

Amperia

Smjernice za programiranje

Verzija 1.0

Istorija revizije

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 30.04.2023. | 0.1 | Prve četiri tačke uvoda | Bojan Pandurević |
| 7.05.2023. | 0.2 | Urađena organizacija i stil koda, komentari i kreirana tabela za narednu sekciju-Imenovanja | Bojan Pandurević |
| 08.03.2023. | 1.0 | Dorada ostalih sekcija | Bojan Pandurević |

Sadržaj

[1. Uvod 4](#_Toc134458256)

[1.1 Svrha 4](#_Toc134458257)

[1.2 Područje primjene 4](#_Toc134458258)

[1.3 Definicije, akronimi i skraćenice 4](#_Toc134458259)

[1.4 Reference 4](#_Toc134458260)

[1.5 Pregled 4](#_Toc134458261)

[2. Organizacija i stil koda 4](#_Toc134458262)

[2.1 Identacija 4](#_Toc134458263)

[2.2 Dužina linije 4](#_Toc134458264)

[2.3 Razbijanje redova 4](#_Toc134458265)

[2.4 Upotreba razmaka i praznih linija 5](#_Toc134458266)

[2.5 Zagrade 5](#_Toc134458267)

[2.6 Pisanje kratkih pojedinačnih linija 5](#_Toc134458268)

[2.7 Povratne vrijednosti 5](#_Toc134458269)

[3. Komentari 6](#_Toc134458270)

[4. Imenovanje 6](#_Toc134458271)

[5. Deklaracije 7](#_Toc134458272)

[5.1 Broj deklaracija po liniji 7](#_Toc134458273)

[5.2 Pozicija 7](#_Toc134458274)

[5.3 Inicijalizacija 8](#_Toc134458275)

[5.4 Deklaracija klasa i interfejsa 8](#_Toc134458276)

[6. Izrazi i izjave 8](#_Toc134458277)

[6.1 Izrazi 8](#_Toc134458278)

[6.2 Jednostavane izjave 8](#_Toc134458279)

[6.3 Složene izjave 8](#_Toc134458280)

[6.4 Return izjave 8](#_Toc134458281)

[6.5 If, if-else, if-else-if else izjave 8](#_Toc134458282)

[6.6 For petlje 9](#_Toc134458283)

[6.7 While petlje 9](#_Toc134458284)

[6.8 Do-while petlje 9](#_Toc134458285)

[6.9 Switch izjave 9](#_Toc134458286)

[6.10 Try-catch izjave 10](#_Toc134458287)

[7. Upravljanje memorijom 10](#_Toc134458288)

[8. Upravljanje greškama i izuzecima 10](#_Toc134458289)

[9. Prenosivost 10](#_Toc134458290)

[10. Ponovno korištenje 10](#_Toc134458291)

[11. Problemi pri kompajliranju 11](#_Toc134458292)

Smjernice za programiranje

# Uvod

*Smjernice za programiranje* sadrže detaljna uputstva za pisanje Java programskog koda za razvoj grafičkog editora Amperia i namijen je za programere koji žele raditi na njemu.

## Svrha

Svrha Smjernica za programiranje je da navede konvencije programiranja kojima će se služiti programeri tokom izrade aplikacije. Ovo je vrlo važno kako bi se izbjegla situacija da se jedan kod piše različitim stilovima što izaziva nepreglednost istog i otežava održavanje aplikacije.

## Područje primjene

*Smjernice za programiranje* utiču na programski kod *Amperia* softverskog alata. Realizacija se ograničava na razvojno okruženje *Eclipse* i programski jezik Java uz dodatak Swing seta alata za GUI.

## Definicije, akronimi i skraćenice

Definicije, akronimi i skraćenice korišteni u *Smjernicama za programiranje* detaljno su objašnjeni u dokumentu Rječnik[1] koji je dio projektne dokumentacije.

## Reference

[1] Rječnik

[2] Java coding standard(<https://www2.hawaii.edu/~walbritt/ics211/materials/standard.htm>)

[3] Code conventions for the Java programming language (https://www.oracle.com/java/technologies/javase/codeconventions-contents.html)

## Pregled

U nastavku dokumenta opisana je organizacija pisanja koda, način pisanja komentara, deklaracija, imenovanja, izraza i izjava, upravljanja greškama i izuzecima kao i rješavanje potencijalnih problema prilikom kompajliranja.

# Organizacija i stil koda

## Identacija

Za jedinicu uvlačenja uzimaju se 4 razmaka, što je ekvivalentno jednom pritisku tastera Tab. Razvojno okruženje Eclipse ima integrisan formataer koji brine o pravilnom uvlačenju dijelova koda.

## Dužina linije

Dužina linije treba da bude manja od 80 karaktera.

## Razbijanje redova

Ukoliko izraz ne može da ispoštuje gore navedeno pravilo potrebno ga je nastaviti pisati u sljedećem redu prema sljedećim pravilima:

* Razbiti nakon zareza,
* Razbiti nakon operatora,
* Poravnanje nove linije izvršiti na način da njen početak bude u istom nivou kao početak izraza u prethodnom redu,
* Ukoliko primjena prethodnih koraka ne doprinosi čitljivosti koda, koristiti uvlačenje od 8 razmaka.

## Upotreba razmaka i praznih linija

* Nije dozvoljena upotreba razmaka između imena metode i lijeve zagrade. Npr. obrisi(int a), a ne obrisi (int a).
* Jedan razmak između ključnih riječi (if, while, for...) i lijeve zagrade.
* Razmak sa obje strane pri korištenju binarnih operatora.
* Jedan razmak poslije zareza.
* Bez razmaka nakon unarnih operatora(++, --, -).

## Zagrade

Vitičaste zagrade se koriste pri pisanju tijela bilo koje funkcije, petlje ili uslova, čak i kada je ono prazno ili sadrži samo jednu liniju koda. Male zagrade se koriste u izrazima sa mnogo operatora radi preglednosti i iskazivanja prioriteta. Npr.

|  |  |
| --- | --- |
| 1 | while((a==b)&&(a%2!=0)) |

## Pisanje kratkih pojedinačnih linija

* Nije dozvoljeno pisanje više izraza u jednoj liniji.
* Odstupanje od ograničenja u vidu maksimalnih 80 karaktera u jednoj liniji postoji kod pisanja URL-a.

## Povratne vrijednosti

Pri pisanju metoda koje vraćaju vrijednost potrebno je izbjegavati redudantni kod.

|  |  |
| --- | --- |
| 1 | if (a) |
| 2 | { |
| 3 | return true; |
| 4 | } |
| 5 | else |
| 6 | { |
| 7 | return false; |
| 8 | } // ovakav način pisanja nije dobar jer stvara nepotreban višak koda. Umjesto toga pišemo na sljedeći način. |
| 1 | return a; |

# Komentari

Pisanje komentara uz kod je jako dobra praksa. Uprkos tome što se nastoji da sam kod bude što čitljiviji, pogodno je imati ova dodatna objašnjenja-kako u toku razvoja softvera, tako i tokom njegovog održavanja.

|  |  |  |
| --- | --- | --- |
| Tip komentara | Pravila korištenja | Primjer |
| **Dokumentacija** | Koristi se prije deklaracija klase, interfejsa ili funkcije u svrhu njihovog dokumentovanja.Nakon komentara generiše se *javadoc.* | /\*\*  \* @Author  \*@Version 1.0  \*  \*  \*/  Public class Klasa |
| **C tip** | Koristi se kako bi svrha koda bila jasna na prvi pogled ili da pruži informaciju koja nije pružena u samom kodu. | /\*  Komentar u svrhu dodatnog objašnjenja svrhe koda.  \*/ |
| **Jednolinijski** | Koristi se unutar funkcije radi dokumentovanja njene logike. Ne koristiti na mjestima gdje nije neophodno, odnosno gdje je logika jasna. | //sortiranje  //dodavanje elementa |

.

# Imenovanje

|  |  |  |
| --- | --- | --- |
| **Tip identifikatora** | **Pravila imenovanja** | **Primjeri** |
| Paketi | Pišu se malim početim slovima. Uzastopne riječi se odvajaju tačkom, a nikako donjom crtom ili kombinacijom velikih i malih slova. U slučaju da se u imenu paketa nalazi niz karaktera koji u Javi predstavlja ključnu riječ, nakon njega se stavlja donja crta. | projekat.klase.main  if\_.podaci |
| Klase | Pišu se velikim početnim slovom. Uzastopne riječi se odvajaju kombinacijom velikih i malih slova tako da se svaka nova riječ piše velikim početnim slovom. Po vrsti riječi su imenice. Ne koristiti skraćenice osim ako su u pitanju pojmovi uopšte poznati po skraćenicama(HTML, URL...) | class Student;  class PrijavaIspita; |
| Interfejsi | Pravila imenovanja ista kao i kod klasa. | interface Kupovina;  interface ServisOpreme; |
| Metode | Pišu se malim početnim slovom, po pravilu su glagoli, a uzastopne riječi se pišu spojeno velikim početnim slovom. | ispisati();  dodajNoviArtikal(Artikal artikal); |
| Varijable | Pisati na isti način kao i metode. Ime varijable treba da bude kratko i jasno, odnosno da na prvi pogled bude razumljivo šta ona predstavlja. Ne davati imena koja sadrže samo 1 karakter, osim kada su u pitanju privremene varijeble-npr. brojači. | int size;  String prezime;  Int i; |
| Konstante | Pišu se u potpunosti velikim slovima, a odvajaju se donjom crtom. | int MAX\_SIZE=45;  double PI=3.14; |

# Deklaracije

## Broj deklaracija po liniji

U jednoj liniji pisati samo jednu deklaraciju radi lakšeg pisanja komentara.

|  |  |
| --- | --- |
| 1 | int indeks; //identifikacioni broj studenta |
| 2 | Boolean obnova; //promjenljiva koja označava da li student obnavlja tekuću godinu studija |

## Pozicija

Deklaracija se vrši na početku bloka koda, unutar vitičastih zagrada. Izbjegavati deklarisanje varijabli tek kada su neposredno potrebne.

|  |  |
| --- | --- |
| 1 | public void metoda() |
| 2 | { |
| 3 | int broj; |
| 4 |  |
| 5 | if (uslov) |
| 6 | { |
| 7 | String tekst: |
| 8 | ... |
| 9  10 | }  ... |
| 11 | } |

Jedini izuzetak ovog pravila je kod for petlji:

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

Zabranjene su lokalne deklaracije koje sakrivaju globalne.

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

## Inicijalizacija

Lokalne varijable inicijalizovati tamo gdje su deklarisane. Od ovoga se odstupa onda kada vrijednost varijable zavisi od nekih prethodnih izračunavanja. Inicijalizaciju vršiti onda kada je zaista potrebna.

## Deklaracija klasa i interfejsa

Deklaracija klasa i interfejsa se počinje otvorenom vitičastom zagradom u istoj liniji sa imenom klase/interfejsa. Zatvorena zagrada nalazi se na kraju sama u jednoj liniji, osim kada je u pitanju prazano tijelo. Tada se zagrade nalaze jedna do druge bez razmaka.

|  |  |
| --- | --- |
| 1 | class Klasa1 { |
| 2 | int broj; |
| 3 | String ime; |
| 4  5  6 | }  class Klasa2{} |

# Izrazi i izjave

## Izrazi

Izrazi izvršavaju operacije nad podacima. Pišu se nedvosmisleno i precizno. Kako bi se ovo ispoštovalo, kod složenijih izraza je potrebno koristiti zagrade.

|  |  |
| --- | --- |
| 1 | x + y / z //neispravno |
| 2  3 | (x + y) / z //ispravno  x + (y / z) //ispravno |

## Jednostavane izjave

Svaka linija treba da sadrži jednu izjavu.

|  |  |
| --- | --- |
| 1 | kolicina++; cijena+=10; //neispravno |
| 2  3 | kolicina++; //ispravno  cijena+=10; //ispravno |

## Složene izjave

## Return izjave

Return izjava ne bi trebala da sadrži zagrade osim onda kada povratnu vrijednost čini uočljivijom.

|  |  |
| --- | --- |
| 1  2  3 | return;  return cijena;  return( pokvarenUredjaj? pokvaren: ispravan) |

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

If, if-else i if-else-if else izjave se pišu na sljedeći način:

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

|  |  |
| --- | --- |
| 1 | if (uslov) //provjera da li je uslov ispunjen |
| 2 | { |
| 3 | izjave1; |
| 4 | } |
| 5 | Else //blok u koji se ulazi u koliko uslov nije ispunjen |
| 6 | { |
| 7 | izjave2; |
| 8 | } |

|  |  |
| --- | --- |
| 1 | if (uslov) //provjera da li je prvi uslov ispunjen |
| 2 | { |
| 3 | izjave1; |
| 4 | } |
| 5 | else if (uslov) //ukoliko nije provjerava se naredni |
| 6 | { |
| 7 | izjave2; |
| 8 | } |
| 9 | Else //blok u koji se ulazi ukoliko ni jedan uslov nije ispunjen |
| 10 | { |
| 11 | izjave3; |
| 12 | } |

## For petlje

For petlja ima sljedeći oblik:

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

Kada imamo više od tri izjave u inicijalizaciji ili u koraku, onda je potrebno inicijalizovati varijable prije for petlje ili na kraju petlje.

## While petlje

While petlja ima sljedeći oblik:

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

## Do-while petlje

Do while petlja ima sljedeći oblik:

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

## Switch izjave

Switch izjave imaju sljedeći oblik:

|  |  |
| --- | --- |
| 1 | switch (uslov) |
| 2 | { |
| 3 | case 1: |
| 4 | izjave1; |
| 5 | break; |
| 6 | case 2: |
| 7 | izjave2; |
| 8 | break; |
| 9 | default: |
| 10 | izjave3; |
| 11 | break; |
| 12 | } |

## Try-catch izjave

Try-catch izjave imaju sljedeći oblik:

|  |  |
| --- | --- |
| 1 | try |
| 2 | { |
| 3 | izjave1; |
| 4 | } |
| 5 | catch (ExceptionClass e) |
| 6 | { |
| 7 | izjave2; |
| 8  9  10 | } finally{  Izjave3;  } |

# Upravljanje memorijom

Koristi se sitem koji radi u pozadini automatski upravlja memorijom i koji se naziva *garbage collector*. On ima ulogu da briše neiškorištene objekte i time oslobađa prostor u memoriji.

# Upravljanje greškama i izuzecima

* Try-catch blok se završava finally blokom.
* Svaku grešku i izuzetak čuvati u log fajlu.
* Prilikom kreiranja novog tipa izuzetka potrebno ga je dokumentovati.
* Potrebno obezbijediti nesmetan rad korisniku pri nastanku izuzetka.
* Ako postoji više tipova izuzetaka, potrebno je postaviti blokove tako da se izuzeci hvataju redom od najniže ka najvišoj podklasi.

# Prenosivost

Programi kodirani u Java programskom jeziku su prenosivi i mogu se kompajlirati na svakom računaru koji ima instaliranu JVM. Izvršni kod se ne mora mijenjati da bi zadovoljio posebne potrebe bilo kog računarskog sistema.

# Ponovno korištenje

Kako bi kod bio pogodan za ponovno korištenje potrebno je pridržavati se sljedećeg:

* Izbjeći ponavljanje koda i pisanje nepotrebnih dijelova koda.
* Klase i metode treba da obavljaju jednu funkciju.
* Napraviti testove za klase i učiniti klase lakim za testiranje.
* Što više koristiti apstrakciju.
* Napraviti kod takvim da bude pogodan za proširivanje u budućnosti.
* Ukloniti poslovnu logiku ili *main* kod iz svakog frejmvork koda.

# Problemi pri kompajliranju

Potrebno je posjedovati sve što je potrebno za uspješno kompajliranje, a što je navedeno u sekciji Prenosivost. Da bi se kod mogao kompajlirati neophodno je otkloniti sve sintaksne greške. U slučaju nastanka greške koristiti poruke u kodu.