|  |
| --- |
| ИнтерТраст |
| Стандарты кодирования |
|  |

|  |
| --- |
| Митавский Д. В. |

Содержание

[Стандарты кодирования CM4/5 2](#_Toc362870271)

[Стандарты 3](#_Toc362870272)

[Форматирование кода 3](#_Toc362870273)

[Именование сущностей 4](#_Toc362870274)

[Исключения 5](#_Toc362870275)

[Журналирование (Logging) 5](#_Toc362870276)

[Кодировка файлов 5](#_Toc362870277)

[Комментарии 5](#_Toc362870278)

[Работа с Git 6](#_Toc362870279)

[Организационные вопросы 6](#_Toc362870280)

[Настройка IDE 7](#_Toc362870281)

[Eclipse 7](#_Toc362870282)

[IntelliJ Idea 8](#_Toc362870283)

# Стандарты кодирования CM4/5

Этот документ описывает стандарты кодирования, используемые в проекте CM4/5. Стандарты должны неукоснительно соблюдаться всеми разработчиками. Данные стандарты во многом соответствуют конвенции кодирования Java, принятой в Oracle (<http://www.oracle.com/technetwork/java/codeconv-138413.html>), однако некоторые моменты, не описанные, остаются на усмотрение разработчиков.

## Стандарты

### Форматирование кода

1. Код не должен содержать символы табуляции.

Вместо символов табуляции используются отступы в 4 пробела (легко настраивается в любой современной среде разработки).

1. Одна строка кода должна содержать не более одного логического выражения или объявления переменной.
2. Максимальная длина строки кода – 120 символов.
3. Тело условных операторов (кроме тернарных) и операторов циклов должно быть заключено в фигурные скобки, даже если тело представляет собой единственную строку кода. Открывающая фигурная скобка находится в конце строки, открывающей условный оператор, закрывающая - на строке, следующей за последней строкой тела условного оператора или цикла. Например, следует использовать следующую конструкцию:

for (int i = 0; i < 10; ++i) {

if (i == 5){

break;

}

}

вместо

for (int i = 0; i < 10; ++i) if (i == 5) break;

1. При использовании типизированных коллекций параметры типов в инициализирующем выражении следует опускать в соответствии с возможностями JDK 1.7:

ArrayList<String> list = new ArrayList<>();

1. Тип коллекции следует типизировать явным образом, даже если предполагается хранение в ней произвольных типов – в этом случае следует описывать тип Object:

Map<String, Object> map = new HashMap<>();

1. При работе с итераторами и при обходе массива следует использовать новый синтаксис JDK 1.5, если это не влечёт усложнения кода:

BigDecimal sum = BigDecimal.ZERO;

for (BigDecimal number : list) {

sum = sum.add(number);

}

1. Следует избегать конкатенации строк через +. Java-машина трансформирует подобный код в вызовы StringBuilder, но делает это не оптимально (новый объект StringBuilder создаётся для каждого знака «+». Если код выполняется редко, использование оператора + для конкатенации допустимо, но это не допустимо в случае многократного использования, например, в цикле.
2. Порядок следования спецификаторов следующий:

<access> static abstract synchronized <unusual> final native

1. Пробелы должны быть использованы в следующих случаях:

for**͏**(BigDecimal number : list)

После точки с запятой

for (int i = 0;͏i < list.size();͏++i)

Вокруг операторов (присваивание, сравнения и прочее):

for (int i͏=͏0; i͏<͏list.size(); ++i)

### Именование сущностей

1. Именование методов и атрибутов классов/интерфейсов соответствует спецификации JavaBeans. В частности, имена переменных и классов не должны содержать (в подавляющем большинстве случаев) символы подчеркивания для разделения слов. Вместо этого используется верхний регистр для начала нового слова в имени.
2. Префиксы и суффиксы (например, символы подчеркивания, или префиксы венгерской нотации) **не допускаются**. Примерами префиксов венгерской нотации являются: "m\_" в названии атрибутов классов или переменных (int m\_ivariable), "I" в названии интерфейсов (ISerializable), "С" в названии классов (CString). Более подробную информацию о венгерской нотации можно найти здесь [[1]](http://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BD%D0%B3%D0%B5%D1%80%D1%81%D0%BA%D0%B0%D1%8F_%D0%BD%D0%BE%D1%82%D0%B0%D1%86%D0%B8%D1%8F).
3. Имена классов, интерфейсов и переменных должны представлять собой правильно составленные выражения на английском языке. Например, переменная zapisVBazeDannyh должна быть переименована в databaseRecord. Если возникают сомнения в правильности именования, необходимо проконсультироваться у коллег.
4. Аббревиатуры в именах классах и переменных должны использоваться, как если бы это было обычное слово. Например класс CrudService, но не CRUDService. Переменная myDao. но не myDAO.
5. Все классы, отображающие на себя конфигурационные элементы должны заканчиваться суффиксом Config.
6. Имена констант и перечислений (enums) представляют собой слова (словосочетания), записанные в верхнем регистре и разделенные символом подчеркивания. Например,

final int MAX\_VALUE = 5;

### Исключения

1. Все исключения, определяемые в системе являются потомками SystemException, который является необрабатываемым RuntimeException.
2. «Сырой» RuntimeException нельзя использовать в коде, даже если ситуация не исправима. Для этого можно использовать системный FatalException.
3. Метод, который может возбуждать конкретные системные исключения обязан явно их специфицировать в операторе **throws**.

### Журналирование (Logging)

1. В коде нельзя использовать System.out и System.err
2. Системой журналирования является **slf4j** с имплементацией Log4j по умолчанию.
3. Прежде, чем использовать конструкции, требующие конкатенации строк, следует проверить необходимость журналирования, чтобы избежать существенного снижения производительности системы. Например вместо такой конструкции:

log.debug("Mathematics constant π = " + Math.PI + ". Approximately");

следует использовать следующую:

if (log.isDebugEnabled()) {

log.debug("Mathematics constant π = " + Math.PI + ". Approximately");

}

### Кодировка файлов

1. Все файлы, содержащие исходный код, имеют кодировку UTF-8
2. property-файлы, использующиеся для локализации приложения, имеют кодировку ISO 8859-1. Символы юникода (например, русские буквы) представляются соответствующими escape-последовательностями (\uXXXX). Современные IDE позволяют выполнять редактирование подобных символов прозрачным для пользователя образом (**обсудить**)

### Комментарии

1. Каждый public, protected или package-local класс (метод, атрибут, константа) должен быть снабжен комментарием, достаточным для понимания
2. Комментарии в коде должны быть написаны на русском языке
3. Каждый файл исходного кода должен быть озаглавлен комментарием, определяющем правообладателя системы (**требуется определить**)

### Работа с Git

1. При получении изменений из репозитория следует использовать метод rebase, а не merge.
2. Комментарии указываются на русском языке.
3. Перед комментарием должен присутствовать идентификатор задачи в Jira, к которому относятся изменения в коде.
4. Желательно не объединять разнородные задачи в единый commit. Например, фикс бага, обнаруженный в ходе работ, и не относящийся к данным работам, желательно вынести в отдельный commit.

### Организационные вопросы

1. Новые папки создаются только после согласования с лицами, ответственными за структуру проекта

## Настройка IDE

### Eclipse

#### Подключение и настройка плагина Checkstyle

Плагин подсвечивает код, который не соответствует стандарту (по умолчанию желтым цветом). При наведении на выделенный код всплывает подсказка с описанием проблемы. Плагин является вспомогательным инструментом и его использование не является обязательным.

#### Установка плагина

На сайте разработчиков пошагово установка плагина.(<http://eclipse-cs.sourceforge.net/downloads.html>)

#### Настройка плагина

**Todo**

#### Настройка форматирования

##### Formatter

Window->Preferences->Java->Code Style->Formatter New, вводим имя, жмём Ок. Открывается окно с закладками:

Indentation: Tab Policy - Spaces Only, indentation size - 4, tab size - 4. Устанавливаем все галочки кроме Align fields in columns, Empty Lines.

New Lines: устанавливаем все галочки, кроме after labels, insert new line after opening brace of array initializer, insert new line before closing brace of array initializer.

Control Statements: устанавливаем галочку keep 'else if' on one line.

Line Wrapping: Maximum line width - 120, устанавливаем галочку never join already wrapped lines.В списке выбираем Expressions->Assignments и внизу в пункте Line wrapping policy выбираем wrap where necessary. Аналогично для пунктов Annotations->Element value-pairs, Method Declarations->Declaration, 'enum' declarations->Constants.Во всех остальных настройках по умолчанию стоит перенос строк, если где-то не будет срабатывать, то надо посмотреть все остальные пункты в этом списке.

Comments: Устанавливаем галочку Enable Javadoc comment formatting. В блоке java doc comment settings устанавливаем все галочки кроме blank line before java doc tags, remove blank lines.

White Space: Заходим в блок Arrays->Arrays initializers и ставим галочки только напротив before opening brace, after comma, before closing brace, between empty braces.

Жмём ОК, в качестве активного профиля должен быть выбран только что созданный.

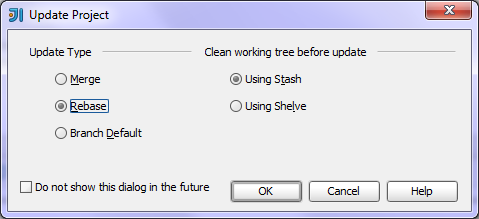
##### Text Editor

Window->Preferences->General->Editors->Text Editors. Устанавливаем все галочки, tab size - 4, print margin column - 120.

### IntelliJ Idea

#### Git Pull

Рекомендованный режим Git Pull – Rebase. Для того, чтобы его использовать, при скачивании изменений (Pull) достаточно выбрать соответствующий способ:



При попытке отправить изменения на сервер и наличии не скачанных обновлений, команда Pull будет выполнена автоматически и при этом будет предложен выбор метода в следующем диалоговом окне.

Если поставить галочки “Do not show this dialog in the future” в этих окнах, Idea будет применять выбранный способ и в дальнейшем.

Чтобы снова показывать данные диалоговые окна перед публикацией, необходимо зайти в меню Settings->Version Control->Confirmation и установить галочку “Update” в группе “Display options dialog when these commands are invoked”.