Модуль #9
Java Code Conventions

Java Code Conventions

- Цели
- Структура файла
- Отступы и длины строк
- Тернарные выражения
- Объявления
- Вертикальные отступы
- Пробельные символы
- Соглашения об именовании
- Java Bean
- Общие рекомендации

Цели

- Зачем нужны соглашения по оформлению кода?

Цели

Зачем нужны соглашения по оформлению кода?

- 80% жизненного цикла приложения состоит из поддержки
- Практически ни одно приложение не поддерживается его автором
- Соглашения по оформлению кода повышают читабельность
- Соглашения по оформлению кода позволяют быстрее понять логику приложения

Структура файла

- Начальный комментарий
  - сведения о разработчиках, короткое описание программы

```
/*
 * Classname
 *
 * Version info
 *
 * Copyright notice
 */
```

- Сведения о пакете

```
package com.luxoft;
```

- Импорты

```
import com.luxoft.Product;
```

Структура файла

- Объявление класса или интерфейса

```
public class Store {
```

- Статические поля (public, protected, private)

```
public static int version;
```

- Поля объекта (public, protected, private)

```
public String title;
```

- Конструкторы

```
public Product() { ... }
```

- Методы

```
public String getTitle() { ... }
```

Горизонтальные отступы и длины строк

- Отступы должны быть равны четырем пробельным символам
- Вложенные элементы должны быть отделены соответствующим количеством отступов
- Длины строк не должны превышать 80 символов
    - Если длина команды превышает 80 символов, то рекомендуется перенести строку с двойным отступом после запятой или скобки
- Длины строк в комментариях JavaDoc не должны превышать 70 символов

Отступы и длины строк

```
function(longExpression1, longExpression2, longExpression3,
        longExpression4, longExpression5);

var = function1(longExpression1,
        function2(longExpression2, longExpression3));

longName1 = longName2 * (longName3 + longName4 - longName5)
        + 4 * longname6;

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

Тернарные выражения

Три варианта оформления тернарных выражений:

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

alpha = (aLongBooleanExpression) ? beta
                                 : gamma;

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

Объявления

- Одна переменная на строку, если предполагается комментарий:

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

- Запрещается объявление переменных и функций в одной строке

```
long dbaddr, getDbaddr(); // WRONG!
```

- Запрещается смешивать разные типы в одной строке

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

Объявления

- Рекомендуется объявлять переменные в начале блока (кода, ограниченного фигурными скобками)

- Не рекомендуется объявлять локальные переменные с именами, совпадающими с переменными более выского уровня

```
int count;

void func() {
    if (condition) {
        int count; // AVOID!
    }
}
```

Объявления классов, интерфейсов и методов

- Скобки не отделяются пробелом от имени метода
- Открывающая фигурная скобка располагается на той же строке, что и объявление
- Закрывающая фигурная скобка располагается на новой строке (за исключением пустых блоков)
- Методы разделяются пустой строкой

```
class Sample extends Object {
    int i;
    int j;

    Sample(int i, int j) {
        this.i = i;
        this.j = j;
    }

    int emptyMethod() {}
}
```

Вертикальные отступы

Отступ в две пустых строки:

- Между классами или интерфейсами в переделах файла с исходным кодом (только для вложенных классов и интерфейсов, невложенные классы и интерфейсы следует размещать в отдельных файлах)

Отступ в одну пустую строку:

- Между методами класса
- Между локальными переменными метода и первым выражением
- Перед комментарием
- Между логическими частями кода в методе для улучшения читаемости

Пробельные символы

Пробельный символ располагается:

- Между ключевым словом и скобкой (за исключением имен методов и открывающими скобками)

```
while (condition) { ... }
```

- После запятых в списке аргументов

```
myMethod(arg1, arg2, arg3);
```

- Вокруг всех бинарных операторов (операторов с двумя операндами)

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

Пробельные символы

- После символа ; в выражении for

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

- После закрывающей скобки при приведении типа

```
(byte) aNum
```

Note: пробельный символ никогда не оделяет унарные операторы от операнда

```
d = s++;
```

Соглашения об именовании

- Соглашения об именовании позволяют сделать программу более читаемой и понятной
- Имя может давать полезную информацию о назначении класса, переменной, метода или его аргумента

Соглашения об именовании – классы

Имена классов должны:

- Представлять собой существительное
- Начинаться с большой буквы
- В случае, если имя класса состоит из нескольких слов, то каждое следующее слово должно начинаться с большой буквы

```
class Raster
class ImageSprite
class InternalProduct
```

Соглашения об именовании – интерфейсы

Имена интерфейсов должны:

- Представлять собой причастие или существительное
- Начинаться с большой буквы
- В случае, если имя интерфейса состоит из нескольких слов, то каждое следующее слово должно начинаться с большой буквы

```
interface Collection
interface Serializable
interface Storing
```

Соглашения об именовании – интерфейсы и реализующие их классы

Существует де-факто несколько соглашений об именовании интерфейсов и классов-имплементоров:

```
class DefaultUser implements User // OK
class Flyer implements Flying // OK
class ServiceImpl implements Service // OK
class Service implements IService // WRONG!
class HumanClass implements Human // WRONG!
class TypeHuman implements Human // WRONG!
```

Соглашения об именовании – методы

Имена методов должны:

- Представлять собой глагол
- Начинаться с маленькой буквы
- В случае, если имя метода состоит из нескольких слов, то каждое следующее слово должно начинаться с большой буквы

Note: Имена методов не должны содержать имя класса

```
public class Product {
    void buy() { ... } // OK
    int getPrice() { ... } // OK
    void store() { ... } // OK
    void storeProduct() { ... } // WRONG!
}
```

Соглашения об именовании – методы доступа к полям объекта

Имена таких методов должны представлять собой слова get- или set- и имя поля с большой буквы. Аргумент метода-сеттера должен совпадать с именем поля.

```
private int size;

public int getSize() {
    return size;
}

public void setSize(int size) {
    this.size = size;
}
```

Соглашения об именовании – переменные

Имена переменных должны:

- Представлять собой существительное
- Начинаться с маленькой буквы
- В случае, если имя переменной состоит из нескольких слов, то каждое следующее слово должно начинаться с большой буквы
- Однобуквенные переменные допустимы только при краткосрочном использовании (в циклах, блоках catch и т.п.)

```
int i; // Только для циклов
int size; // OK
float myWidth; // OK
```

Соглашения об именовании – переменные

- Сокращения допустимы только для общеизвестных аббревиатур или акронимов

```
String sql; // OK
int displayScreenSize; // OK
int dss; // WRONG! Use full name!
HTTPConnection httpConnection; // OK
```

- При объявлении переменных ссылочного (объектного) типа допустимо (и рекомендуется) использовать имя класса с маленькой буквы

```
Product product; // OK
VehicleStore vehicleStore; // OK
VehicleStore vehicle_store; // WRONG!
Product _Product; // WRONG!
Product prod; // WRONG!
```

Соглашения об именовании – аргументы конструктора

В случае, если аргумент конструктора устанавливает значение поля экземпляра, аргумент именуется так же как и поле. В остальных случаях правила те же, что и для переменных.

```
public Product(String name) {
    this.name = name;
}
```

Соглашения об именовании – константы

Имена констант должны:

- Представлять собой существительное
- Состоять из прописных букв (upper case)
- В случае, если имя константы состоит из нескольких слов, то каждое следующее слово должно быть отделено символом _

```
static final int MIN_WIDTH = 4;
static final int MAX_WIDTH = 999;
static final float DEFAULT_BALANCE;
```

Java Bean

JavaBeans — классы в языке Java, написанные по определённым правилам. Эти соглашения поддерживаются многими библиотеками и средами разработки.

- Класс должен иметь публичный конструктор без параметров
- Класс должен предоставлять доступ к своим полям через публичные методы доступа, но сами свойства должны быть приватными
- Класс должен быть сериализуем

Java Bean

```
/* Класс реализует интерфейс Serializable, показывая,
что может быть сериализован */
public class PersonBean implements Serializable {
    // Приватное поле
    private String name;

    // Конструктор по умолчанию (без аргументов)
    public PersonBean() {}

    // Методы доступа
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Общие рекомендации

- Импортируйте конкретные классы, а не пакеты (не используйте *)

```
import com.luxoft.Product; // OK
import com.luxoft.*; // WRONG!
```

- Используйте скобки {} даже для однострочных if, while, for, else…

```
if (condition)
    setSize(newSize) // WRONG!
```

- Только один оператор или вызов метода на одной строчке кода

```
int a = 5; int b = getSize(); // WRONG!
```

Общие рекомендации

- Не игнорируйте исключения

```
try {
    ...
} catch (IOException e) {} // WRONG!
```

- Статические вызовы делайте только с указанием класса, но не переменной

```
public class A {
    public static void staticMethod() { ... }
}

A.staticMethod(); // OK
A a = new A();
a.staticMethod(); // WRONG!
```

Общие рекомендации

- Не пишите гигантских методов
- Сводите к минимуму область видимости переменных и методов
- Используйте коментарии TODO

```
public void authorize(User user) {
    // TODO: Add additional WebSSO check
    ...
}
```