Sistem za upravljanje predmetima u sudovima-Veritas

Smjernice za programiranje

Verzija 1.0

Istorija revizija

|  |  |  |  |
| --- | --- | --- | --- |
| **Datum** | **Verzija** | **Opis** | **Autor** |
| 08.05.2022 | 1.0 | Izrada cjelokupnog dokumenta | Miljan Bodiroga |

Sadržaj

1. Uvod 4

1.1 Svrha 4

1.2 Područje primjene 4

1.3 Definicije, akronimi i skraćenice 4

1.4 Reference 4

1.5 Pregled 4

2. Organizacija koda i stila 4

3. Komentari 5

4. Imenovanje 5

5. Deklaracije 6

5.1 Broj po liniji 6

5.2 Pozicija 6

5.3 Inicijalizacija 7

5.4 Deklaracija klasa i interfejsa 7

6. Izrazi i izjave 7

6.1 Izrazi 7

6.2 Jednostavne izjave 7

6.3 Složene izjave 8

6.4 Return izjave 8

6.5 If, if-else, if-else-if else izjave 8

6.6 For izjave 9

6.7 While izjave 9

6.8 Do-while izjave 9

6.9 Switch izjave 9

6.10 Try-catch izjave 9

7. Upravljanje memorijom 10

8. Upravljanje greškama i izuzecima 10

9. Prenosivost 10

10. Ponovna upotreba 10

11. Problemi pri kompajliranju 11

12. Dodatno: sažetak smjernica 11

Smjernice za programiranje

# Uvod

Smjernice za programiranje predstavlja dokument koji se koristi za definisanje standarda,konvencija i smjernica za pisanje JAVA koda. Uz pomoć ovog dokumenta moguće je brže, efikasnije i razumljivije vršiti razvijanje i izmjenu koda.

## Svrha

Svrha dokumenta se temelji na tome da je potrebo definisati određene standarde kodiranja namjenjene za razvoj programa Veritas, kako ne bi dolazilo do nesporazuma između autora koda ili do nerazumijevanja dijelova koda koji su napisani od strane različitih autora ovog projekta.

## Područje primjene

Razvoj editora koji nam pomaže za upravljanje predmetima u sudovima pod nazivom Veritas se vrši u Eclipse razvojnom okruženju, upotrebom JAVA programskog jezika.

## Definicije, akronimi i skraćenice

Definicije svih nepoznatih pojmova, svi akronimi i skraćenice koje su korišćene u dokumentu “Smjernice za Programiranje” opisane su u dokumentu “Rječnik”, koji se nalazi u sklopu dokumentacije sistema.

## Reference

* (<https://sceweb.uhcl.edu>)
* (<https://www.oracle.com/java/technologies/javase/codeconventions-contents.html>)
* [Software Engineering | Coding - javatpoint](https://www.javatpoint.com/software-engineering-coding)

## Pregled

U daljem toku dokumenta opisana je organizacija koda i njegovog stila, komentari, imenovanja, deklaracije, upravljanje greškama, izuzecima i memorijom, ponovna upotreba, prenosivost, kao i problem pri kompajliranju.

# Organizacija koda i stila

# Definisanje organizacije koda i stila programiranja su izuzetno bitne stavke koje se koriste prilikom pisanja izvornog koda određenog programa. One mogu pomoći programerima da čitaju i razumiju kod u skladu sa zadatim smjernicama, i na taj način se smanjuju šanse za greške i moguće je brže vršiti izmjene na kodu. U ovom slučaju kod se piše u programskom jeziku Java. Smjernice bitne za kod su sljedeće:

* **Uvlačenje redova** (pomaže u identifikovanju kontrola i blokova koda)

Koristiće se varijanta u kojoj se kao jedinica uvlačenja koriste 4 razmaka, što u stvari predstavlja jedan TAB.

* **Dužina redova** (označava koliko maksimalno karaktera može biti u jednom horizontalnom redu)

Linije ne trebaju biti duže od 70 karaktera.

* **Suzbijanje redova** (predstavlja način pisanja linija koje imaju dužinu veću od dozvoljene)

Prenos u drugi red treba izvršiti nakon: zareza, operatora, dostizanja nivoa prethodne linije.

* **Prostor izmedju karaktera i simbola** (prisustvo bijelina povećava čitljivost koda)

Između karaktera i simbola potrebno je obezbjediti razmak.

* **Paketi** (obraćanje pažnje na strukturu koda)

Kako ne bi došlo do nepotrebnog nagomilavanja klasa, funkcija, cjelina koje utiču na čitljivost koda, potrebno je što više koristiti pakete, kako bi se izvršila sistematizacija.

# Komentari

Komentari su jedan vid bilješki u programu i koriste se za dodatna pojašnjenja dizajna i svrhe koda. Treba se truditi da komentari ne budu predugi ili kompikovani. Dužina komentara ne treba da prelazi 70 karaktera. Moguće je koristiti 3 vrste komentara u JAVA kodu:

* Jednolinijski komentari: počinju sa // i završavaju se na kraju linije koda
* Višelinijski komentari: počinju sa /\* i završavaju se sa \*/
* Komentari dokumentacije: počinju sa /\*\* i završavaju se sa \*/

# Imenovanje

|  |  |  |
| --- | --- | --- |
| **Tip identifikatora** | **Pravila imenovanja** | **Primjeri** |
| **Varijable** | Ime varijable se piše malim slovom., a ostale riječi se spajaju velikim prvim slovom. Ime varijable treba ukazuje na njeno značenje, osim ako se koristi u petljama. | i  brojElemenata  vrijemePokretanja |
| **Konstante** | Ime konstante se piše velikim slovima, a ostale riječi se spajaju sa donjom crtom ("\_"). | PI  MAX\_VRIJEDNOST |
| **Metode** | Ime metode se piše malim slovima, a ostale riječi se spajaju sa prvim velikim slovom. Ime metode treba da bude glagol i da ukazuje na njeno značenje. | dodajElement() |
| **Klase** | Ime klase se piše velikim početnim slovom, a ostale riječi se spajaju sa velikim početnim slovima. Ime klase treba da bude imenica koja ukazuje na njeno značenje. | KontekstniMeni |
| **Interfejsi** | Imena interfejsa se pišu na isti način kao i klase. Poželjno je da je naziv interfejsa opisni pridjev ili opisna/deskriptivna imenica. | Interface |
| **Paketi** | Ime paketa se piše malim slovima, a ostale riječi se dodaju sa velikim početnim slovom. Ime paketa treba da ukazuje na grupu klasa koje sadrži taj paket.  Da bi se izbjegla kolizije korištenja istih imena, moraju se poštovati sledeća pravila:   * imena paketa pisana su malim slovima da se izbjegne konflikt sa imenima klasa i interfejsa, * koriste se obrnuti domeni, * kolizije koje nastanu, mogu se rijesiti regijama gdje firma posluje. * paketi jave zapocinju sa *java* ili *javax* u slucaju da postoje kljucne rijeci sadrzane u imenu domene, ili ime počinje brojem, tada se ispred ubacuje znak \_ | main  menuBar |

# Deklaracije

## Broj po liniji

Koristi se jedna deklaracija po liniji, jer je na takav način olakšano komentarisanje. Na primjer:

|  |  |
| --- | --- |
| 1 | int nivo; // nivo indentacije |
| 2 | int velicina; // velicina tabele |

Treba izbjegavati deklaraciju varijabli i metoda na istoj liniji. Primjer:

|  |  |
| --- | --- |
| 1 | long dbard, getDbard; // POGREŠNO |
| 2 | int varijabla, varniz[]; // POREŠNO |

## Pozicija

Stavljati deklaracije samo na početku blokova koda, odnosno na početku unutar vitičastih zagrada “{“i”}“. Izbjegavati deklarisanje varijabli „na zahtjev“, odnosno onda kada su neposredno potrebne.

|  |  |
| --- | --- |
| 1 | public void mojaMetoda() |
| 2 | { |
| 3 | int int1; |
| 4 |  |
| 5 | if (uslov) |
| 6 | { |
| 7 | int int2: |
| 8 | ... |
| 9 | } |
| 10 | } |

Izuzetak je kod indeksa za for petlju, jer se kod Jave moze deklarisati unutar for izraza:

|  |  |
| --- | --- |
| 1 | for (int i = 0; i < duzina; i++) |
| 2 | { |
| 3 | .... |
| 4 | } |

 Treba izbjegavati da lokalne deklaracije sakrivaju globalne, odnosno:

|  |  |
| --- | --- |
| 1 | int broj; |
| 2 | … |
| 3 | func(); |
| 4 | { |
| 5 | if (uslov) |
| 6 | { |
| 7 | int broj; //IZBJEĆI |
| 8 | ... |
| 9 | } |
| 10 | ... |
| 11 | } |

## Inicijalizacija

Pravilo je da se lokalne varijable inicijalizuju gdje su deklarisane. Treba izbjegavati u slučaju ako inicijalna vrijednost zavisi od nekih prethodnih izračunavanja koje uzimaju vrijeme i prostor. Tako, na primjer, treba izbjegavati instanciranje objekata do onoga trenutka kada su zaista potrebni.

## Deklaracija klasa i interfejsa

* Nema razmaka između imena metode i lijeve zagrade “(”
* Lijeva vitičasta zagrada “{”se nalazi na kraju iste linije kao izjava deklaracije
* Desna vitičasta zagrada “}”se nalazi sama u liniji, osim ako je blok prazan onda ta zagrada treba da se pojavi odmah nakon “{”

|  |  |
| --- | --- |
| 1 | class Primjer { |
| 2 | int int1; |
| 3 | int int2; |
| 4 |  |
| 5 | Primjer1 (int i, int j) { |
| 6 | int1 = i; |
| 7 | int2 = j; |
| 8 | } |
| 9 |  |
| 10 | int praznaMetoda () {} |
| 11 |  |
| 12 | ... |
| 13 | } |

# Izrazi i izjave

## Izrazi

Izraze je potrebno pisati nedvosmisleno i precizno. Ukoliko se radi o složenim izrazima potrebno je koristiti zagrade radi razjašnjenja koda.

|  |  |
| --- | --- |
| 1 | x + y / 100 //Nejasno |
| 2 | (x + y) / 100 //Jasno |
| 3 | x + (y / 100) //Jasno |

## Jednostavne izjave

Svaka linija treba da sadrži jednu izjavu.

Primjer:

|  |  |
| --- | --- |
| 1 | argv++; //Pravilno |
| 2 | argc++; //Pravilno |
| 3 | argv++; argc++; //Izbjegavati |

## Složene izjave

Složene izjave su izjave koje sadrže liste izjava unutar zatvorenih zagrada “{izjava}“.

* Ograđene izjave trebaju biti uvučene za jedan nivo više od složenih izjava
* Lijeva vitičasta zagrada treba biti na kraju reda koji započinje složeni izraz; desna vitičasta zagrada treba započeti liniju i biti uvučena do početka složeog izraza.
* Zagrade se koriste oko svih izraza, čak i pojedinačnih izraza, kada su deo kontrolne strukture, kao što je if-else ili for naredba. Ovo olakšava dodavanje izjava bez slučajnog uvođenja grešaka zbog zaborava na dodavanje zagrada.

## Return izjave

Return izjava sa vrijednosti ne bi trebala da koristi zagrade osim ako na neki način povratnu vrijednost čine očiglednijom.

Primjer:

|  |  |
| --- | --- |
| 1 | ... |
| 2 | return; |
| 3 | ... |
| 4 | return izracunajVrijednost(); |
| 5 | ... |
| 6 | return (uslov ? x : y); |
| 7 | ... |

## If, if-else, if-else-if else izjave

if, if-else, if-else-if else izjave treba da imaju sledeću formu:

|  |  |
| --- | --- |
| 1 | if (uslov) |
| 2 | { |
| 3 | izjave; |
| 4 | } |

|  |  |
| --- | --- |
| 1 | if (uslov) |
| 2 | { |
| 3 | izjave; |
| 4 | } |
| 5 | Else |
| 6 | { |
| 7 | izjave; |
| 8 | } |

|  |  |
| --- | --- |
| 1 | if (uslov) |
| 2 | { |
| 3 | izjave; |
| 4 | } |
| 5 | else if (uslov) |
| 6 | { |
| 7 | izjave; |
| 8 | } |
| 9 | else |
| 10 | { |
| 11 | izjave; |
| 12 | } |

## For izjave

For izjave treba da imaju sledeću formu:

|  |  |
| --- | --- |
| 1 | for (inicjalizacija; uslov; korak) |
| 2 | { |
| 3 | izjave; |
| 4 | } |

Prazan for treba da ima sledeću formu:

|  |  |
| --- | --- |
| 1 | for (inicjalizacija; uslov; korak) |
| 2 | {} |

Kada se koristi operator zarez u inicijalizaciji ili koraku u for izjavi, izbjegavati komplikovanje korišćenjem više od 3 varijable. Ako je potrebno, deklarisati izjave prije for petlje (za inisjalizaciju) ili na kraju petlje (za korak).

## While izjave

While izjave treba da imaju slijedeću formu:

|  |  |
| --- | --- |
| 1 | while (uslov) |
| 2 | { |
| 3 | izjave; |
| 4 | } |

## Do-while izjave

Do while izjav treba da imaju slijedeću formu:

|  |  |
| --- | --- |
| 1 | do |
| 2 | { |
| 3 | izjave; |
| 4 | } |
| 5 | while (uslov); |

## Switch izjave

Switch izjave treba da imaju slijedeću formu:

|  |  |
| --- | --- |
| 1 | switch (uslov) |
| 2 | { |
| 3 | case ABC: |
| 4 | izjave; |
| 5 | break; |
| 6 | case DEF: |
| 7 | izjave; |
| 8 | break; |
| 9 | default: |
| 10 | izjave; |
| 11 | break; |
| 12 | } |

## Try-catch izjave

Try-catch izjave treba da imaju slijedeću formu:

|  |  |
| --- | --- |
| 1 | try |
| 2 | { |
| 3 | izjave; |
| 4 | } |
| 5 | catcth (ExceptionClass e) |
| 6 | { |
| 7 | izjave; |
| 8 | } |

# Upravljanje memorijom

JAVA proces alokacije i de – alokacije objekata naziva se Memory management (Upravljanje memorijom). Koristi se automatski sistem za upravljanje memorijom koji se naziva Garbage collector (Sakupljač smeca). Sakupljač smeća osigurava da se neiskoršsteni prostor očisti i memorija oslobodi kada se ne koristi. Postoji i alternativni način oslobađanja memorije, a to je da se sve reference objekta postave na vrijednost ***null***.

# Upravljanje greškama i izuzecima

Rukovanje greškama i izuzecima u programskom jeziku JAVA predstavlja jedan od korisnijij alata tako da se uz pomoć njega može održavati normalan protok aplikacije. Greške koje se pojavljuju mogu pripadati jednoj od sljedećih grupa: *ClassNotFoundException*, *IOException*, *SQLException*, *RemoteException* i tako dalje.

5 ključnih riječi koje se koriste za rukovanje greškama i izuzecima:

**try** – koristi se da bi se odredio blok u kojem očekujemo izuzetak ili grešku. Ova ključna riječ ne može se koristiti sama za sebe već treba koristiti **catch** ili **finally.**

**catch** – koristi se za rukovanjem izuzecima. Obicno se prije ove komande koristi **try**, sto označava da se može koristiti sama za sebe

**finally** – koristi se za izvršavanje važnog koda programa. Izvršava se bez obzira obrađuje li se izuzetak ili ne

**throw** – služi za bacanje izuzetka

**throws** – koristi se za deklarisanje izuzetaka, ne baca izuzetak, vec određuje da u metodi može doći do izuzetaka

# Prenosivost

JAVA programi su prenosivi, sto označava da se izvorni kod programa može pokrenuti na bilo kom računarskom sistemu koji posjeduje JAVA interpreter. Takođe, izvorni program se može kompajlirati na bilo kom računaru koji ima JAVA kompajler.

# Ponovna upotreba

Ponovna upotreba koda u JAVA programskom jeziku je veoma poželjna, kao i kod svih objektno – orijentisanih programskih jezika. Najjednostavniji način ponovne upotrebe koda ogleda se u garanciji osnovnog nivoa, tako da svaka metoda obavlja jedinstven zadatak, a svaka klasa izvodi zadatak u cjelini. To se postiže pridržavnjem određenih pravila prilikom kodiranja:

* Pisati kod jasan, razumljiv,odnosno kod koji se može lako modifikovati i prilagođavati
* Nasledjivanje (koristiti funkcije i osnovne članove osnovne klase u drugim klasama)
* Pozivi funkcija (blokovi koda algoritma ili izvornog koda zasnovanog na logici se mogu izdvojiti u metodu)
* Premjestiti funkcionalnosti iz metoda instance klase
* Ne pisati kod koji nije potreban

# Problemi pri kompajliranju

Problem prilikom kompajliranja i pokretanja koda spada u jednu od dvije kategorije:

* Sintaksna greška - pronalazak greške kompajliranja, program nije uspio da izvrši kompajliranje
* Logička greška – obično se odnosi na grešku koja se javlja nakon uspješnog kompajliranja.

Najčeše greške koje se mogu pojaviti:

* Pogrešno imenovanje JAVA datoteke
* Izostavljena tačka - zarez
* Neusklađene zagrade

# Dodatno: sažetak smjernica

* **Organizacija koda i stila** – predstavlja smjernice koje je potrebno pratiti kako bismo napisali program koji je tačan i razumljiv
* **Komentari** – koristi se kao objašnjenje JAVA koda , čitanje i sprječavanje grešaka prilikom testiranja alternativnog koda
* **Imenovanje –** predstavlja pravila koja treba postovati prilikom imenovanja identifikatora kao sto su klase, varijable, metode, paketi i tako dalje
* **Deklaracije –** standard koji se mora poštovati prilikom navođenja imena i tipa podataka promjenjive
* **Izrazi i izjave –** izrazi koji se koriste prilikom rada sa klasama, varijablama, metodama i tako dalje
* **Upravljanje memorijom –** proces alokacije i dealokacije objekata u JAVA programskom jeziku
* **Rukovanje greskama i izuzecima –** način održavanja toka programa na način da su greške i izuzeci uzeti u obzir
* **Prenosivost –**  korištenje programa napisanih u JAVA progrmskom jeziku na drugim uređajima
* **Ponovna upotreba –** pravila koja se poštuju da bi se ostvarila maksimalna upotreba koda
* **Greske pri kompajliranju –** greške koje se mogu pojaviti prilikom kompajliranja programa