

Smjernice za programiranje

Verzija 1.0

Istorija revizija

|  |  |  |  |
| --- | --- | --- | --- |
| **Datum** | **Verzija** | **Opis** | **Autor** |
| 01.04.2024. | 1.0 | Opis konvencija i pravila koje je potrebno poštovoti tokom pisanja programskog koda softvera Law Bridge | Danijela Kovač |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

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 stil 4

3. Komentari 5

4. Imenovanje 5

5. Deklaracije 6

6. Izrazi i naredbe 6

7. Upravljanje memorijom 8

8. Rukovanje greškama i izuzecima 8

9. Prenosivost 8

10. Ponovna upotreba 9

11. Kompajliranje 9

12. Dodatak: sažetak smjernica 9

Smjernice za programiranje

# Uvod

Dokument Smjernice za programiranje obuhvata opis konencija i pravila o načinu pisanja programskog koda prilikom realizacije softvera Law Bridge.

## Svrha

Dokument Smjernice za programiranje ima za cilj postavljanje standard i pravila koje trebaju biti usvojena od strane svih članova tima koji rade na tazvoju projekta. Ovaj document osigurava jasnoću i dosljednost u interpretaciji zahtjeva tokom procesa programiranja, te sprječava eventualne nesuglasice i nedoumice među članocima tima. Takođe, njegova svrha je i minimiziranje nekonzistentnosti i razlika u stilu pisanja koda među različitim dijelovima projekta, što doprinosi boljoj čitljivosti, održavanju i proširivosti softverskog rješenja.

## Područje primjene

Kroz jasno definisana pravila i smjernice u dokumentu Smjernice za programiranje, osigurava se konzistentnost u načinu programiranja unutar advokatskog ureda Law Bridge, omogućiti tako efikasniju razmjenu znanja i resursa među članovima tima.

## 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

[1] [Software Engineering | Coding - javatpoint](https://www.javatpoint.com/software-engineering-coding)

[2] [College of Science and Engineering | University of Houston-Clear Lake (uhcl.edu)](https://www.uhcl.edu/science-engineering/)

[3] [Code Conventions for the Java Programming Language: Contents (oracle.com)](https://www.oracle.com/java/technologies/javase/codeconventions-contents.html)

## Pregled

U nastavku dokumenta detaljno će biti objašnjene smjernice koje se odnose na organizaciju i stilizaciju koda, pravilno postavljanje komentara, strategije imenovanja varijabli i funkcija, deklaracije i izrazi, efikasno rukovanje greškama i iuzecima, kao i druge ključne elemente koji su relevantni za konzistentan i čitljiv način programiranja.

# Organizacija koda i stil

Prije započinjanja procesa programiranja, ključno je definisati stil pisanja koda i organizacijske principe kako bi se osigurala dosljednost i preglednost. Ovo je od suštinske važnosti zbog olakšavanja snalaženja u kodu i održavanja jedinstvenog izgleda kodova koje su napisali različiti članovi tima. Nedostatak definisanog stila mogao bi rezultirati raznolikošću pristupa kodiranju , što bi otežalo čitanje koda i izvršavanje budućih izmjena ili nadogradnji. Neophodno je poštovati sljedeća pravila pri pisanju koda:

1. Paketiranje: Klase koje čine logičku cjelinu trebaju biti srupisane u isti paket radi lakšeg pronalaženja i organizacije koda.
2. Prelazak u novi red: Saka naredba ili deklaracija treba biti smještena u zaseban red kako bi se poboljšala preglednost koda.
3. Uvlačenje redova: Sadržaj svakog bloka koda treba biti uvučen za jedan tab u odnosu na blok na višem nivou, što olakšava razumijevanje strukture koda.
4. Dužina redova: Linije koda ne bi trebale biti duže od 70 karaktera kako bi se izbjegla horizontalna pomicanja pri čitanju koda.
5. Prelamanje redova: Ako linija sadrži više od 70 karaktera, prelazak u novi red treba napraviti iza zareza ili operatora kako bi se očuvala preglednost.
6. Prazan prostor: Između operatora i operanada treba staviti jedan razmak, blokove koda treba odvajati jednim praznim redom, a suvušan prazan prostor koji ne doprinosi čitljivosti treba eliminisati.

Ovi principu osiguravaju čist i dosljedan stil kodiranja koji olakšava saradnju unutar tima i održavanje softvera dugoročno.

# Komentari

Izvorni kod programa piše se na programskom jeziku JAVA, pa je komentare moguće pisati na jedan od dva sljedeća načina:

* Jednolinijski komentari, pišu se iza znakova //
* Višelinijski komentari, pišu se između znakova /\* i \*/.

Osim ova dva navedena tipa komenata, postoje još i komentari dokumentacije kojji se pišu između znakova /\*\* i \*\*/.

Komentari su neophodni kako bi se dodatno pojasnila svrha i funkcionalnost svake odvojene cjeline koda, ali i za svaku pojedinačnu naredbu kada je to potrebno. Oni igraju ključnu ulogu u olakšavanju razumijevanja koda kasnije, omogućavajući programerima da brže identifikuju svrhu i kontekst svake linije koda. Kroz detaljne komentare, programeri mogu bolje razumijeti razmišljanja i namjere autora koda, štp umanjuje mogućnost nesporazuma i pogrešnih tumačenja. Osim toga, komentari služe kao dokumentacija koja olakšava održavanje koda tokom vremena, omogućavajući timu da brže pronalazi i rješava probleme ili vrši izmjene. Stoga, pravilno postavljani komentari su ključni za održaanje čitljivosti, razumljivosti i dugoročne upotrebljivosti softverskog rješenja.

# Imenovanje

Da bi se postigla jednoličnost koda, od ključne je važnosti definisati pravila za imenovanje paketa, klasa, funkcija, varijabli i ostalih elemenata koji se koriste u programskom kodu. U nastavku su navedena i opisana ova pravila kako bi se osiguralo dosljedno i intuitivno imenovanje elemenata:

1. Imena varijabli: Imena varijabli se pišu bez razmaka, pri čemu prvo slovo prve riječi ostaje malo, dok je prvo slovo svake sljedeće riječi veliko. Prvi znak u imenu mora biti slovo, dok ostali mogu biti slova ili brojevi. Ime varijable treba jasno opisati njen značaj, kako bi se intuitivno razumjelo što predstavlja. Na primjer: varijabla, imeVarijable, imeJedneVarijable, imeVarijable1...
2. Imena konstanti: Imena konstanti pišu se velikim slovima, s odvojenim riječima donjom crtom. Ova imena također moraju biti smislena i povezana s značenjem konstante. Primjeri uključuju: POCETNA\_VRIJEDNOST, PI...
3. Imena metoda: Imena metoda pišu se na isti način kao i imena varijabli. Na primjer: promijeniBoju(), izracunajVrijednost()...
4. Imena klasa i interfejsa: Za imenovanje klasa i interfejsa vrijede ista pravila kao i za imenovanje varijabli i metoda, s tim da imena klasa i interfejsa počinju velikim slovom. Primjeri uključuju: Element, KutijaSaAlatkama...
5. Imena paketa: Imena paketa se pišu malim slovima bez razmaka. Ime paketa treba jasno opisati skup klasa koje paket sadrži, odnosno zajedničku osobinu na osnovu koje su klase povezane i smještene u isti paket. Primjeri imena paketa mogu biti: model, actionlisteners...
6. Pridržavanje ovih pravila omogućuje dosljednost i preglednost u kodiranju, olakšavajući razumijevanje i održavanje programskog koda tijekom vremena.

# Deklaracije

Deklaracije varijabli se najbolje postavljaju na početku blokova naredbi radi postizanja čitljivijeg i preglednijeg koda. Preporučuje se izbjegavanje deklarisanja varijabli po potrebi, tj. tamo gdje se varijabla koristi, jer to može dovesti do nejasnoća u kodu. Inicijalizacija varijabli trebala bi se obaviti odmah prilikom deklaracije, osim ako početna vrijednost zavisi od dijela koda koji se nalazi ispod deklaracije.

Prilikom deklarisanja klasa i metoda, važno je poštovati određena pravila radi smanjenja iskorištenog prostora i povećanja preglednosti koda. To uključuje odsustvo razmaka između imena metode ili klase i otvorene zagrade koja označava početak bloka naredbi. Otvorena vitičasta zagrada treba biti napisana u istom redu kao i deklaracija, dok se zatvorena vitičasta zagrada ne piše u redu gdje se nalazi posljednja naredba bloka, već u novom redu, osim ako je blok naredbi prazan, tada se piše odmah iza otvorene vitičaste zagrade (u istom redu). Ova pravila doprinose čišćem i organiziranijem izgledu koda, olakšavajući njegovo razumijevanje i održavanje.

# Izrazi i naredbe

Prilikom pisanja programskog koda, poptrebno je pisati u zasebnom redu kako bi se olakšalo razumijevanje logike izraza, posebno kada su izrazi složeniji. Ovaj pristup omogućava programerima da jasno vide svaki dio izraza i njegovu ulogu u funkcionalnosti koda. U slučaju složenijih izraza, koriste se zagrade kako bi se definisali prioriteti izvršenja, čime se osigurava tačnost izračunavanja rezultata.

Najčešće korištene naredbe u programskom kodu su if-else i switch naredbe, return naredbe i petlje. Za svaku od ovih naredbi postoje određena pravila koja treba slijediti radi očuvanja čitljivosti i efikasnosti koda. U nastavku su data pravila za pisanje ovih naredbi kako bi se osigurala konzistentnost i jasnoća u kodiranju.

1. Return naredbe imaju sljedeći izgled:



1. If-else if-else naredbe imaju sljedeći izgled:



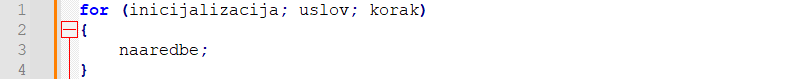




1. Switch naredba treba da ima sljedeći izgled:



1. Naredba For treba da ima sljedeći izgled:



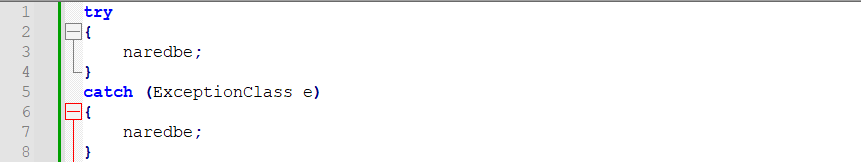
1. Naredba while treba da ima sljedeći izgled:



1. Naredba do while naredbe treba da imaju sljedeći izgled:



1. Naredba try-catch naredbe treba da imaju sljedeći izgled:



# Upravljanje memorijom

Prilikom razvijanja programskog koda, važno je težiti ka pisanju što kompaktnijeg koda kako bi se maksimizirala efikasnost korištenja memorije. Dobra praksa uključuje izbjegavanje nepotrebnih deklaracija, poput stvaranja nove pomoćne varijable za svaku petlju, umjesto toga, ako je moguće, koristiti jednu istu varijablu. Osim toga, umjesto ponovnog pisanja istog koda na više mjesta, preporučuje se smještanje tog koda u jednu funkciju kako bi se uštedio memorijski prostor.

Primjena ovih principa omogućava efikasno iskorištavanje resursa, ali istovremeno je važno osigurati da takva optimizacija ne ugrozi ispravnost rada programa. Potrebno je pažljivo provjeriti da optimizacija koda ne dovodi do nastanka grešaka ili konflikata. Time se postiže ravnoteža između efikasnosti memorije i funkcionalnosti programa, čime se osigurava da aplikacija radi ispravno i učinkovito.

# Rukovanje greškama i izuzecima

U toku procesa razvijanja bilo kojeg dijela programskog koda gdje postoji mogućnost pojave greške ili izuzetka, kao što je na primjer povezivanje s bazom podataka i prikupljanje podataka iz nje, ključno je osigurati da se program može oporaviti u takvim situacijama i nastaviti normalno funkcionirati. U programskom jeziku JAVA, postoji nekoliko elemenata koji omogućavaju upravljanje izuzetcima, uključujući try-catch-finally blokove i throw i throws deklaracije.

U okviru try bloka se nalazi kod u kojem može doći do izuzetka ili greške. U catch bloku se specifično definiše kako se taj izuzetak "hvata" i obrađuje, pružajući mogućnost programu da se prilagodi na određeni način ako dođe do problema. S druge strane, finally dio se koristi za izvršavanje određenog koda bez obzira na to je li došlo do izuzetka ili ne, što je korisno za čišćenje resursa ili izvršavanje završnih operacija.

Ovi mehanizmi omogućavaju programerima da efikasno upravljaju greškama i izuzecima, osiguravajući time stabilnost i pouzdanost njihovih aplikacija.

# Prenosivost

Prilikom pisanja koda važno je osigurati da on bude prenosiv i nevezan za određeni računar ili mašinu, što znači da treba izbjegavati korišćenje apsolutnih putanja. Apsolutne putanje sadrže lokacije na trenutnom računaru na kojem je kod pisan, što bi moglo uzrokovati nemogućnost pokretanja programa na drugim računarima. Umjesto toga, preporučuje se korišćenje relativnih putanja ili dinamičkog pronalaženja putanja unutar samog koda, čime se osigurava da program može pravilno funkcionirati na različitim računarima i platformama. Ova praksa doprinosi većoj fleksibilnosti i prenosivosti softverskih rješenja.

# Ponovna upotreba

Ukoliko se pridržava svih prethodno navedenih pravila prilikom pisanja izvornog koda, to omogućava da cijeli program ili njegovi dijelovi budu ponovno iskoristivi. Važno je organizirati kod na način da svaka cjelina (metoda ili klasa) obavlja precizno određeni posao, što omogućava njihovo lako ponovno korištenje u drugim projektima. U tom scenariju, potrebno je samo znati kako pozvati tu metodu ili klasu, čime se dodatno olakšava razvoj novih aplikacija i poboljšava efikasnost u razvoju softvera. Ova praksa omogućava veću modularnost i ponovno iskorištavanje koda, što doprinosi bržem i efikasnijem razvoju softverskih rješenja.

# Kompajliranje

Prilikom pisanja koda, važno je biti pažljiv kako bi se izbjegle sintaksne greške, koje bi mogle spriječiti kompilaciju koda. Sintaksne greške se odnose na kršenje pravila programskog jezika prilikom pisanja već definisanih imena elemenata, poput promjenljivih, funkcija ili klasa. Osim toga, također je bitno izbjeći logičke greške koje, iako neće spriječiti kompiliranje koda, mogu dovesti do neispravnog funkcionisanja programa. Logičke greške su problemi u algoritmu ili logici programa koji rezultiraju neželjenim ponašanjem ili neispravnim izlazom programa. Stoga je ključno pažljivo razmotriti svaki dio koda kako bi se osigurala njegova tačnost i funkcionalnost. Redovno testiranje i revizija koda također su važni koraci u otkrivanju i ispravljanju sintaksnih i logičkih grešaka prije nego što se program uvede u upotrebu.

# Dodatak: sažetak smjernica

Dokument „Smjernice za programiranje“ obuhvata teme koje se odnose na efikasno pisanje koda. Ovdje su pojedinosti o svakom odjeljku:

* *Organizacija i stil koda:* Ovaj odjeljak pruža opšte smjernice o načinu na koji se kod treba pisati i organizirati kako bi bio čitljiv i jednostavan za razumijevanje.
* *Komentari*: Ovdje se opisuju različiti načini pisanja komentara u kodu, što omogućava programerima da dodaju korisne informacije koje olakšavaju razumijevanje koda drugima ili sebi u budućnosti.
* *Imenovanje*: Pravila za davanje imena elementima programa su detaljno opisana u ovom odjeljku kako bi se osigurala dosljednost i jasnoća u nazivima promjenljivih, funkcija i ostalih elemenata koda.
* *Deklaracije*: Ovdje se navodi na koji način treba pisati deklaracije u kodu kako bi se osigurala pravilna inicijalizacija i upotreba promjenljivih i drugih objekata.
* *Izrazi i naredbe*: Smjernice za pisanje izraza i naredbi pružaju korisne savjete kako bi se osigurala efikasnost i čitljivost koda.
* *Upravljanje* *memorijom*: Ovaj odjeljak daje preporuke za pisanje koda koji optimalno koristi memorijski prostor kako bi se izbjegli problemi s prekomjernom potrošnjom memorije.
* *Rukovanje* *greškama* *i* *izuzecima*: Opisuje se kako pravilno rukovati greškama i izuzecima kako bi se osiguralo stabilno i pouzdano izvršavanje programa.
* *Prenosivost*: Savjetuje se kako pisati kod koji se može izvršavati na različitim računarima i platformama kako bi se osigurala maksimalna prenosivost.
* *Ponovna* *upotreba*: Ovdje se daju smjernice za pisanje koda koji je dizajniran s mogućnošću ponovne upotrebe u drugim projektima, čime se štedi vrijeme i resursi.
* *Kompajliranje*: Opisuje se kako pisati kod tako da se može kompajlirati bez poteškoća, osiguravajući da nema sintaksnih ili logičkih grešaka koje bi mogle ometati proces kompilacije.