

ОАО «Бета Банк»

ИТ-стандарт

**«Правила оформления кода на языке
программирования Java»**

Москва - 20XX

Название:	Корпоративный стандарт ОАО «Бета Банк»			
	Стандарт «Правила оформления кода на языке программирования Java»			
Имя файла:	Правила оформления кода на языке программирования Java v1.2.doc			
Версия:	1.2			
Дата формирования текущей версии:	24.03.2010			
Подразделение – ответственный разработчик:	ДПР			
Уровень доступа:	Общедоступный			
Утверждаю	Ф.И.О.	Подпись	Дата	
Руководитель Блока «Информационные технологии»				
Согласовано		Подпись	Дата	
Первый заместитель руководителя Блока «Информационные технологии»				
Директор Дирекции проектирования и разработки				
Директор Дирекции розничных технологий				
Директор филиала Екатеринбургский				
		Подпись	телефон	
Отв. разработчик:				
Исполнитель:			6087	
Рассылка:				
Блок ИТ				

История изменений

Номер версии	Дата	Автор изменения	Суть изменения
0.7	12.02.2010		Первоначальная версия
0.8	19.02.2010		Учтены замечания А. Столярова, В.Дельвига, А.Павленко
0.9	09.03.2010		Переработана структура документа
1.0	10.03.2010		Добавлен состав профиля настроек Eclipse
1.1	21.03.2010		Учтены замечания Александра Столярова, Алексея Гусева, Александра Павленко, Максима Патрица, Андрея Гусева, Юрия Нещадимова, Алексея Завгороднего, Дмитрия Долгих
1.2	24.03.2010		Учтены замечания, переработана структура документа

Содержание

История изменений	3
Содержание	3

Термины и определения.....	4
Цель и область применения стандарта	6
Вступление в действие, сопровождение и порядок пересмотра стандарта	7
Доступ к стандарту	7
Соглашения, требования и рекомендации для Java	7
1 Общие положения	7
2 Правила оформления исходного кода	8
2.1 Расширения файлов	8
2.2 Порядок организации файлов.....	8
2.3 Правила оформления комментариев	11
3. Форматирование текста исходного кода	15
3.1 Ширина линий и правила переноса длинных строк.....	15
3.2 Разграничители секций	15
3.3 Пробелы. Отступы (абзацы). Скобки.	16
3.4 Операторы цикла/условия/перехода.....	18
4. Соглашение об именовании.....	19
4.1 Наименование пакетов и импорт-выражений.....	20
4.2 Наименование типов (классов и интерфейсов)	21
4.3 Наименование методов	21
4.4 Наименование переменных (полей и экземпляров класса)	22
4.5 Наименование констант	22
4.6 Наименование локальных переменных и параметров методов, обработчиков ошибок.....	22
4.7 Идентификаторы.....	23
5. Профили настроек	23
Литература и интернет-источники.....	24

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ ¹	
Название термина или обозначение	Определение термина или расшифровка обозначения
Банк	ОАО «Бета-Банк»

¹ Изменение данного раздела не требует согласования и утверждения документа.

ООП	Объектно-ориентированное программирование
Java	Язык объектно-ориентированного программирования и технология программирования
ПО	Программное обеспечение, комплекс компьютерных программ (приложений), обеспечивающих обработку, передачу данных
Исходный код или Java-файл	Текст программы на алгоритмическом языке ООП Java
Стандарт	Документ, формулирующий правила и соглашения, устанавливающий необходимые качественные характеристики, которым должен удовлетворять исходный код при разработке ПО на языке Java. Это также документ, регламентирующий требования, предъявляемые к разработчикам в Банке
UTF-8	Кодировка (форма представления) стандарта кодирования символов Юникод (Unicode) версии 2.0, позволяющего представить большинство знаков практически всех письменных языков для компьютерной обработки
Cp1251	Набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для всех русских версий Microsoft Windows
IDE	Интегрированная среда разработки, позволяющая использовать в одной среде несколько различных функций, необходимых для разработки ПО. Обычно IDE включает в себя текстовый редактор, компилятор и/или интерпретатор, средства автоматизации сборки, отладчик и др.
Javadoc	ПО, позволяющее создать набор HTML страниц, которые описывают детали проекта: классы, внутренние классы, интерфейсы, конструкторы, методы и поля. Документация Javadoc создается из структуры исходного кода и из Javadoc комментариев, присутствующих в коде
REQ	Требования Стандарта
ADV	Рекомендации Стандарта
PS	Указание на конфигурацию настроек (профиль) IDE, отображающую описанные в этом документе правила и соглашения

Цель и область применения стандарта

Документ представляет принятый в Банке стандарт кодирования на языке объектно-ориентированного программирования (ООП) Java.

Следование данному стандарту, правилам и соглашениям, описанным в нем, во всех проектах разработки ПО, выполняемых на языке программирования Java, является обязательным для всех разработчиков Банка.

Целью применения стандарта является решение следующих задач:

- 1) обеспечить всем командам разработчиков одинаковый стиль кода;
- 2) обеспечить преемственность разработки и сократить количество времени на передачу исходного кода от одного разработчика другому как внутри одного подразделения, так и между подразделениями;
- 3) сократить время адаптации новых сотрудников и обеспечить их быстрое обучение принятым в Банке стандартам кодирования;
- 4) уменьшить количество ошибок при внесении изменений в код, повысить качество программных разработок;
- 5) уменьшить сроки передачи исходного кода с *outsourse* на *insource*, с *insource* на *outsourse*.

Ответственный за применение стандарта - Директор Дирекции проектирования и разработки.

Данный стандарт состоит из набора требований и ряда рекомендаций.

Требования отмечены знаком **REQ**, рекомендации отмечены знаком **ADV**.

Знак **PS** (profile settings) означает, что данную рекомендацию можно реализовать через установку соответствующего параметра (-ов) в конкретной конфигурации настроек (профиле) используемой среды разработки.²

Требования являются обязательными для исполнения в отличие от рекомендаций. Но следует учитывать, что рекомендации основаны на многолетнем и обобщенном опыте разработки и действительно облегчают жизнь.

В основу документа вошли положения, изложенные в документах «Java Code Conventions» от Sun Microsystems, Java Programming Style Guidelines от Geotechnical Software Services, а также документы, размещенные по ссылкам, указанным в разделе [«Литература и интернет-источники»](#).

Большинство требований и рекомендаций, приведенных в данном стандарте, являются обобщением опыта разработки на Java.

Стандарт обобщает существующие подходы и привычки написания кода в различных подразделениях Банка.

Комментарии по тексту выделены курсивом.

² В разделе «Профили настроек» представлены ссылки на файлы профилей, которые настроены согласно положениям данного документа и которые рекомендуется использовать в работе для удобства следования положениям Стандарта. Конкретная конфигурация параметров и настроек, отображающих описанные в этом документе правила и соглашения, сохранена в профилях *DPR_Eclipse* и *DPR_IDEA* для IDE на основе Eclipse и для IntelliJ IDEA. Профили возможно применять к исходному коду из старых проектов, а также форматировать с их помощью исходный код, подготовленный в ином стиле, чем предписывает Стандарт.

Примеры выделены курсивом и моноширинным шрифтом на 2 пикселя меньше, чем основной текст.
 Если приведен пример неверного кодирования, то текст выделен красным шрифтом.
 Предпочтительный вариант оформления кода отмечен синим цветом.

Вступление в действие, сопровождение и порядок пересмотра стандарта

- Стандарт вступает в действие с момента подписания.
- Владельцем стандарта является Блок ИТ.
- Владелец имеет право в любой момент инициировать внеочередной пересмотр стандарта при наличии производственной необходимости, либо по прошествии 12 месяцев

Решение о проведении внеочередного пересмотра стандарта утверждается Директором Дирекции проектирования и разработки.

Доступ к стандарту

- Доступ сотрудников Банка к стандарту осуществляется без ограничений.
- Доступ к стандарту третьих лиц допустим по согласованию и с разрешения Руководителя Блока «Информационные технологии».

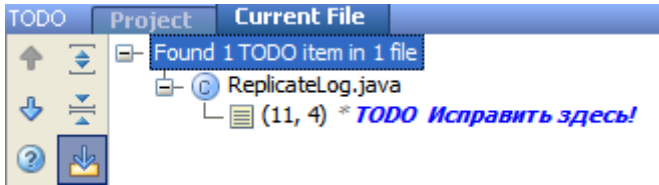
СОГЛАШЕНИЯ, ТРЕБОВАНИЯ И РЕКОМЕНДАЦИИ ДЛЯ JAVA		
1 Общие положения		
1.1	ADV	В пределах всего проекта используйте единый стиль форматирования. <i>Применяйте один тип профиля настроек в IDE.</i>
1.2	ADV	Избегайте дублирования кода с похожей функциональностью. <i>Необходимо перепроектировать файл с исходным кодом таким образом, чтобы эта функциональность была сосредоточена в одном месте.</i>
1.3	ADV	С учетом целесообразности, где возможно, предпочтительно используйте стандартные методы и стандартные библиотеки, а не разрабатывайте собственный код.
1.4	ADV	Предпочитайте минимальные классы монолитным. <i>Небольшие классы легче писать, тестировать и использовать. Минимальный класс проще в употреблении, его легче понять и проще использовать. Монолитные классы сложнее сделать корректными и безопасными.</i>
1.5	ADV	Указывайте модификаторы доступа public и private, даже если возможно их пропустить.

1.6	REQ	Используйте при кодировании следующие идентификаторы: только буквы ('A' до 'Z' и 'a' до 'z') и цифры ('0' через '9'). В виде исключения используйте знак подчеркивания. Не применяйте знаки доллара или не-ASCII символы. В качестве кодировки текста исходного кода используйте UTF-8. Если текст набран с использованием Cp1251, продолжайте использовать Windows-1251.
1.7	REQ	Все наименования, кроме комментариев разработчика, должны быть написаны с использованием латиницы.
1.8	ADV	Выполняйте детальный анализ кода для максимально возможного устранения предупреждающих сообщений.
1.9	ADV	Старайтесь применять максимум инкапсуляции для экземпляров объектов. Используйте минимально необходимый уровень доступа. Используйте get-, set- методы вместо прямого открытия public для переменных класса. <pre> public class NewColor { public int nColor; //... }; public class NewColor { public int getColor() const; bool setColor(int nColor); private int nColor; //... }; </pre>
2 Правила оформления исходного кода		
2.1 Расширения файлов		
2.1.1	REQ	Помещайте исходный код каждого public-класса в отдельный файл с названием, совпадающим с названием класса и расширением *.java
2.1.2	REQ	Файлы с байт-кодом должны иметь расширение: *.class
	PS	2.2 Порядок организации файлов
2.2.1	REQ	Java-файлы должны иметь следующую структуру <u>модуля компиляции</u> : <ul style="list-style-type: none"> • Заголовочный комментарий; • Объявления пакетов; • Импорт-выражения; • JavaDoc-комментарий описания класса

		<ul style="list-style-type: none"> • Объявления классов и интерфейсов; <p>Объявления пакетов размещайте в Java-файле после заголовочного комментария, затем располагайте отсортированные и сгруппированные импорт-выражения, которые всегда указывайте в явном виде. Располагайте импорт-выражения по принципу – наиболее фундаментальные, авторитетные группы вверху, сразу после объявления пакетов, наиболее частные группы – внизу. Сортируйте в пределах групп по алфавиту. Не используйте символ * в импорт-выражениях.</p> <p><i>К наиболее фундаментальным, авторитетным группам относятся следующие:</i></p> <pre> com.ibm.* com.sun.* java.* javax.* net.* org.ietf.* org.omg.* org.w3c.* org.xml.* org.apache.* org.* com.* sun.* sunw.* oracle.* import javax.naming.InitialContext; import javax.servlet.ServletContext; import javax.servlet.ServletException; import org.apache.commons.logging.Log; import org.apache.commons.logging.LogFactory; import ru.bank.ws.AS.WSCommonTypes10.WSAccessException; import ru.bank.ws.AS.WSCommonTypes10.WSTechnicalException; import ru.bank.ws.AS.WSUtils.Core.WSExceptions; import ru.bank.ws.AS.WSUtils.Core.WSExtentionHelper; import com.bank.forex.Statement; import com.bank.forex.sql.ResultSet; import com.bank.forex.*; </pre>
2.2.2	REQ	<p>Java-файлы должны иметь следующую структуру <u>классов и интерфейсов</u>:</p> <ul style="list-style-type: none"> • JavaDoc комментарии описания класса/интерфейса;

		<ul style="list-style-type: none"> • Заголовок класса/интерфейса; • Переменные класса; <i>(в порядке следования модификаторов: public, protected, private)</i> • Нестатические переменные; <i>(в порядке следования модификаторов: public, protected, private)</i> • Конструкторы; • Методы; <i>(последовательность модификаторов не регламентируется, старайтесь сгруппировать методы по функциональности, а не по объему и инкапсуляции.. Например, private методы класса могут быть помещены между двумя методами public, чтобы сделать чтение и понимание кода доступнее).</i>
2.2.3	REQ	В одном файле может находиться только один public-класс (интерфейс) Допускается размещение в том же файле второстепенных классов, логически связанных с основным.
2.2.4	REQ	Первым по счету в Java-файле размещайте public класс или интерфейс.
2.2.5	REQ	Разделяйте логически связанные части исходного кода отступами и переводами строк, а также дополнительными комментариями. <i>Правильный в отношении логики выполнения программы, но неправильный в отношении оформления код затрудняет сопровождение кода несколькими разработчиками.</i>
2.2.6	REQ	Сохраняйте структуру файлов с исходным кодом небольшими размерами, до 500 - 1000 строк кода с учетом целесообразности деления.
2.2.7	REQ	Ограничивайте описание public-класса примерно 500 строками (до 10 экранов текста).
2.2.8	REQ	Избегайте длинных методов и глубокой вложенности. Желательно, чтобы размер одного метода не превышал 50 строк кода. Оптимальным размером является код из 10-30 строк. <i>Чрезмерно большой по размеру метод и чрезмерная вложенность блоков (например, блоков if, for, while, do) делают метод более трудным для понимания и сопровождения, причем зачастую без каких бы то ни было оснований.</i>
2.2.9	REQ	Избегайте методов с большим количеством аргументов (> чем 5), используйте структуры для передачи большого числа параметров.
2.2.10	REQ	Размещайте на одной строке не более одного оператора, за исключением вызова операторов через точку. <pre>private String title, publisher; private String title; private publisher;</pre>

		<code>dto.setKurss(rs.getBigDecimal("kurss").toString());</code>
		2.3 Правила оформления комментариев
2.3.1	ADV	<ul style="list-style-type: none"> • Поясняющий логику работы и описательный текст комментариев пишите на русском языке; • Избегайте таких комментариев, которые могут устареть в процессе развития кода; • Комментарии не должны дублировать код и затруднять чтение кода; • Использование блочных комментариев предпочтительно для пояснений логики работы программы; • Для пояснения параметров используйте комментарий в одну строку; • Предпочтительно использовать однострочный комментарий типа <code>//</code> вместо <code>/*</code> комментарий <code>*/</code> там, где это возможно; • Основным типом комментариев являются документирующие комментарии (Javadoc), признаком которых служит знак <code>@</code>. <i>Остальные типы комментариев используются по усмотрению разработчика для пояснений и для того, чтобы «закомментировать» блок кода;</i> • Включайте документирующие комментарии на раннем этапе процесса разработки, а затем периодически проверяйте и дополняйте их по мере развития кода, вместо того, чтобы просто дожидаться окончания этапа кодирования, и только затем дописывать тэги для Javadoc; • Актуализируйте шаблоны комментариев, которые генерирует IDE при создании класса, удаляйте излишние. Язык таких комментариев допустимо оставить в оригинале (английский); <pre> /** * Created by IntelliJ IDEA. * User: ФИО * Date: DD.MM.YYYY * Time: 17:22:52 * To change this template use File Settings File * Templates. // Отредактировать */ public class Test1 { } </pre> <ul style="list-style-type: none"> • Подробную информацию о документирующих комментариях см. в документе "Как написать документирующие комментарии для Javadoc Tool" http://Java.sun.com/products/jdk/Javadoc/writingdoccomments.html

2.3.2	ADV	<p>Подробно комментируйте исходный код для понимания логики работы программы: <i>В качестве обязательного требования этот раздел применяйте к интерфейсным классам - это необходимый минимум.</i> REQ</p> <ul style="list-style-type: none"> Документирующие комментарии для JavaDoc должны предшествовать классу, полю, конструктору или объявлению метода; Для метода любого типа видимости описывайте назначение, принцип действия, параметры и возвращаемые значения, потокобезопасность и выбрасываемые исключения; Как правило, все операторы ветвления, условные операторы, циклы, присвоения значений флагам и индикаторам и т.п. снабжайте комментариями; Как правило, вновь добавленные переменные класса и константы снабжайте однострочными комментариями; Для полей документируйте особенности сериализации;
2.3.3	REQ	<p>Для стандартных заголовочных блочных комментариев используйте следующий стиль:</p> <pre> /* * \$Id\$ * * Название класса * * \$Revisions\$ \$Date\$ * * Copyright (c) ОАО Бета-Банк YYYY */ </pre>
2.3.4	ADV	<p>Если участок кода еще не реализован, размещайте блочный комментарий TODO</p> <pre> if (a == b) { /** * TODO Исправить здесь! * */ } </pre> 
2.3.5	ADV	<p>Используйте тэги в документирующих комментариях в следующей последовательности и для описания следующих элементов языка:</p> <ul style="list-style-type: none"> * @author (только для классов и интерфейсов) * @version (только для классов и интерфейсов) * @param (только для методов и конструкторов)

		<ul style="list-style-type: none"> * @return (только для методов) * @exception (или @throws) * @see * @since * @serial (или @serialField или @serialData) * @todo
2.3.6	<div>REQ</div> <div>PS</div>	<p>Текст документирующих комментариев оформляйте согласно следующим правилам:</p> <ul style="list-style-type: none"> • Первая строка должна содержать бланковый комментарий с разделителем /**; • Ведущие звездочки являются обязательными во всех строках документирующего комментария; • Между ведущей звездочкой и текстом всегда размещайте один пробел; • Первое предложение должно содержать краткое изложение функциональности класса, метода, т.к. Javadoc автоматически размещает его в качестве резюме; • Документирующий комментарий должен состоять из двух неповторяющихся разделов - сначала следует раздел описания, затем раздел тэгов; • Если у вас есть несколько блоков в описательном разделе комментария, отделяйте их тэгом <p>; • Вставляйте бланковый комментарий между разделами описания и списком тэгов, которые начинаются с символа "@" ; • Строка, которая начинается с символа "@", состоит из 2-х частей: 1) обозначение тэга, 2) описание тэга; • Последняя строка содержит бланковый комментарий с разделителем */; • Ограничивайте длину строки 70 символами, максимально 80 символов; • Записывайте текст в виде неполных предложений, лаконичными фразами без ущерба для описания логики реализации; <p>Получение метки. Здесь получаем метку. Ссылка на объект «кнопка». Этот метод получает ссылку на объект «кнопка».</p> <ul style="list-style-type: none"> • Используйте скобки в случае, если есть аргументы; <p>Метод <code>add(int, Object)</code> добавляет элемент в заданную позицию в массиве <code>ArrayList</code>. Метод <code>add</code> позволяет добавлять элементы. Метод <code>add()</code> позволяет добавлять элементы.</p> <ul style="list-style-type: none"> • Не дублируйте тэг @link;
2.3.7	<div>ADV</div>	<p>Стиль оформления для документирующего комментария:</p>

```

/**
 * Возвращает объект Image, который затем может быть нарисован на экране.
 * Аргумент URL должен указывать абсолютное значение {@link URL}.
 * <p>
 * Возвращается значение, несмотря на то, что существует или
 * нет изображение.
 * Графические примитивы, из которых состоит изображение, постепенно будут
 * появляться на экране.
 *
 * @param url абсолютное значение URL, указывающее местоположение
 * изображения
 * @param name имя объекта
 * @return объект-изображение по указанному адресу
 * @see ссылка на изображение
 */
public Image getImage(URL url, String name)
{
    try {
        return getImage(new URL(url, name));
    } catch (MalformedURLException e) {
        /**
         * @todo Исправить здесь !
         */
        return null;
    }
}

```

Результатом применения JavaDoc будет следующий текст:

```

getImage
public Image getImage(URL url, String name)

```

Возвращает объект Image, который затем может быть нарисован на экране. Аргумент URL должен указывать абсолютное значение url.

Этот метод всегда возвращает значение, несмотря на то, что существует или нет изображение. Графические примитивы, из которых состоит изображение, постепенно будут появляться на экране.

Parameters:

url - абсолютное значение URL, указывающее местоположение изображения

		<p><i>name</i> - <i>местоположение объекта</i></p> <p>Returns:</p> <p><i>объект-изображение по указанному адресу</i></p> <p>See Also:</p> <p>Image // это ссылка</p> <p>To Do:</p> <p>Исправить здесь!</p>
		<h3>3. Форматирование текста исходного кода</h3>
	PS	<h4>3.1 Ширина линий и правила переноса длинных строк</h4>
3.1.1	REQ	<p>Устанавливайте границу печати в редакторе на 80 символов. Старайтесь избегать строк длиной более 80 символов, и не используйте строки длиной более 120 символов.</p> <p>Если строка превысила размер 80-и символов, где целесообразно, используйте синтаксис переноса строки.</p>
3.1.2	REQ	<p>Синтаксис переноса строки:</p> <ul style="list-style-type: none"> • Перенос после запятой; • Перенос перед оператором, т.е. начинайте новую строку с оператора; • Выравнивание новой строки на уровне начала разделяемой части строки; • Если у метода много аргументов, тогда используйте выравнивание по открывающей скобке; <p><i>Допускается перенос параметров метода не по открывающейся скобке, а с отступом 8 пробелов, если имя метода заканчивается вблизи правой печатной границы;</i></p> <ul style="list-style-type: none"> • Предпочтителен перенос более короткой части строки; <pre>longMethodCall(expr1, expr2, expr3, expr4, expr5); int nRes = num1 + num2 + num3 + num4;</pre> <p>Не допускается разрывать переносами квалифицированные идентификаторы и вызовы, а также тернарные операторы.</p>
		<h4>3.2 Разграничители секций</h4>
3.2.1	REQ	<p>Для разделения логически связанных частей исходного кода (секций) допускается использовать только пустую строку.</p>

		<pre> //////////////////////////////////// /** * \$Id\$ */ </pre> <hr/>
3.2.2	REQ	<p>Используйте две пустых строки для разделения объявлений классов/интерфейсов.</p> <p>Используйте одну пустую строку для разделения следующих секций кода:</p> <ul style="list-style-type: none"> • групп импорт-выражений; • блочных комментариев; • между описанием локальных переменных в методе и его первым объявлением; • между методами; • между секциями кода внутри метода для улучшения читабельности;
	PS	3.3 Пробелы. Отступы (абзацы). Скобки.
3.3.1	REQ	Устанавливайте размер отступа, кратный 4-м стандартным символам пробела.
3.3.2	REQ	Не используйте символов табуляции, клавишу табуляции настраивайте на генерацию 4-х пробелов.
3.3.3	REQ	<p>Не помещайте пробелы до или после левой скобки или перед правой скобкой в объявлениях массивов, до левой скобки в объявлении методов.</p> <pre> args [i] = 0; args[i] = 0; </pre>
3.3.4	REQ	<p>Окружайте пробелами двоеточия, кроме использования в case-выражениях.</p> <pre> case 100: case 100 : </pre>
3.3.5	REQ	<p>Ставьте пробел до и после бинарных операторов (+, -, *, &, ==, и т.д.), кроме точки.</p> <pre> sum=aValue+ bValue; sum = aValue+ bValue; sum = aValue + bValue; z = 2*x + 3*y; z = 2 * x + 3 * y; z = (2 * x) + (3 * y); </pre>
3.3.6	REQ	<p>Делайте пробельный интервал:</p> <ul style="list-style-type: none"> • после запятой (,); <i>// например, в списке параметров</i>

		<ul style="list-style-type: none"> • после зарезервированных слов; // например, между ключевым словом и скобкой • перед открывающей скобкой; • после точки с запятой (;); // в заголовке for • после операции приведения типа; <p>но не ставьте между именем метода и скобкой, перед/после унарного оператора.</p> <pre> a = (b + c) * d; a=(b+c)*d; while (true) { while(true){ doSomething(a, b, c, d); doSomething(a,b,c,d); for (i = 0; i < 10; i++) { for(i=0;i<10;i++){ void doIt(a, b, c) { void doIt (a, b, c) { a++; a ++ (int) value </pre>
3.3.7	ADV	<p>При объявлении и инициализации переменных используйте «табличное» форматирование:</p> <pre> string name = "Mr. Ed"; int myValue = 5; Test aTest = Test.TestYou; </pre>
3.3.8	REQ	<p>Открывающая фигурная скобка «{» должна начинаться после названия класса или его метода на той же строке. Перед ней один стандартный символ пробела.</p>
3.3.9	REQ	<p>Между скобками и условием пробел не ставится. Исключение: если выражение внутри скобок сложное (более одного оператора), то оно отделяется от скобок пробелами.</p> <pre> If ((nVeryLongOp1) && (nVeryLongOp2)) { int i = (!sMobnum.empty() && IsBlocked(sMobnum)) ? 0 : 1; </pre>
3.3.10	REQ	<p>Выравнивайте соответствующие друг другу операторы цикла/условия/перехода по вертикали. Соответствующая закрывающая фигурная скобка «}» должна находиться на уровне открывающей и охватывать весь блок кода.</p>
3.3.11	REQ	<p>Код, который следует под открывающей фигурной скобкой «{», должен находиться на смещении 4-х символов пробела.</p>

3.3.12	REQ	Для отступа в коде с использованием операторов if, if-else, if else-if else, switch, как правило, используйте 4 пробела.
3.3.13	REQ	Используйте фигурные скобки для if, while, for даже когда в выражение входит только одна строка кода, а также при записи нескольких операторов в ветви then или в ветви else. В последнем случае размещайте операторы на нескольких строках с отступами.
3.3.14	REQ	Скобки вокруг возвращаемого выражения не ставятся, кроме случаев, когда их использование улучшает читаемость. <pre>return disk.getSize(); return (disk.getSize()); return (size ? size : defaultSize);</pre>
	PS	3.4 Операторы цикла/условия/перехода
3.4.1	REQ	Оформляйте операторы if следующим образом: <pre>If (condition) { statements; } If (condition) statements; If (condition) { statements; } else { statements; } If (condition) { statements; } else { statements; } if (condition) { statements; } else if (condition) { statements; } else { statements; }</pre>
3.4.2	REQ	Оформляйте операторы for следующим образом: <pre>for (initialization; condition; update) { statements; } for (initialization; condition; update);</pre>




3.4.3	REQ	<p>Оформляйте операторы while следующим образом:</p> <pre> while (condition) { statements; } while (condition); </pre>
3.4.4	REQ	<p>Оформляйте операторы do-while следующим образом:</p> <pre> do { statements; } while (condition); </pre>
3.4.5	REQ	<p>Оформляйте операторы switch следующим образом:</p> <pre> switch (condition) { case ABC: statements; case DEF: statements; break; case XYZ: statements; break; default: statements; break; } </pre>
3.4.6	REQ	<p>Оформляйте операторы try-catch-(finally) следующим образом:</p> <pre> try { statements; } catch (ExceptionClass e) { statements; } try { statements; } catch (ExceptionClass e) { statements; } finally { statements; } </pre>
<h2>4. Соглашение об именовании</h2>		
4.1	REQ	<p>Для наименований применяйте следующие нотации (соглашения):</p> <p>PascalCasing (<i>каждое логическое слово должно начинаться с заглавной (большой) буквы</i>); <i>TestCounter</i></p> <p>camelCasing (<i>первый символ первого логического слова со строчной (маленькой) буквы, остальные логические слова с большой, подобно Pascal Casing</i>); <i>testCounter</i></p>

		<p>UPPER CASE (только заглавные буквы); <i>COUNTER</i></p> <p>lower case (используются только строчные буквы); <i>counter</i></p> <p>Не используйте венгерскую нотацию (в том числе суффиксную и постфиксную приставку для именования переменных), кроме случаев доработок старого кода</p> <pre>private String sFirstName; // s для String private String mFirstName; private String _FirstName; private String firstName; ArrayList lstCurrencies;</pre>
		4.1 Наименование пакетов и импорт-выражений
4.1.1	REQ	Используйте lower case соглашение для наименования пакетов.
4.1.2	ADV	<p>Старайтесь ограничивать название пакета 10 символами и не используйте в имени пакета несколько логических слов.</p> <pre>ru.bank.forex ru.bank.bankforex</pre>
4.1.3	ADV	Предпочтительнее в имени пакета (и в именах частных импорт-выражений) отражать компонентную структуру приложения, однако допускается применение других стратегий именования пакетов, специфичных для конкретного проекта, если такое является целесообразным.
4.1.4	REQ	<p>Иерархия в имени пакета и в именах частных импорт-выражений должна начинаться с доменного имени ru.bank:</p> <pre>ru.bank. ru.alfaforex.orrpp ru.upris.orfes ru.bank.orpfemm //допустимо использовать аббревиатуру //отдела или управления ADV</pre> <p>Далее идет разделение:</p> <pre>ru.bank.forex.server // серверная часть, которая в свою очередь делится на модули: ru.bank.forex.server.blog ru.bank.forex.server.qp ru.bank.forex.server.bbc ru.bank.forex.utils // классы утилиты и различные хелперы</pre>

		4.2 Наименование типов (классов и интерфейсов)
4.2.1	REQ	Для именования классов, конструкторов классов, интерфейсов используйте PascalCasing соглашение.
4.2.2	REQ	<p>Если имя содержит несколько слов, то каждое слово начинайте с заглавной буквы. <i>Исключение: для известных акронимов используйте UPPER CASE формат.</i></p> <p>Класс (конструктор) должны именоваться как существительное в единственном или множественном числе, без использования каких-либо префиксов в наименовании класса.</p> <p><i>TestCounter</i></p> <p>Аналогично задаются имена интерфейсов, хотя они не обязательно должны быть существительными. Можно использовать английский суффикс able.</p> <p>В качестве префикса не используйте заглавную букву “I” в наименовании интерфейса.</p> <p><i>Runnable</i> <i>ISerializable</i></p>
		4.3 Наименование методов
4.3.1	REQ	Используйте camelCasing соглашение - имена методов начинайте с маленькой буквы. Если имя содержит несколько слов, то каждое следующее слово начинается с заглавной буквы.
4.3.2	ADV	Везде, где возможно, называйте методы похожим образом, как в стандартных классах используемого языка, для того, чтобы они были легко понятны всем разработчикам.
4.3.3	ADV	<p>Имена методов должны быть глаголами и обозначать действия, которое совершает данный метод, т.е. преимущественно используйте конструкцию глагол-объект для именования методов.</p> <ul style="list-style-type: none"> • методы, предназначенные для чтения и изменения значения переменной, начинайте с get и set соответственно; <i>getName () ,</i> • метод, возвращающий длину, называйте <i>getSize () ;</i> • метод, который проверяет булевское условие, начинайте с префикса «is» или «has»; <i>isVisible () // компонента графического пользовательского интерфейса</i> • метод, который преобразует объект в формат F, называйте <i>toF () ;</i> <i>toString () // приводит любой объект к строке.</i>

4.3.4	ADV	<p>Старайтесь давать осмысленные имена методам.</p> <pre>showUserInfo () getRequestByName ()</pre>
		4.4 Наименование переменных (полей и экземпляров класса)
4.4.1	REQ	Используйте camelCasing соглашение - имена переменных начинайте с маленькой буквы. Если имя содержит несколько слов, то каждое следующее слово начинается с заглавной буквы.
4.4.2	REQ	<p>Имена переменных должны быть существительными.</p> <pre>testCounter</pre>
4.4.3	ADV	Как правило, используйте следующие переменные i, j, k, l, m, n для оформления циклов.
		4.5 Наименование констант
4.5.1	REQ	Используется UPPER CASE соглашение - имена констант записывайте полностью заглавными буквами.
4.5.2	REQ	<p>Если имя состоит из нескольких слов, то между ними ставьте знак подчеркивания.</p> <pre>COLOR_RED</pre>
4.5.3	REQ	<p>Если константы образуют группу, тогда рекомендуется использовать одно или несколько одинаковых слов в начале имен:</p> <pre>COLOR_RED RED_COLOR COLOR_GREEN COLOR_BLUE</pre>
4.5.4	ADV	Не используйте при записи длинных целых констант заглавную латинскую букву I, ее легко спутать с единицей, и букву O, которую легко принять за ноль.
4.5.5	ADV	Строки, используемые в основном для диагностики, как, например, сообщения в лог или описатели исключений, надо оставлять прямо в коде, если они не используются больше одного раза. Иначе эти строки объявляются как константы.
		4.6 Наименование локальных переменных и параметров методов, обработчиков ошибок.
4.6.1	ADV	Имена локальных переменных, как правило, устанавливайте довольно короткие, но тем не менее они должны быть осмыслены.

4.6.2	ADV	<p>Старайтесь объявлять переменные в начале блока (часть кода, обрaмленная {}), в котором они будут использоваться.</p> <pre> int totalWide; int firstWide = 20; int secondWide = 12; firstWide = doFoo(firstWide, secondWide); doBar(firstWide, secondWide); totalWide = firstWide + secondWide; int secondWide = 12; int firstWide = doFoo(20,secondWide); doBar(firstWide, secondWide); int totalWide = firstWide + secondWide </pre>
		4.7 Идентификаторы
4.7.1	REQ	<p>Имена идентификаторов в программе должны отражать назначение данного класса, переменной, метода или параметра. Создавайте имена идентификаторов настолько короткими, насколько это возможно, без потери смыслового значения. Избегайте похожих имен идентификаторов.</p> <pre> int a int priceCurr check1(ByRef obj) checkObjText(ByRef obj) </pre>

		5. ПРОФИЛИ НАСТРОЕК
	IDE	
5.1	Eclipse	  DPR_Eclipse_clean.xml ³ DPR_Eclipse_f.xml ⁴
5.2	IDEA	 DPR_IDEA .xml ⁵

³ Рекомендуется использовать для очистки кода. См. интернет-источник № 2.

⁴ Рекомендуется использовать для форматирования кода

⁵ Рекомендуется использовать для форматирования кода

--	--	--

		ЛИТЕРАТУРА И ИНТЕРНЕТ-ИСТОЧНИКИ
1.		«Java Code Conventions» . Sun Microsystems.
2.		«Java Programming Style Guidelines». Geotechnical Software Services.
3.		«How to Write Doc Comments for the Javadoc Tool». Sun Microsystems.
		Интернет-источники
1.		http://Java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html
2.		http://www.ibm.com/developerworks/ru/library/os-eclipse-clean/index.html
3.		http://Java.sun.com/j2se/Javadoc/writingdoccomments/#examplesresult
4.		http://www.arinnav.ru/js/Java02.htm
5.		http://www.skipy.ru/index.html
6.		http://www.interface.ru/home.asp?artId=3567