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

Standardi i dokumentacija kod konstruisanja

Lekcija 03

PRIRUČNIK ZA STUDENTE

KI301 - KONSTRUISANJE SOFTVERA

Lekcija 03

STANDARDI I DOKUMENTACIJA KOD KONSTRUISANJA

- Standardi i dokumentacija kod konstruisanja
- Poglavlje 1: Softverski standardi
- Poglavlje 2: Dokumentovani programi
- Poglavlje 3: Vizuelna struktura programskog koda
- Poglavlje 4: Interna dokumentacija softvera
- Zaključak

L3 - STANDARDI I DOKUMENTACIJA KOD KONSTRUISANJA

L3 - Softverski standardi i softverska dokumentacija kod konstruisanja softvera - Sadržaj

Sadržaj:

- Softverski standardi
- Dokumentovani programi
- Vizuelna struktura programskog koda
- Interna dokumentacija softvera

Poglavlje 1

Softverski standardi

STANDARDI SOFTVERA

Ovde navodimo neke bitne standarda softvera, i objašnjenje ovih standarda.

Sledeći standardi su relevantni za konstruisanje softvera:

- Standardi za format i sadržaj dokumenata,
- Standardi za programske jezike, npr. standardi za C++ I Javu
- Standardi za platforme
- Standardi za notacije modela softvera, npr. UML notacija
- IEEE Std 1016-1998, Recommended Practice for Software Design Descriptions, itd.

IEEE Standards**:**

Vrlo korisni izvori informacija su IEEE (Institute for Electric and Electronic Engineers, USA) standardi. IEEE standardi su razvijeni od strane grupa kompetenih inženjera, iz prakse i sa fakulteta, koji su eksperti u pojedinim oblastima. Svaki standard sadrži rezime oblasti koju pokriva, i takodje sadrži objašnjenje za izgled odgovarajuće dokumentacije za rad u dotičnoj oblasti. Više nacionalnih i internacionalnih organizacija učestvuje u tome. IEEE je preuzela vodjstvo u definisanju standarda u softverskom inženjerstvu. Neki standardi su prihvaćeni od ISO, International Standards Organization, ili IEC, International Engineering Consortium.

Prethodno navedeni standardi su tzv. eksterni standardi. Ali, pored eksternih standard mogu postojati i interni standardi, tj. standardi na nivou neke organizacije ili korporacije.

IEEE STANDARDS

Postoji čitava lista IEEE Standarda.

IEEE Standards-lista: Ime nekog standarda sadrži redni broj nekog dokumenta standarda, godinu usvajanja standarda, i ime standarda. Evo nekih relevantnih standarda,

Software development standards:

IEEE Std 830-1998, Recommended Practice for Software Requirement Specification,

IEEE Std 1233-1998, Guide for Developing System Requirements Specifications,

IEEE Std 1016-1998, Recommended Practice for Software Design Descriptions,

IEEE Std 1063-2001, Standard for Software User Documentation,

itd.

Software Quality-Assurance Standards:,

IEEE Std 1008-1993, Standard for Software Unit Testing,

IEEE Std 829-1998, Standard for Software Test Documentation,

IEEE Std 730-2002, Standard for Software Quality Assurance Plans,,

Itd.

MANAGEMENT STANDARDS

Evo liste standarda za upravljanje softverskim projektima.

Evo liste standarda za upravljanje softverskim projektima:

- IEEE, Standard for Software Project Management Plans,
- IEEE, Standard for Developing Sofware Life Cxcle Processes
- IEEE, Standard for Software Productivity Metrics
- itd.

KORISNI LINKOVI

Pogledajmo donje linkove, u cilju boljeg upoznavanja sa standardima softvera.

Pogledati linkove:

- http://www.tcworld.info/e-magazine/translation-and-localization/article/standards-for-software-documentation/
- http://www.literateprogramming.com/documentation.pdf
- https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/QualityMan/docstandards.html
- http://ieeexplore.ieee.org/document/565312/?reload=true

VIDEO VEŽBA

Video vežba -IEEE Standards for Software Engineering.

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

ZADACI ZA VEŽBU / PITANJA ZA SAMOPROVERU ZNANJA

Evo liste pitanja, za individualnu vežbu.

Zadaci za vežbu:

- Proučiti IEEE Standard for Software Productivity Metrics
- Za izabrani projekat, Ilustrovati primenu sledećeg standarda: Standard for Software Productivity Metrics

Pitanja:

- Objasniti od kojih komponenti se sastoji naziv nekog standarda ?
- Opisati komponente standarda za format i sadržaj dokumenata softvera ?
- Nabrojati standardne relevantne za konstruisanje softvera?

Poglavlje 2

Dokumentovani programi

DOKUMENTACIJA PROGRAMA

Programska dokumentacija je ponos programera, jer je to finalni rezultat njegovog rada.

Po pravilu, programeri vole da pišu programsku dokumentaciju. Dobra programska dokumentacija je ponos programera, jer je to finalni rezultat njegovog rada. Softverska dokumentacija može da ima puno formi, a jedan specifičan oblik su i komentari u programu. Dokumentacija softverskog projekta obuhvata kako informacije u okviru istinga programskog koda, ali i izvan tih listinga, odvojeni dokumenti koji opsuju razvoj pojedinih edinica softvera. Kod velikih projekata, većina dokumntacije je izvan listinga, i ovo je tzv. eksterna dokumentacija, *external documentation*. Eksterna konstrukciona dokumentacija je na višem nivou softvera u poredjenju sa izvornim kodom, i na nižem nivou softvera u poredjenju sa zahtevima softvera i arhitekture softvera.

Detaljni dizajnerski dokument, detailed-design document,opisuje detaljni tj. *low-level* dizajn (dizajn na nižem nivou softvera). On opisuje dizajn na nivou klase (class-level) i na nivou podprograma(routine-level), odluke i alternative koje su azmotrene u toku procesa detaljnog dizajna. Za razliku od eksterne dokumentacije, interna dokumentacija je dokumentacija u okviru listinga programskog koda. To je najdetaljnija dokumentacija, na nivou programskog koda. Ova unutrašnja dokumentacija obuhvata komentare u programu, izbor imena varijabli i konstanti je vrlo bitan, izbor imena podprograma, dobar raspored (*layout)* pojedinih delova programskog koda, itd.

TIPIČNI SOFTVERSKI DOKUMENTI

Pogledajmo listu tipičnih softverskih dokumentata.

Evo kratkog opisa tipičnih softverskih dokumenata: 1) Statement of work (Preliminarni kratki opis funkcionalnosti softvera-od strane korisnika);

Specifikacija softverskih zahteva (*Software requirements specification*) – Opisuje šta ce i kako raditi završeni softver.

2) **Model objekata/klasa** (Object/class model) – Opisuje glavne tzv. objekte/klase podataka softvera. I **Korisnički scenariji** (Use case scenarios) – Opis serije nizova mogućih ponašanja softvera sa korisničke tačke gledišta.

3) **Projektni vremenski plan** (Project schedule) – Opis redosleda zadataka i procena vremena i potrebnih napora.

4) **Plan testiranja softvera** (Software test plan) – Opis testiranja softvera. **Testovi prihvatljivosti** (Acceptance tests)– Testovi naznačeni od strane korisnika da odrede prihvatljivost softvera. **Izveštaj o testiranju** (Test report) - Specificira koji testovi su obavljeni i kako se softver ponašao. Izveštaj o defektima (Defect report) – Opisuje nezadovoljstvo korisnika sa konkretnim nezadovoljstvom korisnika zbog konkretne greške ili nedostataka softvera.

5) **Softverski dizajn** (Software design) – Opisuje strukturu softvera. **Arhitektonski dizajn** (Architectural design) – Opisuje strukturu na gornjim hijerarhijskim nivoima softvera, zajedno sa interkonekcijama (interkonekcijama izmedju pojedinih modula tj. blokova). **Detaljni dizajn** (Detailed design) – Opisuje dizajn modula ili dizajn objekata (klase) tj dizajn na najnižem hijerarhijskom nivou softvera.

6) **Plan obezbedjenja kvaliteta** (Quality assurance plan) – Preciziranje aktivnosti koje će biti izvršene u cilju obezbedjenja kvaliteta.

7) **Korisničko uputstvo** (User manuel)- Objašnjava kako da se koristi završen softver.

8) **Izvorni kod** (Source code) tj. programske instrukcije – Listing softverskih instrukcija (program u izvornom obliku/kodu).

9) Test report **(**Opisuje testove i kako se sistem ponaša), 10) Defect report (Opisuje nezadovoljstvo korisnika, tj. greške i nedostatke softvera)

DATA DICTIONARY

Ovde ukratko opisujemo tzv. rečnik podataka projekta tj. Project Data Dictionary.

Data-dictionary tj. rečnik podataka je baza podataka koja opisuje sve značajne podatke u projektu. Npr.ovaj alat sadrži

- u velikim projektima stotine ili hiljade definicije klasa
- na velikim projektima omogućuje da se ima pregled imenovanja klasa itd, i izbegavanja korišćenja istih imena ili nesaglasnih imena
- ovaj alat omogućuje optimizaciju izbora imena u projektu za pojedine varijable, klase, itd

Pogledati link:

- https://www.tutorialcup.com/dbms/data-dictionary.htm - Data Dictionary and Types of Data Dictionary
- http://www.cs.unb.ca/~fritz/cs3503/dd.htm - Data dictionary-example

DATA DICTIONARY-PRIMER

Evo ilustracije alata Data Dictionary

Rečnik podataka tj. Data dictionary je u stvari tabela podataka o svakom elementu u sistemu. npr. na početku projekta, u fazi zahteva, svi podaci iz problemskog domena (problem domain). Kako projekat napreduje, dodaju se podaci iz arhitektonskog dizajna, pa detaljnog dizajna, itd.

Tipičan red rečniku podataka sadrži naziv podatka, naziv klase gde je podatak lociran, tip podatka, i semantiku tj. značenje podatka. Evo primera za softver biblioteke:

Naziv Klasa Tip Veličina Semantika

Autor Knjiga String <40 znakova Ime i prezime

ISBN Knjiga String 10-20 znakova International Standard Book Number

Naslov Knjiga String <50 znakova Naslov knjige

Pozajmioc Pozajmioc Objekt Registrovani član biblioteke

itd.

KORISNI LINKOVI

Pogledajmo donje linkove.

Pogledati linkove:

https://en.wikipedia.org/wiki/Software_documentation

http://www.wikihow.com/Write-Software-Documentation

http://www.tcworld.info/e-magazine/translation-and-localization/article/standards-for-software-documentation/

VIDEO VEŽBA

Video vežba -Software Architecture Document.

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

VIDEO VEŽBA 2

Video vežba - Software documentation tutorial

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

PITANJA ZA SAMOPROVERU ZNANJA

Odgovorimo na pitanja.

Pitanja:

- Šta je to eksterna softverska dokumentacija?
- Šta su to standardi za softversku dokumentaciju?
- Šta je to Data Dictionary?

Poglavlje 3

Vizuelna struktura programskog koda

"TEOREMA" FORMATIRANJA, I STILOVI FORMATIRANJA

Teorema formatiranja: dobar vizuelni raspored pokazuje logičku strukturu programa.

Vizuelna struktura programskog koda se bavi formatiranjem i estetikom programskog koda tj. vizuelnim aranžiranjem programskog koda (layout of program source code). Tehnike formatiranja ne utiču na brzinu izvršavanja programa , upotrebu memorije, već one utiču na brzinu razumevanja programa, brzinu pregleda programa, i brzine revizije programa. Ove tehnike onomogućuku onima koji nisu pisali program da ga lako shvate i po potrebi midifikuju. U cilju postizanja dobro-formatiranog programa, primenljuje se teorema formatiranja, koja glasi: dobar vizuelni raspored pokazuje logičku strukturu programa. Dakle cilj formatiranja programa je da se što bolje vizuelno prikaže logička struktura programa. Postoji niz tehnika za to, i ove tehnike ćemo navesti i objasniti ovde, kroz niz primera. Dakle, vizuelni raspored programskog koda treba da što bolje prikaže logičku strukturu programskog koda. Dobra vizuelna prezentacija programskog koda omogućuje nekom ko čita program da ga brzo i lako razume, i ove tehnike su bitne.

Npr. for-petlja , u cilju dobrog formatiranja, može da se predstavi koristeći dva načina tj. dva stila, naime, ovako:

```
for (.........)

.....{iskaz1;

.....iskaz2.}
```

ili ovako:

```
for(....) { iskaz1;

}
```

CILJEVI DOBROG STRUKTUIRANJA PROGRAMSKOG KODA

Ima niz ciljeva koji se postižu primenom tehnike dobrog struktuiranja programskog koda.

Koji god stil formatiranja tj. metoda struktuiranja programskog koda se koristi, potrebno ga je koristiti konzistentno, dakle u celom programu, a ne različiti stilovi u različitim delovima programa.

Mogu se navesti sledeći ciljevi dobrog struktuiranja:

- tzv. "beli prostor" ("white space") , npr. "uvlačenje" instrukcija ("indentation"), se koristi da vizualizuje logičku strukturu programskog koda
- povećava se čitljivost tj. razumljivost programa
- omogućuje se lako modifikovanje programa

Npr., prazne linije treba koristiti da se razdvoje dva logička bloka. Naime, jedan logički blok je grupa iskaza koji logički pripadaju zajedno u jednu grupu, Dva logička bloka treba razdvojiti praznom linijom, na sličan način kao što se u nekoj knjizi paragrafi razdvajaju praznom linijom. Npr.:

```
iskaz1;

iskaz2;

.

iskaz3;

iskaz4;
```

prethodno vidimo dva bloka iskaza razdvojeni praznom linijom.

BELI PROSTOR

"Uvlačenje" pojedinih iskaza (indentation of statements) se koristi da prikaže logičku strukturu programskog koda.

Tzv. "beli prostor" tj. metafora "white space", koristi se za postizanje dobrog struktuiranja programskog koda. Beli prostor koristi razmake, nove redove, prekide linija i prazne linije, u cilju prikazivanja logičke strukture pojedinih sekcija programa. Na sličan način kao kad pišete knjigu, i koristite nove redove, razmake, poglavlja, itd., isto tako kad se piše program, u cilju bolje čitljivosti i lakše razumljivosti se korsti tehnika belog prostora.

Sledeće su tehnike dobrog struktuiranja:

- grupisanje povezanih iskaza
- prazne linije razdvajaju grupe iskaza tj. nepovezane iskaze

- uvlačenje pojedinih iskaza (indentation of statements) se koristi da prikaže logičku strukturu programskog koda, i pravilo uvlačenja iskaza je da se neki iskaz uvlači ispod onog iskaza koji je logički na višem nivou
- zagrade se koriste da se pojasne iskazi koji imaju više elemenata, one ne koštauju ništa a puno pobiljšavaju jasnoću izraza

PRIMERI IF-ISKAZA I WHILE-ISKAZA

Ovde dajemo primere formatiranja u C++.

Evo primera if-instrukcije:

if (iskaz0) {

....iskaz1;

....iskaz2;

}

gde vidimo da su iskazi između velikih zagrada uvučeni iza ključne reči if.

Ili primer while-instrukcije:

while (iskaz0) {

.....iskaz1;

.....iskaz2;

}

Ovaj stil formatiranja je standardan u C++ i Javi.

ALTENATIVAN STIL

Mogu se koristiti alternativni stilovi formatiranja.

Alternativan stil za if-instrukciju i while -instrukciju je sledeći:

```
if (.............................)
   {
     iskaz1;
     iskaz2;
     ...........
    }

ili

while (.............................)
        {
```

```
        iskaz1;.
        iskaz2;
        ...........
        }
```

PRIMER FOR-PETLJE

Evo primera for-petlje.

Evo kako treba da izgleda dobro-formatirana for-petlja:

```
for (iskaz0.)
     {

      iskaz1;
      iskaz2;
      ............
      }
```

a evo pogrešnog formatiranja for-petlje:

```
for iskaz0)

{

      iskaz1;
      iskaz2;
      ............
}
```

FORMATIRANJE GOTO-INSTRUKCIJE

U principu treba izbegavati goto-instrukciju, ali ako je neophodno da se koristi onda postoje određena pravila formatiranja ove instrukcije.

Evo nekih korisnih pravila zaformatiranje goto-instrukcija:

- izbegavati goto-instrukcije, jer one kvare logičku strukturu programa
- koristiti velika slova za goto-label, jer ovo čini vidljivom goto-instrukciju
- koristiti posebnu liniju za goto-instrukciju da bi je učinili lako vidljivom
- koristiti posebnu liniju za goto-label

Evo primera dobrog formatiranja goto-instrukcije:

```
.........................
if (iskaz0) {
         iskaz1;
         goto XXXX;

}

XXXX:
         iskaz2;

}
```

FORMATIRANJE DEKLARACIJA PODATAKA

Ovde nabrajamo pravila za formatiranje deklaracija podataka.

Evo pravila zaformatiranje deklaracija podataka:

- koristiti samo jednu deklaraciju podataka u jednoj liniji
- deklarisati podatke blizu lokacije gde se prvi put oriste, umesto grupisanja svih deklaracija zajedno u jednom velikom bloku
- redosled deklaracija može biti prema tipu deklaracije, umesto prema alfabetskom redosledu

Evo primera dobrog formatiranja deklaracija podataka:

```
int index1;

int index2;

float flNumber1;

float flNumber2;
```

FORMATIRANJE KOMENTARA U PROGRAMU

Evo pravila za formatiranje komentara.

Komentari u programu mogu dramatično da poboljšaju čitljivost programa. Formatiranje komentara u rogramu je značajno u cilju obezbeđenja dobre čitljivosti programa. Evo pravila za formatiranje komentara:

- ispred komentara treba staviti praznu liniju
- komentari se uvlače da ne remete logičku strukturu programskog koda

Evo primera za razdvajanje komentara u Javi:

```
//komentar1

iskaz1;

/

//komentar2

iskaz2;
```

A evo primera uvlačenja komentara:

```
for (.................)
...... //komentar1
```

FORMATIRANJE PODPROGRAMA

Evo liste formatrijaćih pravila za podprograme.

Evo liste formatrijaćih pravila za podprograme:

- argumenti podprograma se uvlače
- svaki argument se piše u posebnom redu
- koriste se prazne linije da razdvoji zaglavlje podprograma i telo podprograma

Evo npr. u C++ kako treba da izgleda zaglavlje podprograma:

```
public int Ime(
      int index1;
      int index2;

)
```

FORMATIRANJE KLASE

Ovde navodimo listu pravila za formatiranje klase.

Kod klasnog interfejsa treba poštovati sledeći redosled zaređanje pojedinih članova klasnog interfejsa:

- komentar u zaglavlju koji opisuje klasu
- konstruktori i destruktori
- javni podprogrami tj. metode
- zaštićeni podprogrami (metode)
- privatni podproframi

A implementacija klase se raspoređuje po sledećem redosledu:

- komentar u zaglavlju koji detaljnije opisuje klasu
- podaci klase
- javni podprogrami
- zaštićeni podprorami
- privatni podprogrami

A evo liste pravila u vezi sa formatiranem klasa:

- po pravilu u jedan fajl staviti samo jednu klasu
- ako imamo nekoliko klasa u jednom fajlu potrebno je razdvojiti klase sa nekoliko praznih linija
- redosled podprograma u pojedinim grupama može biti alfabetski
- ispred i posle svakog podprograma stavlja se prazna linija

POKAZNA VEŽBA

Evo primera formatiranja klase.

U nastavku je dat primer programskog koda koji nije formatiran po definisanim pravilima. Samim tim, takav programski kod je nečitljiv i nerazumljiv. Progamer mora poštovati pravila u toku pisanja programskog koda i izvršiti njegovo formatiranje. Često se može desiti da se progameri menjaju tako da pravilno formatiranje programskog koda omogućava da se novi članovi razvojnog tima brzo i efikasno prilagode i nastave sa radom na programu.

```
package business; /** *
Singleton klasa koja sluzi za komunikaciju biznis dela aplikacije i gui dela *
@author Student */ public class Business { private static Business instance = new
Business(); // Instanca singleton klase private Kalkulator k = new Kalkulator(); //
Instanca klase kalkulator /** * * Privatni konstruktor */ private Business(){ } /**
* * Metoda koja vraca instancu ove singlton klase */ public static Business
getInstance(){ return instance; } /** * * Metoda koja nadovezuje broj nula */
public void nula(){ k.nula(); } /** * * Metoda koja nadovezuje broj jedan */ public
void jedan(){ k.jedan(); } /** * * Metoda koja nadovezuje broj dva */ public void
dva(){ k.dva(); } /** * * Metoda koja nadovezuje broj tri */ public void tri(){
k.tri(); } /** * * Metoda koja nadovezuje broj cetiri */ public void cetiri(){
k.cetiri(); } /** * * Metoda koja nadovezuje broj pet */ public void pet(){
k.pet(); } /** * * Metoda koja nadovezuje broj sest */ public void sest(){
k.sest(); } /** * * Metoda koja nadovezuje broj sedam */ public void sedam(){
k.sedam(); } /** * *
Metoda koja nadovezuje broj osam */
public void osam(){ k.osam(); } /** * * Metoda koja nadovezuje broj devet */
public void devet(){ k.devet(); } /** * * Metoda koja setuje drugi broj */
public String getDrugiBroj(){ return k.getDrugiBroj(); } /** * * Metoda koja sabira
*/ public void plus(){ k.plus(); } /** * * Metoda koja oduzima */ public void
minus(){ k.minus(); } /** * * Metoda koja mnozi */ public void puta(){ k.puta(); }
/** * * Metoda koja deli */ public void podeljeno(){ k.podeljeno(); } /** * *
Metoda koja poziva racunanje */ public double jednako(){ return k.jednako(); } /**
* * Metoda koja pravi decimalni broj */ public void tacka(){ k.tacka(); } /** * *
Metoda koja brise samo poslednji unet broj */ public void c(){ k.c(); } /** * *
Metoda koja brise sve */ public void ac(){ k.ac(); } /** * * Metoda koja racuna
```

```
sinus */ public void sinus(){ k.sinus(); } /** * * Metoda koja racuna kosinus */
public void cos(){ k.cos(); } /** * * Metoda koja racuna tangens */ public void
tan(){ k.tan(); } /** * * Metoda koja racuna Arkus sinus */ public void asinus(){
k.asinus(); } /** * * Metoda koja racuna arkus kosinus */ public void acos(){
k.acos(); } /** * * Metoda koja racuna arkus tangens */ public void atan(){
k.atan(); } /** * * Metoda koja racuna koren */ public void sqrt(){ k.sqrt(); } /**
* * Metoda koja racuna faktorijel */
public void faktorijal(){ k.faktorijal(); } /** * * Metoda koja racuna prirodni
logaritam */ public void logN(){ k.logN(); } /** * Metoda koja racuna logaritam od
deset * */ public void log10(){ k.log10(); } /** * Metoda koja racuna kvadrat * */
public void stepen() { k.stepen(); } /** * Metoda koja racuna absolutno * */ public
void abs() { k.abs(); } }
```

Primer formatiranja programskog koda u razvojnom okruženju NetBeans:

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

U okviru ovog dela dat je pravilno formatiran programski kod u Java programskom jeziku. Prikazano je na koji način je potrebno organizovati jednu klasu (klasa Business) i kako je potrebno pisati metode u okviru same klase.

```
package business;

/**
 * Singleton klasa koja sluzi za komunikaciju biznis dela aplikacije i gui dela
 * @author Student
 */
public class Business {

    private static Business instance = new Business(); // Instanca singleton klase
    private Kalkulator k = new Kalkulator(); // Instanca klase kalkulator

    /**
     *
     * Privatni konstruktor
     */
    private Business(){

    }

    /**
     *
     * Metoda koja vraca instancu ove singlton klase
     */
    public static Business getInstance(){
        return instance;
    }

    /**
     *
     * Metoda koja nadovezuje broj nula
     */
```

```
    public void nula(){
        k.nula();
    }

    /**
     *
     * Metoda koja nadovezuje broj jedan
     */
    public void jedan(){
        k.jedan();
    }

    /**
     *
     * Metoda koja nadovezuje broj dva
     */
    public void dva(){
        k.dva();
    }

    /**
     *
     * Metoda koja nadovezuje broj tri
     */
    public void tri(){
        k.tri();
    }

    /**
     *
     * Metoda koja nadovezuje broj cetiri
     */
    public void cetiri(){
        k.cetiri();
    }

    /**
     *
     * Metoda koja nadovezuje broj pet
     */
    public void pet(){
        k.pet();
    }

    /**
     *
     * Metoda koja nadovezuje broj sest
     */
    public void sest(){
        k.sest();
    }

    /**
```

```
     *
     * Metoda koja nadovezuje broj sedam
     */
    public void sedam(){
        k.sedam();
    }

    /**
     *
     * Metoda koja nadovezuje broj osam
     */
    public void osam(){
        k.osam();
    }

    /**
     *
     * Metoda koja nadovezuje broj devet
     */
    public void devet(){
        k.devet();
    }

    /**
     *
     * Metoda koja setuje drugi broj
     */
    public String getDrugiBroj(){
        return k.getDrugiBroj();
    }

    /**
     *
     * Metoda koja sabira
     */
    public void plus(){
        k.plus();
    }

    /**
     *
     * Metoda koja oduzima
     */
    public void minus(){
        k.minus();
    }

    /**
     *
     * Metoda koja mnozi
     */
    public void puta(){
        k.puta();
```

```
    }

    /**
     *
     * Metoda koja deli
     */
    public void podeljeno(){
        k.podeljeno();
    }

    /**
     *
     * Metoda koja poziva racunanje
     */
    public double jednako(){
        return k.jednako();
    }

    /**
     *
     * Metoda koja pravi decimalni broj
     */
    public void tacka(){
        k.tacka();
    }

    /**
     *
     * Metoda koja brise samo poslednji unet broj
     */
    public void c(){
        k.c();
    }

    /**
     *
     * Metoda koja brise sve
     */
    public void ac(){
        k.ac();
    }

    /**
     *
     * Metoda koja racuna sinus
     */
    public void sinus(){
        k.sinus();
    }

    /**
     *
     * Metoda koja racuna kosinus
```

```
     */
    public void cos(){
        k.cos();
    }

    /**
     *
     * Metoda koja racuna tangens
     */
    public void tan(){
        k.tan();
    }

    /**
     *
     * Metoda koja racuna Arkus sinus
     */
    public void asinus(){
        k.asinus();
    }

    /**
     *
     * Metoda koja racuna arkus kosinus
     */
    public void acos(){
        k.acos();
    }

    /**
     *
     * Metoda koja racuna arkus tangens
     */
    public void atan(){
        k.atan();
    }

    /**
     *
     * Metoda koja racuna koren
     */
    public void sqrt(){
        k.sqrt();
    }

    /**
     *
     * Metoda koja racuna faktorijel
     */
    public void faktorijal(){
        k.faktorijal();
    }
```

```
    /**
     *
     * Metoda koja racuna prirodni logaritam
     */
    public void logN(){
        k.logN();
    }

    /**
     * Metoda koja racuna logaritam od deset
     *
     */
    public void log10(){
        k.log10();
    }

    /**
     * Metoda koja racuna kvadrat
     *
     */
    public void stepen() {
        k.stepen();
    }

    /**
     *  Metoda koja racuna absolutno
     *
     */
    public void abs() {
        k.abs();
    }
}
```

POKAZNA VEŽBA-NASTAVAK

Evo primera formatiranja podprograma tj. klasne metode.

Klasne metode u okviru programa omogućavaju izvršavanje određenih funkcija programa. Primer nepravilno formatirane metode:

```
/** * Metoda mnozenje */ public void puta(){
if(!drugiBroj.equals("")){ if(prviBroj.equals("")){ prviBroj=drugiBroj;
drugiBroj=""; }
else{ if(znak.equals("+")){ rezultat = Double.parseDouble(prviBroj) +
 Double.parseDouble(getDrugiBroj()); }else if(znak.equals("-")){
 rezultat = Double.parseDouble(prviBroj) - Double.parseDouble(getDrugiBroj());
}else if
(znak.equals("*")){ rezultat = Double.parseDouble(prviBroj) *
Double.parseDouble(getDrugiBroj()); }
else if(znak.equals("/")){ rezultat = Double.parseDouble(prviBroj) /
```

```
Double.parseDouble(getDrugiBroj()); }
prviBroj=""+rezultat; drugiBroj=""; } znak="*"; } }
```

Radi se o metodi množenja u okviru klase, i na osnovu datog isečka programskog koda nije moguće utvrditi šta programski kod radi. Nakon formatiranja programskog koda u nekom od razvojnih okruženja (u ovom slučaju NetBeans) metoda je formatirana i izgleda kao u nastavku.

Primer pravilno formatirane metode u klasi:

```
/**
    * Metoda mnozenje
    */
    public void puta(){
        if(!drugiBroj.equals("")){
            if(prviBroj.equals("")){
                prviBroj=drugiBroj;
                drugiBroj="";
            }else{
                if(znak.equals("+")){
                    rezultat = Double.parseDouble(prviBroj) +
Double.parseDouble(getDrugiBroj());
                }else if(znak.equals("-")){
                    rezultat = Double.parseDouble(prviBroj) -
Double.parseDouble(getDrugiBroj());
                }else if(znak.equals("*")){
                    rezultat = Double.parseDouble(prviBroj) *
Double.parseDouble(getDrugiBroj());
                }else if(znak.equals("/")){
                    rezultat = Double.parseDouble(prviBroj) /
Double.parseDouble(getDrugiBroj());
                }
                prviBroj=""+rezultat;
                drugiBroj="";
            }
            znak="*";
        }
    }
```

ZADACI ZA SAMOSTALAN RAD

Evo zadataka za individualnu vežbu.

Zadaci:

- Dati jedan primer dobrog formatiranja klase?
- Dati jedan primer lošeg formatiranja klase?
- Dati jedan primer dobrog formatiranja podprograma tj. klasne metode?
- Dati jedan primer lošeg formatiranja podprograma?

Poglavlje 4

Interna dokumentacija softvera

SAMO-DOKUMENTUJUĆI PROGRAMSKI KOD

Ovde navodimo pravila za samo-dokumentujući programski kod.

Dokumentacija softverskog projekta se sastoji od

- source code listings (interna dokumentacija) tj. listing programa koji uključuje instrukcije programa i komentare u programu
- ostali separatni dokumenti (eksterna dokumentacija), npr.detailed-design document , itd.

Interna dokumentacija obuhvata komentare u programu i same instrukcije u programu. Medjutim, ne samo komentari u programu već i sam programski kod predstavlja vrlo bitan deo dokumentacije. Programski kod treba da je self-documenting tj. samo-dokumentujući tj. da je jasan i dobro formatiran.

Evo pravila zasamo-dokumentujući kod za klase u programu:

- ime klase treba da bude dobro odabrano, da opisuje centralni cilj klase
- klasni intrefejs treba da jasno i očigledno opisuje kako se koristi klasa
- klasni inrfejs treba da je dobra aabstrakcija klase da omogućuje da se klasa posmatra kao crna kutija tj. da ne treba da ulazite u detalje implementacije da bi koristili tu klasu

Evo pravila za samo-dokumentujuće podprograme (tj. klasne metode):

- ime podprograma treba tačno da izražava šta podprogram radi
- svaki podprogram treba da obavlja jedan dobro-definisan cilj
- interfejs podprograma treba da je jasan i očigledan

VRSTE KOMENTARA

Ovde diskutujemo vrste komentara i tehnike komentarisanja unutar programa.

U programu treba staviti razne vrste komentara. Npr.

- objašnjenje programskog koda za komplikovane ili delikatne delove programskog koda (explanation comments)
- rezime programskog koda za pojedine delove programskog koda daje sažetu informaciju o tom delu programa (summary comments)
- objašnjenja o tome šta pojedini delovi programskog koda rade (intent comments)

- generalni komentari o programu, npr. copyright notice, confidentiality notice, version number, itd.

Tehnike komentarisanja su sledeće:

- komentarisanje pojedinih linija u programu, npr. za neke komplikovane ili važne linije u programu
- komentarisanje pojedinih paragrafa u programu
- komentarisanje deklaracija podataka
- komentiranje podprograma
- komentarisanje klasa
- komentarisanje fajlova, npr. fajla sa više klasa ili sa više podprograma
- komentarisanje celokupnog programa

TEHNIKA KOMENTARISANJA U PROGRAMU

Data je lista preporuka za dobro komentarisanje u programu.

Evo generalne liste preporuka za dobro komentarisanje tj. komentiranje u programu:

- komentari treba da omoguće lako i brzo razumevanje programa
- komentari u programu treba da opišu nameru tj. cilj programa i da sumarizuju šta program radi
- komplikovani delovi programa treba da su komentarisani
- komentari treba da su jasni i tačni
- komentari treba da su konzistentni tj. da imaju konzinstentan stil tako da se lako mogu modifikovati
- u programu je korisno ubaciti pseudokod (pseudocode), jer onda komentari mogu biti kraći

Evo liste preporuka za dobro komentarisanje iskaza i paragrafa u programu:

- ne treba koristiti komentare na kraju linija tzv. endline comments tj. komentare u produžetku instrukcije
- ponavljajući komentari i suviše obimni komentari treba da se izbegavaju
- izbegavati skraćenice
- treba praviti razliku između glavnih i sporednih komentara, dakle na neki način vizuelno označiti razliku između glavnih komentara i sporednih

A evo liste preporuka za dobro komentarisanje deklaracija podataka:

- opsezi vrednosti za numeričke podatke treba komentarisati
- limiti na ulazne podatke treba da su komentarisani
- svaka globalna varijabla treba da se komentariše tamo gde je deklarisana

TEHNIKA KOMENTARISANJA PODPROGRAMA I KLASA

Ovde se specificiraju preporuke za komentarisanje klasnih metoda i klasa i fajlova i programa.

Za kontrolne strukture(if, while,for, itd) preporuke su sledeće:
svaka kontrolnastruktura treba da je komentarisana na svom početku, a krajevi se komentarišu samo za dugačke kontrolne strukture

Navedimo pravila za komentarisanje podprograma tj. klasnih metoda:

- cilj svakog podprograma treba da je komentarisan
- ostali važni detalji, npr. ulaz i izlaz, limiti, itd. treba da su komentarisani

Evo preporuka za komentarisanje klasa, fajlova i programa:

- program treba da ima opis ukupne organizacije programa
- cilj svake klase i svakog fajla treba da je opisan
- ime autora pojedinih delova programa treba da su dati u programu zajedno sa njihovim mejl-adresom

POKAZNA VEŽBA

Evo primera komentarisanja klase.

Komentarisanje klase omogućavaju da kod bude razumljiv i onima koji ga nisu pisali. Ukoliko programer ne izvrši komentarisanje klase, takav kod će biti nečitljiv i često će drugim programerima trebati više vremena kako bi utvrdili šta ta klasa radi i kako se izvršava. Komentari se pišu pre naziva klase. U nastavku je dat primer klase u Java programskom jeziku:

```
package business;

/**
 * Klasa kalkulator u kojoj se nalazi logika izračunavanja
 * @author Student
 */
public class Kalkulator {
    private String prviBroj, drugiBroj, znak;
    private double rezultat;

    public Kalkulator() {
        ocisti();
    }
```

Komentar klase je:

/**
* Klasa kalkulator u kojoj se nalazi logika izračunavanja

Na osnovu komentara jasno je da se u klasi kalkulator nalazi programska logika koja vrši izračunavanja u okviru programa.

POKAZNA VEŽBA-NASTAVAK

Evo primera komentarisanja podprograma.

Kao što se vrši komentarisanje klasa, tako je potrebno izvršiti i komentarisanje programskog koda u okviru same klase. U nastavku je dat primer programskog koda gde se vide komentari:

```
public class Business {

    private static Business instance = new Business(); // Instanca singleton klase
    private Kalkulator k = new Kalkulator(); // Instanca klase kalkulator

    /**
     *
     * Privatni konstruktor
     */
    private Business(){

    }
```

Komentari su:

- // Instanca singleton klase
- // Instanca klase kalkulator

i odnose na instancu određenih klasa (singleton i kalkulator).

Primer komentarisanja Java programskog koda u Eclipse razvojnom okruženju:

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

ZADACI ZA SAMOSTALAN RAD

Zadaci za individualnu vežbu.

Zadaci:

- Dati jedan primer dobrog komentarisanja podprograma?
- Dati jedan primer dobrog komentarisanja klase?
- Dati jedan primer lošeg komentarisanja podprograma?
- Dati jedan primer lošeg komentarisanja klase?

DOMAĆI ZADATAK

Na osnovu urađenih primera u vežbama potrebno je uraditi prvi domaći zadatak koji je univerzalan za sve studente.

Domaći zadatak se imenuje:

KI301-DZ01-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-DZ01.**

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

Napomena: Prvi domaći zadatak je univerzalan i važi za sve studente, dok ostale domaće zadatke dobijate od asistenta nakon realizacije prethodnog.

Domaći zadaci treba da budu realizovani u razvojnom okruženju koje je definisano domaćim zadatkom i da predstavljaju jedinstveno rešenje svakog studenta. Prepisivanje i preuzimanje programskog koda sa interneta strogo je zabranjeno.

Poglavlje 5

Zaključak

ZAKLJUČAK

Sledeći standardi su relevantni za konstruisanje softvera:

- Standardi za format i sadržaj dokumenata,
- Standardi za programske jezike, npr. standardi za C++ I Javu
- Standardi za platforme
- Standardi za notacije modela softvera, npr. UML notacija
- IEEE Std 1016-1998, Recommended Practice for Software Design Descriptions, itd.

Vizuelna struktura programskog koda se bavi formatiranjem i estetikom programskog koda tj. vizuelnim aranžiranjem programskog koda (layout of program source code). Tehnike formatiranja ne utiču na brzinu izvršavanja programa , upotrebu memorije, već one utiču na brzinu razumevanja programa, brzinu pregleda programa, i brzine revizije programa. Ove tehnike onomogućuku onima koji nisu pisali program da ga lako shvate i po potrebi midifikuju. U cilju postizanja dobro-formatiranog programa, primenljuje se teorema formatiranja, koja glasi: dobar vizuelni raspored pokazuje logičku strukturu programa.

LITERATURA

Linkovi:

http://www.tcworld.info/e-magazine/translation-and-localization/article/standards-for-software-documentation/

http://www.literateprogramming.com/documentation.pdf

https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/QualityMan/docstandards.html

http://ieeexplore.ieee.org/document/565312/?reload=true

https://en.wikipedia.org/wiki/Software_documentation

http://www.wikihow.com/Write-Software-Documentation

http://www.tcworld.info/e-magazine/translation-and-localization/article/standards-for-software-documentation/