PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET
METROPOLITAN
BEOGRAD

# KI301 - KONSTRUISANJE SOFTVERA

## Pravila kodiranja

Lekcija 04

PRIRUČNIK ZA STUDENTE

# KI301 - KONSTRUISANJE SOFTVERA

## Lekcija 04

## *PRAVILA KODIRANJA*

# L4: PRAVILA KODIRANJA

## *L4: Pravila kodiranja- Sadržaj:*

Sadržaj:

- Upotreba varijabli kod programiranja
- Upotreba tipova podataka
- Upotreba sekvencijalnih instrukcija
- Upotreba If-instrukcije
- Upotreba petlji
- Vežba - Pravila kodiranja

# Poglavlje 1

# Upotreba varijabli kod programiranja

## PREPORUKE ZA INICIJALIZACIJU VARIJABLI

*Postoji čitava lista preporuka za inicijalizaciju varijabli, koje treba primenljivati.*

Greške kod inicijalizacije varijabli su česte u programima. I ovo oduzima puno vremena kod debugging-a. Zato, izbegavanje problema sa inicijalizacijom varijabli je vrlo bitno. Evo tipičnih razloga za greške inicijalizacije varijabli:

- varijabli nije nikad zadata vrednost
- jedan deo objekta je inicijalizovan ali ne ceo (nekoliko članova u objektu nije inicijalizovano)
- varijabli nije alocirana memorija
- varijabla je inicijalizovana, ali ta vrednost više ne važi već treba zadati novu vrednost

Evo preporuka za inicijalizaciju varijabli:

- inicijalizovati varijablu odmah kada je deklarisana, npr. Java i C++
- varijable treba deklarisati i inicijalizovati blizu mesta gde se prvi put koristi, npr. u Javi C++
- treba koristiti konstante kada je to moguće (u Javi ključna reč "final" a u C++ ključna reč "const")
- često imamo tzv. brojače i akumulatore (counters, accumulators), i česta greška je da kod ponovne upotrebe ovih varijabli da se zaboravi njihovo resetovanje (dakle, pojedine varijable treba resetovati kod ponovne upotrebe)
- treba izvršiti proveru da li je potrebna reinicijalizacija neke varijable, npr. kod ponovne upotrebe petlje ili kod ponovne upotrebe funkcije tj. metode
- ako je potrebna reinicijalizacija varijable, onda to uraditi tamo gde se vrši ponavljanje petlje ili ponavljanje pozivanje metode
- varijable deklarisane u nekom podprogramu treba inicijalizovati u tom podprogramu
- klasne varijable treba inicijalizovati u konstruktoru
- ako je memorija klasnih varijabli alocirana u konstruktoru, onda u destruktoru ova memorija treba da se oslobodi
- konstante se inicijalizuju samo jednom
- ako kompajler upozorava da varijabla nije inicijalizovana, treba to uzeti u obzir
- treba vršiti validaciju ulaznih parametara podprograma

# IMENA VARIJABLI

## *Ovde navodimo preporuke za imenovanje varijabli.*

Neki jezici dozvoljavaju upotrebu "implicitnih" varijabli tj. varijabli bez imena, ali treba izbegavati implicitne varijable jer se lakše prave greške sa implicitnim varijablama nego sa eksplicitnim varijablama.

Evo preporuka zaimenovanje varijabli:

- ime varijable treba da tačno i u potpunosti opisuje šta dotična varijabla reprezentuje
- ime varijable po pravilu treba da opisuje problem koji se rešava a ne način rešavanja pomoću konkretnog programskog jezika
- treba optimizirati dužinu imaena varijable da ne bude suviše dugačka ali ne i suviše kratka
- ako ime varijable treba da sadrži reči kao što su Max, Min, Total, Sum itd., onda tu reč (tzv. "modifikator varijable") treba staviti na kraj imena a ne na početak, jer prvo treba naznačiti smisao varijable a tek onda modifikator varijable

Specifični tipovi podataka zaHTEVAJU SPECIFIČNE PREPORUKE:

- Indeksi petlji (loop-index variable) trebaju da imaju imena koja imaju značenje a ne samo "i" ili "j" ili "k"
- tzv. "flag" varijable treba imenovati da imaju značenje a ne samo reč "flag" jer sama reč "flag" ne objašnjava zašta služi ta varijabla, a ove "flag" varijable se obično koriste da opišu status programa
- privremene varijable su varijable koje služe da memorišu međurezultate nekoh proračuna, i obično im se daju imena bez značenja, ali to treba izbegavati, naime i ovim "privremenim" varijablama treba dati ime sa značenjem
- bulove varijable treba imenovati tako da impliciraju vrednost "true" ili "false", npr. "uspeh" ili "gotovo", itd.
- kod enumerated-type varijabli koristiti prefiks, npr. Color_Red, Color_Green,Color_Blue, gde prefiks označava grupu imena, npr. Enum Color
- kod imenovanja imenovanih konstanti, ime imenovane konstante treba da označava značenje te konstante a ne njenu konkretnu vrednost, npr. TEMPERATURE_MAX a ne npr.FIFTY_DEGREES

# KONVENCIJE ZA IMENOVANJE VARIJABLI

## *Ovde navodimo konvencije za imenovanje varijabli.*

Konvencije za imenovanje varijabli mogu biti vrlo korisne. Konvencije za imenovanje varijabli su korisne u sledećim slučajevima:

- ako više programera rade na istom projektu
- ako se planira da neki drugi programer izvrši modifikaciju programa u cilju korektivnog održavanja, što je praktično uvek slučaj

- ako se program pregleda tj. vrši inspekcija od strane drugog programera
- ako je program ogroman

Evo konvencija za imenovanje varijabli:

- razlikovati imena varijabli i imena podprorama, npr. variableName i RoutineName(), dakle imena varijabli i objekata počinju malim slovima a imena podprograma velikim slovima
- razlikovati imena klasa i imena objekata
- identifikovati globalne varijable, npr. sa prefiksom "g", npr. varijabla g_Temperature
- identifikovati klasne varijable ("member variables") sa "m", npr. m_Ime
- identifikovati imenovane konstante, npr. u Javi i C++ se koriste velika slova npr. TEMPERATURE_MAX
- identifikovati enumeracije (enumerated type) pomoću prefiksa "E"
- ako imamo ime varijable sa više reči, onda svaka reč počinje velikim slovom ili praviti razmake sa crtom, npr. room_temperature ili roomTempoerature
- pojedini jezici imaju svoje konvencije, i treba ih koristiti, npr. u C++ su imena varijabli i imena podprograma su kompletno u malim slovima, npr. room_temperature, a u Javi imena varijabli i metoda koriste malo slovo za prvu reč a velika slova za ostale reči, npr.roomTemperature

# PRIMER

## *Pokazni primer .*

U nastavku je dat primer programskog koda u okviru koga su definisani nazivi varijabli:

```
int prviBroj, drugiBroj // Atributi u kojima se cuva prvi broj, drugi broj
    double rezultat; // Atribut u kojem se cuva rezultat
    boolean tacnost; // Atribut u kojem se tacnost iskaza
```

Nazivi varijabli su:

- prviBroj
- drugiBroj

i napisani su po konvenciji. Varijable u svom imenu imaju dve reči i shodno konvenciji potrebno je prvu reč napisati malim slovima a zatim drugu reč spojiti sa prvom i početno slovo druge reči mora biti veliko.

# ZADACI ZA INDIVIDUALNU VEŽBU

## *Zadaci za samostalan rad*

1. Napisati konstantu po definisanoj konvenciji.

2. Napisati globalnu varijablu i predstaviti je u programskom kodu.

3. Definisati varijablu koja sadrži tri reči u nazivu.

# Poglavlje 2

# Upotreba tipova podataka

## UPOTREBA BROJEVA

*Ovde dajemo preporuke za upotrebu brojeva, dakle fundamentalnih tj. "ugrađenih" tipova podataka.*

Za upotrebe brojeva generalno (celih brojeva tj.integers i decimalnih brojeva tj. floating-point numbers) možemo navesti sledeće preporuke:

- treba predvideti deljenje sa nulom
- izbegavati tzv. "magićne brojeve" tj. brojeve koji se pojavljuju u programu a ne zna im se značenje
- kod konverzije jednog tipa u drugi tip koristiti eksplicitnu konverziju da bi bilo očigledno da se vrši konverzija npr. y=z - (float) j
- izbegavati poređenje celih brojeva i decimalnih brojeva, jer to po pravilu ne funkcioniše, npr. if (x>i) then .....

Evo preporuka za upotrebu celih brojeva:

- proveriti deljenje celih brojeva, jer i/j npr. 7/10 nije 0.7 već 0
- kod množenja ili sabiranja celih brojeva može doći do integer overflow greške, npr. maksimalni ceo broj 65535 za unsigned 16-bit cele brojeve
- proveriti integer overflow grešku za međurezultate a ne samo za finalne rezultate, npr. iskaz i*i/1000 ima međurezultat i*i koji treba proveriti

Evo preporuka za upotrebu decimalnih brojeva (floating-point numbers) :

- izbegavati sabiranje ili oduzimanje brojeva koji imaju veoma različite vrednosti, npr. jednog veoma velikog broja i jednog veoma malog broja
- izbegavati poređenje jednakosti decimalnih brojeva, npr. if (x==y)...., gde su x i y decimalni brojevi
- predvideti greške zaokruživanja decimalnih brojeva

## UPOTREBA OSTALIH FUNDAMENTALNIH TIPOVA

*Ovde dajemo preporuke za upotrebu stringova, bulovih varijabli, enumeracija, imenovanih konstanti, i nizova, dakle fundamentalnih tj. "ugrađenih" tipova podataka.*

Evo sugestija za upotrebu stringova (strings) i znakova (characters):

- izbegavati "magične" stringove, tj. stringove čije značenje nije jasno
- izbegavati "magične" znakove

Evo sugestija za upotrebu enumeracija:

- ponekad, enumeracije su zgodna alternativa za bulove varijable, jer bulove varijable imaju dve vrednosti (true ili false), dok enumeracije mogu imatri više od dve vrednoasti, npr. tri vredniosti: Status_Bad, Status_Fair, Status_VeryGood

A evo sugestija za upotrebu bulovih varijabli:

- često je korisno kod kondicionalnih testova, npr. if(.....)...., upotrebiti dodatne bulove varijable da bi testovi bili jasnije ili jednostavnije dokumentovani

Evo preporuka za upotrebu imenovanih konstanti:

- koristiti imenovane konstante umesto "magičnih" brojeva

Konačno, evo preporuka za upotrebu nizova:

- proveriti da se indeks niza kreće unutar dimenzije niza, a ne da prevazilazi dimenziju niza, jer u nekim jezicima biće javljena greška, a u nekim jezicima neće biti javljena greška ali ćete dobiti pogrešan rezultat

# UPOTREBA NEOBIČNIH TIPOVA PODATAKA- "STRUKTURE"

## *Ovde navodimo preporuke za upotrebu tzv. "struktura".*

Termin "structure" označava tip podataka koji obuhvata više različitih tipova podataka. To su user-created tipovi podtaka, , u C i C++ jeziku "structs". U Javi i C++, klase se ponašaju kao strukture ako klasa se sastoji samo od javnih podataka a bez javnih metoda. Ali, klase imaju prednost u odnosu na strukture, jer nude privatnost podataka i funkcionalnost preko metoda tj. funkcija. Ipak, ponekad je upotreba podataka bez odgovarajućih metoda korisno.

Navedimo preporuke za upotrebu struktura:

- koristiti strukture da se pokaže da su neke grupe podataka u međusobnoj vezi
- strukture omogućavaju jednostavnije manipulisanje blokovima podataka koji su u međusobnoj vezi
- liste parametara podprograma se nekad može pojednostaviti upotrebom strukture
- upotreba struktura preko grupisanja povezanih podataka, može da pojednostavi program i samim tim olakša održavanje programa

# UPOTREBA NEOBIČNIH TIPOVA PODATAKA- "GLOBALNI PODACI"

*Ponekad je korisno primeniti globalne podatke, a povesti računa o povećanom riziku. Ipak, treba izbegavati primenu globalnih podataka.*

Globalne varijable su dostupne bilo gde u programu. Za razliku od lokalnih varijabli koje su dostupne lokalno, npr. klasne varijable su dostune samo u okviru klase. Iskustvo govori da upotreba globalnih podataka je riskantnija od upotrebe lokalnih podataka, tj. izaziva više greški. Kod složenih programa, jasno je da upotreba klasa uz minimalniu upotrebu globalnih podataka je cilj, jer je to jedna od pretpostavki o.o. programiranja (data hiding).Osim toga, globalni podaci smanjuju mogućnost od reupotrebe pojedinih delova programa tj. kvare modularnost programa. Ali, ponekad je korisno primeniti globalne podatke, a povesti računa o povećanom riziku.

Evo razloga za upotrebu globalnih podataka:

- neki podaci se koriste u celom programu, i onda je korisno takve podatke koristiti kao globalne podatke
- globalni podaci mogu da pojednostave program tamo gde se neki podaci koriste u većem delu programa ili se često pojavljuju listi parametara čitavog niza podprorama

Navedimo preporuke za upotrebu tzv. globalnih podataka (globalnih varijabli) tj. tzv. global data:

- izbegavati globalne podatke kad god je to moguće, i koristiti ih samo u izuzetnimsituacijama
- primeniti access-routines umesto globalnih podataka ( pri tome organizovati program tako da rasporediti ove podatke od globalnog značaja po klasama i za njih koristiti acces-routines,dakle access-routines i odgovarajuće podatke organizovati po klasama, a ne treba staviti sve ove podatke i access-routines u jednu klasu)
- ako koristite globalne podatke, njihova imena treba da očigledno ukazuju da su to globalni podaci, npr. koristiti prefiks g_
- napraviti listu svih globalnih podataka u programu

# PRIMER

*Pokazni primer.*

Tipovi promenljivih su:

```
int a, b, c;         // Deklariše tri celobrojne vrednosti, a, b, i c.
int a = 10, b = 10;  // Dodeljivanje vrednosti promenljivih
byte B = 22;         // Dodeljuje vrednost promenljivih B.
double pi = 3.14159; // Dodeljuje vrednost u vidu decimalnog broja PI.
boolean a = true;    // Deklariše da je promenljiva a "tačna"
```

Tip promenljive bira programer u odnosu na to koji podatak mu je potreban. Ako će aplikacija koristiti samo cele brojeve ima mogućnost odabira integer tipa podatka, dok se za brojeve sa decimalnim zarezom koristi double tip promenljive.

Primer tipova promenljivih u Java programskom jeziku:

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

# ZADACI ZA SAMOSTALAN RAD

## *Zadaci za individualnu vežbu.*

1. Definisati četiri različita tipa promenljivih po izboru.

2. Predstaviti programskim kodom jednu privatnu i dve javne promenljive.

3. Koja je razlika između privatnih i javnih promenljivih u programskom kodu? Napisati primer u Java programskom jeziku.

# Poglavlje 3

# Upotreba sekvencijalnih instrukcija

## SEKVENCIJALNA ORGANIZACIJA PODPROGRAMA

*Često kod sekvencijalnih instrukcija je bitan redosled instrukcija, npr. instrukcija1 učitava podatke, instrukcija2 vrši proračun na učitanim podacima, a instrukcija3 štampa rezultate prora*

Kod statement-centred tj. proceduralnog programiranja, najednostavniji oblik organizacije u okviru nekog podprograma su sekvencijalne instrukcije, tj. niz instrukcija koje slede jedna za drugom. Npr.

instrukcija1;

instrukcija2;

instrukcija3;

Često kod sekvencijalnih instrukcija je bitan redosled instrukcija, npr. instrukcija1 učitava podatke, instrukcija2 vrši proračun na učitanim podacima, a instrukcija3 štampa rezultate proračuna. Dakle, u ovakvim slučajevima redosled instrukcija se ne može menjati, tj. redosled instrukcija je vrlo važan.

Tu možemo da uvedemo koncept "uzajamne zavisnosti" tj. međuzavisnosti tj. "dependencies". U prethodnom primeru, treća instrukcija zavisi od druge instrukcije a druga instrukcija od prve instrukcije. U prethodnom primeru, uzajamna zavisnost instrukcija je očigledna, medjutim, u mnogim primrrima to nije očigledno, i u složenim programima imamo puno međusobnih veza koje nisu očigledne, nisu toliko eksplicitne već implicitne, pa o tome treba voditi računa. U mnogim primerima, međusobna zavisnost je sakrivena, i ovo važi i za proceduralno programiranje i za o.o.p.. Posebno kod o.o.p., ako npr. u nekom podprogramu imamo pozivanje nekih podprograma, npr. inicijalizacija nekih klasnih varijabli, može biti vrlo bitan redosled instrukcija, jer može biti sakriveno da postoji međuzavisnost između nekih podprograma.

## MERE ZA OTKRIVANJE I POJAŠNJAVANJE MEĐUZAVISNOSTI POJEDINIH PODPROGRAMA

*Ovde navodimo koje mere možemo preduzezi da bismo otkrili tj. pojasnili međusobnu zavisnost pojedinih podprograma.*

Možemo preduzeti određene mere da bi otkrili tj. pojasnili međusobnu zavisnost pojedinih delova programa:

- imena pojedinih podprograma su dobro izabrana tj. imena podprograma tačno opisuju šta podprogram radi, i na osnovu imena podprograma može se otkriti međuzavisnost između pojedinih podprograma
- treba koristiti parametre podprograma da bi se utvrdila međuzavisnost između pojedinih podprorama, naime ako imamo nekoliko podprograma koji operišu sa istim podacima, onda treba ispitati međusobnu zavisnost podprograma
- treba ubaciti komentare u podproram koji ukazuju na međusobnu zavisnost pojedinih podprograma, tako da one međusobne zavisnosti koje nisu očigledne treba naznačiti preko komentara u podprogramu
- ubaciti u program assertions ili error-handling code ili tzv. status variables (tj. flags) da bi se proverila međusobna zavisnost pojedinih podprograma

Npr. ako imamo neki klasni konstruktor, u njemo može biti zadata statusna varijabla da je jednaka false, i kada se pozove konstruktor, onda se pored inicijalizacije klasnih podataka vrši promena statusne varijable da bude true.

# KONCEPT GRUPISANJA POVEZANIH INSTRUKCIJA

*Ovde diskutujemo princip grupisanja instrukcija koje su u međusobnoj vezi. Dakle treba razlikovati instrukcije koje su dependent i koje su related.*

Postoji još jedan koncept, a to je grupisanja instrukcija u blokove i to instrukcija koje su međusobno povezane (related), a ne moraju da budu direktno međuzavisne (dependent). Dakle, ako imamo niz instrukcije koje su u međusobnoj vezi treba ih grupisati u blok. Tj. iz takvog bloka treba izbaciti instrukcije koje nisu u međusobnoj vezi. To je tzv. princip blizine (Principle of Proximity), koji nalaže da instrukcije koje su u međusobnoj povezanosti držimo blizu između sebe i grupišemo ih u blokove, a da instrukcije koje nisu u vezi sa instrukcijama u nekom bloku izbacimo iz tog bloka.

Evo primera grupisanja povezanih (related) instrukcija:

- instrukcije rade sa istim podacima ali nisu u direktnoj međusobnoj zavisnosti
- instrukcije obavljaju slične zadatke
- instrukcije su u međusobnoj vezi zbog direktne međusobne zavisnosti instrukcija (dependencies)

U svakom pdprogramu treba grupisati instrukcije koje su u međusobnoj vezi i formirati blokove povezanih instrukcija. Ako to nije urađeno kako treba, onda imamo preklapanje pojedinih blokova instrukcija, a ako je urađeno kako treba,

- onda imamo niz blokova koji su jedan iza drugog
- ili su ugnežđeni blokovi
- a ako nije urađeno grupisanje povezanih instrukcija kako treba, onda imamo preklapanje blokova instrukcija

Dakle za svaki podprogram,

- treba grupisati instrukcije koje su u međusobnoj vezi
- i treba izvršiti proveru preklapanja blokova instrukcija, pa ako ima preklapanja blokova onda izvršiti pregrupisavanje instrukcija
- Takođe, mogu se uvesti da se formiraju novi podprogrami za pojedine ugneždene blokove

# PRIMER

## *Pokazni primer.*

Sekvencijalno grupisanje vrši se kao u primeru:

```
int a = 5;
int b = 12;
int c = a*a + b + 7;

Primer sekvencijalnog grupisanja
```

U okviru datog primera, svaka linija programa će biti izvršena po definisanom redosledu.

# ZADACI ZA SAMOSTALAN RAD

## *Zadaci za individualnu vežbu.*

1. Prikazati primer sekvencijalnog grupisanja za množenje četiri promenljive.

2. Prikazati preskakanje instrukcija u programskom kodu korišćenjem if else funkcije.

3. Opisati koncept grupisanja povezanih instrukcija na proizvoljnom primeru.

# Poglavlje 4

# Upotreba If-instrukcije

## UPOTREBA KONDICIONALNIH INSTRUKCIJA

*Ovde diskutujemo upotrebu kondicionalnih instrukcije.*

Kondicionalne instrukcije mogu biti tipa:

- if-then
- if-then-else
- if-then-else-if

Kod upotrebe if-then instrukcije treba voditi računa

- da se moninalna putanja napiše prvo, a alternativna putanja posle, a ne obrnuto
- razlikovati > od >=, i razlikovati < od <=
- ako imamo komplikovane test izraze if(.....), može se komplikovani test izraz zameniti odgovarajućom bulovom funkcijom

Što se tiče if-then-else-if instrukcija, treba reći da se često u praksi koriste lančaneif-then-else-if instrukcije, dole je dat primer. Ovo je slično sa case-iskazom tj. case-instrukcijom, ali case-instrukcija je manje fleksibilna od lančane if-then-else-if instrukcije. Naime neki jezici, npr. C++ i Java, samo parcijalno podržavaju case-instrukciju, tj.case-instrukcija ima niz ograničenja pa je potrebno koristiti lančanu if-then-else-if .

```
if (.......){.......................}

else if(.....................) { ................}

else if(..................){...................}

else if(.............){...................}
```

## PRIMER

*Pokazni primer*

Upotreba kondicionalnih instrukcija prikazana je na sledećem primeru:

```
class IfElseDemo {
    public static void main(String[] args) {
```

```
        int brojPoena = 76;
        char ocena;

        if (brojPoena >= 90) {
            ocena = '5';
        } else if (brojPoena >= 80) {
            ocena = '4';
        } else if (brojPoena >= 70) {
            ocena = '3';
        } else if (brojPoena >= 60) {
            ocena = '2';
        } else {
            ocena = '1';
        }
        System.out.println("Ocena = " + ocena);
    }
}
```

Za potrebe primera korišćene su IF i ELSE IF instrukcije. Sam kod se odnosi na kratak program koji će kada se unese broj bodova u promenljivu brojPoena, korisniku na ekranu ispisati ocenu.

# ZADACI ZA SAMOSTALAN RAD

## *Zadaci za individualnu vežbu.*

1. Prikazati upotrebu if kondicionalne instrukcije.

2. Napisati program koji za broj koji je veći od 5 ispisuje "Broj je veći od 5" a za brojeve manje od 5 ispisuje "Broj je manji od 5".

3. Definisati programski kod koji za uneti datum ispisuje u kojoj trećini meseca se nalazi. Koristiti else if funkciju.

# Poglavlje 5

# Upotreba petlji

## GREŠKE KOD PETLJI

*Upotreba petlji je jedan od najsloženijih aspekta programiranja, i tu često nastaju greške.*

U C++ i Java imamo petlje:

- tipa for iwhilesa testom na početku, npr. for(.....){.....} i while(.....){.....}
- i tipa do-while sa testom na kraju petlje
- U jeziku C# imamo foreach-petlju koja izvršava neku operaciju na svakom elemebtu nekog niza ili nekog kontejnera.

Petlje sa izlazom u sredini su

- petlje sa break- instrukcijom (C,, C++, Java)
- igoto-instrukcijom (C, C++)

Kod upotrebe petlji mogu nastati mnoge greške, i to je bitan faktor za kvalitet softvera, npr.

- pogrešna inicijalizacija petlje
- pogrešno ugnežđavanje
- pogrešno završavanje petlje
- itd

Petlje sa mnogo break-instrukcija nisu poželjne jer ukazuju na nejasno razmišljanje o strukturi petlje, već je bolje koristiti niz petlji nego jednu petlju sa mnogo izlaza. Napomenimo da Java omogućujelabeled-break-instrukcije. Preporuka mnogih programera je da treba izbegavati break-instrukciju i continue-instrukciju. I danas traje rasprava da li ove instrukcije pripadaju ili ne tzv. strukturnom programiranju. Ove instrukcije mogu da naprave problem sa kompajlerom, pa ih je bolje izbegavati, ili ako ih koristimo treba biti svestan da možda nisu ispravne.

## PREPORUKE ZA PETLJE

*Ovde navodimo neke korisne preporuke nastale iz iskustva.*

Evo nekih preporuka o dužini petlje:

- dužina petlje treba da je ne više od 15 ili 20 linija, ili izuzetno do jedne strane (50 linija)

- ne treba vršiti ugnežđavanje više od tri nivoa
- kod dugih petlji, deo petlje se može zameniti pozivanjem podprograma
- kod kratkih petlji imamo upotrebu break i continue tj. višestruke izlaze, ali ove petlje se mogu pretvoriti u duže petlje sa jednim izlazom

Treba napomenuti da su petlje i nizovi često u vezi, naime, često se petlja koristi upravo da operiše sa nekim nizom tj. array. Ovo važi za C++ i Javu. Ali u nekim jezicima to nije slučaj, npr. Fortran90 ili APL jezici omogućuje moćne operacije sa nizovima ali bez upotrebe petlji.

Greške kod petlji se mogu klasifikovati:

- izbor vrste petlje
- ulazak u petlju
- inicijalizacija petlje
- dužina petlje
- ugnežđavanje petlji
- indeks petlje
- izlaz iz petlje

# PRIMER

## *Pokazni primer.*

Primer FOR petlje:

```
class ForPokazna {
    public static void main(String[] args){
         for(int i=1; i<11; i++){
              System.out.println("Broj je: " + i);
         }
    }
}

Izlaz:

Broj je:  1
Broj je:  2
Broj je:  3
Broj je:  4
Broj je:  5
Broj je:  6
Broj je:  7
Broj je:  8
Broj je:  9
Broj je:  10
```

Program se sastoji od FOR petlje koja će izvršavati kod koji se nalazi u samoj petlji sve dok je promenljiva **i** manja od broja 11. U ovom primeru ispisaće celobrojnu vrednost promenljive **i** od broja 1 do 10.

# ZADACI ZA SAMOSTALAN RAD

## *Zadaci za individualnu vežbu.*

1. Prikazati nepravilan način upotrebe petlji u programskom kodu.

2. Primenom for petlje napisati program koji ispisuje sve brojeve od 20 a manje od 50.

3. Prikazati upotrebu do while petlje na konkretnom primeru.

# Poglavlje 6

# Vežba - Pravila kodiranja

## PRIMER 1

### *Pokazni primer 1.*

Primer konvencije za imenovanje varijabli:

```
int g_Konstanta;
string m_Ime;
int BROJ_MAX;
```

U kodu su definisane promenljive:

g_Konstanta koja se odnosi na globalnu varijablu, postoji mogućnost prepoznavanja iste zbog prefiksa "g".

m_Ime koja se odnosi na klasnu varijablu, postoji mogućnost prepoznavanja pomoću prefiksa "m".

BROJ_MAX se odnosi na imenovanu konstantu i može se prepoznati po velikim slovima.

## PRIMER 2

### *Pokazni primer 2.*

Tipovi promenljivih u odnosu na to da li su vidljivi u ostalim klasama:

```
private int jbmg;
private int brojTelefona;
private boolean bracniStatus ;

public string ime;
public string prezime;
public int godine;
```

Podela promenljivih se vrši uglavnom na privatne i javne. Privatne promenljive se koriste samo u okviru klase, dok se javne promenljive mogu koristiti u ostalim klasama programa. Na progameru je da definiše tip promenljive shodno potrebama u programu (da li će biti vidljiva samo u klasi kojoj se nalazi ili u svim klasama u okviru programa).

# PRIMER 3

## *Pokazni primer 3.*

Primer nesekevencijalnog grupisanja:

```
int a = 5;
int b = 12;
int c;
if (a > b) {
    c = a;
} else {
    c = b;
}

Primer nesekvencijalnog grupisanja, zato sto ce se jedna instrukcija preskociti.
```

U okviru ovog primera dat je programski kod u kome će biti preskočena jedna od instrukcija ( c = a ili c = b) što znači da je ovakav kod nesekvencijalno grupisan. Da bi ovaj kod bio sekvencijalno grupisan potrebno je onemogućiti preskakanje neke instrukcije u samom kodu.

# PRIMER 4

## *Pokazni primer 4.*

Kao drugi primer za kondicionalne instrukcije, napisan je sledeći programski kod:

```
public class Test {

   public static void main(String args[]) {
      int x = 30;

      if( x < 20 ) {
         System.out.print("Ovo je IF");
      }else {
         System.out.print("Ovo je ELSE");
      }
   }
}
```

Radi se o jednostavnom primeru koji će korisniku ispisati poruku "Ovo je IF" u zavisnosti da li je zadati broj X manji od 20 ili poruku "Ovo je Else" ukoliko je broj veći od 20. U ovom kodu su korišćene IF i ELSE instrukcije. Primer korišćenja if kondicionalne instrukcije:

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

# PRIMER 5

## *Pokazni primer 5.*

Primer DO WHILE petlje:

```
public class Test {

   public static void main(String args[]) {
      int x = 10;

      do {
         System.out.println("Vrednost x je: " + x );
         x++;
      }while( x < 20 );
   }
}


Izlaz:

Vrednost x je : 10
Vrednost x je : 11
Vrednost x je : 12
Vrednost x je : 13
Vrednost x je : 14
Vrednost x je : 15
Vrednost x je : 16
Vrednost x je : 17
Vrednost x je : 18
Vrednost x je : 19
```

U ovom primeru prikazana je DO WHILE petlja, koja će izvršavati kod sve dok se uslov ne ispuni. U ovom slučaju početna vrednost promenljive x je 10, a uslov je da program ispisuje vrednost promeljive u okviru konzole i povećava promenljivu za 1, sve dok se ne ispuni uslov da je x <20.

# ZADACI ZA INDIVIDUALNU VEŽBU

## *Zadaci za samostalan rad .*

1. Napisati konstantu po definisanoj konvenciji.
2. Napisati globalnu varijablu i predstaviti je u programskom kodu.
3. Definisati varijablu koja sadrži tri reči u nazivu.
1. Definisati četiri različita tipa promenljivih po izboru.
2. Predstaviti programskim kodom jednu privatnu i dve javne promenljive.
3. Koja je razlika između privatnih i javnih promenljivih u programskom kodu? Napisati primer u Java programskom jeziku.

1. Prikazati primer sekvencijalnog grupisanja za množenje četiri promenljive.
2. Prikazati preskakanje instrukcija u programskom kodu korišćenjem if else funkcije.
3. Opisati koncept grupisanja povezanih instrukcija na proizvoljnom primeru.
1. Prikazati upotrebu if kondicionalne instrukcije.
2. Napisati program koji za broj koji je veći od 5 ispisuje "Broj je veći od 5" a za brojeve manje od 5 ispisuje "Broj je manji od 5".
3. Definisati programski kod koji za uneti datum ispisuje u kojoj trećini meseca se nalazi. Koristiti else if funkciju.
1. Prikazati nepravilan način upotrebe petlji u programskom kodu.
2. Primenom for petlje napisati program koji ispisuje sve brojeve od 20 a manje od 50.
3. Prikazati upotrebu do while petlje na konkretnom primeru.

# DOMAĆI ZADATAK 4

*Domaći zadatak 4se dobija direktno od asistenta nakon poslatog prvog domaćeg zadatka ili na poseban zahtev studenta.*

Domaći zadatak 4 je individualan za svakog studenta i dobija se po definisanim instrukcijama.

Domaći zadatak se imenuje:
**KI301-DZ04-ImePrezime-brIndeksa** gde su vrednosti Ime, Prezime i br.Indeksa vaši podaci.

Domaći zadatak je potrebno poslati na adresu asistenta: nebojsa.gavrilovic@metropolitan.ac.rs sa naslovom (subject mail-a) **KI301-DZ04.**

**Posebno je potrebno voditi računa o pravilnom imenovanju mail-a prilikom slanja domaćih zadataka.**

**Napomena:**
Domaći zadaci treba da budu realizovani u zadatku navedenom razvojnom okruženju i da predstavljaju jedinstveno rešenje svakog studenta. Prepisivanje i preuzimanje programskog koda sa interneta strogo je zabranjeno.

# Poglavlje 7

# Zaključak

## ZAKLJUČAK

Greške kod inicijalizacije varijabli su česte u programima. I ovo oduzima puno vremena kod debugging-a. Zato, izbegavanje problema sa inicijalizacijom varijabli je vrlo bitno. Evo tipičnih razloga za greške inicijalizacije varijabli:

- varijabli nije nikad zadata vrednost
- jedan deo objekta je inicijalizovan ali ne ceo (nekoliko članova u objektu nije inicijalizovano)
- varijabli nije alocirana memorija
- varijabla je inicijalizovana, ali ta vrednost više ne važi već treba zadati novu vrednost

Petlje sa izlazom u sredini su:

- petlje sa break- instrukcijom (C,, C++, Java)
- i goto-instrukcijom (C, C++)

Kod upotrebe petlji mogu nastati mnoge greške, i to je bitan faktor za kvalitet softvera, npr.

- pogrešna inicijalizacija petlje
- pogrešno ugneždavanje
- pogrešno završavanje petlje
- itd
- Petlje sa mnogo break-instrukcija nisu poželjne jer ukazuju na nejasno razmišljanje o strukturi petlje, već je bolje koristiti niz petlji nego jednu petlju sa mnogo izlaza. Napomenimo da Java omogućujelabeled-break-instrukcije. Preporuka mnogih programera je da treba izbegavati break-instrukciju i continue-instrukciju. I danas traje rasprava da li ove instrukcije pripadaju ili ne tzv. strukturnom programiranju. Ove instrukcije mogu da naprave problem sa kompajlerom, pa ih je bolje izbegavati, ili ako ih koristimo treba biti svestan da možda nisu ispravne.

## LITERATURA

1. Code Complete: A practical handbook of software construction, by S. McConnell, Microsoft Press, ISBN 0-7356-1967-0, https://www.amazon.com/Code-Complete-Practical-Handbook-Construction/dp/0735619670
2. SWEBOK-V3, https://www.computer.org/web/swebok/v3
3. Theory and Problems of Software Engineering - Schaum’s Outline Series, by David Gustafson, ISBN 0-07-137794-8, http://www.mhebooklibrary.com/doi/abs/10.1036/0071377948