Visoka tehnička škola Niš

Savremene računarske tehnologije

Predmet: **Objektno orijentisano programiranje - OOP**

Prof. dr Zoran Veličković, dipl. inž. el.

2019/20.

Prof. dr Zoran Veličković, dipl. inž. el.

Objektno orijentisano programiranje - OOP

UVOD: Programski jezici i programske paradigme (1)

Sadržaj

- **Uvod**
  - Predmet izučavanja
  - Predispitne obaveze
  - Način polaganja ispita
  - Literatura
- **Jezici**
  - Govorni
  - Pisani
  - Programski
- **Klasifikacija programskih jezika**
  - Mašinski jezik
  - Asemblerski jezik
  - Viši programski jezici (kompajleriski i interpreterski)
- **Komponente programskih jezika**
  - Sintaksa
  - Semantika
  - Pragmatika
- **Programske paradigme**
  - Imperativna
  - Proceduralna
  - Deklarativna
  - Funkcionalna
  - Logička
  - Objektno orijentisana

Osnovne informacije o predmetu

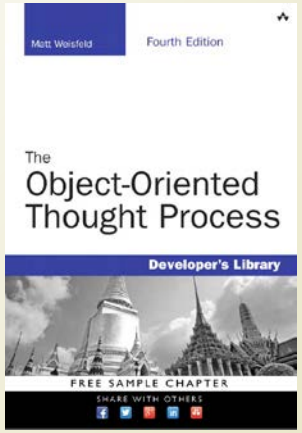
- **KURIKULUM**
  - Predavanja: 2 časa
  - Laboratorijske vežbe: 2 časa.
- Predavanja su **obavezna**!
- Laboratorijske vežbe su **obavezne**!
- Kako do OBAVEZNIH 30 bodova?
  - Predispitne obaveze (30 bodova).
  - Ispitne obaveze (30-70 bodova).
- **Način polaganja ispita**
  - Predispitne obaveze + KOLOKVIJUMI + Ispit = max 100 bodova.
  - max (10 + 20) + (20 + 20) = max 70 bodova u toku semestra.
  - Max 30 bodova na Ispitu.

Izvod iz programa

- **PREMET IZUČAVANJA**
  - Objektno orijentisana programska paradigma.
  - Principi OOP-a.
  - Java OO programski jezik.
  - Alati za razvoj aplikacija u Javi: JDK i Eclipse.
- **TEME KOJE SE OBRAĐUJU**
  - Programske paradigme i programski jezici
  - Principi OOP
  - Java – OO programski jezik
  - Prostor imena, paketi i tipovi podataka
  - Nizovi, stringovi, operatori i upravljacke naredbe
  - Osnove klasa
  - Nasleđivanje klasa
  - Upravljanje pristupom
  - Interfejsi i polimorfizam
  - Pojam i obrada izuzetaka
  - Višenitno programiranje
  - Generički tipovi
  - Grafičko programiranje
  - JWS i Apleti
  - Kolekcije

Literatura

- Z. Veličković, OO programiranje – predavanja, VTŠ Niš, 2020.
- Z. Veličković, Praktikum laboratoriskih vežbi: OO programiranje, Niš, 2018.
- M. Weisfeld, The Object-Oriented Thought Process, Pearson Education, Inc. 2013.
- H. Schildt, Java J2SE 5: kompletan priručnik, Mikro knjiga 2006 (Biblioteka).
- Beleške sa predavanja;
- http://www.vtsnis.edu.rs;
- http://www.oracle.com;
- http://php.com.



Uvod

- U savremenom svetu se termin "**PROGRAMIRANJE**" koristi u najrazličitim kontekstima.
- U svetu računarstva, pod programiranjem se podrazumeva pisanje **NIZA INSTRUKCIJA** kojima se nalaže računaru da izvrši **ODREĐENI ZADATAK**.
- **UREĐENI NIZ RAČUNARSKIH INSTRUKCIJA** kojim se rešava određeni problem se naziva **PROGRAM**.
- Shodno prethodnioj definiciji, osoba koja piše program je **PROGRAMER**.
- Za pisanje programa koristi se **SKUP** dobro definisanih **RAČUNARSKIH INSTRUKCIJA**.
- Ovaj dobro definisani **SKUP**
  - **INSTRUKCIJA** i
  - **PRAVILA**

  se koristi za pisanje programa se naziva **PROGRAMSKI JEZIK**.
- Programer mora detaljno **POZNAVATI** programski jezik koji koristi kako bi efikasno pisao programe - **PROGRAMIRAO**.

Govorni, pisani i programski jezici

- U **SVAKODNEVNOJ KOMUNIKACIJI** čovek sa čovekom komunicira pomoću **GOVORNOG JEZIKA** kao što su npr. engleski, nemački ili srpski.
- Međutim, govorni jezik nije jedini način komunikaciju između ljudi - ljudi mogu komunicirati korišćenjem i **PISANIH JEZIKA -** bez izgovaranja bilo koje reči.
- Da bi se obavila uspešna komunikacija - neophodna je sposobnost obeju strana da **NEDVOSMISLENO RAZUMEJU** ono što je saopšteno sa druge strane.
- Ako jedna osoba zna da govori engleski, a druga zna nemački, mogu li one međusobno komunicirati (**Ne**)?!
- Da bi se ovi sagovornici razumeli neophodno je uključiti u kominikaciju **PREVODIOCE** sa engleskog na njemački jezik (ili obrnuto), dakle, sagovornici će u tom slučaju **MOĆI** da komuniciraju iako se ne razumiju direktno!
- Već znamo, programer može putem pisanog **PROGRAMSKOG JEZIKA** naložiti računaru da obavi neki specijalizovani zadatak.
- Da bi računar razumeo instrukcije napisane u nekom (višem) **PROGRAMSKOM JEZIKU** mora postojati **PREVODILAC** u programski jezik koji računar prepoznaje.

Mašinski jezik

- Računari razumeju instrukcije zapisane samo u tzv. **BINARNOM** formatu.
- **BINARNI FORMAT** zapisivanja računarskih instrukcija predstavlja niz **0** i **1**, i u svetu računara se naziva **MAŠINSKI JEZIK** ili mašinski kod.
- Svaki računar ima svoj **FIKSNI SKUP** osnovnih instrukcija koje podržava, tako da jedan računar može da koristi binarni **kod 0010** kao instrukciju za sabiranje dva broja, dok drugi može koristiti **binarnu sekvencu 0101** za **ISTU SVRHU**!
- Dakle, programi pisani na mašinskom jeziku **ZAVISE OD REALIZOVANOG HARDVERA** - MAŠINE.
- Programi napisani na mašinskim jezikom se **VEOMA TEŠKO**, pišu, čitaju, razumeju i modifikuju.
- Progam za sabiranje brojeva 12 i 15 na mašinskom jeziku može izgledati ovako:

```
0010010010 1001010000100110

0001000100 0101001000100101O
```

Asemblerski jezik

- Za neku drugu **HARDVERSKU PLATFORMU** program sabiranja dva broja bi u binarnom smislu izgledao **SASVIM DRUGAČIJE**.
- Da bi se ovaj posao **POJEDNOSTAVIO**, kreirana je druga **FORMA ZAPISIVANJA** programskog koda koja se naziva **ASEMBLERSKI JEZIK**.
- Asemblerski jezik omogućava korišćenje **OZNAKA** (mnemonika) za pisanje **INSTRUKCIJA** računaru.
- U asembleru je **LAKŠE** pisati, čitati i razumeti instrukcije u odnosu na mašinski jezik.
- Za razliku od binarnih kodova koji se koriste u mašinskom jeziku, asemblerski jezik koristi **MNEMONIKE** (li, add, t1, t0) za **KODIRANJE INSTRUKCIJA**, npr.:

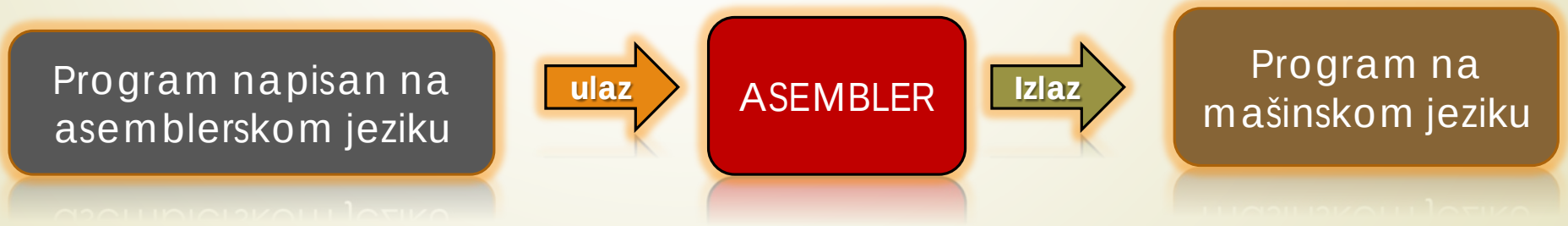
```
li $t1, 15

add $t0, $t1, 12
```

- Sa druge strane, na **VIŠEM PROGRAMSKOM JEZIKU** bi se moglo jednostavno napisati:

```
int x = 15 + 12;
```

Asembler

- Ako se uporede programi napisani na mašinskom i asemblerskom jeziku, evidentno je da je asemblerski jezik **LAKŠI** za **PISANJE**, **ČITANJE** i **RAZUMEVANJE**.
- Treba uočiti da postoji korespondencija **JEDAN NA JEDAN** između instrukcije na **mašinskom jeziku** i **asemblerskog** jezika za datu arhitektura računara.
- Instrukcije napisane na asemblerskom jeziku **MORAJU BITI PREVEDENE** na **MAŠINSKI JEZIK** kako bi ih računar mogao izvršiti.
- Program koji **PREVODI** instrukcije napisane na asemblerskom jeziku u mašinski jezik se naziva **ASEMBLER** (eng. *assembler*):



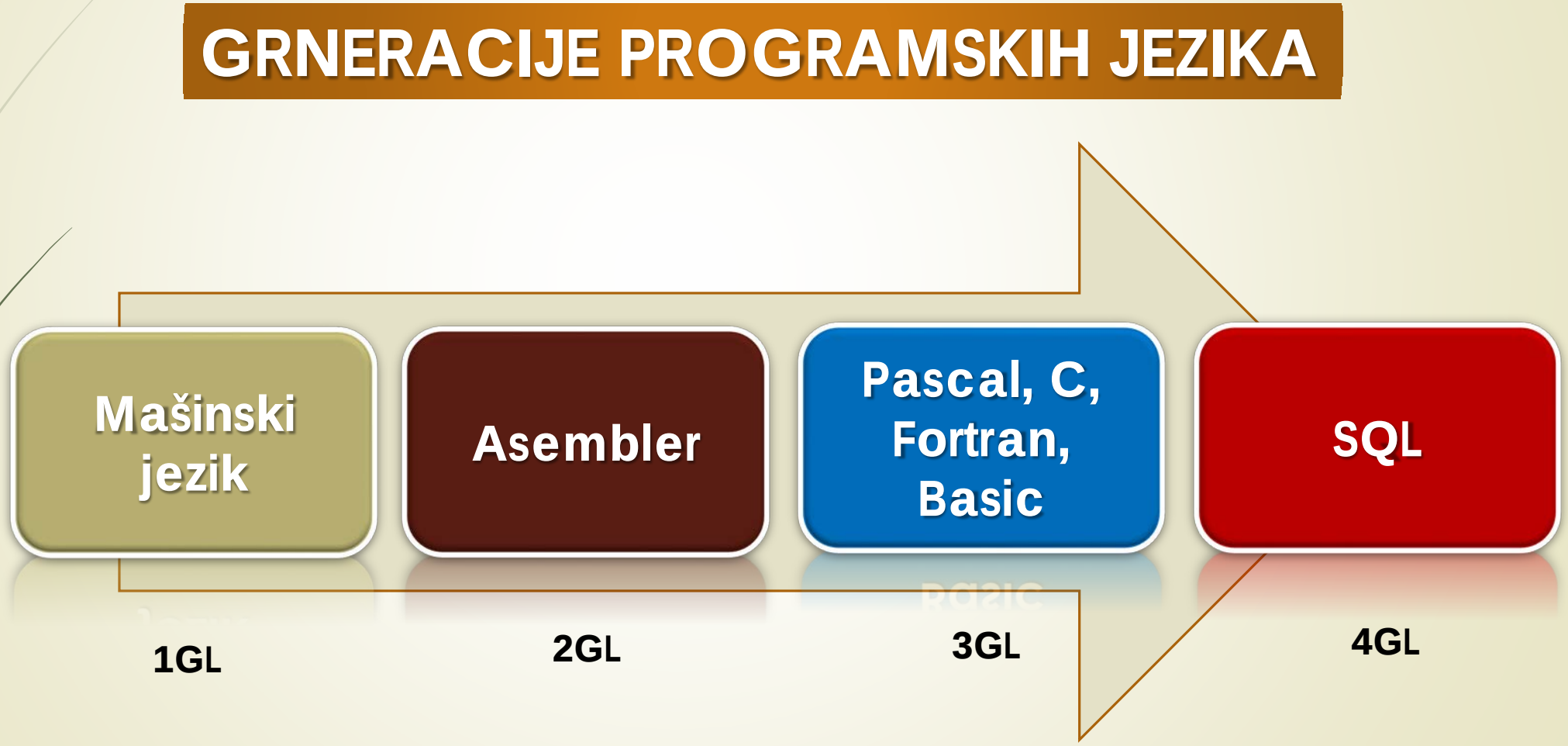
Programski prevodioci: kompajleri i interpreteri

- Ako se prevođenje programa iz višeg programskog jezika u mašinski obavlja u **ISTO VREME KADA I IZVRŠAVANJE**, onda se govori o **INTERPRETERIMA**.
- **INTERPRETER** uzima po jedan programski iskaz iz višeg programskog jezika, **PREVODI GA** u mašinski kod i **IZVRŠAVA** ga.
- Generalno, programski jezici **INTERPRETERSKOG TIPA** se **SPORIJE IZVRŠAVAJU** u odnosu na programske jezike kompajlerskog tipa.
- Programski jezici interpreterskog tipa su recimo **Matlab**, **Python** i **Ruby**.
- Sa druge strane, ako se prevođenje programa iz višeg programskog jezika u mašinski **NE OBAVLJA U VREME IZVRŠENJA**, onda se govori o **KOMPAJLIRIMA**.
- Kompajleri prevode celokupni izvorni kod višeg programskog jezika u mašinski kod i **TEK ONDA SE IZVRŠAVAJU**.
- Programski jezici kompajlerskog tipa su recimo **C**, **C++, C#** i **Java**.

Klasifikacija programskih jezika

- Generalno, programski jezici se mogu klasifikovati na jezike **PRVE** (1GL), **DRUGE** (2GL), **TREĆE** (3GL) i **ČETVRTE** (4GL) generacije.
- Što je nivo programskog jezika **VIŠI**, to je on **BLIŽI** govornom ljudskom jeziku!
- Jezici prve generacije su poznati kao **MAŠINSKI JEZICI** (1GL).
- **ASEMBLERSKI JEZIK** je poznat kao jezik druge generacije (2GL).
- **PROCEDURALNI PROGRAMSKI JEZICI** visokog nivoa kao što su C, C++, Java i C#, zahtevaju kreiranje **ALGORITAMA** uz korišćenje jezičke sintakse i pripadaju trećoj generaciji programskih jezika (3GL).
- **NE-PROCEDURALNI** programski jezici visokog nivoa, **NE ZAHTEVAJU PISANJE ALGORITAMA** za rešavanje programa i poznati su kao programski jezici četvrte generacije (4GL).
  - **STRUKTUIRANI UPITNI JEZIK SQL** (engl. *Structured Query Language*) je najrašireniji 4GL programski jezik i koristi se za komuniciraju sa bazama podataka.

Generacije programskih jezika



Komponente programskih jezika

- Programski jezici se sastoje od sistema **NOTACIJA** (programskih iskaza) koje se koriste za **PISANJE INSTRUKCIJA** za računare.
- Programski jezik se sastoji od **TRI** komponente:
  - SINTAKSE,
  - SEMANTIKE i
  - PRAGMATIKE,
- **SINTAKSA** se koristi za kreiranje **VAŽEĆIH** programskih konstrukcija (iskaza) korišćenjem dostupnih notacija.
- **SEMANTIKA** se bavi **ZNAČENJEM** programskih konstrukcija.
- **PRAGMATIKE** se bavi **PRIMENOM** programskih jezika u praksi.
- Slično kao i pisani jezik, programski jezik ima **REČNIK** i **GRAMATIKU**.

Rečnik, gramatika i semantika

- **REČNIK** programskog jezika sastoji se od skupa **REČI**, **SIMBOLA** i znakova **INTERPUNKCIJE**.
- **GRAMATIKA** programskog jezika definiše **PRAVILA** koja se koristie nad **REČNIKOM** kako bi se kreirao **VAŽEĆI PROGRAMSKI ISKAZ**.
- Na pisanom jeziku se može kreirati **GRAMATIČKI ISPRAVNA REČENICA** ali koja **MOŽDA NEMA SMISLA** - ovakav luksuz u programskim jezicima **NIJE DOZVOLJEN**,.
- Takođe, **DVOSMISLENOSTI** u programskim jezicima **NIJE DOZVOLJENA**!
- **SEMANTIKA** u programskim jezicima objašnjava **ZNAČENJE** sintaktički važećih programskih konstrukcija, odnosno daje odgovore na pitanje "**Šta radi konkretan program**"?
- Dakle, sintaksno ispravna programska konstrukcija **NE MORA** biti semantički ispravna.
- Program mora biti **SINTAKTIČKI** i **SEMANTIČKI ISPRAVAN** pre nego što ga može **IZVRŠITI RAČUNAR**!
- Pomoćni programi integrisani u razvojna okruženja **PROVERAVAJU** ova dva uslova pre izvršavanja.

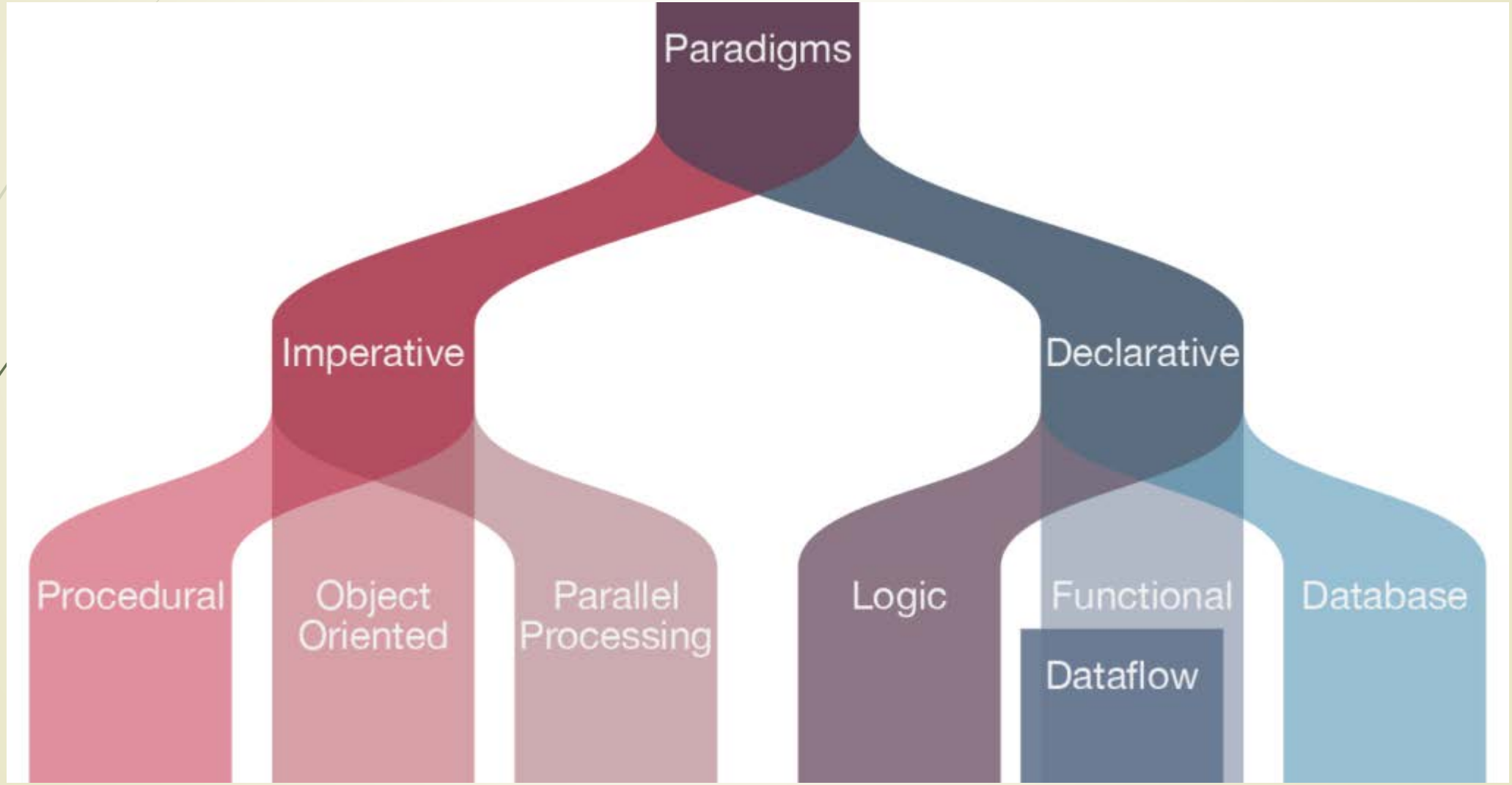
Programske paradigme (1)

- Generalno se pod **PARADIGMOM** podrazumeva **TEORIJA** ili **GRUPA IDEJA** o tome **KAKO** nešto treba uraditi ili napraviti.
- **PROGRAMSKA PARADIGMA** po Robert V. Floidu je **KREIRANJE KONCEPTA** obavljanja nekog izračunanja, odnosno, opis **KAKO** treba **STRUKTURIRATI** i **ORGANIZOVATI** zadatke koji se izvršavaju na računaru.
- Programska paradigma stupa na scenu **PRE** nego što se počne sa **PISANJEM PROGRAMA** pomoću programskih jezika.
- Programske paradigme se koriste u fazi **ANALIZE** kada se koriste određene paradigme za **RASČLANJIVANJE PROBLEMA** i njegovo rešenje na određeni način.
- Pojedini programski jezici pružaju **SREDSTVA ZA PRIMENU** određene paradigme programiranja.
- Programski jezik može biti **POGODAN** za programiranje pomoću jedne programske paradigme, ali **NE** i za drugu!

Programske paradigme (2)

- Prema **Niklaus E. Wirth**-u (1984 god. dobio **Tjuringovu nagradu** za razvoj inovativnih programskih jezika) programski jezici se sastoje samo od **DVE** komponente:
  - **PODATAKA** – zapravo strukture podataka i
  - **ALGORITMA**.
- **PODACI** se koriste **ZA PREDSTAVLJANJE INFORMACIJA**, dok je **ALGORITAM** skup koraka koji se izvršavaju nad podacima kako bi se došlo do do rešenja problema.
- Različite **PROGRAMSKE PARADIGME** uključuju **RAZLIČITE POGLEDE** na rešavanje nekog problema kombinovanjem podataka ali i algoritama.
- U programiranju se poznate sledeće programske paradigme:
  - **Imperativna** paradigma
  - **Proceduralna** paradigma
  - **Deklarativna** paradigma
  - **Funkcionalna** paradigma
  - **Logička** paradigma
  - Objektno orijentisana paradigma - **OOP**

Programske paradigme (3)



Imperativna paradigma

- **IMPERATIVNA PARADIGMA** je poznata i kao **ALGORITAMSKA PARADIGMA**.
- U **IMPERATIVNOJ PARADIGMI**, program se sastoji od **podataka** i **algoritma** koji manipulišu **PODACIMA**, tako da podaci u određenom trenutku definiše **STANJE PROGRAMA**.
- **STANJE PROGRAMA** se **MENJA** kao se **NAREDBE IZVRŠAVAJU** prema određenoj programskoj sekvenci.
- Jezici koji podržavaju **IMPERATIVNO PROGRAMIRANJE** obezbeđuju:
  - **promenljive** (memorijske lokacije),
  - **operacije dodeljivanja** vrednosti promenljivama,
  - **programske konstrukcija** za **kontrolu toka** programa.
- U imperativnom programiranju se kreiraju dobro definisani **KORACI** za rešavanje problema.

Imperativna paradigma - primer

```
int num = 15;                    // num čuva vrednost 15 u ovoj tački programa
int counter = 0;                 // counter čuva 0 u ovoj tački programa
while (counter < 10) {
    num = num + 1;               // Modifikacija podataka u promeljivoj num
    counter = counter + 1;       // Modif. podataka u promenljivoj counter
}
// num čuva vrednost 25 u ovoj tački programa
```

Proceduralna paradigma (1)

- **PROCEDURALNA PARADIGMA** je slična imperativnoj paradigmi sa jednom razlikom: ona kombinuje **VIŠESTRUKE NAREDBE** u jedinici koja se naziva **PROCEDURA**.
- **PROCEDURA** se izvršava kao jedna celina (jedinica).
- Izvršavanje komandi procedura je poznata kao **POZIVANJE** ili pozivanje na proceduru.
- Programski jezkci koji koriste **PROCEDURE** za rešavanje problema se nazivaju **PROCEDURALNI PROGRAMSKI JEZICI**.
- Program na proceduralnom jeziku sastoji se od **PODATAKA** i **NIZA POZIVA NA PROCEDURE** koji manipulišu podacima.
- Primer procedure **`addTen()`** je dat u nastavku.

Proceduralna paradigma - primer

```
void addTen(int num) {
int counter = 0;
    while (counter < 10) {
    num = num + 1;                  // Modifikacija podataka u num
    counter = counter + 1;          // Modifikacija podataka u counter
 }
// num je inkrementiran za 10 u ovoj tački programa
}
------------- Primena procedure ----------------
int x = 15;                     // x čuva 15 u ovoj tački
addTen(x);                     // poziv addTen procedure koja će inkrementovati x za 10
// x čuva vrednost 25 u ovoj tački programa
```

Proceduralna paradigma (2)

- Programski kodovi kod imperativnih i proceduralnih paradigmi su **SLIČNIH STRUKTURA**.
- Korišćenje **PROCEDURA** rezultira **MODULARNIM KODOM** i povećava mogućnost **PONOVNOG KORIŠĆENJA** koda.
- U **PROCEDURALNOJ PARADIGMI**, jedinica programiranja **NIJE NIZ KOMANDI**!
- Umesto toga, sekvencu koda smestite u **PROCEDURU**, tako da se program sastoji od **NIZA PROCEDURA**.
- Nuspojava korišćenja procedura je da one **MODIFIKUJU DEO PODATAKA** prilikom izvršavanja svoje logike.
- Programski jezici koji **PODRŽAVAJU** proceduralnu paradigmu su:
  - **C,**
  - **C++,**
  - **Java.**

Deklerativna paradigma

- Kod **DEKLARATIVNE PARADIGME**, program se sastoji **OD OPISA PROBLEMA**, dok **RAČUNAR PRONALAZI REŠENJE**.
- Program **NE PRECIZIRA** kako doći do rešenja problema, već je to **POSAO RAČUNARA** da dođe do rešenja opisanog problema.
- **FUNKCIONALNA i LOGIČKA** paradigma su **PODTIPOVI** deklarativne paradigme.
- U imperativnoj paradigmi **PODACI SE MODIFIKUJU** kako se program izvršava.
- U deklarativnoj paradigmi, podaci se **DOSTAVLJAJU ALGORITMU** kao ulazni podaci i **OSTAJU NEPROMENJENI** kako se algoritam izvršava.
- Pisanje **UPITA BAZI PODATAKA** pomoću strukturiranog jezika upita (**SQL**) spada u programiranje zasnovano na **DEKLARATIVNAOJ PARADIGMI**.
- Korisnik određuje **PODATKE KOJE ŽELI**, a programi podrške baze određuju **KAKO** će pronaći i dostaviti tražene podatke.

Funkcionalna paradigma

- **FUNKCIONALNA PARADIGMA** je zasnovana na konceptu **MATEMATIČKIH FUNKCIJA**.
- Kod funkcionalne paradigme, za razliku od proceduralne, **NEMA NUSPOJAVA**.
- Dakle, vrednosti su **NEPROMENLJIVE** - nova vrednost se dobija primjenom funkcije na ulaznu vrednost.
- U funkcionalnom programiranju, **PONOVLJENI ZADATAK** se izvodi pomću **REKURZIJE**.
- Funkcija uvek daje **ISTI IZLAZ** kada se primeni na **ISTI ULAZ**.

```
int add(x, n) {
if (n == 0) {
 return x;
} else {
return 1 + add(x, n-1);       // funkcija add se poziva rekurzivno
}
}
```

Logička paradigma

- Za razliku od imperativne paradigme, **LOGIČKA PARADIGMA** se fokusira na „**ŠTA**“ a ne na „**KAKO**“ rešiti.
- **TREBA TAČNO** opisati problem, dakle **ŠTA** treba rešiti, a program će odrediti **NAJBOLJI ALGORITAM** za rešenje.
- U logičkoj paradigmi, program se sastoji od **SKUPA AKSIOMA** i **IZJAVU O CILJU**.
- Izjava o cilju predstavlja **TEOREMU**, a program koristi **ZAKLJUČKE** da bi dokazao teoremu unutar teorije.
- Skup aksioma je skup **ČINJENICA** i **PRAVILA** zaključivanja koji čine teoriju.
- **LOGIČKO PROGRAMIRANJE** koristi matematički koncept **RELACIJA** iz teorije skupova.
- **RELACIJE** u teoriji skupova su definisane **KAO PODSKUP KARTEZIJANSKOG PROIZVODA** dva ili više skupova.

Objektno orijentisana paradigma (1)

- U **OBJEKTNO ORIJENTISANOJ** (OO) paradigmi, program se sastoji od **INTERAKCIONIH OBJEKATA**.
- Objekt kapsulira **PODATKE** i **ALGORITME** koji rade sa tim podacima.
- **PODACI** definišu **STANJE OBJEKTA**, a **ALGORITMI** (metode) definišu **PONAŠANJE OBJEKTA**.
- Objekti između sebe **KOMUNICIRAJU** slanjem poruka.
- Kada objekat **PRIMI PORUKU**, on reaguje izvršavanjem jednog od svojih algoritama, koji mogu **MODIFIKOVATI** njegovo stanje.
- Podaci i algoritmi su **RAZDVOJENI** u imperativnoj i funkcionalnoj paradigmi.
- Kod **OBJEKTNO ORIJENTISANE PARADIGMIE** podaci i algoritmi su **KOMBINOVANI** u **JEDNOM ENTITETU**, koji se naziva **OBJEKT**.

Objektno orijentisana paradigma (2)

- **KLASE** su osnovne jedinice programiranja u objektno orijentisanoj paradigmi.
- Slični objekti su grupisani u jednu definiciju koja se zove **KLASA**.
- Definicija klase se koristi za **KREIRANJE OBJEKTA** - objekt je poznat i kao **INSTANCA** klase.
- Klasa se sastoji od **PROMENLJIVIH** i **METODA** instance.
- **VREDNOSTI PROMENLJIVIH** instance objekta određuju **STANJE OBJEKTA**.
- Različiti objekti klase održavaju **SVOJA STANJA ODVOJENO** - svaki objekat klase ima **SVOJU KOPIJU** promenljive instance.
- Stanje objekta čuva se **PRIVATNO ZA TAJ OBJEKAT** – stanju objekta se **NE MOŽE PRISTUPITI** ili direktno modifikovati izvan objekta.
- **METOD** je kao procedura (ili potprogram) u proceduralnoj paradigmi - metode mogu **PRISTUPITI/MODIFIKOVATI** stanje objekta.