



## Code Style

---

- code conventions



## Why Have Code Conventions

- до **80% стоимости** ПО приходится на **поддержку**
- обычно поддержкой занимается **не автор**
- соглашения **улучшают читабельность** кода, позволяя разработчикам и другим инженерам быстрее вникать в (чужой) код

**maintenance ~ up to 80% cost**



## Примеры стандартов

### **JPL Institutional Coding Standard for the C Programming Language**

Лаборатория Реактивного  
Движения NASA

### **MISRA C/C++**

Motor Industry Software Reliability Association  
Используется при разработке встраиваемых систем  
(aerospace, telecom, medical devices, defense, railway and others)



## Общая структура файла с исходным кодом

- beginning comments (license, etc)
- **package and import statements**
- class/interface documentation comment
- **class or interface statement**
- class/interface implementation comment
- **class (static) variables**
- **instance variables**
- **constructors**
- **methods**

порядок



## Class (static) variables order

- public class variables
- protected
- package level (no access modifier)
- private

**ORDER!**

## Instance variables order

- public
- protected
- package level (no access modifier)
- private

**ORDER!**

## Constructors

## Methods

Группируются **по функциональности**, не по «видимости».

Целью является повышение читабельности кода.



## Indentation (сдвиг)

**Unit: 4 spaces** (основной размер выравнивая)

**Tabs: 8 spaces**

## Line length

- **Original requirement:** not longer than **80 characters**  
(not handled well by many terminals and tools)
- **Be rational and don't exceed boundaries**



## Перенос строчек (общие принципы)

- перенос после **запятой**
- перед **операторами**
- лучше использовать переносы **верхнего уровня**
- необходимо **выравнивать** новые строки  
по выражению на предыдущей строке
- если правила ведут к плохой читабельности,  
используется 8 пробелов



## Перенос строчек – примеры

```
function(longExpression1, longExpression2, longExpression3,  
    longExpression4, longExpression5);  
  
var = function(longExpression1,  
    func2(longExpression2,  
        longExpression3));
```

## Arithmetic expressions

```
longName = longName2 * (longName3 + longName4 - longName5)  
    + 4 * longname6;
```

**PREFER**

```
longName = longName2 * (longName3 + longName4  
    - longName5) + 4 * longname6;
```

**AVOID**





## Перенос строчек – примеры (объявление методов)

```
someMethod(int anArg, Object anotherArg, String xArg,  
            Object andStillAnother) {  
    ...  
}
```

**Conventional**

```
private static synchronized horkingLongName(int anArg,  
    TAB Object anotherArg, String yetAnotherArg,  
    TAB Object andStillAnother) {  
    ...  
}
```

**Indent 8 spaces  
to avoid very deep indents**

```
private static synchronized horkingLongName(int anArg,  
    TAB Object anotherArg, String yetAnotherArg,  
    TAB Object andStillAnother)  
{  
    ...  
}
```

**Another variant**



## Перенос строчек (общие принципы)

**If statements** (обычно используют правило 8 пробелов)

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

**ТАК НЕ НАДО!**

Легко пропустить  
эту строку

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

**ЛУЧШЕ – ТАК !**

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

**ИЛИ ТАК !**



## Перенос строчек (общие принципы)

**If statements** (обычно используют правило 8 пробелов)

Предыдущий пример

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

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

Можно и так




## Перенос строчек (общие принципы)


### Ternary expressions

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

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



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





## Блочные комментарии

- Используются для
  - описания файлов (заголовки, лицензии, ...)
  - методов
  - структур данных
  - описания алгоритмов
- Должны иметь отступ, совпадающий с описываемым кодом
- Перед комментарием **ДОЛЖНА** быть добавлена **пустая строка**

### Пример

```
[BLANK LINE]
/*
 * Here is a block comment with some very special formatting
 *
 *     one
 */
```



## Однострочные комментарии

- Должны иметь отступ, совпадающий с дальнейшим кодом
- Если комментарий не может быть записан в одну строку, должен использоваться блочный комментарий
- Перед комментарием ДОЛЖНА быть добавлена **пустая строка**

## Пример

```
someCode();  
[BLANK LINE]  
if (condition) {  
    [BLANK LINE]  
    /* Handle the condition. */  
    ...  
}
```



## End of line comments

- Не должны использоваться на нескольких строках подряд для написания комментариев

```
if (foo > 1) {  
    // Do a double-flip.  
    ...  
}  
else {  
    return false;           // Explain why here.  
}
```

- Но могут использоваться на нескольких строках подряд для комментирования кода

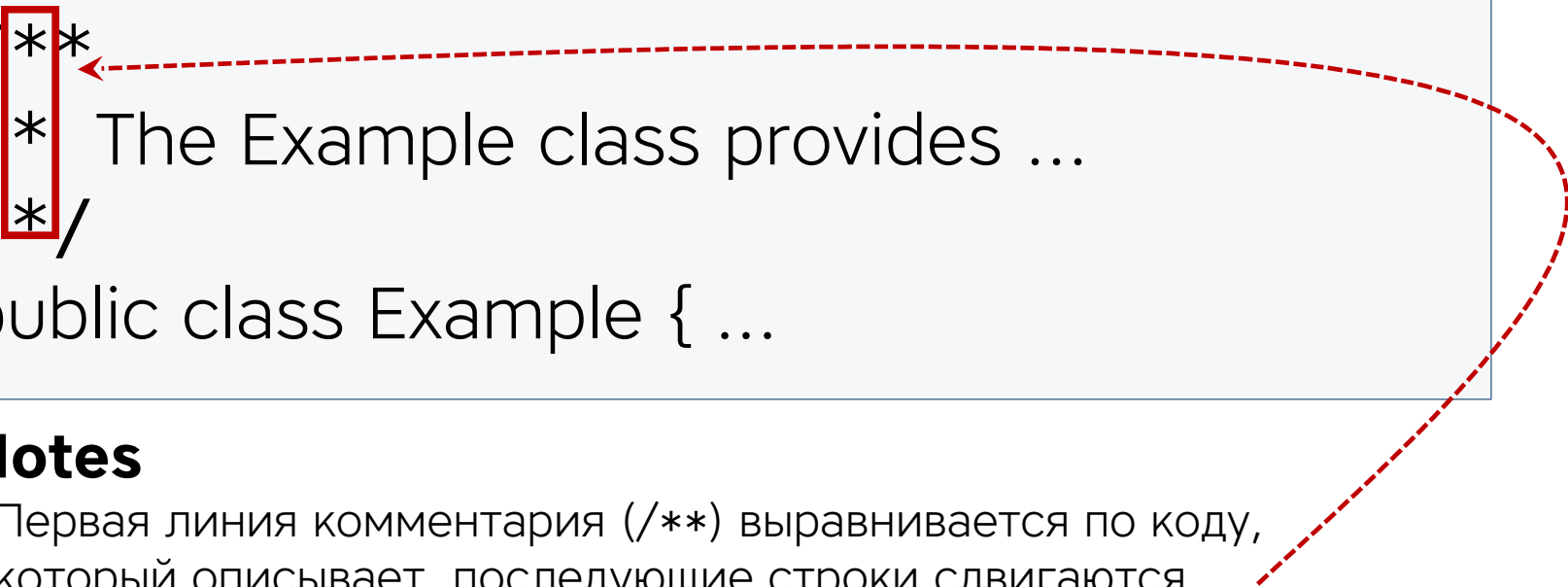
```
//if (bar > 1) {  
//  
//    // Do a triple-flip.  
//    ...  
//}  
//else {  
//    return false;  
//}
```



## Комментарии javadoc

- Описывают Java классы, интерфейсы, конструкторы, методы и поля
- Каждый комментарий заключается в **/\*\*...\*/**
- Должны вставляться непосредственно перед **declaration**

```
/**  
 * The Example class provides ...  
 */  
public class Example { ...
```



## Notes

Первая линия комментария (/\*\*) выравнивается по коду, который описывает, последующие строки сдвигаются на 1 символ для выравнивания «звездочек».

Детали реализации класса (implementation details) пишутся в отдельном комментарии, сразу после определения класса (class XXX).





## Declarations (style & initialization)

- Одно определение на строке

```
int level; // indentation level  
int size;  // size of table
```

```
int level, size;
```

**BAD**



- Не следует помещать разные типы на одной строке:

```
int foo,  fooarray[];
```

**BAD**





## Declarations (style & initialization)

- Альтернативный вариант определения переменных

```
int    level;           // indentation level
int    size;            // size of table
Object currentEntry;    // entry
```



### NOTE

Желательно инициализировать переменные в месте определения (если не требуются предварительные вычисления)



## Declarations (placement)

- Всегда помещайте объявления **в начале блока**

```
void myMethod() {  
    int int1 = 0;    // beginning of method block  
  
    if (condition) {  
        int int2 = 0;    // beginning of "if" block  
        ...  
    }  
}
```

- Определение переменных в середине блока (перед использованием) ведет к **сложностям с чтением кода и усложняет переносимость.**



## Declarations (placement)

- Определение переменных в середине блока (перед использованием) ведет к **сложностям с чтением кода и усложняет переносимость**.
- Исключением являются индексы в циклах:

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





## Declarations (placement)

Избегайте объявления, которые  
**перекрывают переменные верхнего уровня**

```
int count;
...
myMethod() {
    if (condition) {
        int count = 0;
        ...
    }
}
```

**AVOID**



## Class & interface declarations

- **Не должно быть пробела** между именем метода и скобкой
- Открывающая "{" ставится на той же строке кода, что и название
- Закрывающая "}" ставится на отдельной строке, выравнивается по выражению, открывающему блок
- Методы разделяются пустой строкой

### Пример

```
class Sample extends Object {  
    int ivar1;  
    int ivar2;  
  
    Sample(int i, int j) {  
        ivar1 = i;  
        ivar2 = j;  
    }  
  
    ...  
}
```

!?!



## Simple statements

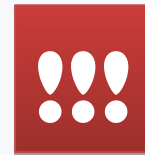
Каждая строка должна

**содержать только одно выражение:**

```
argv++;           // Correct
```

```
argc-- ;          // Correct
```

```
argv++; argc-- ; // AVOID!
```





## Compound statements - "{ statements }"

- «Внутренние» выражения сдвигаются на 1 уровень вправо
- Открывающая "{" ставится на той же строке кода
- Закрывающая "}" ставится на отдельной строке, выравнивается по выражению, открывающему блок
- Скобки используются **ВСЕГДА** вокруг выражений, являющихся частью управляющих выражений **(даже однострочных)**



```
if (condition) {  
    System.out.println("condition is true!");  
}
```





## if, if-else, if else-if-else statements

```
if (condition) {  
    statements;  
}
```

```
if (condition) {  
    statements;  
} else {  
    statements;  
}
```

```
if (condition) {  
    statements;  
} else if (condition) {  
    statements;  
} else {  
    statements;  
}
```

**за такое**

```
if (condition)  
    statement;
```

**расстрел на месте !!!**



## Cycle statements

```
for (initialization; condition; update) {  
    statements;  
}
```

«Пустой» цикл записывает так:  

```
for (initialization; condition; update);
```

```
while (condition) {  
    statements;  
}
```

```
while (condition);
```

```
do {  
    statements;  
} while (condition);
```

```
for (condition)  
    statement;
```

**расстрел!!!**

```
while (condition)  
    statement;
```

**расстрел!!!**

```
do statement;  
while (condition)
```

**расстрел!!!**



## Switch statements

```
switch (condition) {  
  case ABC:  
    statements;  
    /* falls through */  
  
  case DEF:  
    statements;  
    break;  
  
  case XYZ:  
    statements;  
    break;  
  
  default:  
    statements;  
    break;  
}
```

- Если case «проваливается» в следующий, обязательно необходим комментарий на месте опущенного `break` `/* falls through */`
- Каждый switch должен содержать default case
- Default case может обходиться без `break`, но он избавляет от ошибок при добавлении новых case`ов



## White spaces | blank lines

Пустые строки повышают читабельность за счет визуального разделения секций кода, которые логически объединены.

**2 пустых строки** используются в следующих случаях:

- Между секциями исходного файла
- Между определениями классов и интерфейсов

**1 пустая строка** используется в следующих случаях:

- Между определением методов
- Между определением локальных переменных в методе и первым выражением
- Перед блочным и однострочными комментариями
- Между логическими секциями внутри метода



## White spaces | blank spaces

- Ключевые слова с последующей скобкой **должны быть разделены пробелом**

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



- Пробел **НЕ** используется **между названием метода и скобкой** (параметры).  
Это помогает различать методы и управляющие конструкции.

```
println("hello");
```



- Пробел **обязателен после запятой** в списке параметров

```
Arrays.asList("A", "B", "C", "D");
```





## White spaces | blank spaces

- Все бинарные операторы (за исключением точки) должны **отделяться пробелом от обоих операндов**

```
a += c + d;  
a = (a + b) / (c * d);  
  
println("size is " + foo + "\n");
```

- Унарные операторы **никогда не отделяются от операнда**

```
while (d++ != s-- ) {  
    n++;  
}
```



## White spaces | blank spaces

- Выражения, задающие цикл for, отделяются пробелами

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



- Приведение типа отделяется пробелом

```
myMethod((byte) aNum, (int) (cp + 5));
```





## Именованние

Имена должны быть короткими, но «смысловыми», должны прояснять смысл их использования.

Односимвольные имена должны избегаться за исключением временных переменных. Стандартные имена для них:

**i**, **j**, **k**, **m** and **n** for integers

**c**, **d** and **e** for characters

```
int          i;  
char         c;  
float        myWidth;
```





## Именование

Имена должны быть короткими, но «смысловыми», должны прояснять смысл их использования.

Имена переменных, определенных как константы уровня класса записываются в верхнем регистре со словами, разделенными подчеркиванием (" \_ ")

```
static final int MIN_WIDTH    = 4;  
static final int MAX_WIDTH    = 999;  
static final int GET_THE_CPU = 1;
```



## Общие правила

### Доступ к переменным класса

Don't make any instance or class variable public without good reason. Often, instance variables don't need to be explicitly set or gotten.

One example of appropriate public instance variables is the case where the class is essentially a data structure, with no behavior (like struct in c/c++).

### Обращение к статическим методам и полям

Желательно обращаться через имя класса, а не объекта.

```
classMethod();           //OK
AClass.classMethod();    //OK

anObject.classMethod();  // AVOID!
```

### Константы

Числовые константы не должны использоваться в коде напрямую, за исключением значений -1, 0, и 1, которые часто появляются в циклах





## Общие правила

### Присвоение значений переменным

- Нежелательно использовать присвоение нескольким переменным в одном выражении, падает читабельность

```
fooBar.fChar = barFoo.lchar = 'c';
```



- Не используйте вложенные присваивания

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

should be written as:

```
a = b + c;
```

```
d = a + r;
```



## Общие правила

### Скобки

Всегда используйте скобки в длинных выражениях, это позволяет избежать проблем с приоритетом операций.

Если приоритет понятен Вам, совершенно не обязательно, что он так же воспринимается другими.

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



**двойной расстрел!!!**

```
if ((a == b) && (c == d)) // RIGHT
```



# Code Style

---

- code conventions
- examples of bad code



**From the internet...**

```
try {  
    ...  
} catch (Exception e ) {}
```



## From the internet...

```
class Singletons {  
    public static final MyException myException =  
        new MyException();  
}
```

```
class Test {  
    public void doSomething() throws MyException {  
        throw Singletons.myException;  
    }  
}
```



## From the internet...

```
if (expensiveFunction() > aVar) {  
    aVar = expensiveFunction();  
}
```

```
for (int i = 0; i < expensiveFunction(); ++i) {  
    System.out.println(expensiveFunction());  
}
```





## From the internet...

```
if () {  
    if () {  
        if () {  
            if () {  
                if () {  
                    if () {  
                        if () {  
                            ....  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```



## From the internet...

```
try {  
    j = 0;  
    while (true)  
    {  
        array[j] = [some calculations];  
        j++;  
    }  
}  
catch(Exception e) {}
```



## **Code Conventions for the Java Programming Language**

[https://www.google.ru/?](https://www.google.ru/?q=Code+Conventions+for+the+Java+Programming+Language)

[q=Code+Conventions+for+the+Java+Programming+Language](https://www.google.ru/?q=Code+Conventions+for+the+Java+Programming+Language)

## **Google Java Style**

<https://google.github.io/styleguide/javaguide.html>



**FIN**