, већ и на својој историји прегледања. Знајући да ће задржати приватност током прегледавања ваше веб локације и знајући да их све што преузму, пријаве или купе неће довести у ризик због недостатка сигурности, помоћи ће вам да изградите поверење, што је од виталног значаја за хватање потенцијалних клијената и затварање продаје.

Поред тога, ХТТПС штити вашу веб локацију од нарушавања безбедности, које могу на крају оштетити вашу репутацију, па чак и коштати вас новца ако се догоде.

**Испуњавање услова за стварање АМП страница**

Ако желите да можете да користите АМП (убрзане мобилне странице), мораћете да имате ХТТПС. Гоогле је АМП створио као начин за учитавање садржаја на мобилне уређаје по много бржој брзини. У својој основи, АМП је некако попут уклоњеног ХТМЛ-а. АМП садржај је истакнут на Гоогле-овим СЕРП-овима како би створио боље мобилно искуство за кориснике паметних телефона и таблета.Ако вам је стварање веб локације прилагођене мобилним уређајима (а требало би да буде, с обзиром на све већу важност рангирања у претрази за мобилне уређаје и локалног СЕО-а), прелазак на ХТТПС неопходан.

## Sta je hash?

Heš funkcije se prvenstveno koriste za generisanje **fiksne dužine izlaznih podataka** koji se ponašaju kao reference na originalne podatke. Ovo je korisno kada su originalni podaci suviše glomazni da bi se koristili u celosti. Jedna prakticna primena je struktura podataka koja se zove [heš tabela](https://sr.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88_%D1%82%D0%B0%D0%B1%D0%B5%D0%BB%D0%B0) u kojoj su podaci smešteni asocijativno. Linearna pretraga imena osobe u listi traje duže kako se dužina liste povećava, ali heširana vrednost može biti skladištiti referencu na originalni podatak, pa tako dobijamo konstantno vreme za pretragu.

Druga upotreba je u [kriptografiji](https://sr.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%98%D0%B0), nauci kodiranja i očuvanja podataka. Heš funkcije predstavljaju osnovni princip za PGP algoritam za validaciju podataka. Takođe se koriste da bi se ubrzalo traženje stavki u bazama podataka, detektovanje dupliranih ili sličnih vrednosti u velikom fajlu i pronalaženje sličnih segmenata u DNK sekvenci. Heš funkcija treba da bude deterministički odredjena, tj. kada se dva puta pozove nad identičnim podacima (npr. dve niske koje sadrže potpuno iste karaktere), funkcija treba da proizvede istu vrednost. To je od ključnog značaja za ispravnost gotovo svih algoritama na osnovu heširanja. Heš funkcije obično nisu invertibilne, što znači da nije moguće rekonstruisati ulaznu vrednost *x* samo iz njene heš vrednosti *h(x)*. U mnogim aplikacijama uobičajeno je da se nekoliko različitih vrednosti slika u istu heš vrednost i to stanje zovemo **heš kolizijama**. Ovi sudari mogu izazvati zabunu medju objektima, što može doprineti da algoritmi rade sporije i da budu manje precizni. Heš funkcije su dizajnirane tako da se što više smanji verovatnoća sudara. Za kriptografske namene, heš funkcije su projektovane na takav način da je nemoguće rekonstruisati bilo kakav ulaz samo iz heš funkcije, bez trošenja velikog računarskog vremena.

## Gde bi skladistio podatke forme koje korisnik ne treba da menja?

Probaj nesto nac za ovo!!!

## Koje su metode transakcije i kada bi ih pozivao?

Metode transakcije su COMMIT i ROLLLBACK.

Naredba COMMIT signalizira administratoru transakcija o uspešnom

završetku transakcije. Baza podataka ponovo se nalazi u konzistentnom stanju

i sva dejstva transakcije (privremena ažuriranja) mogu biti trajno preneta na

bazu podataka. Nasuprot, naredba ROLLBACK signalizira administratoru

transakcija o neuspešnom završetku transakcije. U tom slučaju potrebno je sva

dejstva nad bazom podataka takve transakcije poništiti. Naredbe COMMIT i

ROLLBACK se zadaju direktno ili automatski od strane SUBP.

## Da li moze try bez catch-a?

*try* blok obuhvata korišćenu funkciju i “hvata” izuzetak, odnosno sprečava da on zaustavi program

*catch* blok definiše proceduru za obradu izuzetka

*finally* blok definiše proceduru koja treba da se izvrši i u slučaju uspešnog i u slučaju neuspešnog izvršavanja funkcije (čak i ukoliko u try ili catch blokovima stoji return naredba)

mora da postoji *try* blok i bar jedan od blokova *catch* ili *finally*, a mogu i oba

## Throw exception vs throws?

Основна разлика је у томе што изузетак Тхров преписује траг стека и то отежава проналажење оригиналног броја линије кода који је изузео изузетак.

Throw у основи задржава информације о стеку и додаје их у стек, осим у случају да су бачене.

Дакле, сваки пут када поново баците изузетак помоћу изјаве „Баци“, он чува оригинални траг стека, а ако поново баците изузетак помоћу „Баци ек“, траг стека се ресетује на тачку на коју је изузетак избачен.

Prouci ovo jos malo!

## Sta je Map/Dictionary?

Kolekcija ključ-vrednost parova.

Objekat bilo koje klase može da bude ključ ili vrednost.

Ključevi moraju biti jedinstveni, duplikati nisu dozvoljeni.

Dictionary u Јави је апстрактна класа која представља спремиште за складиштење кључа / вредности и делује као мапа. Ако имамо пар кључ / вредност, тада можемо вредност сачувати у објекту Дицтионари. Једном када је вредност ускладиштена у речничком објекту, можемо је преузети помоћу њеног кључа. У Речнику се сваки не-нулл објекат може користити као кључ и као вредност.

Hashtable = Dictionary

* Hashtable metode sinhronizovane.

## Indexi?

Indeksi se koriste iz sledećih razloga:

Obezbedjivanje bržeg pristupa po kolonama koje se indeksiraju bez čitanja cele tabele,

Obezbedjivanje jedinstvenosti vrednosti kolona koje čine indeks (kalone koje čine indeks mogu imati ulogu primarnog ključa).

Opšti oblik naredbe za kreiranje indeksa je:

CREATE (UNIQUE) INDEX naziv\_indeksa

ON naziv\_tabele (naziv\_obl, [ASC DESC) (naziv obl, (ASC | DESC])...) [drugi parametri);

Po jednoj tabeli može se kreirati više indeksa. Svaki indeks povećava vreme potrebno za održavanje tabele. Svaki indeks mora imati jedinstveno ime.

Klauzule ASC DESC odredjuju rastući ili opadajući redosled uredjenja indeksa. Klauzula UNIQUE odredjuje da tabela ne može sadržavati dve D-torke sa istim sadržajem obeležja koja čine indeks. Ovim mehanizmom se može obezbediti uloga primarnog ključa. Ako klauzula UNIQUE nije navedena mogu postojati dve ili više n-torki sa istim vrednostima indeksa.

Posle izdavanja naredbe za kreiranje indeksa on se automatski održava.

Zahtev za kreiranja UNIQUE indeksa nad tabelom koja sadrži podatke koji ne ispunjavaju uslov jedinstvenosti neće biti izvršen.

Indeks se briše naredbom: DROP INDEX naziv indeksa;

Primer: Kreirati indeks nad tabelom NASTAVNIK po obeležju PREZIME\_IME

Rešenje: CREATE INDEX ABC\_IME ON NASTAVNIK (PREZIME\_IME);

Indeks ABC\_IME omugućava pristup podacima tabele NASTAVNIK po abecedi obelcžja PREZIME\_IME Mogu postojati više nastavnika sa istim prezimenom i imenom

Primer: Kreirati indeks tabele NASTAVNIK koji može imati ulogu primarnog ključa.

Rešenje: CREATE UNIQUE INDEX NASTAVNIK\_KEY ON NASTAVNIK (S\_NAS);

## No-SQL baze?

Tri popularne NoSQL baze podataka — MongoDB, Neo4j i Apache Cassandra.

Prvo moramo da napravimo digresiju i postavimo neke osnove za ovo naše igralište. NoSQL, uprkos uvreženom mišljenju, ne isključuje SQL. Zapravo, NoSQL znači: „Ne samo SQL” (“*Not only SQL*”). Mnoge SQL baze podataka imaju jezičku podršku nalik na SQL, zato što to ima smisla prilikom ispitivanja podataka.

Šta to, onda, razlikuje i klasifikuje NoSQL od SQL baza podataka? Sve se svodi na jednu stvar: Fleksibilnost! NoSQL baza podataka uobičajeno ne zahteva rigidnu strukturu, ne prisiljava vas da ispaštate zbog vaših odluka vezanih za dizajn i proklinjete dan kada ste tu kolonu podesili na *numeric*, a ne *varchar*… Ne, NoSQL baza podataka će želeti da bude vaš prijatelj i dozvoliće vam da usput adaptirate vaš model. Znamo da ništa u životu nije džabe, pa šta je onda cena za ovakvu funkcionalnost? Pa, bez rigidnih pravila i strukture, vaši podaci mogu da ispaštaju od nedoslednosti, a vi možete završiti sa nepovezanim podacima bez referenci, ukoliko niste veoma pažljivi u kasnijoj primeni… Sa više moći dolazi i više odgovornosti. Međutim, stvarno je vredno toga, pošto su NoSQL baze podataka *de facto* brže i mogu da sadrže više podataka.

Prema tome kako NoSQL baze čuvaju podatke, možemo ih podeliti na:

bazirane na kolonama,

bazirane na dokumentima,

ključne vrednosti,

bazirane na grafovima,

multimodelne.

Spring Data je, grubim rečima, JPA za NoSQL baze podataka. Ona zapravo dodaje sloj apstrakcije na JPA, dok je implementacija ostavljena posebnim drajverima za NoSQL baze podataka. Takođe dodaje sopstvenu, vrlo kul, funkciju implementacije bez koda za obrasce spremišta (*repository*) i kreiranje upita (*queries*) baza podataka od metodnih imena.

<https://startit.rs/nosql-igraliste-ili-tri-nosql-baze-podataka-i-njihove-karakteristike/>

Poslednjih godina su se razvili noviji sistemi koji obezbeduju veliku horizontalnu skal- ¯ abilnost za jednostavne operacije citanja i pisanja nad podacima distribuiranim na više ˇ servera izuzetno velikom brzinom dok relacioni modeli baza podataka ne omogucavaju ´ ili omogucavanju jako malu horizontalnu skalabilnost. Horizonstalna skalabilnost znaciˇ da se lako može dodati novi cvor (server) kada je to potrebno i to ne uti ˇ ce na rad sistema. ˇ Mnogi od ovakvih sistema su svrstani grupu NoSQL i za sve modele skladista podataka koji su svrstani u ovu grupu vaze sledece kljucne osobine: ´ – Sposobnost ponavljanja i distribucije podataka na vise servera. 1 ACID predstavlja akronim za osobine transakcija u relacionim bazama podataka = atomicnost ˇ (eng. Atomicity), konzistentnost (eng. Consistency), izolaciju (eng. Isolation), i trajnost (eng. Durability). 4 Sanja Mijalkovic 1061/2013 ´ – Obezbedena horizontalna skalabilnost i jednostavne operacije izme ¯ du više servera. ¯ – Jednostavan spoljašnji interfejs ili protokol (za razliku od SQL-a) – Uglavnom podržavaju slabiji konkurentni model od ACID osobina transakcija (koje su zastupljenje u SQL-u). – Sposobnost da dinamicki dodaju nove atribute. ˇ – Efikasno korišcenje distribuiranih indeksa. ´ ACID osobine uglavnom nisu obezbedjene u ovakvim modelima. Ideja je da se zaobidju ogranicenja ACID osobina jer se na taj nacin zaobilazi nepotrebna kompleksnost, ˇ podižu se performanse i omogucava skalabilnost. Obi ´ cno se garantuje da važi skup prin- ˇ cipa BASE2 . Stanje sistema se menja tokom vremena, može doci do problema pri doda- ´ vanju novog cvora u sistem jer se tada sadržaj mora umnožiti i ne postoji garancija da ˇ ce´ svi cvorovi sadržati identi ˇ cne kopije podataka u svakom trenutku, pa se teži ta ˇ cki u kojoj ˇ ce to biti ispunjeno i svi podaci konzistentni. [ ´ 4], [10]

Iz tih izazova nastale su nove vrste baza podataka kod kojih se podaci ne organiziraju prema principima relacijskog modela nego, prema puno jednostavnijem i “slobodnijem” modelu podataka, zavisno od vrste takve jedne baze podataka. Jedna od najuočljivijih karakteristika većine ovih baza Vol. 4, No. 1, 2016. DOI: 10.19279/TVZ.PD.2016-4-1-06 45 POLYTECHNIC & DESIGN Vol. 4, No. 1, 2016. podataka je to da one ne zahtijevaju definiciju sheme podataka, kao što je to slučaj s relacijskim bazama podataka. NoSQL baze podataka su zbog toga dinamičke, u smislu da je tip i broj atributa nekog entiteta “otvoren”, to jest može ga se po potrebi mijenjati bez da to utječe na druge podatke. Prednost ovog pristupa je u tome što je jednostavnije preslikati strukturu podataka u nekom programskom jeziku na sličnu sturkturu podataka neke takve baze podataka, čime se pojednostavljuje interakcija između aplikacije i baze podataka. Još jedna važna karakteristika NoSQL baza podataka je to da su one napravljene za distribuiran način upotrebe – dok su relacijske baze podataka napravljene za rad na jednom računalu, NoSQL sustavi napravljeni su tako da se više njih raspoređuje na veći broj računala gdje svaki radi sa svojom skupinom podataka. Time se značajno ubrzava prihvat i obrada podataka.

Iako postoji nekoliko vrsta NoSQL baza podataka, u ovom radu opisane su četiri takve tehnologije: ključ/vrijednost, graf, dokument i stupčane baze podataka. Ključ-vrijednost sustavi spadaju u one jednostavnije. Takvi sustavi podatke smještaju u obliku parova, gdje je prvi član ključ, a drugi vrijednost koja je pridružena tom ključu. Ovakve baze podataka imaju odlične performanse, ali nisu praktične za kompleksne modele podataka. Graf-sustavi bazirani su na modelu grafa i manje se koriste, ali su izuzetno korisne u radu s podacima koji su jako povezani. Ovakve se baze podataka sastoje od čvorova i veza među njima, te omogućuju efikasno prolaženje kroz čvorove slijedeći njihove veze. Dokument-sustavi rade s podacima koji se sastoje od jedinstvenog identifikatora i vrijednosti, što se skupa naziva dokumentom. Vrijednost, međutim, može sadržavati i drugi dokument, tako da ovakvi sustavi podržavaju ugnježđene vrijednosti, što omogućava veliku fleksibilnost strukturiranja podataka. Sustavi bazirani na stupcima (stupčane baze podataka) podatke koji se nalaze u istom stupcu (ako ih promatramo kroz dvodimenzionalnu tablicu) fizički smještaju zajedno, što omogućava dobre performanse za određene vrste pretraživanja.

## Rest?

Nastao kao rezultat rada na doktoratu: Roy Fielding: “Architectural Styles and the Design of Network-based Software Architectures”, University of California, Irvine, 2000 λ Predstavlja pokušaj da se web i njegovi principi iskoriste na smislen način. λ Na primer: “The application by selecting links, such as /user/tom, and operations such as GET or DELETE (state transitions), resulting in the next resource (representing the next state of the application) being transferred to the user for their use.”

Representational state transfer (REST) or RESTful Web services are one way of providing interoperability between computer systems on the Internet. λ Ovaj model se zasniva na prenosu prikaza stanja resursa, gde resurs može biti bilo koji smisleni, adresibilni koncept λ "Web resources" were first defined on the World Wide Web as documents or files identified by their URLs, but today they have a much more generic and abstract definition encompassing every thing or entity that can be identified, named, addressed or handled, in any way whatsoever, on the Web.

pojam resursa: svaki entitet na webu je resurs. Npr. web sajt, HTML strana, XML fajl, web servis, fizčki uređaj, ...

λ adresa resursa: svaki resurs je identifikovan svojim URIjem nad resursima se obavljaju jednostavne operacije

λ GET - čitanje ;POST - kreiranje; PUT – ažuriranje; DELETE – brisanje. λ In a RESTful Web service, requests made to a resource's URI will elicit a response that may be in XML, HTML, JSON or some other defined format. The response may confirm that some alteration has been made to the stored resource, and it may provide hypertext links to other related resources or collections of resources.

Stateless je jedna od osnovnih osobina REST servisa.

λ Stateless označava da nema čuvanja sesije na serveru.

λ Svaki zahtev od klijenta je nezavisan.

λ Zahtev sadrži sve potrebne informacije, a sesija ako je to potrebno može se čuvati i na klijentu.

λ Stanje (sesija) se može transportovati na server kako bi mogla da se obavi autentifikacija.

klijent se obraća resursu putem URI-ja

λ dobija reprezentaciju resursa

λ ta reprezentacija pomera klijenta u novo stanje

λ klijent se zatim obraća drugom resursu, itd.

λ seoba klijenta iz stanja u stanje = transfer

λ Servis koji se kreće u granicama REST principa naziva se RESTful servisom. So REST is the architecture and RESTful is an adjective.

Nećemo ljude da teramo da se koristi/prave neki novi protokol protokol, ili da smišljaju smišljaju novi programski programski jezik. Već je zadatak definisanje dizajn šablona koji opisuje kako bi web trebalo da radi tako da predstavlja okvir za razne web standarde i dizajn web servisa

λ REST nije standard - W3C ga neće propisati kao standard

λ REST stimuliše upotrebu standarda: HTTP, URL, XML/HTML/GIF/JPEG/ XML/HTML/GIF/JPEG/itd. (formati formati za reprezentaciju reprezentaciju resursa)

Bazira se na HTTP protokolu i njegovim metodama.

## Kolekcije - Mape – HashMap?

Kolekcija (*Collection*)je objekat koji sadrži grupu objekata koji se tretiraju kao jedna celina.

Primeri: skup, lista, asocijativna mapa itd.

Kolekcije mogu da sadrže objekte bilo kog tipa koji se mogu dodavati, preuzimati, uklanjati, sortirati itd.

**Java arhitektura za rad sa kolekcijama Collections Framework**

Objedinjuje sve komponente programskog jezika Java za rad sa kolekcijama.

Sadrži tri značajne celine:

Interfejsi

Definišu zajedničko ponašanje (funkcionalnost) za sve kolekcije.

Implementacije

Konkretne klase koje implementiraju interfejse i predstavljaju strukture podataka (lista, skup, asocijativna mapa itd.)

Operacije

Metode koje pružaju mogućnost manipulisanja kolekcijama.





Osnovne operacije za rad sa kolekcijama:

* + - Provera da li je kolekcija prazna
    - Provera da li se objekat nalazi u kolekciji
    - Uzimanje objekta iz kolekcije
    - Dodavanje objekta u kolekciju
    - Uklanjanje objekta iz kolekcije
    - Iteriranje kroz kolekciju
  + Svaka od operacija ima odgovarajući metod koji je implementira za svaku od tipova kolekcija.

**Osobine kolekcija:**

* *Uređena*
  + Elementi su uređeni i pristupa im se u specifičnom redosledu (npr. redosled kojim su dodati ili pozicije na koje su dodati)
* *Sortirana*
  + Elementi su sortirani i pristupa im se u redosledu po kome su sortirani (npr. abecedno)
* *Indeksirana*
  + Elementima se pristupa pomoću indeksa (npr. za indeks 4 možemo dobiti četvrti element po redu)
* *Jedinstveni elementi*
  + Kolekcija ne dozvoljava duplikate elemenata

**Iterator:**

* Iterator je objekat koji se koristi da se označi pozicija u kolekciji i pomoću koga se može pomerati kroz kolekciju od elementa do elementa.

Sintaksa:

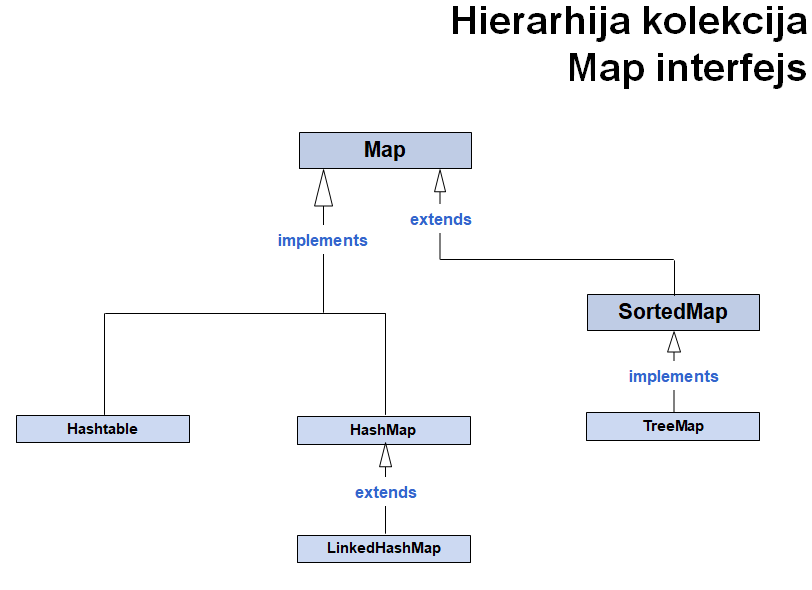
Iterator <*promenljiva*> = <*ObjekatKolekcije*>.iterator();

Primer:

ArrayList myList = new ArrayList();

Iterator myIterator = myList.iterator();

Iterator interface ima metode kojima se može proći kroz kolekciju, proveriti da li postoji sledeći element na koji bi se moglo iterirati i ukloniti tekući element iz kolekcije.



**Lista:**

Dozvoljeni su duplikati elemenata.

Operacije vezane za poziciju elementa su dozvoljene.

Korisnik može ali ne mora da koristi iterator da bi pristupao elementima kolekcije.

Korisnik može da pristupa elementima za datu poziciju na osnovu indeksa, kao i da dodaje elemente na poziciju koju želi.

**Mapa:**

Kolekcija ključ-vrednost parova.

Objekat bilo koje klase može da bude ključ ili vrednost.

Ključevi moraju biti jedinstveni, duplikati nisu dozvoljeni.

Vrste mapa:

* TreeMap
* HashMap
* LinkedHashMap

**HashMap:**

* Implementira Map interfejs.
* Dozvoljava null elent.
* Nema garancija da će bilo kakav redosled iteriranja elemenata biti održan.

**TreeMap:**

* Implementira SortedMap.
* Ključevi se čuvaju u rastućem redosledu po rezultatu compareTo() ili compare() metoda.

**LinkedHashMap:**

* Nasleđuje HashMap.
* Implementirana je kao dvostruko spregunta lista.
* Očuvan je redosled dodavanje ključeva.

**Hashtable:**

* Slična HashMap
* Razlika je u tome što su kod klase Hashtable metode sinhronizovane.
* Ova je stara klasa čija se upotreba više ne preporučuje.
* Korisiti ConcurrrentHashMap ako je potrebna sinhornizacija.

**Ključne informacije:**

* *Collections Framework* sadrži:
  + 1. Interfejse
    2. Implementacije
    3. Operacije
* Za Listu je važan indeks.
* Za Skup je važna jedinstvenost elemenata, duplikati nisu dozvoljeni.
* Za Mapu je važna jedinstvenost ključeva.

Za ostalo pogledaj prezentaciju iz UOP-a. Prezentacija:05\_03\_Kolekcije

[05\_03\_Kolekcije.ppt](../Documents/UOP/06.Kolekcije_WebCD_DVD_Prodavnica/05_03_Kolekcije.ppt)

## Spring?

OSA PREZENTACIJE. Osa1.pdf I pusti takodje I snimak sa tim prezentacijama. Spring krece od 18. Slajda.

[osa1.pdf](../Documents/Osnove%20softverskih%20arhitektura/Predavanja/osa1.pdf)

## Dependency injection?

Poznato i pod nazivom Dependency Injection

Spring kontejner upravlja ºivotnim ciklusom objekata

Programer pi²e programski kod u kojem samo koristi objekte

Prezentacija: osa4.pdf, pogledaj I video vezan za te prezentacije

[osa4.pdf](../Documents/Osnove%20softverskih%20arhitektura/Predavanja/osa4.pdf)

## Spring bean?

Obično se navodi iznad metode koja je unutar klase koja je anotirana sa @Configuration

Razdvaja deklaraciju i definiciju

Kreira se jedan bean, a ne koliko kontejner odluci

Pomocu ove anotacije nije moguce napraviti REST servis - za to se koristi @Controller

Prezentacija: osa3.pdf, pogledaj I video vezan za tu prezentaciju!

[osa3.pdf](../Documents/Osnove%20softverskih%20arhitektura/Predavanja/osa3.pdf)

## Enkapsulacija?

**Enkapsulacija** je još jedan od osnovnih principa objektno orijentisanog programiranja. Enkapsulacija podrazumeva skrivanje podataka od spoljnih pristupa. Primena enkapsulacije je izuzetno bitna, pogotovo kod velikih projekata. Poštovanjem pravila enkapsulacije obezbeđujemo da objekti imaju strogo kontrolisane ulaze i izlaze, a samim tim smanjujemo mogućnost greške, logičke nedoslednosti ili grešaka u programu.  
  
Skrivanje podataka obezbeđujemo upotrebom ključne reči **private**. Na taj način niko van te klasea ne može neplanirano da promeni podatak. Svako pristupanje podacima vrši se isključivo preko javnih metoda kreiranih za te potrebe a to su metode - *getXXX()* i *setXXX()*.

Metoda getXXX() služi za dohvatanje podataka, tj. ona vraća vrdnost neke promenljive, dok metoda setXXX() služi za postavljanje neke nove vrednosti promenljive. Metode "geteri" moraju vraćaju isti tip podatka koji je i promenljiva čiji su oni "geter". Metode koje setuju podatke, "seteri", moraju da primaju podatak istog tipa kao što je i podatak koji setuju.  
Obezbeđujući samo "seter" ili "geter" metod, možemo učiniti klasu read-only ili write-only. Na taj način obezbeđujemo kontrolu nad podacima.  
Primer:

UOP prezentacije. Prezentacija: [03\_01\_uvod\_u\_objektno.pptx](../Documents/UOP/03.Uvod_u_objektno/03_01_uvod_u_objektno.pptx)

## Observer pattern?

THE OBSERVER PATTERN (bihevijoralni) Obrazac koji omogucava nadgledanje objekata: imamo objekat koji dozvoljava nadgledanje ("observable" objekat) i objekte koji ga nadgledaju ("observers") tako sto se registruju za pracenje i bivaju obavesteni o svim izmenama na nivou objekta kada se one dese.

Uzorak promatrača koristi se kada postoji odnos jedan-prema-više između objekata, na primjer, ako je jedan objekt modificiran, njegovi ovisni objekti moraju se automatski prijaviti. Uzorak promatrača spada u kategoriju obrazaca ponašanja.

Implementacija

Uzorak promatrača koristi tri klase glumaca. Predmet, promatrač i klijent. Subjekt je objekt koji ima metode za pričvršćivanje i odvajanje promatrača na klijentski objekt. Stvorili smo apstraktnu klasu Observer i konkretnu klasu Subject koja proširuje klasu Observer.

ObserverPatternDemo, naša demo klasa, koristit će objekt Predmet i objekt klase kako bi prikazao obrazac promatrača na djelu.

## Lambda izrazi?

Jednostavan način rada sa metodom objekta automatski generisane

anonimne lokalne klase koja implementira interfejs sa samo 1 metodom

Sličnost sa globalnom funkcijom

nema (eksplicitno) definisanu klasu kojoj pripada

Razlike u odnosu na globalnu funkciju

postoji automatski generisana klasa "omotača funkcije i njen objekat

- funkcija, odnosno njena klasa je lokalna

• Doprinose konciznoj notaciji i eleganciji programa

• Često se metod svede na izračunavanje izraza - odatle naziv "izraz"

naziv "izraz ne odražava prirodu, u pitanju je funkcija,

an ne izraz

Na mestu definisanja - ne izračunava se izraz

stvara se objekat anonimne klase koja implementira interfejs sa 1 metodom

**Definisanje lambda izraza:**

• Opšti oblik definicije lambda izraza:

(parametri) -> {telo)

parametri - uobičajena lista tipova i imena

tipovi mogu da se izostave, ako se o njima može zaključiti iz konteksta

moraju se ili navesti svi tipovi ili izostaviti svi tipovi

ako u listi postoji samo 1 parametar bez tipa - zagrade ) mogu da se izostave

• zaprade() su obavezne ako nema parametara

telo- uobičajeno telo funkcije (blok koji koristi parametre)

• ako samo jedna naredba (return), mogu da se izostave zagrade i red return

• tada se telo svodi na izraz koji izračunava vrednost lambda izraza

tip lambda izraza (rezultata metoda) - ne navodi se u definiciji izraza

• io njemu se zaključuje iz konteksta, kao i za pare metre

• tip izraza u return naredbi mora bli kompatibilan sa tipom lambda izraza

• tip može biti i void

**Primer definicije:**

(int k) -> {return k\*k; }

k -> k\* k; // skraćeni oblik

(double a, double b) => [return a+b; }

(a,b) -> a+b;

(int x, y) -> x+y // ! Greška

n-> {int s=0; for (int i=0; i<n; St-i++); return s;}

**Ciljni tip i cilj lambda izraza:**

• Cilj (target) lambda izraza - referenca na lambda objekat

• Lambda objekat - objekat anonimne klase omotača lambda izraza

• Lambda klasa - automatski definisana, ostvaruje funkcijski interfejs

• Funkcijski interfejs - interfejs sa 1 apstraktnim metodom

može imati anotaciju: FunctionalInterface

• Funkcijski interfejs predstavlja tip cilja lambda izraza (ciljni tip)

treba ga razlikovati od tipa rezultata lambda izraza

• Zaključivanje o tipu parametara lambda izraza i tipu rezultata

na osnovu tipa parametara i tipa rezultata metoda ciljnog tipa

tip rezultata lambda izraza mora da bude kompatibilan sa tipom metoda

**Definisanje cilja lambda izraza:**

• U definiciji cilja (reference na lambda objekat)

koristi se ciljni tip (funkcijski interfejs)

Primer:

GFunctionalInterface

interface FI{ int fifint x); }

interface FD double fd (double x); }

class A

void m()

FI ciljFi-x->\*\*; // int fi(int x) [return x\*x;

**Izračunavanje lambda izraza**:

U naredbi u kojoj se izračunava lambda izraz  
- funkciji se pristupa preko cilja (reference) i imena metoda interfejsa  
• Primer (nastavak prethodnog):

int [i-cilisi.fi](http://i-cilisi.fi/) (3);  
FD ciljd = x -> Math.exp(-0.1\*\*) \*Math.sin(x);  
double d=ciljed. Id (2.0);  
ciljd - X > X\* X;  
d=ciljd. fd (2.0);  
ciljed = (int x) -> XX; //! Greška

**Pristup podacima iz lambda izraza:**

• Iz lambda izraza je dozvoljen pristup podacima

kao kod lokalnih klasa

Primer:

@FunctionalInterface

interface FI int fi(int x); }

public class A

static int sa=1; int a=2;

void m(int p) {

int 1-3;

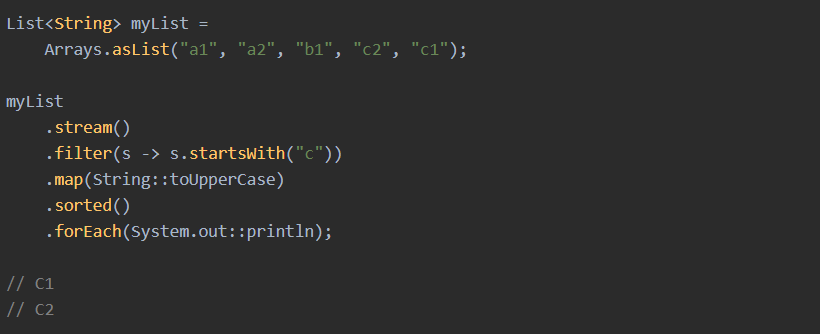
FI ciljei=x->x=1+p+a+sa;

System.out.println("i -" + cilji.fi10));

}

public static void main(String alll new AC).m4); }

## Stream API?

Tok predstavlja niz elemenata i podržava različite vrste operacija za izvođenje izračunavanja na tim elementima: 

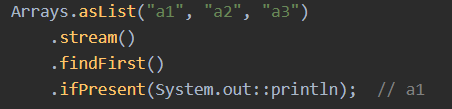
Stream operacije su ili intermedijarne ili terminalne. Intermedijarne operacije vraćaju tok tako da možemo povezati više posrednih operacija bez korištenja zareza. Terminalske operacije su ili nevažeće ili vraćaju rezultat koji nije u toku. U gornjem primjeru filter, map i sort su posredne operacije dok je forEach terminalna operacija. Potpuni popis svih dostupnih operacija toka pogledajte Stream Javadoc. Takav lanac operacija toka, kao što se vidi u gornjem primjeru, poznat je i kao operativni cjevovod.

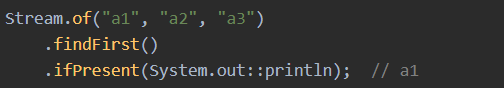
Većina operacija toka prihvaća neku vrstu parametra lambda izraza, funkcionalno sučelje koje specificira tačno ponašanje operacije. Većina tih operacija mora biti i bez ometanja i bez državljanstva. Sta to znaci?

Funkcija ne ometa kada ne modificira osnovni izvor podataka toka, npr. u gornjem primjeru nijedan lambda izraz ne mijenja myList dodavanjem ili uklanjanjem elemenata iz kolekcije.

Funkcija je bez stanja kada je izvršenje operacije determinističko, npr. u gornjem primjeru nijedan lambda izraz ne ovisi o bilo kojoj promjenjivoj varijabli ili stanju iz vanjskog opsega koji bi se mogli promijeniti tijekom izvođenja.

**Različite vrste tokova #**

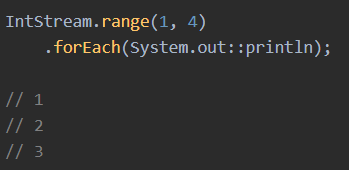
Potoci se mogu kreirati iz različitih izvora podataka, posebno iz zbirki. Popisi i skupovi podržavaju nove metode stream () i parallelStream () kako bi kreirali sekvencijalni ili paralelni tok. Paralelni tokovi mogu raditi na više niti i bit će obrađeni u kasnijem odjeljku ovog vodiča. Za sada se fokusiramo na sekvencijalne tokove:

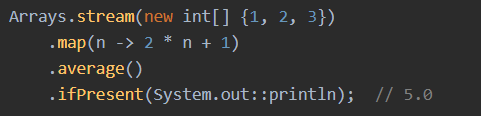
Pozivanje toka metode () na listi objekata vraća redoviti tok objekta. Ali ne moramo stvarati kolekcije da bismo mogli raditi s streamovima kao što vidimo u sljedećem uzorku koda:

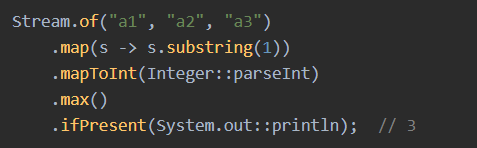
Samo upotrijebite Stream.of () da kreirate tok od gomile referenci na objekt.

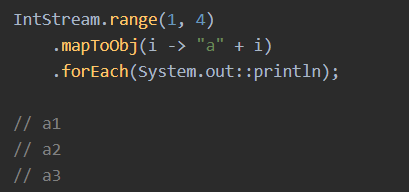
Pored redovnih tokova objekata, Java 8 isporučuje i posebne vrste tokova za rad s primitivnim tipovima podataka int, long i double. Kao što ste mogli pretpostaviti, to su IntStream, LongStream i DoubleStream.

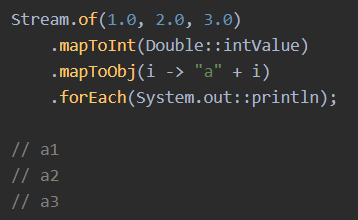
IntStreams može zamijeniti uobičajeni for-loop koristeći IntStream.range ():



Svi ti primitivni tokovi rade baš kao i redovni tokovi objekata sa sljedećim razlikama: Primitivni tokovi koriste specijalizirane lambda izraze, npr. IntFunction umjesto Function ili IntPredicate umjesto Predicate. A primitivni tokovi podržavaju dodatne terminalne agregatne operacije sum () i prosjek ():

Ponekad je korisno transformirati redovni tok objekta u primitivni tok ili obrnuto. U tu svrhu potoci objekata podržavaju posebne operacije mapiranja mapToInt (), mapToLong () i mapToDouble:

Primitivni tokovi se mogu transformirati u objektne tokove putem mapToObj ():

Evo kombiniranog primjera: tok dvojnika prvo se preslikava u int tok, a zatim se mapira u objektni tok nizova:

**Pogledaj vise na ovom linku:** [**https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/**](https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/)

## Ključna riječ static?

Definiše statičke atribute i metode

**class StaticTest {**

**int a = 1;  
 static int i = 47;  
 static void metoda() { i++; }  
}**

vezuju se za klasu, a ne za objekat klase

Vrednost atibuta se ne čuva u objektima, već se skladišti u *Field Data* prostoru koji se nalazi u *Class Data* (prostor namenjen za skladištenje metapodataka i informacija za klasu) koji pripada prostoru *Method Area* koji pripada delu memorije *Metaspace.*

Statički atributi i metode postoje i bez kreiranje objekta zato im se treba pristupiti preko imena klase

**StaticTest.i++;**

Statički atributi imaju istu vrednost u svim objektima

Ako promenim statički atribut u jednom objektu, on će se promeniti i kod svih ostalih objekata

* Namena statičkih metoda:
  + pristup i rad sa statičkim atributima
  + opšte metode za koje nije potrebno da se kreira objekat
* Primeri upotrebe:

**// out je staticki atribut**

**System.out;**

Math.PI;

**// ovo ostalo su staticke metode**

**Math.random();**

**Math.sin();**

**public static void main(String[] args) {...}**

* Kada pozovemo klasu Hello

**java Hello**

* Poziva se njena funkcija main, ovo se u stvari dešava kada pokrenemo program

**Hello.main(args)**

* Statički blok se izvršava samo jednom, prilikom prvog korišćenja klase
* Unutar statičkog bloka može se pristupati samo statičkim atributima i mogu se pozivati samo statičke metode

**class Test {**

**static int a;**

**static int b;**

**int c;**

**static void f() {**

**b = 6;**

**}**

**static {**

**a = 5;**

**// c = 1; //zabranjeno**

**f();**

**}**

**}**

## Overloading vs overriding?

**Method overloading**:

U klasi može da postoji više metoda sa istim imenom

Razlikuju se po parametrima (broju i/ili tipu ulaznih parametara)

Metode se nikada ne razlikuju po povratnoj vrednosti

* U klasi može da postoji više metoda sa istim imenom (method overloading)
  1. razlikuju se po parametrima (broju i/ili tipu ulaznih parametara)

**Method overriding:**

Redefinisanje metoda je pojava da u klasi naslednici postoji metoda istog imena i parametara kao i u baznoj klasi

Cilj je definisati/izmeniti/proširiti funkcionalnost metode roditeljske klase

Redefinisane metode mogu se anotirati u kodu Anotacija @Override

Primer:

klasa A ima metode metoda1() i metoda2()

klasa B nasleđuje klasu A i takođe ima metode metoda1() i metoda2(), ali samo metoda1() je redefinisana

## Izuzeci šta su, kakvi postoje, šta znači da su neki unchecked?

Prezentacija UOP, [05\_01\_izuzetci ukratko.pptx](../Documents/UOP/05.Izuzeci_IO/05_01_izuzetci%20ukratko.pptx)

## MVC pattern, kako funkcioniše?

Pdf OWP, [02\_MVC.pdf](../Documents/OWP/05%20-%20MVC%20EL%20JSTL/02_MVC.pdf)

## JDBC, šta ide gdje u try catch blok?

Java ima definisan standardni interfejs za komunikaciju sa bazama podataka nazvan JDBC. (Java DataBase Connectivity)

• JDBC definiše skup klasa i interfejsa koji se koriste za pristup bazama podataka. Za komunikaciju sa serverima najčešde se koristi TCP/IP mrežna konekcija.

• Razvio Sun Microsystems kao uniforman skup interfejsa za pristup heterogenim relacionim bazama podataka.

Pisanjem Java koda se zadaju SQL naredbe ka bazi podataka posredstvom JDBC API.

• Same SQL naredbe mogu biti napisane ručno i ugnježdene u Java kod.

• JDBC API se koristi za povezivanje argumenata poziva sa parametrima SQL upita, započinjanje izvršavanja upita, navigaciju kroz rezultujudi skup

JDBC API - namenjen aplikativnim programerima, koji definiše komunikaciju na relaciji aplikacija – JDBC menadžer

• JDBC drajver API, interfejs nižeg nivoa namenjen programerima drajvera, koji definiše način komunikacije na relaciji JDBC menadžer –drajver konkretnog DBMS

[..\Documents\OWP\06 - Servleti i Baze Podataka\Servleti i baze podataka.pdf](../Documents/OWP/06%20-%20Servleti%20i%20Baze%20Podataka/Servleti%20i%20baze%20podataka.pdf)