DIZAJN I RAZVOJ SOFTVERA

Uvod: Od strukturnog ka objektnom

Principi OO programiranja

Uvod u Javu

Osnovni podaci - osoblje

Predavač: **dr Stefan Vujović**

kabinet: **Laboratorija za obradu signala**

mail: **stefanv@ucg.ac.me**

konsultacije: **svakim danom, po dogovoru**

Saradnik: **MSc Slavko Kovačević**

kabinet: **Laboratorija za obradu signala**

mail: **skovacevic@ucg.ac.me**

konsultacije: **svakim danom, po dogovoru**

Osnovni podaci - bodovanje

- Bodovanje
  - Kolokvijum **50** poena
  - Ispit **50** poena
  - Izlaskom na popravni kolokvijum/ispit, poništavaju se prethodno osvojeni bodovi i računaju se oni osvojeni na popravnom kolokvijumu/ispitu

- Ocjene se formiraju po pravilu:

$50 \leq E < 60$
$60 \leq D < 70$
...
$90 \leq A \leq 100$

Literatura

- Osnovna literatura:
  - Prezentacije sa predavanja
  - Vježbe (https://github.com/slavzilla/drs)
- Dodatna literatura:
  - Java: How to Program (9th Edition) – *P. Deitel, H. M. Deitel*
  - Java The Complete Reference, 8th Edition – *Herbert Schildt*
- Internet:
  - http://docs.oracle.com/javase/10/docs/api/

Svaka inovacija postojećeg ili kreiranje novog programskog jezika nastaje iz potrebe da se riješi neki fundamentalni problem koji nije rješiv u postojećim jezicima.

Od strukturnog ka OO programiranju

- Pojava jezika C je iz osnova promijenila pristup programiranju.
- Jezik C je nastao iz potrebe za strukturnim, efikasnim jezikom visokog nivoa koji je u stanju da zamijeni programiranje u asembleru pri sistemskom programiranju.
- Prije C-a, programeri su obično birali jezik namijenjen određenoj aplikaciji: FORTRAN ima upotrebu u naučnim aplikacijama, ali nije dobar za sistemsko programiranje, BASIC je bio jednostavan za naučiti, ali nije bio previše moćan. Programiranje u asembleru daje efikasne programe, ali je asembler složen za rad.
- Prvi programski jezici, uključujući Basic, Cobol i FORTRAN, nisu projektovani na osnovu strukturnih principa. Upotreba naredbe **GOTO** bila je vrlo česta, što je značajno narušavalo preglednost programskog koda.
- Sa druge strane, strukturni jezici kao što je Pascal nisu bili pretjerano efikasni i nisu imali široku upotrebu.

Od strukturnog ka OO programiranju

- Početkom 70. godina XX vijeka, računarska revolucija se uveliko zahuktala i potražnja za softverom je prevazilazila mogućnosti programera da ga isporuče. Sa druge strane, računarski hardver se toliko raširio da je dostigao kritičnu masu. Po prvi put su programeri dobili naizgled neograničen pristup svojim mašinama, što je donijelo slobodu u eksperimentisanju i omogućilo programerima da počnu da prave svoje alatke. Dakle, sve je vapilo za novim jezikom koji bi predstavljao kvalitativan skok u aktuelnom programiranju.
- Programski jezik C je stvorio Dennis Ritchie. C je nastao iz jezika B, koga je stvorio Ken Thompson, a ovaj je nastao iz jezika BCPL, koga je stvorio Martin Richards. Tokom više godina, standard jezika C je bio onaj koji se koristio na UNIX-u, i koji je opisan u knjizi *The C programming language* (Prentice Hall, 1978), čiji su autori Brian Kernighan i Denis Ritchie. Jezik C je zvanično standardizovan od strane ANSI instituta.

Od strukturnog ka OO programiranju

- Smatra se da nastanak C-a predstavlja početak modernog doba u programskim jezicima. To nije samo programski, već i programerski jezik. Za razliku od jezika prije pojave C-a, koji su obično nastajali kao rezultat akademskih eksperimenata, C su zamislili, kreirali i razvili programeri, pa zato on odražava programerski pristup programiranju. C su stvorili ljudi koji su ga stvarno i koristili. Time je ovaj jezik vrlo brzo pronašao svoje mjesto među programerima.
- Početkom 80. godina, programeri su široj javnosti otkrili mogućnosti računara, pa su zahtjevi korisnika računara postajali sve veći, a samim tim je i rastao obim posla povjeravan računarima. Broj i veličina projekata na kojima su programeri radili su rasli i strukturno programiranje više nije moglo da izađe u susret svim zahtjevima.

Od strukturnog ka OO programiranju

- Pisanje, testiranje i ispravljanje programa je postao mukotrpan posao zbog čestih izmjena programa. Često se dešavalo da je za male zahtijevane funkcionalne promjene potrebno izvršiti znatne promjene u kodu nekog modula ili njegovo ponovno pisanje. Zatim, veze modula sa ostalim modulima programa su bile veoma složene, pa je promjena jednog modula zahtijevala značajno mijenjanje čitavog programa. Problem je predstavljalo i testiranje novog sistema i otklanjanje grešaka koje bi se pojavile kao posljedica mijenjanja nekih modula.
- Odgovor na softversku krizu predstavljalo je OO programiranje. Iako se kao koncept pojavilo znatno prije softverske krize, sa njenom pojavom uočena je neophodnost ovog koncepta. Naime, američka komunikaciona kompanija AT&T (AT&T - American Telephone and Telegraph) je trebala da početkom osamdesetih realizuje određeni projekat.

Od strukturnog ka OO programiranju

- Realizacija ovog projekta iziskivala je značajne prepravke postojećih programa. Bjarne Stroustrup je shvatio da korišćenje strukturnog programiranja nije baš najbolje rješenje za ovaj problem. Stoga je 1979. kreirao OO verziju programskog jezika C, koji je prvobitno nazvan *C sa klasama*, a 1983. je preinačen u C++. Prvi C++ kompajler se pojavio 1986, a standardizovan je 1997. Ovo je danas vrlo popularan programski jezik.
- C++ proširuje C dodajući mu OO osobine. Pošto se temelji na C-u, C++ nasljeđuje sve njegove osobine i prednosti, što i jeste razlog uspjeha ovog jezika.
- C++ nije stvoren kao potpuno nov programski jezik, već sa ciljem da se poboljša jezik koji se pokazao izuzetno uspješnim.

Nastanak Jave

- OO programiranje je, krajem osamdesetih i početkom devedesetih, uzelo maha, i činilo se da je pronađen savršen programski jezik koji će izaći u susret svakom programerskom izazovu. Ipak, kao mnogo puta u prošlosti, javila se potreba da se računarski jezici pomaknu za još jedan stepen u svojoj evoluciji. U to doba, Internet već poprima svoj sadašnji oblik, što će začeti novu revoluciju na polju programiranja.
- Javu je prvobitno, 1991. godine, koncipirao James Gosling sa svojim kolegama iz korporacije Sun Microsystems, Inc. Prvo ime jezika bilo je Oak, koje je 1995. godine preinačeno u Java. Iznenađujuće, ali glavni pokretač razvoja Jave nije bio Internet, već potreba za jezikom nezavisnim od računarske platforme, koji bi se mogao koristiti za programiranje različitih elektronskih uređaja u domaćinstvu (mikrotalasne pećnice i slično).

Nastanak Jave

- Kao kontroleri ovih uređaja koriste se različiti procesori, pa je upotreba jezika C i C++ za njihovo programiranje ograničena, jer su ovi jezici, kao i većina drugih, usmjereni na određeni procesor. Iako se program napisan na C++ jeziku može prevesti za bilo koji procesor, za to je potreban kompajler namijenjen konkretnom procesoru. Kompajleri su skupi i sporo se prave, što implicira potrebu za lakšim i jeftinijim rešenjem. Java je rođena upravo iz pokušaja Goslinga i saradnika da kreiraju prenosivi jezik koji ne zavisi od platforme i čiji se kôd može izvršavati na različitim procesorima i u različitim okruženjima.
- U to doba, pojavljuje se ključni faktor za razvoj Jave, World Wide Web (WWW). Da se WWW pojavio u neko drugo vrijeme, pretpostavka je da bi se Java koristila samo za programiranje kućne elektronike. Međutim, sa pojavom WWW-a, Java se probija u prvi plan računarskih jezika, jer su za WWW bili potrebni prenosivi programi.

Nastanak Jave

- WWW i Internet su povezali mnoštvo različitih računara i operativnih sistema, pa je potreba za prenosivim programima postala urgentna. Isti problem, zbog koga je Java prvobitno smišljena se pojavio i na Internetu, samo u mnogo većem obimu. Ovime se fokus primjene Jave, sa kućne elektronike, premješta na programiranje za Internet.
- Dakle, iako je početni motiv razvoja Jave bilo kreiranje jezika nezavisnog od platforme, za njegov uspjeh na visokom nivou najzaslužniji je Internet.
- Većinu svojih osobina Java duguje jezicima C i C++. Autori Jave su znali da će ova dva široko rasprostranjena jezika uticati na brzo prihvatanje Jave. Sa C i C++, Java ima zajedničke one osobine koje su ova dva jezika učinile ekstremno popularnim. Javu su oblikovali, testirali i unaprijeđivali profesionalni programeri. To je jezik utemeljen na potrebama i iskustvu osoba koje su ga stvorile, zbog čega on predstavlja jezik programera.

Nastanak Jave. Bajt kôd

- Sličnost između Jave i jezika C++ navodi na pogrešnu pretpostavku da Java predstavlja C++ verziju za Internet. Iako je Java nastala pod uticajem jezika C++, ona ne predstavlja poboljšanu verziju tog jezika, već je napravljena da riješi drugačiji skup problema od onih kojima je namijenjen C++. Oba ova jezika će postojati uporedo još mnogo godina.
- Java obezbjeđuje bezbjednost i prenosivost tako što Java prevodilac ne generiše izvršni kod već tzv. **bajt kôd** (eng. bytecode), koji predstavlja optimizovan skup instrukcija kojeg u trenutku izvršavanja interpretira Javin izvršni sistem poznatiji kao **Javina virtuelna mašina** (Java virtual machine - JVM).
- Dakle, JVM je interpreter bajt kôda.

Bajt kôd

- Prevođenje Java programa u bajt kôd omogućava lakše izvršavanje u različitim okruženjima, jer je za različite platforme potrebno napraviti samo različite virtuelne mašine.
- Iako se Javine virtuelne mašine razlikuju na različitim platformama, sve one razumiju isti Javin bajt kôd, tj. mogu da izvrše svaki Javin program. Kada bi se Java programi direktno prevodili u izvršni kôd, onda bi morale postojati različite verzije programa za svaki procesor priključen na Internet, što nas vraća na početni problem prenosivosti. Izvršavanje bajt kôda pomoću JVM-a predstavlja najjednostavniji način pravljenja prenosivih programa.
- Činjenica da JVM, a ne neposredno procesor, izvršava Java programe čini te programe bezbjednijim. Svojim djelovanjem, tj. upravljanjem izvršenja programa u potpunosti, JVM sprečava da program koji se izvršava utiče na okolni sistem.

Bajt kôd

- Po pravilu, programi prevedeni u izvršni oblik se brže izvršavaju od programa koji se interpretiraju.
- U Javi, ova razlika u vremenu izvršavanja nije velika. Razlog tome je činjenica da je bajt kôd u velikoj mjeri optimizovan.
- Iako je Java izvorno namijenjena interpretiranju, tehnički ne postoji prepreka da se, radi bržeg izvršavanja, bajt kôd prevede u izvršni kôd. Zato je firma Sun ubrzo posle objavljivanja Jave ponudila svoj JIT (Just In Time) prevodilac pod imenom HotSpot.
- Kada JVM uključuje JIT prevodilac, tokom izvršavanja se određeni djelovi bajt kôda prevode u izvršni kôd. Ipak, nije moguće cio Java program odmah prevesti u izvršni kôd, zbog različitih provjera koje Java izvodi tokom izvršavanja programa.
- Zato Java prevodi kôd po potrebi, tokom izvršavanja, i to samo one djelove koji će bitno ubrzati izvršavanje. Preostali dio kôda se i dalje interpretira. Ovaj način „prevođenja po potrebi“ pruža bolje performanse izvršavanja, dok se bezbjednost i prenosivost programa ne smanjuju, jer izvršavanjem i dalje upravlja JVM.

Osobine Jave

- Osobine Jave:
  - **Jednostavnost** Java je koncipirana tako da može lako da se nauči i efikasno koristi. Uz određeno iskustvo u programiranju i poznavanje osnova OO programiranja, učenje Jave je lak zadatak.
  - **Objektna orijentisanost** Iako je bila pod velikim uticajem svojih prethodnika, Java kôd nije dizajniran da bude kompatibilan sa bilo kojim drugim jezikom. Java je slobodno pozajmljivala principe iz mnogih postojećih objektnih softverskih okruženja. Objektni model Jave je jednostavan i lako se proširuje, dok prosti tipovi nisu realizovani kao objekti radi boljih performansi.
  - **Robustnost** U cilju povećanja robustnosti, Java forsira programera da greške ispravlja u ranoj fazi. Sa druge strane, pošto je strogo tipiziran jezik, Java provjerava kôd u trenutku njegovog prevođenja, ali i tokom izvršavanja. Java vrši automatsko dealociranje memorije (eng. *garbage collection*), što predstavlja izvor suptilnih fatalnih grešaka u C i C++. Java obezbjeđuje OO obradu izuzetaka.

Osobine Jave

- **Nezavisnost od platforme** Slogan dizajnera Jave bio je „Napiši jednom; izvršavaj bilo gdje, bilo kad i zauvijek“, što je dobrim dijelom i postignuto.
- **Interpretiranost i visoka efikasnost** Bajt kôd se može izvršavati na svakom računaru koji ima JVM. U kombinaciji sa JIT prevodiocem, djelovi bajt kôda se prevode u mašinski kod, čime se postižu bolje performanse. Pošto JVM upravlja procesom izvršavanja, ovakvo izvršavanje ne umanjuje nezavisnost kôda od platforme.
- **Višenitnost** Java podržava višenitno programiranje. JVM ima elegantno i potpuno rješenje za sinhronizovanje više procesa.
- **Distribuiranost** Distribuirano izvršavanje podrazumijeva nekoliko računara na mreži koji rade zajedno. Java ima integrisanu mogućnost rada u mrežnom okruženju. Pristupanje Internet adresi se u osnovi ne razlikuje od pristupanja fajlu.
- **Dinamičnost** Tokom izvršenja Java programa, vrši se stalna provjera tipova podataka na osnovu čega se razrješavaju pristupi objektima. Na ovaj način je omogućeno pouzdano dinamičko povezivanje kôda, što je suštinski bitno u okruženju gdje se fragmenti bajt kôda dinamički ažuriraju tokom rada programa.

Naredbe i podaci

- Svi računarski programi sastoje se od dva elementa, **naredbi** i **podataka**.
- Dvije koncepcije razvoja programa su organizacija programa oko onoga „ko radi“ i oko onoga „sa čime se radi“.
- Prvi način predstavlja procesno orijentisan model po kome se program definiše kao sekvenca naredbi koje rade sa podacima. Ovaj model uspješno koriste proceduralni jezici kakav je C. Ne predstavlja dobro rješenje kada program postane veoma obiman.
- OO programiranje predstavlja drugi pristup koji je nastao kao odgovor na teškoće nastale usložnjavanjem programa. Ovim pristupom se program organizuje oko podataka (objekata) i skupa dobro definisanih načina pristupa tim podacima.

Apstrakcija

- Apstrakcija predstavlja vrlo bitan element OO programiranja.
- Ljudi se sa složenim situacijama bore apstrahovanjem. Na primjer, niko ne tretira automobil kao skup velikog broja pojedinačnih djelova, već ga poima kao objekat koji se ponaša na jasno definisan način. Ovakvo apstrahovanje omogućava da koristimo automobil bez znanja o tome kako on radi.
- Hijerarhijska klasifikacija nam omogućava da složene sisteme razdvojimo na slojeve, tj. cjeline kojima se lakše upravlja. Na primjeru automobila, gledajući spolja, automobil predstavlja jedinstven objekat. Iznutra, automobil se sastoji iz više podsistema: za upravljanje, kočenje, signalizaciju, ozvučenje itd.
- Svaki od podsistema se sastoji od djelova koji su još više specijalizovani. Na primjer, ozvučenje se sastoji od radio uređaja, CD plejera i zvučnika.

Apstrakcija

- Svaki od ovih podobjekata karakteriše sopstveno jedinstveno ponašanje i ti se objekti mogu tretirati kao cjeline koje reaguju na poruke kojima im se saopštava da nešto urade. Na primjer, kada vozač pritisne kočnicu, on **šalje poruku** kočionom sistemu da koči.
- Koncept reagovanja objekata na poruke koje im se šalju predstavlja suštinu OO programiranja!
- OO programiranje predstavlja moćan i prirodan uzor za pisanje programa koji doživljavaju neizbežne promjene.
- Kada su objekti dobro definisani, a pristup njima čist i pouzdan, odbacivanje ili zamjena pojedinačnih djelova sistema ne predstavljaju problem.

Tri principa OOP - Enkapsulacija

- Osnovni mehanizmi za ostvarivanje OO modela programiranja su: **enkapsulacija**, **nasljeđivanje** i **polimorfizam**.
- Enkapsulacija (eng. *encapsulation*) predstavlja mehanizam koji povezuje naredbe i podatke sa kojima te naredbe rade, štiteći i jedne i druge od spoljnih uplitanja i zloupotreba.
- Pristupanje enkapsuliranim podacima i naredbama strogo se kontroliše pomoću dobro definisanih standarda.
- Ilustrujmo ovaj koncept na primjeru automobila i ponašanju automatskog mjenjača. Automatski mjenjač enkapsulira veliki broj podataka o motoru uključujući koliko ste pritisnuli pedalu gasa, nagib puta itd.
- Mi kao korisnici na ovu složenu strukturu možemo da utičemo samo na jedan način – pomjeranjem ručice mjenjača. Štaviše, ono što se događa unutar mjenjača ne utiče na objekte izvan njega. Na primjer, mijenjanje brzina ne utiče na ozvučenje ili farove automobila.
- Iako postoji veliki broj različitih realizacija automatskih mjenjača, sa gledišta vozača svi oni rade jednako.

Tri principa OOP - Enkapsulacija

- Ista ideja može da se primijeni i na programiranje.
- Moć enkapsuliranog objekta je u tome da svako zna kako može da mu pristupi i koristi bez obzira na princip rada.
- **Klasa** predstavlja osnovu enkapsulacije u Javi.
- Klasa definiše strukturu i ponašanje (podatke i naredbe) zajedničke za sve objekte te klase.
- Objekti klase se nazivaju **primjercima** ili **instancama** klase.
- Klasa predstavlja logičku konstrukciju, a objekat fizičku realnost.
- Podaci i naredbe koji čine klasu se nazivaju **članovima klase** (eng. *class members*).
- Podaci koji su dio klase se nazivaju **podaci članovi** (eng. *data members*) ili **članovi promjenljive** (eng. *member variables*).

Tri principa OOP - Enkapsulacija

- Procedure koje rade sa podacima klase se nazivaju **članovi metode** (eng. *member methods*) ili samo **metode** (u C i C++ metode se nazivaju funkcijama).
- Metode definišu način korišćenja podataka klase, u smislu dozvoljenih operacija koje se mogu vršiti nad njima.
- Metode i podaci klase mogu da budu **privatni** ili **javni**.
- Javni (eng. *public*) dio klase predstavlja ono što spoljni korisnici klase treba ili mogu da znaju.
- Privatnim (eng. *private*) metodama i podacima može da pristupi samo kôd klase.
- Pristup privatnim članovima klase se može izvršiti samo pomoću javnih metoda, koje jasno definišu šta se može uraditi sa podacima klase, na taj način obezbjeđujući se od nedozvoljenih akcija.
- Dobro napisana javna metoda je ona koja otkriva tačno onoliko privatnosti koliko treba, ništa manje i ništa više od toga.

Tri principa OOP - Nasljeđivanje

- **Nasljeđivanje** (eng. *inheritance*) je proces kojim jedan objekat dobija svojstva drugoga, uz mogućnost da ta svojstva unaprijedi.
- Nasljeđivanje je važno jer podržava koncept hijerarhijske klasifikacije (odozgo nadolje). Na primjer, pudlica je dio klase pas, koja predstavlja dio klase sisar, koja je dioo veće klase životinja.
- Bez postojanja hijerarhije, za svaki objekat bi se morale eksplicitno zadati sve njegove karakteristike. Sa druge strane, nasljeđivanje omogućava da za objekat definišemo samo one karakteristike koje ga čine jedinstvenim.
- Klase iz kojih se kreiraju nove klase nasljeđivanja se nazivaju **natklase** ili **superklase** (eng. *superclass*), dok se izvedene klase nazivaju **potklasama** (eng. *subclass*).
- Nasljeđivanje je povezano i sa enkapsulacijom - ako klasa enkapsulira neke osobine, onda će svaka potklasa imati te osobine, kao i dodatne osobine po kojima se ona izdvaja.
- Ovo je ključna koncepcija koja omogućava da složenost OO programa raste linearno, a ne geometrijski.

Tri principa OOP - Polimorfizam

- **Polimorfizam** (eng. *polymorphism*) je riječ grčkog porijekla i znači mnogo oblika.
- U OO programiranju, polimorfizam predstavlja osobinu koja omogućava da se jedan način pristupa koristi za opštu klasu akcija. Specifičnost akcije biće određena u zavisnosti od konkretne prirode situacije.
- Posmatrajmo primjer niza. Algoritmi koji obavljaju standardne operacije sa nizom su isti, samo se mijenja tip elemenata. Na primjer, traženje maksimuma niza se radi na potpuno isti način za nizove cijlih i realnih brojeva.
- U jezicima koji nisu objektno orijentisani, za svaki tip niza je potrebno pisati posebnu proceduru, pri čemu se imena procedura moraju razlikovati. Zahvaljujući polimorfizmu, u Javi možemo definisati opšte procedure za obradu nizova koje će imati ista imena.

Tri principa OOP - Polimorfizam

- Generalno govoreći, polimorfizam omogućava pravljenje opšteg načina pristupa za rad sa grupom srodnih aktivnosti. To doprinosi smanjenju složenosti, omogućavajući da se istim načinom pristupa zada opšta klasa akcija.
- Prevodilac je taj koji bira specifičnu akciju u zavisnosti od konkretne situacije, dok programer treba samo da zna opšti način pristupa.
- Kada se primijene na odgovarajući način, enkapsulacija, nasljeđivanje i polimorfizam stvaraju programsko okruženje koje podržava razvoj mnogo robustnijih i prilagodljivijih programa nego što to omogućava procesno-orijentisan model.
- Dobro dizajnirana hijerarhija klasa predstavlja osnovu za ponovno korišćenje koda u čiji je razvoj i provjeravanje već uloženo vreme i trud. Enkapsulacija omogućava da se tokom vremena mijenja unutrašnjost klasa ne remeteći programski kôd koji koristi te klase. Polimorfizam omogućava pravljenje čistog, smislenog, čitljivog i elastičnog koda.

Prvi Java program

```
public class PrviPrimjer {

  /* Program koji ilustruje štampanje na ekranu
     pomoću metoda print, println i printf. */

  public static void main(String[] args) {
    System.out.print("Započeli smo ");// print štampa tekst i ostaje u istom redu
    // println štampa liniju teksta i prelazi u novi red
    System.out.println("sa Java programiranjem,");
    // printf štampa formatirani tekst. Ima istu sintaksu kao u jeziku C.
    System.out.printf("i dobićemo ocjenu %c ako se budemo %s :-)",'A',"trudili");
  }
}
```

// - jednolinijski komentari

/* */ - višelinijski komentari

/** */ - Javadoc komentari. Služe za ubacivanje programske dokumentacije direktno u program. javadoc program čita ove komentare i koristi ih za pripremu programske dokumentacije u HTML formatu.

Deklaracija klase

- Svaki Java program sadrži bar jednu klasu.
- Ključna riječ **class** služi za deklaraciju klase, i odmah nakon nje dolazi ime klase.
- Po Java konvenciji, ime klase počinje velikim slovom, pri čemu su ostala slova mala. Izuzetak je kada ime klase sadrži više riječi, kada je prvo slovo svake riječi veliko, kao u slučaju naše klase `PrviPrimjer`. Ovakav zapis se još naziva i CamelCase ili Upper CamelCase.
- Ime klase može sadržati slova (velika i mala), cifre, podvlaku (_) i znak dolar ($), pri čemu ne može započeti cifrom.
- Java je case sensitive.
- `public` klase moraju biti smještene u fajlu koji ima isto ime kao klasa i ekstenziju .java. U suprotnom, dolazi do greške pri kompajliranju. Tako se klasa `PrviPrimjer` mora naći u fajlu PrviPrimjer.java.
- Tijelo klase je oivičeno velikim zagradama `{}`.

Deklaracija klase

- Svaka Java aplikacija mora imati metodu `main`, koja predstavlja polaznu tačku aplikacije.
- Kao kod klasa, tijelo metode je oivičeno velikim zagradama.
- Java aplikacija može sadržati desetine klasa, od kojih jedna mora da ima metodu `main` od koje počinje izvršavanje.
- Kod Java apleta (programi ugrađeni u Internet pretraživače), se ne koristi `main` iz razloga što pretraživači koriste različite načine za pokretanje apleta.

Metode print, println i printf

- `System.out` je **standardni izlazni objekat** (eng. *standard output object*), i pomoću njega se prikazuje (štampa) tekst u komandnom prozoru.
- Sve tri metode štampaju tekst predstavljen stringom (parametar metode).
- U slučaju metode `print`, tekst se odštampa i kursor ostaje u istom redu, tj. dalje štampanje se vrši u nastavku prethodnog. Za razliku od `print`, naredba `println` štampa tekst i automatski prelazi u novi red.
- Metoda `printf` štampa formatiran tekst, tj. dozvoljava da se tekst formatira (mijenja) u zavisnosti od dodatnih argumenata ove metode. Prvi parametar metode `printf` je tekst koji se formatira i on se sastoji od fiksnog teksta i specifikatora formata. Specifikatore formata ćemo prepoznati po karakteru % unutar stringa.
- Unutar stringova argumenata kod sve tri metode smo mogli koristiti *escape* sekvence (`\n`, `\t`, `\0` itd.).
- Konačno, svaka Java naredba se mora završiti sa tačka-zarezom. Sintaksna greška je ako se tačka-zarez izostavi na kraju naredbe.

Kompajliranje i izvršenje Java aplikacije

- Ilustrovaćemo kompajliranje i izvršavanje Java aplikacije koristeći komandni prozor i odgovarajuće Java alate.
- Ukoliko koristimo neko integrisano razvojno okruženje, kao što su Eclipse ili NetBeans, program se jednostavno pokreće sa palete alatki.
- Prvo se pozicioniramo u direktorijum koji sadrži izvorni kod (u našem slučaju, fajl `PrviPrimjer.java`). Na primjer, ako se fajl `PrviPrimjer.java` nalazi u direktorijumu A:\Java\PrviPrimjer\src, sa komandom

  ```
  cd /d A:\Java\PrviPrimjer\src
  ```

  se pozicioniramo u željeni direktorijum.
- Kompajliranje se vrši pozivom Java kompajlera `javac`, praćeno imenom Java fajla, tj.

  ```
  javac PrviPrimjer.java
  ```

Kompajliranje i izvršenje Java aplikacije

- `javac` naredbom se, u tekućem direktorijumu, kreira fajl `PrviPrimjer.class` koji sadrži bajt kod verziju programa.
- Bajt kôd je nezavisan od platforme i sadrži instrukcije koje izvršava JVM. To nije kôd koji se može direktno izvršiti. Da bi izvršili program, mora se pozvati `java` komanda praćena imenom fajla, ovaj put bez ekstenzije, tj. na sledeći način:

  ```
  java PrviPrimjer
  ```

- Ova komanda pokreće JVM koja učitava `PrviPrimjer.class` fajl i izvršava ga, počev od metode `main`. Ako se navede ekstenzija class u `java` komandi, JVM neće izvršiti program.

Učitavanje podataka

```
import java.util.Scanner;
public class DrugiPrimjer {

  public static void main(String[] args) {
    Scanner unos = new Scanner(System.in);

    int x1, x2, x3;
    double y1, y2, y3;

    System.out.print("Unijeti dva cijela broja: ");
    x1 = unos.nextInt();
    x2 = unos.nextInt();

    System.out.print("Unijeti dva realna broja: ");
    y1 = unos.nextDouble();
    y2 = unos.nextDouble();

    x3 = x1 + x2;
    y3 = y1 * y2;

    System.out.printf("Zbir %d + %d = %d,\na proizvod %.2f * %.2f = %.3f",
                          x1,x2,x3,y1,y2,y3);
  }
}
```

Učitavanje podataka. Deklaracija promjenljivih

- **`import`** deklaracijom naznačavamo da naša aplikacija koristi klasu `java.util.Scanner`.
- Java sadrži ogroman broj predefinisanih klasa koje možemo koristiti pri razvoju aplikacije. Ove klase su grupisane u **pakete** (eng. *packages*), imenovane grupe srodnih klasa, i zajedno se nazivaju *Java class library* ili *Java Application Programming Interface* (Java API).
- Klasa **`Scanner`** se nalazi u paketu **`java.util`**.
- `import` deklaracija se takođe mora završiti tačka-zarezom.
- Deklaracija (navođenje imena i tipa) promjenljivih može da se izvrši bilo gdje u programu.
- Ime promjenljive mora biti ispravan identifikator, dok tip može biti jedan od primitivnih tipova, kakvi su `int`, `float` i `double`, ili referencijskih tipova (eng. *reference types*).

Učitavanje podataka. Deklaracija promjenljivih

- Klasa `Scanner` omogućava učitavanje podataka (brojeva, stringova) iz raznih izvora (tastatura ili fajl).
- Da bi koristili klasu `Scanner`, moramo deklarisati promjenljivu te klase i dodijeliti joj referencu na kreiranu instancu te klase.
- Nova instanca klase se kreira pomoću ključne riječi **`new`**.
- Argument u malim zagradama, **`System.in`**, predstavlja standardni ulazni objekat (eng. *standard input object*) i omogućava čitanje podataka koje unosi korisnik sa standardnog ulaznog uređaja - tastature. Pročitane podatke `Scanner` prevodi u tip podatka koji učitavamo.
- U naredbama `x1 = unos.nextInt()` i `y1 = unos.nextDouble()`, metode `nextInt` i `nextDouble` služe za učitavanje cijelog i realnog broja, respektivno.
- Klasa `System` je dio paketa `java.lang` i ona se ne uvodi `import` deklaracijom na početku programa.
- Podrazumijevano se paket `java.lang` uvodi u svaki Java program.

Deklaracija i inicijalizacija promjenljivih

- Pri imenovanju Java promjenljivih, koristićemo mixedCase zapis, po kome su sva slova mala, osim velikih početnih slova druge i svake naredne riječi u imenu. Ime počinje malim slovom. Ovaj zapis je isti kao CamelCase zapis pri čemu je prvo slovo promjenljive malo (npr. `x`, `brojacRijeci`, `opetNekiBrojac`).
- Imena Java paketa će biti pisana u skladu sa CamelCase zapisom, kao što je slučaj kod klasa (`Klasa`, `MojaKlasa`).
- Pokušaj čitanja vrijednosti neinicijalizovane promjenljive dovodi do greške u kompajliranju!

```
int x1;
System.out.printf("Vrijednost x1 je %d",x1);   Greška!!!
```

Primitivni Java tipovi

- Java sadrži osam primitivnih tipova podataka.
- Radi efikasnosti, primitivni tipovi nisu realizovani kao objekti.

Tip	Veličina u bitima	Opseg vrednosti
`boolean`	Specifičnost JVM na svakoj platformi	`true` ili `false`
`char`	16	'\u0000' to '\uFFFF' (0 to 65535)
`byte`	8	–128 do +127 (-2^7 to $2^7 - 1$)
`short`	16	–32,768 do +32,767 (-2^{15} to $2^{15} - 1$)
`int`	32	–2,147,483,648 do +2,147,483,647 (-2^{31} to $2^{31} - 1$)
`long`	64	–9,223,372,036,854,775,808 do +9,223,372,036,854,775,807 (-2^{63} to $2^{63} - 1$)
`float`	32	*Negativni opseg*: $-3.4028234663852886 \times 10^{38}$ do $-1.4012984632481707 \times 10^{-45}$ *Pozitivni opseg*: $1.4012984632481707 \times 10^{-45}$ do $3.4028234663852886 \times 10^{38}$
`double`	64	*Negativni opseg*: $-1.7976931348623157 \times 10^{308}$ do $-4.94065645841246544 \times 10^{-324}$ *Pozitivni opseg*: $4.94065645841246544 \times 10^{-324}$ do $1.7976931348623157 \times 10^{308}$

Primitivni Java tipovi

- Veličina tipa `int` je fiksna i iznosi 32 bita, za razliku od jezika C i C++ koji dozvoljavaju variranje ove veličine u zavisnosti od okruženja. Razlog za ovo je prenosivost koda.
- Još jedna razlika u odnosu na C i C++ je da Java ne poznaje neoznačene (eng. unsigned), isključivo pozitivne, celobrojne promjenljive.
- Tipovi `float` i `double` se zapisuju u skladu sa IEEE 754 standardom, dok se za `char` promjenljive koristi ISO Unicode skup karaktera.
- Da zaključimo, za razliku od C i C++, primitivni tipovi u Javi su prenosivi na sve platforme koje podržavaju Javu.

Java kao strogo tipiziran jezik

- Program je **tipski siguran** (eng. *type-safe*) ako nije moguće primijeniti operacije na nekompatibilne tipove podataka.
- Jezici se dijele na **strogo tipizirane** i **slabo tipizirane**.
- Strogo tipizirani su oni koji ne dozvoljavaju izraze u kojima se pojavljuju operandi nepredviđenog tipa, tj. zabranjuju formiranje (ili izvršenje) tipski nesigurnih programa.
- Slabo tipizirani dozvoljavaju ovakve izraze, ali samo u situacijama u kojima postoji unaprijed definisan lanac transformacija kojima se data promjenljiva može transformisati u promjenljivu predodređenog tipa.
- Java je strogo tipiziran, a C i C++ su slabo tipizirani jer dozvoljavaju da se zaobiđu tipska pravila.
- Java, kao C i C++, zahtijeva da se pri deklaraciji navede i tip promjenljive.

Java aritmetički operatori

Operator	Operacija
+	Sabiranje
–	Oduzimanje
*	Množenje
/	Dijeljenje
%	Modulo

- Operator dodjele vrijednosti je =
- Izraz tipa
  `2.3 = X`
  nije dozvoljen jer se u konstantu ne može upisati vrijednost.
- Dozvoljeno je `x = y = z = 0`

Java operatori poređenja. Uslovni operator

Operator	Operacija
&&	I
||	ILI
&	boolean I
|	boolean ILI
^	boolean isključivo ILI
!	negacija

- Razlika između dejstva operatora && i &, (kao i || i |) je vrlo suptilna.
- Kod && i || se podizrazi izvršavaju sve dok se ne ustanovi da li je celokupni izraz `true` ili `false`. Ovakav način izvršavanja logičkih izraza se još naziva i **kratkospojno izvršavanje** (eng. *short-circuit evaluation*).
- Kod boolean I i ILI operacija, ne primjenjuje se kratkospojno izvršavanje, već se svi podizrazi uvijek izvrše.

- Uslovni operator (?:) je jedini ternarni operator u Javi. Ima sintaksu:

```
uslov ? izraz1 : izraz2;
```

Ako je uslov tačan, vrijednost izraza je `izraz1`, a u suprotnom `izraz2`.

- Na primjer, `y = (x>3) ? (x+2) : (x-1);` promjenljivoj y dodjeljuje vrijednost `x+2` ako je `x>3`, ili vrijednost `x-1` u suprotnom.

Kombinovani operatori dodjele. ++ i --

- Izraz

  `x = x + 2`

  se može skratiti sa

  `x += 2`

- Postoje i operatori `-=`, `*=`, `/=` i `%=`.

- Izraz

  `x = x + 1`

  se može napisati i kao `x++` ili `++x`.

  postfiksno inkrementiranje (uzmi staru vrijednost x u izrazu, pa uvećaj x)

  prefiksno inkrementiranje (uvećaj x, pa uzmi novu vrijednost x u izrazu)

- Postoji i operator dekrementiranja `--`.

Prvenstvo i asocijativnost operatora

Operator	Opis	Asocijativnost
`++`	unarno postfiksno inkrementiranje	s desna na lijevo
`--`	unarno postfiksno dekrementiranje	
`++`	unarno prefiksno inkrementiranje	s desna na lijevo
`--`	unarno prefiksno inkrementiranje	
`+`	unarni plus (pozitivan predznak)	
`-`	unarni minus (negativan predznak)	
`!`	unarna logička negacija	
`~`	unarni komplement nad bitovima	
`(tip)`	unarna eksplicitna konverzija (cast operator)	
`new`	kreiranje objekata	
`*`	množenje	s lijeva na desno
`/`	dijeljenje	
`%`	ostatak pri dijeljenju (modulo)	
`+`	sabiranje ili nadovezivanje stringova	s lijeva na desno
`-`	oduzimanje	
`<<`	pomjeranje ulevo	s lijeva na desno
`>>`	pomjeranje udesno	
`>>>`	pomjeranje udesno uz dodavanje nula	
`<`	manje od	s lijeva na desno
`<=`	manje od ili jednako	
`>`	veće od	
`>=`	veće od ili jednako	
`instanceof`	poređenje tipova	

Operator	Opis	Asocijativnost
`==`	poređenje jednakosti	s lijeva na desno
`!=`	poređenje nejednakosti	
`&`	I nad bitovima	s lijeva na desno
	boolean logičko I	
`^`	isključivo ILI nad bitovima	s lijeva na desno
	boolean logičko isključivo ILI	
`|`	ILI nad bitovima	s lijeva na desno
	boolean logičko ILI	
`&&`	uslovno I	s lijeva na desno
`||`	uslovno ILI	s lijeva na desno
`?:`	uslovni operator	s desna na lijevo
`=`	dodjela vrijednosti	s desna na lijevo
`+=`	dodjela uz sabiranje	
`-=`	dodjela uz oduzimanje	
`*=`	dodjela uz množenje	
`/=`	dodjela uz dijeljenje	
`%=`	dodjela uz ostatak	
`&=`	dodjela uz I nad bitovima	
`^=`	dodjela uz isključivo ILI nad bitovima	
`|=`	dodjela uz ILI nad bitovima	
`<<=`	dodjela uz pomjeranje ulijevo	
`>>=`	dodjela uz pomjeranje udesno	
`>>>=`	dodjela uz pomjeranje udesno uz dodavanje nula	

Kontrola toka programa

- U Javi postoji niz naredbi kojima se omogućava izvršenje naredbe koja nije naredna u sekvenci, što se naziva **transferom kontrole**.
- Tokom šezdesetih godina prošlog vijeka, postalo je jasno da je neselektivna upotreba transfera kontrole uzrok mnogih problema u razvoju softvera. Glavni krivac je bila `goto` naredba, koja se koristila u većini jezika tog doba.
- Java nema `goto` naredbu. Ipak, ovo je jedna od Javinih rezervisanih ključnih riječi i kao takva se ne može koristiti kao identifikator u programu.
- Bohm i Jacopini su pokazali u svom radu da se programi mogu pisati bez korišćenja `goto` naredbe, pa je ključni izazov programerima bio da izbace `goto` iz upotrebe.
- Sedamdesetih godina XX vijeka većina programera je prihvatila strukturno programiranje, rezultujući u čistijim programima, jednostavnijim za modifikovanje i otklanjanje grešaka.
- Bohm i Jacopini su takođe pokazali da se svi programi mogu napisati koristeći tri kontrolne strukture – **sekvence**, **selekcije** i **ciklusa**.

Sekvenca

- Sekvenca je kao struktura ugrađena u Javu. Ukoliko se drugačije ne naznači, naredbe se izvršavaju jedna za drugom, u redosljedu kako su navedene u sekvenci.
- Sekvenca može sadržati proizvoljno mnogo naredbi.
- Blok predstavlja sekvencu naredbi uokvirenih velikim zagradama:

```
{
  naredba1;
  naredba2;
  ...
  naredbaN;
}
```

- Java doživljava blok kao jednu naredbu.
- Ukoliko blok sadrži samo jednu naredbu, zagrade se ne moraju navoditi.

Selekcija

- Selekcija se u Javi vrši koristeći naredbe **`if`**, **`if...else`** i **`switch`**.
- `if` naredba se koristi kada želimo izvršiti naredbu ili blok ako je zadovoljen određeni uslov. Sintaksa je:

```
if(uslov)
   naredba ili blok;
```

- Ako je uslov zadovoljen, izvršava se `naredba ili blok`. Ako nije, ta se naredba ili blok preskaču i kontrola prelazi na prvu naredbu nakon selekcije.
- Kada grana sadrži samo jednu naredbu, nema potrebe koristiti zagrade.
- Izvršavanje jedne naredbe ako je uslov ispunjen, a u suprotnom druge, se obavlja pomoću `if...else` naredbe na sledeći način:

```
if(uslov)
   naredba1 ili blok naredbi1;
else
   naredba2 ili blok naredbi2;
```

Selekcija

- Postoji i ugnježdena `if...else` naredba sa više uslova:

```
if(uslov1)
   naredba1 ili blok naredbi1;
else if(uslov2)
   naredba2 ili blok naredbi2;
else if(uslov3)
   naredba3 ili blok naredbi3;
...
else
   naredbaN ili blok naredbiN;
```

- Uslovi se provjeravaju redom i za prvi koji je zadovoljen izvršavaju se odgovarajuće naredbe. Ukoliko nijedan od uslova nije zadovoljen, izvršavaju se naredbe u `else` grani.