Java kódolási konvenciók

Fordítás: Erdei Szabolcs (erdeisz@westel900.net), Viczián István (viczus@freemail.hu)
Eredeti szöveg: http://java.sun.com/docs/codeconv/
Fordítás: http://dragon.unideb.hu/~vicziani

Bevezetés

Miért van szükség kódolási konvenciókra?

Számos ok van, amiért fontos a kódolási konvenciók használata:

- A szoftver fejlesztési idejének 80 százalékát fordítják karbantartásra.
- A karbantartás nehezen kivitelezhető egy ember által a fejlesztés egész ideje alatt.
- A kódolási konvenciók növelik a forrás olvashatóságát, a fejlesztőknek lehetővé teszi, hogy az ismeretlen kódot gyorsabban és mélyebben értsék meg.
- Ha közzéteszel egy forráskódot, biztosnak kell lenned abban, hogy ugyanolyan jól felépített, és tiszta, mint az összes eddigi munkád.

Készítők

Ebben a dokumentumban leírt kód konvenciókat a Sun Microsystems, Inc. dolgozói készítették a Java nyelv specifikációjára építve. Név szerint: King, Patrick Naughton, Mike DeMoney, Jonni Kanerva, Kathy Walrath, and Scott Hommel. A dokumentumot magyarra fordította Erdei Szabolcs (erdeisz@westel900.net) és Viczián István (viczus@freemail.hu).
Fordításkor a forrástöredékeket nem fordítottuk, így ékezetes karaktereket sem használunk bennük. A forrásban szereplő megjegyzések szövegeit azonban fordítottuk, és ékezetes karaktereket is használtunk.

Fájlnevek

Ez a fejezet felsorolja az általánosan használt kiterjesztéseket, és neveket.

Kiterjesztések

Egy Java nyelvű forrás fájlnak `.java` kiterjesztéssel kell rendelkeznie, míg a lefordított bájtkód a `.class` kiterjesztést kapja.

Általánosan használt nevek

Általánosan használt neveket használjunk makefile estén (`GNUmakefile`), ha a gnumake alkalmazást használjuk az alkalmazás fordítására, és az adott könyvtár tartamának szöveges leírását tartalmazó fájl esetén (`README`).

Fájlok felépítése

Egy fájl szakaszokból áll, melyeket üres sorokkal kell elválasztani, illetve ajánlott megjegyzéseket tenni, azonosításképpen.
A 2000 programsornál hosszabb fájlokat lehetőleg kerüljük.
A dokumentum végén található egy példa a megfelelően formázott Java forrásfájlra.

Java forrásfájlok

Minden Java forrásfájl tartalmaz egy publikus osztályt vagy interfészt. Ha ehhez tartoznak privát osztályok vagy interfészek, akkor azokat nyugodtan elhelyezhetjük ebbe a fájlba. A publikus osztálynak vagy interfésznek kell az elsőnek lennie ebben a forrásban.

A Java forrásnak követnie kell a következő sorrendet:

- Kezdő megjegyzések.
- Csomag- illetve importdeklarációk.
- Osztály- és interfészdeklarációk

Kezdő megjegyzések.

Minden forrásnak az elején egy C-stílusú megjegyzésnek kell lennie, amely tartalmazza az osztály nevét, verzióval kapcsolatos információkat, dátumot és copyright megjegyzéseket:

```
/*
 * Osztálynév
 *
 * Verzióval kapcsolatos információk
 *
 * Dátum
 *
 * Copyright megjegyzések
 */
```

Csomag- és importdeklarációk

Az első sor a Java forrásban, mely nem megjegyzés, az a csomagdeklaráció. Ezután az importdeklarációk következhetnek. Például:

```
package java.awt;

import java.awt.peer.CanvasPeer;
```

Az egyedi csomag azonosító első tagja mindig kisbetűkből álljon, és top-level domain azonosító legyen, azaz com, edu, gov, mil, net, org vagy a kétbetűs ország azonosító, melyeket a ISO Standard 3166, 1981definiálja.

Osztály és interfész deklarációk

A következő táblázat leírja az osztály- vagy interfészdeklaráció elemeit olyan sorrendben, ahogy a forrásban szerepelniük kell.

	Az osztály- vagy interfészdeklaráció része	**Megjegyzések**
1	Osztály/interfész dokumentációs megjegyzés (/** */)	Lásd a ``Dokumentációs megjegyzések" című fejezetet.
2	Osztály/interfész fej	
3	Osztály/interfész implementációs megjegyzések, ha szükséges	Ez a megjegyzés tartalmazhat osztály vagy interfész szintű kiegészítő információkat, melyek nem kerültek be a dokumentációs megjegyzésbe.

4	Osztályváltozók	Először a nyilvános, majd a protected (protected), majd a félnyilvános tagok, végül a privát tagok.
5	Példányváltozók	Követve az előbbi sorrendet.
6	Konstruktorok	
7	Metódusok	Ezeket a metódusokat inkább a működés alapján kell csoportosítani, mint a hozzáférhetőség vagy láthatóság alapján. Például egy privát metódus szerepelhet két publikus metódus között. A cél, hogy a kód olvasása és megértése egyszerűbb legyen.

Behúzás

A behúzás alapegysége négy szóköz karakter. A négy szóköz elhelyezésének pontos módja nincs specifikálva (lehet használni szóközt vagy tab-ot is). A tab-nak pontosan nyolc szóközből kell állnia, nem négyből.

Sorhossz

A 80 karakternél hosszabb sorok kerülendőek, mivel több terminál és fejlesztőeszköz nem tudja megfelelően kezelni.
Megjegyzés: A példák a dokumentációkban általában rövidebbek, a sorhosszak nem nagyobbak, mint 70 karakter.

Sortörések

Ha egy kifejezés nem fér el egy sorban, akkor a sortörést a következő alapelvek szerint kell végezni:

- Sortörés egy vessző után.
- Sortörés egy operátor előtt.
- Magas szintű törések preferálása az alacsonyszintűekkel szemben.
- Az új sor elhelyezése az előző sorban lévő azonos szintű kifejezés kezdetéhez igazítva.
- Ha a fenti szabályok a kód összezavarásához vezetnének, vagy a kód a jobb margónál sűrűsödne, akkor használjunk nyolc szóközt a többszörös behúzás helyett.

Álljon itt két példa a metódushívásokon belüli sortörésekre:

```
someMethod{longExpression1, longExpression2, longExpression3,
      longExpression4, longExpression5);
var = someMethod1(longExpression1,
            someMethod2(longExpression2,
                  longExpression3));
```

A következő két példa az aritmetikai kifejezéseken belüli sortörésekre hoz példát. Az első az ajánlott, mivel a sortörés a magasabb helyen lévő zárójelezett kifejezésen kívül van.

```
longName1 = longName2 * (longName3 + longName4 - longName5)
          + 4 * longName6; // Ajánlott
```

```
longName1 = longName2 * (longName3 + longName4
          - longName5) + 4 * longName6; // Elkerülendő
```

Következik két példa a metódusdeklaráció behúzására. Az első a hagyományos eset. A második esetben a második és a harmadik sort nyolc szóközzel húztuk be, ugyanis a sorok a jobb oldalon sűrűsödnének a konvencionális megoldás használata esetén.

```
// Konvencionális behúzás
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
           Object andStilAnother) {
    ...
}

// Többszintű behúzás esetén nyolc szóköz használata
private static synchronized horkingLongMethodName(int anArg,
      Object anotherArg, String yetAnotherArg,
      Object andStilAnother) {
   ...
}
```

Sortörésnél `if` kifejezés esetén a nyolc szóközös szabályt alkalmazzuk, ha a konvencionális (4 szóközös) szabály a törzset túl bonyolulttá tenné. Példa:

```
//Ne használjuk ezt a behúzást
if ((condition1 && condition2)
    || (condition3 && condition4)
    || (condition5 && condition6)) { //A rossz sortörések miatt
    doSomethingAboutIt();            //ez a sor könnyen
}                                    //eltéveszthető

//Helyette használjuk ezt
if ((condition1 && condition2)
        || (condition3 && condition4)
        || (condition5 && condition6)) {
    doSomethingAboutIt();
}

//Vagy ezt
if ((condition1 && condition2) || (condition3 && condition4)
        || (condition5 && condition6)) {
    doSomethingAboutIt();
}
```

Álljon itt három példa a feltételes kifejezés használatára:

```
alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongBooleanExpression) ? beta
                                 : gamma;

alpha = (aLongBooleanExpression)
      ? beta
      : gamma;
```

Megjegyzések

A Java programokban kétfajta megjegyzések lehetnek: implementációs és dokumentációs megjegyzések. Az implementációs megjegyzések vannak C++ nyelvű forrásszövegben is, melyeket a /* */, vagy a // szimbólumok jelölnek. A dokumentációs megjegyzések csak Java nyelvben léteznek, ezeket a /** */ zárójelpárokkal kell jelölni. A javadoc programmal ezek a megjegyzések bekerülnek egy HTML fájlba sok egyéb információval együtt.

Az implementációs megjegyzésekkel a forráskód szövege magyarázható. A dokumentációs megjegyzések ezzel szemben a specifikációt magyarázzák, mellőzve az implementációs kérdéseket, és azokhoz a fejlesztőkhöz szólnak, melyeknek nincs szükségük az adott rész forráskódjára.

A megjegyzések áttekintést is nyújtsanak, illetve olyan plusz információkat tartalmazzanak, amelyek nem szembetűnők a kód egyszeri elolvasásakor. A megjegyzéseknek csak olyan idevágó információkat kellene tartalmaznia, melyek segítik a program olvashatóságát, illetve megértését.

Nem triviális tervezői döntések magyarázata odaillő, viszont kerülni kell az olyan információk ismétlését, melyek adottak (és világosak) a kódban. A redundáns adatok módosítása többletmunkát eredményez, és könnyen elévülhet. Ezért kerülni kell minden olyan megjegyzést, mely a program változása során érvényét veszthetí.

Megjegyzés: A megjegyzések aránytalanul nagy mennyisége takargathatja a kód gyenge minőségét. Ezért új megjegyzés beszúrása helyett érdemes elgondolkozni a kód érthetőbbé tételén.

A megjegyzéseket nem szabad körülrajzolni karakterekkel. A megjegyzésekbe sohasem lehet speciális karaktereket használni.

Implementációs megjegyzések formátumai

Az implementációs megjegyzéseknek négy alakja lehet: blokk, egysoros, utó- és sorvégi megjegyzések.

Blokk megjegyzések

A blokk megjegyzések használhatóak fájlok, metódusok, adatstruktúrák és algoritmusok leírására. Blokk megjegyzések használhatóak minden fájl és metódus elején. Ezen kívül máshol is elhelyezkedhetnek, például metódusokon belül. A blokk megjegyzéseknek egy metóduson belül azonos szinten kell lennie, mint a kód, melyet magyaráz.

A blokk megjegyzésnek egy üres sorral kell kezdődnie, mely elválasztja a megjegyzést a kódtól.

```
/*
 * Ez egy blokk megjegyzés.
 */
```

A blokk megjegyzések kezdődjenek /*- szimbólumsorozattal, mely jelzi a behúzást végző programnak (indent), hogy a megjegyzést nem lehet átformázni.

```
/* -
 * Ez egy blokk megjegyzés speciális formátumozással,
 * melynél az újraformázás kerülendő.
 *
 *    egy
 *          kettő
 *                három
 */
```

Megjegyzés: Ha nem használjuk az behúzást végző programot, akkor nem kell alkalmazni a /*- karaktersorozatot, de érdemes, abban az esetben, ha más akarja használni a mi kódunkon.

Egysoros megjegyzések

Rövid megjegyzések egy sorban alkalmazhatóak, azon a szinten, melyen az előző sor is van. Ha a megjegyzés nem írható egy sorba, akkor a blokk megjegyzést kell alkalmazni. Az egysoros megjegyzés előtt egy üres sornak kell állnia. Példa:

```
if (feltetel) {

    /* Ha a feltétel teljesül. */
    ...
}
```

Utómegjegyzések

Nagyon rövid megjegyzések kerülhetnek egy sorba a kódrészlettel, melyet magyaráznak, viszont megfelelő távolságra kell attól tenni. Ha több mint egy megjegyzés tartozik egy köteg programsorhoz, akkor azokat ugyanolyan távolságra kell tenni.

Itt egy példa az utómegjegyzésekre:

```
if (a == 2) {
    return TRUE;          /* speciális eset */
} else {
    return isPrime(a);    /* csak páratlan számok esetén */
}
```

Sorvégi megjegyzések

A // karakterekkel megjegyzésbe lehet tenni egy teljes sort, illetve egy sor végét. Nem szabad használni többsoros megjegyzések esetén, viszont lehet, ha a forráskódból több egymás utáni sort akarunk megjegyzésbe tenni. Példa a három használati módra:

```
if (foo > 1) {

    // Dupla dobás.
    ...
}
else {
    return false;       // Miért itt?
}
//if (foo > 1) {
//
//    // Dupla dobás.
```

```
//    ...
//}
//else {
//    return false;
//}
```

Dokumentációs megjegyzések

További információk a ``How to Write Doc Comments for Javadoc" című írásban, melyben megtalálhatók a dokumentációs megjegyzések tag-jei (@return, @param, @see):

http://java.sun.com/products/jdk/javadoc/writingdoccomments.html

Több információért a dokumentációs megjegyzésekkel és a javadoc programmal lásd a javadoc honlapját:

http://java.sun.com/products/jdk/javadoc/

A dokumentációs megjegyzések jellemzik a Java osztályokat, interfészeket, konstruktorokat, metódusokat és adattagokat. Minden dokumentációs megjegyzést a /** és */ karaktersorozatok határolnak, osztályonként, interfészenként és tagonként egyszer. Ennek a megjegyzésnek a deklaráció előtt kell szerepelnie:

```
/**
 * Az Example osztály ...
 */
public class Example { ...
```

Megjegyezzük, hogy a legfelső szintű osztályok és interfészek nincsenek behúzva, míg azok tagjai igen. Az osztályokhoz és interfészekhez tartozó dokumentációs megjegyzések első sora (/**) nincs behúzva, a következő sorok be vannak húzva egy karakterrel (hogy a csillagok egymás alá essenek). A tagoknál, beleértve a konstruktort is, a behúzás az első sorban négykarakternyi, míg öt a rákövetkezőkben.

Ha több információt kell megadni az osztályhoz, interfészhez, változóhoz vagy metódushoz, de nem akarjuk, hogy a dokumentációban megjelenjen, implementációs blokk, vagy egysoros megjegyzést kell használni azonnal a deklaráció után.

Dokumentációs megjegyzéseket nem lehet egy metóduson vagy konstruktoron belül hagyni, ugyanis a Java a megjegyzés *utáni* első deklarációra vonatkoztatja a megjegyzést.

Deklarációk

Deklarációk soronkénti száma

Soronként egy deklaráció ajánlott, a megjegyzések miatt is. Azaz

```
    int level; // szint
    int size;  // tábla mérete
```

ajánlott

```
int level,size;
```

helyett.

Egy sorba ne tegyünk különböző típusú deklarációkat. Például:

```
int foo, fooarray[];  //HIBÁS!
```

Megjegyzés: A fenti példák egy szóköz helyet hagytak a típus és a név között. Másik elfogadható választási lehetőség a tabok használata, azaz:

```
int    level;            // szint
int    size;             // tábla merete
Object currentEntry;     // kiválasztott táblabejegyzés
```

Inicializálás

Érdemes a lokális változókat ott inicializálni, ahol deklarálva lettek. Az egyetlen eset, amikor ez nem használható, ha a kezdőérték valamilyen számítás eredménye.

Elhelyezés

A deklarációkat a blokk elejére tegyük. (Blokk a kapcsos zárójelek (``{ és }") által határolt rész.) Ne várjunk a deklarációval addig, míg a változót először használatba vesszük.

```
void myMethod() {
  int int1 = 0;     // metódus törzsének kezdete

  if (condition) {
      int int2 = 0; // if törzsének kezdete
      ...
  }
}
```

A szabály alóli kivétel a `for` ciklus fejében deklarált változó:

```
for (int i = 0; i < maxLoops, i++) { ... }
```

Azon deklarációk, melyek egy magasabb szinten deklarált nevet elfednek, kerülendőek. Például nem deklarálhatunk ugyanolyan névvel egy másik változót:

```
int count;
...
myMethod() {
  if (condition) {
      int count;          // KERÜLENDŐ!
      ...
    }
    ...
}
```

Osztály és interfész deklarációk

Java osztályok és interfészek deklarációjánál a következő szabályokra kell ügyelni:

- Nincs szóköz a metódus név és a formális paraméterlistát kezdő nyitó zárójel között.
- A metódus fejével egy sorban kell a nyitó kapcsos zárójelnek (``{``) lennie.
- A záró kapcsos zárójelnek (``}'') egymagában kell a sorban állnia, egy oszlopban a fej első karakterével, kivéve, mikor a metódus törzse üres, ekkor egyből a nyitó zárójel után kell állnia.

```
class Sample extends Object {
    int ivar1;
    int ivar2;

    Sample(int i, int j) {
      ivar1 = i;
      ivar2 = j;
    }

    int emptyMethod() {}

    ...
}
```

- Metódusokat üres sorral kell egymástól elválasztani.

Utasítások

Egyszerű utasítások

Minden sornak csak egy utasítást szabad tartalmaznia, például:

```
    argv++;    // Helyes
    argc++;    // Helyes
    argv++; argcc++;     // KERÜLENDŐ!
```

Összetett utasítások

Az összetett utasítások olyan utasítások listája, melyek kapcsos zárójelek között vannak: ``{ utasítások } ``.

- A közrefogott utasításokat egy szinttel beljebb kell húzni.
- A nyitó zárójelnek azon sor végén kell szerepelnie, mely megkezdi az összetett utasítást; a csukó zárójelnek sor elején kell szerepelnie, behúzva annyival, mint a kezdő utasítás.
- Zárójeleket akkor is használni kell, ha egy vezérlési szerkezet törzseként csak egy utasítást szeretnénk szerepeltetni. Ez megkönnyíti a törzs bővítését, anélkül, hogy valami hibát vétenénk a zárójelek elhagyása miatt.

A return utasítás

A `return` utasításnál nem szükséges használni a zárójeleket, ha a visszatérési érték így is nyilvánvaló. Példák:

```
    return;
```

```
return myDisk.size();

return (size ? size : defaultSize);
```

Az if, if-else, if else-if else utasítások

Az `if-else` szerkezeteket a következő alakban kell használni:

```
if (feltetel) {
  utasitasok;
}

if (feltetel) {
  utasitasok;
} else {
  utasitasok;
}

if (feltetel) {
  utasitasok;
} else if (feltetel) {
  utasitasok;
} else {
  utasitasok;
}
```

Megjegyzés: Az `if` szerkezeteknek mindig tartalmazniuk kell zárójeleket. Azaz kerülendő a következő forma:

```
if (feltetel)  // KERÜLENDŐ! HIÁNYZÓ ZÁRÓJELEK!
  utasitas;
```

A for utasítás

A `for` utasításnak a következő formát kell követnie:

```
for (inicializalas; feltetel; leptetes) {
  utasitasok;
}
```

Egy üres törzzsel rendelkező `for` utasításnak a következőképpen kell kinéznie:

```
for (inicializalas; feltetel; leptetes);
```

Ha az inicializálás vagy léptetés során használjuk a vessző operátort, azaz több változót is deklarálunk, tiltott a háromnál több változó használata. Ha szükséges, akkor ezt tegyük meg a for ciklus előtt, vagy a ciklus törzsének végén.

A while utasítás

A `while` utasítást a következő formában kell használni:

```
while (feltetel) {
  utasitasok;
```

```
    }
```

Az üres while utasítás alakja:

```
    while (feltetel);
```

A do-while utasítás

Formája:

```
    do {
        utasitasok;
    } while (feltetel);
```

A switch utasítás

A `switch` utasítást a következő alakban kell használni:

```
    switch (feltetel) {
    case ABC:
      utasitasok;
      /* továbblép */
    case DEF:
      utasitasok;
      break;

    case XYZ:
      utasitasok;
      break;

    default:
      utasitasok;
      break;
    }
```

Minden esetben, ha a vezérlés a következő eseten folytatódik (azaz hiányzik a `break` utasítás, használni kell egy megjegyzést a `break` helyén. Jelen esetben a `/* továbblép */` megjegyzést használtuk.

A try-catch utasítás

A `try-catch` utasítást a következő alakban kell használni:

```
try {
    utasitasok;
} catch (ExceptionClass e) {
    utasitasok;
}
```

A `try-catch` utasítást követheti `finally` utasítás, a vezérlés mindenképp eljut erre az ágra, függetlenül attól, hogy a `try`, vagy a `catch` blokk sikeresen végrehajtódott-e. Használata:

```
try {
    utasitasok;
} catch (ExceptionClass e) {
```

```
    utasitasok;
} finally {
    utasitasok;
}
```

White Space

Üres sorok

Az üres sorokkal a logikailag összetartozó kódrészeket emelhetjük ki, ami növeli az olvashatóságot.

Két üres sort kell használni a következő esetekben:

- a forrás fájl két bekezdése között;
- osztály és interfészdefiníciók között.

Egy üres sort kell használni a következő esetekben:

- metódusok között;
- egy adott metódus lokális változói és az első utasítás között;
- blokk vagy egysoros megjegyzés előtt;
- az egy metóduson belüli logikai szakaszok között az olvashatóság növelése érdekében.

Szóközök

Szóközöket kell használni a következő esetekben:

- Ha egy kulcsszót kerek zárójel követi, akkor el kell választani egy szóközzel. Például:

```
while (true) {
    ...
}
```

Megjegyzendő, hogy szóköz nem használható a metódus neve és nyitó kerek zárójele közt. Ez segít elkülöníteni a kulcsszavakat a metódushívásoktól.

- Az argumentum listában szóköznek kell követnie minden vesszőt.
- Minden kétoperandusú operátort a . kivételével el kell választani egy szóközzel az operandusaitól. Tilos szóközt használni az unáris operátor és operandusa között, mint például az előjelváltás („-”), a növelő („++”) és a csökkentő („—”) operátorok esetén. Példák:

```
a += c + d;
a = (a + b) / (c * d);

while (d++ = s++) {
    n++;
}
printSize("size is " + foo + "\n");
```

- A kifejezéseket egy `for` utasításban el kell választani szóközökkel. Példa:

```
for (expr1; expr2; expr3)
```

- A típuskényszerítéseket szóköznek kell követnie. Példa:

```
myMethod((byte) aNum, (Object) x);
myMethod((int) (cp + 5), ((int) (i + 3)) + 1);
```

Elnevezési konvenciók

Az elnevezési konvenciók könnyebben olvashatóvá és így jobban érthetőbbé teszik a forráskódot. Információt adhatnak az azonosító funkciójáról, ami javítja a kód érthetőségét.

Azonosító típus	**Az elnevezés szabályai**	**Példák**

Csomagok	Az egyedi csomagnevek prefixét mindig kis ASCII karakterekkel kell írni és a top-level domain nevek egyikének kell lennie. Jelenleg lehet com, edu, gov, mil, net, org vagy az országok angol kétbetűs azonosítókódjának egyike, melyeket az ISO 3166 és 1981 szabványok definiálnak.
	A csomag nevének többi komponensét a szervezet saját belső elnevezési konvencióinak megfelelően változtathatjuk. Például a csomagnevek tartalmazhatnak divízió, osztály, hivatal, projekt, gép és felhasználó neveket.
		`com.sun.eng`
		`com.apple.quicktime.v2`
		`edu.cmu.cs.bovik.cheese`

Osztályok	Az osztálynéveknek főneveknek kell lenniük és minden belső szó első betűjét nagybetűvel kell írni. Törekedni kell, hogy az osztálynevek egyszerűek és kifejezőek legyenek. Kerülendőek a rövidítések, kivéve nagyon nyilvánvaló esetben, mint az URL vagy HTTP.	`class Raster;` `class ImageSprite;`
Interfészek	Az interfész nevekben a főneveket az osztálynevekhez hasonlóan nagybetűvel kell kezdeni.	`interface RasterDelegate;` `interface Storing;`

Metódusok	A metódus neveknek kisbetűvel kezdődő igéknek kell lennie, de a belső szavakat nagybetűvel kell kezdeni.	`run();`
		`runFast();`
		`getBackground();`

Változók	A `for` változókat kivéve minden példány-, osztály- és osztályhoz tartozó konstans kezdőbetűjének kisbetűnek kell lennie. A belső szavak nagybetűvel kezdődjenek. A	`int i;`
		`char c;`
		`float myWidth;`

	változó nevek nem kezdődhetnek aláhúzás _ vagy dollárjellel $, habár mindkettő szintaktikailag megengedett. A változónevek lehetőleg rövidek és kifejezőek legyenek. A változónév megválasztásakor ügyeljünk arra, hogy a név tükrözze az alkalmi szemlélő számára a használatának szerepét. Az egy-karakteres változó neveket kerüljük kivéve az ideiglenes, "eldobható" változókat. Az általános nevei az ideiglenes változóknak `i, j, k, m` és `n` az egész típusúak számára; `c, d` és `e` a karakter típusúak számára.	

Konstansok	Az osztályszintű konstansoknak deklarált változókat és a konstansokat csupa ANSI nagybetűvel kell írni és a szavakat aláhúzás jellel ("_") kell elválasztani.	`static final int MIN_WIDTH = 4;`
		`static final int MAX_WIDTH = 999;`
		`static final int GET_THE_CPU = 1;`

Programozási gyakorlatok

Példány vagy osztályváltozóhoz való hozzáférés szabályozása

Egyetlen példány vagy osztályváltozót se tegyünk ok nélkül publikussá. Gyakran a példányváltozónak nem szükséges explicit értéket adni, mert ez egy metódus hívás mellékhatásaként fog bekövetkezni.
A publikus példányváltozó indokolt használatára példa az olyan osztály, ami valójában csak adat struktúra és nem tárolja az adatok viselkedését. Más szavakkal ha `struct` deklarációt használnál (ha létezne a Java nyelvben) osztálydeklaráció helyett.

Hivatkozás osztály változókra és metódusokra

Kerüljük az objektumok használatát statikus valtozó vagy metódus eléréséhez. Ilyenkor csak az osztálynév használata a megengedett. Példák:

```
classMethod();              //OK
AClass.classMethod();       //OK
anObject.classMethod();     // KERÜLENDŐ!
```

Konstansok

Numerikus konstansokat nem szabad közvetlenül a kódba szerepeltetni a -1, 0 és 1 kivételével, melyek `for` ciklus fejbén szerepelhetnek.

Értékadás

Kerülendő több változónak egy utasításban értéket adni, mert nehezen olvasható. Példa:

```
fooBar.fChar = barFoo.lchar = 'c'; // KERÜLENDŐ!
```

Ne használjuk az értékadás operátort olyan helyeken, ahol könnyen összetéveszthető az összehasonlító operátorral. Példa:

```
if (c++ = d++) {        //KERÜLENDŐ! (A Java nem engedi.)
    ...
}
```

Helyette:

```
if ((c++ = d++) != 0) {
    ...
}
```

Ne használjunk egymásba ágyazott értékadásokat a futásidő csökkentésére. Ez a fordító dolga. Példa:

```
d = (a = b + c) + r;        // KERÜLENDŐ!
```

Helyette:

```
a = b + c;
d = a + r;
```

Vegyes gyakorlatok

Zárójelek

Általában véve jó programozói gyakorlat a zárójelek bő alkalmazása a kifejezésekben, hogy elkerüljük a precedencia problémákat. Így azok számára is egyértelmű lesz, akik nem látják át elsőre a kifejezést vagy nincsenek tisztában az operátorok precedenciájával.

```
if (a == b && c == d)     // KERÜLENDŐ!
if ((a == b) && (c == d)) // Helyes
```

Visszatérési értékek

Próbáljuk a program struktúráját úgy kialakítani, hogy tükrözze a szándékot. Például:

```
if (booleanExpression) {
    return true;
} else {
    return false;
}
```

Helyett:

```
return booleanExpression;
```

Hasonlóan,

```
if (condition) {
    return x;
}
return y;
```

Helyette:

```
return (condition ? x : y);
```

Kifejezés a "?" előtt a feltételes operátorban

Ha egy kifejezés bináris operátort tartalmaz a ? előtt egy háromoperandusú ?: operátorban, akkor zárójelezni kell. Példa:

```
(x >= 0) ? x : -x;
```

Speciális megjegyzések

Használjuk az XXX szót annak jelzésére a megjegyzésben, ha valami még hibás, de működik. Használjuk a FIXME szót pedig annak a jelölésére, hogy valami hibás és nem működik.

Kód példák

Java forrásfájl példák

A következő példa bemutatja hogyan formázzuk az egyszerű publikus osztályt tartalmazó Java forrás fájlt. Az interfészeket hasonlóan kell formázni.

```
/*
 * @(#)Blah.java        1.82 99/03/18
 *
 * Copyright (c) 1994-1999 Sun Microsystems, Inc.
 * 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information of
 * Sun Microsystems, Inc. ("Confidential Information").  You shall
 * not disclose such Confidential Information and shall use it only
 * in accordance with the terms of the license agreement you entered
 * into with Sun.
 */


package java.blah;

import java.blah.blahdy.BlahBlah;

/**
 * Az osztály leírása ide jön.
 *
 * @version 1.82 18 Mar 1999
 * @author  Firstname Lastname
```

```
 */
public class Blah extends SomeClass {
    /* Itt következhetnek az osztály implementációs megjegyzései */

    /** classVar1 dokumentációs megjegyzés */
    public static int classVar1;


    /**
     * classVar2 dokumentációs megjegyzés,
     * egy sornál hosszabb eset
     */
    private static Object classVar2;

    /** instanceVar1 dokumentációs megjegyzés */
    public Object instanceVar1;

    /** instanceVar2 dokumentációs megjegyzés */
    protected int instanceVar2;

    /** instanceVar3 dokumentációs megjegyzés */
    private Object[] instanceVar3;

    /**
     * ...constructor Blah dokumentációs megjegyzés...
     */
    public Blah() {
        // ...Itt jön az implementáció...
    }

    /**
     * ...method doSomething dokumentációs megjegyzés...
     */
    public void doSomething() {
        // ...Itt jön az implementáció...
    }

    /**
     * ...method doSomethingElse dokumentációs megjegyzés...
     * @param someParam leírás
     */
    public void doSomethingElse(Object someParam) {
        // ...Itt jön az implementáció...
    }
}
```