ОТ ИСПОЛНИТЕЛЯ
СОГЛАСОВАНО

ОТ ЗАКАЗЧИКА
УТВЕРЖДАЮ
Руководитель проекта

_____ А.И.Власов
«___»_____ 2006 года

СОГЛАШЕНИЕ ПО РАЗРАБОТКЕ

МОСКВА 2006

СОДЕРЖЕНИЕ

1 ОБЩИЕ ПОЛОЖЕНИЯ

1.1 НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

Настоящая документированная процедура устанавливает требования к исходным текстам программ и правила использования инструментальных средств среды программирования.

2 НАИМЕНОВАНИЕ ФАЙЛОВ

Данный раздел включает в себя описание используемых названий и расширений файлов.

2.1 РАСШИРЕНИЯ ФАЙЛОВ

JSF приложения используют следующие расширения файлов

Тип файла	**Расширение**
Файл с исходным кодом Java	.java
Скомпилированный файл Java	.class
Java-архив	.jar
Файлы конфигурации приложения	.xml
Файл с исходным кодом серверной страницы java	.jsp
Файл визуального представления компонента JSF	.xhtml
Файл каскадных таблиц стилей	.css
Файлы скриптов для СУБД	.sql

2.2 ИМЕНА ФАЙЛОВ

Правила формирования имен файлов определяются применяемыми инструментальными средствами. Например, для файлов с исходным кодом Java имя файла совпадает с названием класса или интерфейса который описан в данном файле.

Если новый файл не относится к категории, для которой правило формирования наименования определяется применяемым инструментальным средством, то имя файла согласовывается с руководителем разработки.

В именах файлов любых типов НЕЛЬЗЯ использовать символы подчеркивания (вместо подчеркивания необходимо использовать дефис).

Ниже перечислены наиболее часто используемые имена фалов в JSF приложениях

Имя файла	**Описание**
README	Рекомендуемое имя для файла, включающего описание содержимого каталога, в котором он находится
Index.jsp	Имя файла начальной страницы для каталога в котором она находится.

3. СТРУКТУРА ФАЙЛОВ

Файл состоит из разделов, которые должны быть отделены друг от друга пустыми строками и дополнительными комментариями, идентифицирующими каждый раздел. Файлы длиннее 2000 строк тяжелы для восприятия. Их нужно избегать.

Пример файла исходного кода Java, форматированный должным образом см. п. 12.1 «Примеры оформления исходного кода Java»

Пример файла исходного кода серверной страницы Java, форматированный должным образом см. п. 12.2 «Примеры оформления исходного кода JSP»

3.1 ФАЙЛЫ ИСХОДНОГО КОДА JAVA

Каждый файл исходного кода Java содержит единственный `public` класс или интерфейс. Если private классы и интерфейсы связаны с `public` классом, допускается поместить их в одном файле. Описание `public` класса или интерфейса должно быть первым в файле.

Файлы исходного кода Java имеют следующую структуру:

- Начальные комментарии (см. "Начальные комментарии")
- Объявление Пакета (`packages`) и Импортируемых библиотек (`import`)
- Объявление класса (`class`) или интерфейса (`interface`) (см. "Объявление класса/интерфейса")

3.1.1 НАЧАЛЬНЫЕ КОММЕНТАРИИ

Все файлы исходного кода должны начинаться с c-style комментариев которые включают в себя название класса, информацию о версии, дата создания и информацию об авторских правах.

```
/*
* Имя класса
*
* Информация о версии
*
* Дата
*
* Информация об авторских правах
*/
```

3.1.2 ОБЪЯВЛЕНИЕ ПАКЕТА И ИМПОРТИРУЕМЫХ БИБЛИОТЕК

Первой не комментированной строкой любого файла исходного кода Java должна быть строка, содержащая имя пакета, начинающаяся со слова `package`. После нее могут следовать произвольное число строк описывающих импортируемые библиотеки, начинающиеся со слова `import`. Например:

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

3.1.3 ОБЪЯВЛЕНИЕ КЛАССА/ИНТЕРФЕЙСА

Таблица описывает части объявления класса/интерфейса. Пример файла исходного кода Java иллюстрирующий описываемые блоки см. п. 11.1 «Примеры оформления исходного кода Java»

	Раздел объявления класса/интерфейса	Описание
1	Документируемые комментарии класса/интерфейса(`/**...*/`)	См. “Документируемые комментарии” Чтобы посмотреть информацию, которая описывается в данных комментариях.
2	Объявление `class` или `interface`	
3	Комментарии реализации класса/интерфейса (`/*...*/`), если необходимо	Данные комментарии содержат любую информацию о реализации класса/интерфейса, которая не будет использована в документируемых комментариях.
4	Константы (`final`)	Сначала описываются переменные класса `public`, затем переменные `protected`, затем переменные пакетного уровня (доступ на изменение запрещен), и затем `private` переменные.
5	Переменные класса (`static`)	Сначала описываются переменные класса `public`, затем переменные `protected`, затем переменные пакетного уровня (доступ на изменение запрещен), и затем `private` переменные.
6	Переменные экземпляра	Сначала описываются `public` переменные, затем `protected`, затем переменные пакетного уровня (доступ на изменение запрещен), и затем `private` переменные.
7	Конструкторы	
8	Методы	Методы должны быть сгруппированы по функциональности, а не по уровню доступа или области видимости. Например: описание `private` метода класса может располагаться между описаниями двух `public` методов класса. Основная цель – сделать код более понятным и читаемым.

3.2 ФАЙЛЫ СЕРВЕВНЫХ СТРАНИЦ JAVA

3.3 ФАЙЛЫ SQL

Каждый запрос должен иметь вначале описание всех возвращаемых колонок, или набор документированных индексов, применяемых для доступа к этим полям как элементам массива, полученного в результате вызова GetRows.

Для формирования текста запросов рекомендуется использовать следующий шаблон:

```
<Select>|<update>|<insert><…>
        [Колонка 1]
        ...
        [Колонка n]
from
        <dbo>.[Таблица 1] [as]…
        [right | left|outer|inner] join <dbo>.[Таблица Y] [as] <alias> on <Condition1>…
        …
        <dbo>.[Таблица n] as…
[where
        search_condition1
        search_condition2... ]
[ GROUP BY
        group_by_expression ]
[ HAVING
        search_condition ]
[ ORDER BY
        order_expression [ ASC | DESC ] ]
```

3.3.2 ТРЕБОВАНИЕ К ОФОРМЛЕНИЮ SQL-ПРОГРАММ ПРИ ПРОВЕДЕНИИ ИЗМЕНЕНИЙ В БАЗЕ ДАННЫХ.

Имена sql-программ должны включать следующие атрибуты: **NNXXXX.SQL**
где:

- NN - порядковый номер SQL-файла при запуске;
- XXXX - имя подсистемы разработчика - 3 - 4 символа

Например: **2exam.sql** .

Необходимо предусмотреть возможность запуска sql-файлов в назначенной последовательности несколько раз.

При выполнении sql-программы должен включаться вывод результатов в файл. Для этого необходимо добавить:

- в начало файла:
  SPOOL xxxxxxxx.lst
  где xxxxxxxx - имя файла sql-программы;
- в конец файла:
  SPOOL off.

Необходимо, чтобы в sql-программе при выполнении действий с объектом базы данных отражалось имя этого объекта, например:

PROMPT создается TABLE1
CREATE TABLE1 ...

Следует включить в команду CREATE INDEX опцию TABLESPACE и предусмотреть возможность интерактивной подстановки имени табличного пространства администратору базы данных в момент выполнения sql-программы:

CREATE INDEX i_table1 ON table1 ...TABLESPACE &&tablespace;

Перед командой создания синонимов должна ставиться команда удаления синонимов с тем же именем:

DROP PUBLIC SYNONYM xxx. ..
CREATE PUBLIC SYNONYM xxx ...

В случае необходимости выборочной выгрузки таблиц необходимо подготовить файл параметров с названием таблиц в следующем формате:

tables=username.table_name1
tables=username.table_name2

где username - владелец таблиц, table_name - название таблицы.

Данные в файл параметров использовать только прописные буквы.

4 ОТСТУПЫ

4.1. ОТСТУПЫ ДЛЯ ФАЙЛОВ ИСХОДНОГО КОДА JAVA

Для обозначения нового абзаца необходимо использовать 4 пробела. Принцип задания абзаца (пробелы или табуляция) не определяется данным соглашением. Табуляция может быть размером 8 пробелов (не 4).

4.1.1 ДЛИНА СТРОКИ

Необходимо избегать строк длинной более 80 символов, так как они не поддерживаются многими терминалами.

Примеры кода, используемые в документации, должны иметь меньшую длину строки(не более 70 символов).

4.1.2 РАЗРЫВ СТРОК

В случае, когда выражение не помещается на одну строку необходимо разрывать строку в соответствии с правилами приведенными ниже:

- Разрывать строки после запятой
- Разрывать строки перед оператором
- Предпочтительнее высокоуровневые разрывы, нежели низкоуровневые
- Новую строку выравнивать с начала выражения на том же уровне как и в предыдущей строке
- Если при выполнении правил код получается не читаемым или сильно смещается к правому краю необходимо отступить 8 пробелов

Ниже приведены примеры разрыва строк:

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

ый вариант
являе которое яв-
ляется выражением более высокого уровня.

```
longName1 = longName2 * (longName3 + longName4 - longName5)
             + 4 * longname6; // Предпочтительное
longName1 = longName2 * (longName3 + longName4
                         - longName5) + 4 * longname6; // Допустимое
```

Ниже представлены два примера выделения частей текста различными отступами. Первый является общепринятым. Во втором примере используется отступ всего в 8 пробелов, так как при использовании общепринятого правила вторая и третья строки были бы сильно смещены вправо.

```
//Общепринятые отступы
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
           Object andStillAnother) {
           ...
}
//Отступ 8 пробелов для избежания чрезмерного смещения текста вправо

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

При разрыве строки для объявления выражений `if` должен использоваться отступ в 8 пробелов, т.к. общепринятый отступ (4 пробела) усложняет просмотр содержимого. Например:

```
//НЕ СЛЕДУЕТ ИСПОЛЬЗОВАТЬ ТАКОЙ ОТСТУП
if ((condition1 && condition2)
    || (condition3 && condition4)
    ||!(condition5 && condition6)) { //ИЗ-ЗА НЕВЕРНОГО РАЗРЫВА СТРОКИ
    doSomethingAboutIt();            //ЭТА СТРОКА СЛИВАЕТСЯ С ОБЪЯВЛЕНИЕМ
}
//НЕОБХОДИМО ИСПОЛЬЗОВАТЬ СЛЕДУЮЩИЙ ОТСТУП
if ((condition1 && condition2)
        || (condition3 && condition4)
        || (condition5 && condition6)) {
     doSomethingAboutIt();
}
//ВОЗМОЖНО ИСПОЛЬЗОВАНИЕ СЛЕДУЩЕГО ОТСТУПА
if ((condition1 && condition2) || (condition3 && condition4)
```

Ниже приведены 3 допустимых способа форматирования тройных выражений:

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


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

В Java-программах имеется два вида комментариев: комментарии реализации и документируемые комментарии. Комментарии реализации обозначаются при помощи символов `/*...*/`, и `//`. Документируемые комментарии (известные как “doc comments”) обозначаются при помощи символов `/**...*/`. Документированные комментарии могут быть преобразованы в файл HTML при помощи утилиты javadoc.

Комментарии реализации предназначены для комментирования внешнего кода или для комментирования особенностей реализации. Документируемые комментарии используют для описания спецификации кода и предназначены в первую очередь для разработчиков.

Комментарии должны давать общее представление о коде и предоставлять дополнительную информацию, которая не следует непосредственно из просмотра самого кода.

Комментарии должны включать только информацию, которая является важной для понимания программы. Например, информация о том, как строится соответствующий пакет или в какой директории он находится, не должна быть включена в комментарии.

Описание нетривиальной или неочевидной реализации допустимо, но необходимо избегать дублирования информации, если она однозначно следует из кода. У избыточных комментариев велика вероятность устареть. Необходимо отказаться от комментариев, которые могут быстро устареть, т.к. код постоянно изменятся.

Иногда большое количество комментариев отражает низкое качество кода. Если появляется необходимость добавить комментарий, необходимо посмотреть можно ли сделать код более понятным. Комментарии не должны быть отделены от кода большим количеством символов или разрывов строки. Комментарии не должны включать специальных символов, таких как, перевод страницы и возврат

4.2. ОТСТУПЫ ДЛЯ СЕРВЕРНЫХ СТРАНИЦ JAVA

4.3. ОТСТУПЫ ДЛЯ ФАЙЛОВ SQL

5. КОММЕНТАРИИ

5.1 КОММЕНТАРИИ В ФАЙЛАХ ИСХОДНОГО КОДА JAVA

5.1.1 ФОРМАТ КОММЕНТАРИЕВ РЕАЛИЗАЦИИ

Возможно использовать 4 типа комментариев реализации: блочные, однострочные, замыкающие, end-of-line комментарии.

5.1.1.1 БЛОЧНЫЕ КОММЕНТАРИИ

Блочные комментарии используются для описания файлов, методов, структур данных, и алгоритмов. Блочные комментарии могут использоваться в начале каждого файла и перед реализацией каждого метода. Они так же могут использоваться в других местах, например при описании кода реализации методов. Блочные комментарии при описании кода фунций и методов должны иметь тот же отступ что и описываемый участок кода.

Блочные комментарии должны начинаться с новой строки, для того чтобы зрительно отделить их от кода.

```
/*
 * Это комментарий.
 */
```

Блочный комментарий может начинаться с символов `/*-`, которые говорят о том что блочный комментарий не должен быть переформатирован. Например:

```
/*-
 * Это блочный комментарий со специальным форматированием, форматирование которого
 * запрещено
 *
 *   один
 *           два
 *                   три
 */
```

5.1.1.2 ОДНОСТРОЧНЫЕ КОММЕНТАРИИ

Короткие комментарии, помещающиеся в одну строку, смещенную на уровень описываемого кода. Если комментарий не может быть написан на одной строке, его следует писать в формате блочного комментария. Однострочный комментарий должен начинаться с новой строки. Пример однострочного комментария:

```
if (условие) {
     /* Пояснения к условию. */
     ...
}
```

5.1.1.3 ЗАМЫКАЮЩИЕ КОММЕНТАРИИ

Очень короткие комментарии, которые могут поместиться на одной строке с описываемым кодом, но при этом они должны быть смещены для того, чтобы их зрительно можно было отличить от кода. Если более одного короткого комментария требуется к разделу кода, они должны быть смещены на одинаковое расстояние.

Пример замыкающих комментариев:

```
if (a == 2) {
     return TRUE;          /* специальное условие */
} else {
     return isPrime(a);    /* работает только при нечетных a */
}
```

Комментарии обозначенные `//` могут комментировать строку или часть строки. Этот тип комментариев не следует использовать, если требуется написать комментарий в несколько строк, однако его можно использовать для того, чтобы закомментировать блоки кода. Пример:

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

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

5.1.2 ДОКУМЕНТИРУЕМЫЕ КОММЕНТАРИИ

Пример описываемых в данном разделе документируемых комментариев можно найти п. 11.1 «Примеры оформления исходного кода Java».

Более детальную информацию можно найти в руководстве “Как писать документируемые комментарии для Javadoc” по адресу:

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

В этом руководстве имеется информация о тэгах документируемых комментариев (`@return`, `@param`, `@see`,).

Также подробную информацию по javadoc можно найти на официальном сайте:

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

Документируемые комментарии описывают Java классы, интерфейсы, конструкторы, методы и поля.

Каждый документируемый комментарий заключен между символами `/**...*/`, по одному комментарию на класс, интерфейс или другой объект. Этот тип комментариев должен начинаться непосредственно перед объявлением.

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

Объявление классов и интерфейсов пишется без отступа, в то время как нижеследующие определения с отступом. Первая строка документируемых комментариев (/**) для классов и интерфейсов пишется без отступа. Каждая последующая строка документируемых комментариев должна иметь отступ в 1 пробел (выравнивание по идет по символам *).

Если необходимо записать информацию о классе, интерфейсе, переменной или методе, которая не требуется в документации, следует использовать блочные или однострочные комментарии непосредственно после объявления. Например, детали реализации класса должны быть описаны в блочном комментарии реализации следующим за объявлением класса, а не в документируемых комментариях.

Документированные комментарии не должны располагаться внутри блока описания метода или конструктора, т.к. Java связывает документируемые комментарии с первым объявлением, которое следует сразу после их окончания.

5.2. КОММЕНТАРИИ СЕРВЕРНЫХ СТРАНИЦ JAVA

5.3. КОММЕНТАРИИ ФАЙЛОВ SQL

Для каждого SQL-запроса (исключая тривиальные типа "SELECT COUNT(*) FROM <имя_таблицы>") должен быть комментарий, описывающий назначение запроса, например: "получаем список избирательных комиссий, подчиненных комиссиям, находящимся на территории с идентификатором nAreaID".

5.3.1 END-OF-LINE КОММЕНТАРИИ

Комментарии обозначенные -- могут комментировать строку или часть строки. Этот тип комментариев не следует использовать, если требуется написать комментарий в несколько строк, однако его можно использовать для того, чтобы закомментировать блоки скрипта. Необходимо чтобы между символами -- и текстом комментария должен быть пробел. Пример:

```
-- Это комментарий.
```

5.3.2 БЛОЧНЫЕ КОММЕНТАРИИ

Блочные комментарии используются если необходимо написать комментарии длинной более одной строки. Блочные комментарии могут использоваться в начале каждого файла и перед описанием каждого sql-запроса. Блочные комментарии при описании кода фунций и методов должны иметь тот же отступ что и описываемый участок кода.

Блочные комментарии должны начинаться с новой строки, для того чтобы зрительно отделить их от кода.

```
/*
 Это комментарий.
*/
```

5.3.3. КОММЕНТИРОВАНИЕ ХРАНИМЫХ ПРОЦЕДУР, ФУНКЦИЙ И VIEW

В начале процедуры, функции или VIEW должно быть описано назначение и все параметры.

Если процедура, функция или VIEW используется компонентом или файлом с именованием, отличным от наименования процедуры, функции или VIEW, необходимо в описании указать, кем используется процедура, функция или VIEW.

6 ОБЪЯВЛЕНИЯ В ФАЙЛАХ ИСХОДНОГО КОДА JAVA

6.1 КОЛИЧЕСТВО ОБЪЯВЛЕНИЙ В СТРОКЕ

Рекомендуется делать одно объявление в строке, т.к. это облегчает чтение кода и комментирование. Другими словами:

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

предпочтительнее чем

```
int level, size;
```

Не делать объявления разнотипных объектов в одной строке. Например:

```
int foo, fooarray[]; //НЕ ВЕРНО!
```

В приведенных примерах используется разрыв в один пробел между идентификатором и типом. Возможно так же использовать для их разделения символ табуляции. Например:

```
Int           level;        // Уровень вложенности
Int           size;         // Размер таблицы
Object        currentEntry; // Выбранная ячейка в таблице
```

6.2 ИНИЦИАЛИЗАЦИЯ

Необходимо стараться инициализировать локальные переменные в месте их объявления. Исключениями являются случаи при которых для инициализации переменной требуется провести предварительные расчеты.

6.3 РАСПОЛОЖЕНИЕ

Размещать объявления необходимо в начале блоков (фрагментов кода ограниченного фигурными скобками “{“ и “}”). Не следует объявлять переменную в месте ее первого использования, т.к. это затрудняет чтение кода

```
void myMethod() {
       int int1 = 0;            // Начало блока метода
       if (condition) {
                int int2 = 0;  // начало блока «if»
                ...
       }
}
```

Единственное исключение из этого правила касается индексов для цикла `for`, которые могут быть объявлены в Java непосредственно в выражении `for`. Пример:

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

При совпадении деклараций в различных областях видимости кода необходимо использовать вызовы переменных через `this` во всех вложенных блоках кода. Например:

```
int count;
...
myMethod() {
       if (condition) {
              int count;
              this.count = 1; // Работа с локальной переменной
              ...
       }
       ...
}
```

При описании классов и интерфейсов необходимо придерживаться следующих правил:

- Не делать пробелов между названием метода и открывающейся скобкой “(“ указывающей на начала блока объявления параметров.
- Фигурная скобка “{” должна ставиться в конце строки, на которой делается объявление
- Фигурная скобка “}” должна находится на отдельной строке и иметь такое же смещение как и объявление. В случае, когда блок выполнения пуст фигурная скобка “}” должна следовать непосредственно за ”{”.
- Методы должны разделяться пустой строкой

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

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

       int emptyMethod() {}
       ...
}
```

7 ОПЕРАТОРЫ В ФАЙЛАХ ИСХОДНОГО КОДА JAVA

7.1 ПРОСТЫЕ ОПЕРАТОРЫ

Каждая строка должна содержать не более одного оператора

Пример:

```
argv++;             // Верно
argc++;             // Верно
argv++; argc--;     // СЛЕДУЕТ ИЗБЕГАТЬ!
```

7.2 СОСТАВНЫЕ ОПЕРАТОРЫ

Составные операторы – это операторы, которые представляют из себя список операторов заключенный в фигурные скобки “{” и “}”.

Вложенные операторы должны быть смещены на один уровень относительно составной оператор.

Открывающая фигурная скобка “{” должна находится в конце строки которая, на которой определяется составной оператор. Закрывающая фигурная скобка “}” должна стоять в начале строки и иметь тоже смещение что и строка объявления составного оператора.

Фигурные скобки необходимо использовать повсеместно во всех операторах, даже в одиночных, где скобки являются частью структуры кода, таких как операторы `if-else` или `for`. Это упрощает добавление операторов, снижая риск забыть добавить скобку.

7.3. ОПЕРАТОР RETURN

При написании оператора `return` не должны использоваться скобки кроме тех случаев когда их использование позволяет сделать код более понятным. Например:

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

7.4. ВЫРАЖЕНИЯ IF, IF-ELSE, IF ELSE-IF ELSE

Выражения `if-else` должны иметь следующую форму:

```
if ( условие) {
    выражения;
}

if ( условие) {
    выражения;
} else {
    выражения;
}

if ( условие) {
    выражения;
} else if ( условие) {
    выражения;
} else {
     выражения;
}
```

Выражение `if` необходимо всегда использовать с фигурными скобками {}. Необходимо избегать упрощенную запись.

```
if ( условие)   //Необходимо избегать! Необходимо использовать {}!
      выражение;
```

7.5 ВЫРАЖЕНИЯ FOR

Выражения `for` должны иметь следующую форму:

```
for ( инициализация; условие; обновление переменной) {
      выражения;
}
```

условия и обновление переменной) должно иметь следующую форму:

```
for ( инициализация; условие; обновление переменной);
```

При использовании запятой в разделах инициализация или обновление переменной выражения `for`, необходимо избегать более трех переменных. Если необходимо, следует разделить выражения на до цикла `for` (для инициализации) или после `for` (для описания других значений при обновлении переменной)

7.6 ВЫРАЖЕНИЯ WHILE

Выражения `while должны иметь следующую форму:`

```
while ( условие) {
      выражения;
}
```

Пустое выражение `while` должно иметь следующий вид:

```
while ( условие);
```

7.7 ВЫРАЖЕНИЯ DO-WHILE

Выражения `do-while` должны иметь следующий вид:

```
do {
    выражения;
} while ( условие);
```

7.8 ВЫРАЖЕНИЯ SWITCH

Выражения `switch` должно иметь следующую форму:

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

В случае когда выбран блок с дальнейшим исполнением кода (блок case не содержит оператора `break`), необходимо добавлять комментарий. Это изображено на примере в блоке с комментарием `/* falls through */`.

Каждое выражение `switch` должно включать вариант, который выполняется по умолчанию (блок `default`). Оператор `break` в данном случае является излишним, но он предотвращает ошибку если в последствии будет добавлен другой блок `case`.

7.9 ВЫРАЖЕНИЯ TRY-CATCH

Выражения `try-catch` должны иметь следующий вид:

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

В выражениях `try-catch` может так же использоваться блок `finally`, который выполняется в независимости от того успешно ли завершил свою работу блок `try`.

```
try {
      выражения;
} catch (ExceptionClass e) {
      выражения;
} finally {
      выражения;
}
```

8. Общие рекомендации по применению хранимых процедур, функций и View в файлах SQL.

View рекомендуется применять только для форматированного вывода и не должны участвовать в Update- и Delete-операторах (и в сложных выборках).

Все хранимые процедуры должны возвращать 0 в случае успеха или отрицательное число с кодом ошибки.

Рекомендуется избегать **неоправданной** рекурсии в хранимых процедурах и функциях.

Создание общих синонимов должно прописываться в тех же SQL-программах, что и создание объектов.

Требуется писать комментарии в тексте хранимых процедур и комментировать таблицы, представления и их поля с помощью SQL команды COMMENT.

Необходимо использовать существующую систему разграничения доступа или, по крайней мере, предусмотреть в коде представлений и хранимых процедур места, куда в будущем можно было бы вставить процедуры системы разграничения доступа. Эти процедуры отвечают на вопрос «имеет ли текущий пользователь право сделать данное дейст-

вие над данным объектом», а представления показывают совокупность разрешенных действий над объектами для текущего пользователя.

Хранимые процедуры не должны находиться в состоянии ожидания освобождения заблокированных другими пользователями записей длительное время. Для этого перед попыткой захвата данных необходимо убедиться в отсутствии чужих блокировок (SELECT ... FOR UPDATE ...NOWAIT), и в случае наличия блокировок пользователю должно быть выдано соответствующее сообщение.

При написании хранимых процедур нужно считать, что на Delphi-клиенте выбран режим работы SHARED NOAUTOCOMMIT.

Ошибки хранимых процедур, как логические так и подключения к ORACLE, должны обрабатываться в самих процедурах с выполнением ROLLBACK, по крайней мере, до точки вызова процедуры. При этом к клиенту не должен возвращаться текст сообщения ORACLE SERVER об ошибке. Вместо этого клиенту должен возвращаться код ошибки и подробное описание ошибки, созданное разработчиком.

В начале процедуры должны быть комментарии, отражающие следующие сведения:

- наименование подсистемы;
- наименование модуля;
- версия и дата последнего обновления;
- фамилия разработчика;
- краткое назначение;
- комментарий к каждому входному и выходному параметру;
- производит ли процедура фиксацию транзакций во время своей работы;
- до какой точки производится откат транзакций в случае ошибки;
- оставляет ли процедура заблокированные записи в таблицах после выполнения своей работы.

9 ПРОБЕЛЫ, ТАБУЛЯЦИЯ, ПУСТЫЕ СТРОКИ

9.1 ПРОБЕЛЫ, ТАБУЛЯЦИЯ, ПУСТЫЕ СТРОКИ В ФАЙЛАХ ИСХОДНОГО КОДА JAVA

9.1.1 ПУСТЫЕ СТРОКИ

Пустые строки улучшают читаемость кода разбивая его на секции.

Две пустые строки в двух случаях:

- Между секциями файла исходного кода
- Между определениями класса и интерфейса

Одна пустая строка используется в следующих случаях:

- Между методами
- Между локальными переменными и первым выражением в теле метода
- Перед блочными или однострочными комментариями
- Между логическими блоками внутри метода для улучшения читаемости кода

9.1.2 ПРОБЕЛЫ

Пробелы должны использоваться в следующих случаях:

- Ключевое слово и следующая за ним открывающая скобка должны быть разделены пробелом. Например:

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

Пробелы не должны разделять название метода и следующую за ним открывающую скобку. Это помогает отличать ключевые слова от вызова методов.

- Пробел должен стоять после запятой в списке аргументов.
- Все бинарные операторы исключая . должны быть разделены при помощи пробела. Пробел никогда не разделяет унарные операторы такие как унарный минус, инкремент (“++”),и декремент (“--”) от их операндов. Например:

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

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

- Выражения в цикле `for` должны быть разделены пробелами. Например:

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

10. СОГЛАШЕНИЕ ОБ ИМЕНАХ

10.1 СОГЛАШЕНИЕ ОБ ИМЕНАХ В ФАЙЛАХ ИСХОДНОГО КОДА JAVA

Соглашение об именах делает программы более понятными и упрощает их чтение. Также соглашение может дать информацию о функции, выполняемой тем или иным идентификатором. Например, является ли запись константой, пакетом или классом, что может быть полезным для понимания кода.

Тип идентификатора	**Правила именования**	**Пример**

Пакеты	Префикс уникального имени пакета всегда пишется в нижнем регистре ASCII символов и должен указывать на название одного из доменов верхнего уровня, таких как com, edu, gov, mil, net, org или один из двухбуквенных кодов стран на английском языке (коды определены стандартом ISO 3166, 1981).	`com.sun.eng`
		`com.apple.quicktime.v2`
		`edu.cmu.cs.bovik.cheese`

Классы	Названия классов должны быть именами существительными. Регистр написания – смешанный, с первой строчной буквой и заглавной буквой каждого нового слова. Необходимо давать классам простые и понятные названия. Необходимо использовать целые слова без сокращений и аббревиатур (за исключением когда аббревиатуры являются общеизвестными, такие как URL или HTML).	`class Raster;` `class ImageSprite;`
Интерфейсы	Интерфейсы именуются по такому же принципу что и классы.	`interface RasterDelegate;` `interface Storing;`

Методы	Названия методов состоят из символов различного регистра. Первая буква названия всегда строчная, а каждая последующая буква нового слова – заглавная	`run();`
		`runFast();`
		`getBackground();`

Переменные	За исключением переменных, все объекты, классы и константы класса именуются в смешанном регистре, начиная со строчной буквы. Каждое новое слово начинается с заглавной буквы. Названия переменной не должны начинаться с символов _ или знака доллара $, несмотря на то, что эти символы допустимы.
	
	Имена переменных по возможности должны быть короткими, но при этом отражать функции переменной.
	
	Выбор имени переменной должен быть мнемонический, т.е. указывающий случайному читателю Назначение той или иной переменной.
	
	Односимвольные имена являются исключением для временных «одноразовых» переменных. Обычно таким переменные дают следующие имена:
	
	i, j, k, м, and n для целых типов;
	
	c, d, e для символьных типов.
		`int i;`
		
		`char c;`
		
		`float myWidth;`

Константы	Имена переменных объявленных константами внутри класса и ANSI констант должны содержать символы только в верхнем регистре с словами разделяемыми символом подчеркивания (“_”). ANSI констант следует избегать для упрощения отладки.	`static final int MIN_WIDTH = 4;`
		
		`static final int MAX_WIDTH = 999;`
		
		`static final int GET_THE_CPU = 1;`

10.2 СОГЛАШЕНИЕ ОБ ИМЕНАХ В СЕРВЕРНЫХ СТРАНИЦАХ JAVA

10.3 СОГЛАШЕНИЕ ОБ ИМЕНАХ SQL

10.3.1 ОБЩИЕ СОГЛАШЕНИЕ ОБ ИМЕНАХ SQL

Настоятельно рекомендуется использование псевдонимов имен колонок.

База данных должна состоять из модулей (совокупности объектов), каждый из которых по возможности должен являться независимой единицей и должен быть способен работать максимально автономно от других модулей.

Взаимодействие между модулями должно осуществляться с помощью хранимых процедур и представлений, что имеет целью сделать возможным изменение структуры базы модуля без переделки других модулей, которые с ним взаимодействуют. Запрещается прямое обращение к таблицам другого модуля. При таком подходе при изменении структуры базы (таблиц) в модуле для сохранения работоспособности других модулей будет достаточно переделать соответствующие представления и процедуры.

Взаимодействие клиентской части и серверной рекомендуется осуществляться с помощью хранимых процедур и представлений.

База данных должна создаваться под единым ORACLE-пользователем (владельцем объектов). Конечные пользователи должны обращаться к процедурам, функциям, представлениям владельца объектов через общие синонимы (PUBLIC SYNONYM).

Конечные пользователи системы не имеют привилегий на действия с объектами БД.

Все ORACLE-права должны раздаваться через роли, которые создаются администратором по согласованию с разработчиком. Разработчики должны включать привилегии на создаваемые объекты в ранее созданные роли.

С целью обеспечения уникальности и мнемоничности имен объектов базы необходимо придерживаться СОГЛАШЕНИЯ (Префиксное наименование таблиц) о наименовании объектов:

Имена таблиц и хранимых процедур модуля должны начинаться с какого-либо определенного префикса (2-4 буквы), за которым следует знак подчеркивания «_». Префикс должен быть одним и тем-же для всех таблиц и процедур одного модуля.

Имена создаваемых общих синонимов должны совпадать с именами самих объектов.

Имена последовательностей (SEQUENCE) должны начинаться с символов «S_», за которыми следует префикс модуля с последующим подчеркиванием, после чего идет смысловое имя индекса. При создании последовательности для первичного ключа таблицы рекомендуется такое имя последовательности: S_ИМЯ_ТАБЛИЦЫ.

Имена представлений (VIEW) должны начинаться с символов «V_», далее префикс модуля и смысловая часть (например: имя основной таблицы).

Имена индексов (INDEX) должны начинаться с символов «I_», далее имя таблицы (или его сокращение), после чего следует смысловая часть. Например: для первичных ключей рекомендуется применять аббревиатуру «_PK», для FOREIGN KEY - имена столбцов или сокращений, на которые осуществляется ссылка.

Имена ограничений на таблицы (CONSTRAINT) должны начинаться с символов «C_» (ограничения типа PRIMARY KEY должны начинаться с символов «I_», т.к. в соответствии с этим именем создается индекс), далее аналогично именам последовательностей.

Имена триггеров (TRIGGER) должны начинаться с символов «T_», далее аналогично именам последовательностей.

Имена связей (DB_LINK) должны начинаться с символов «L_», далее аналогично именам последовательностей.

Данные правила комментирования распространяются на все случаи использования запросов – в исходных текстах программ, хранимых процедурах и т.д.

Перед созданием нового объекта (база данных, таблица, хранимая процедура, и.т.п) необходимо выполнять проверку на существование объекта с таким же именем

10.3.2 ПРАВИЛА ИМЕНОВАНИЯ ХРАНИМЫХ ПРОЦЕДУР, ФУНКЦИЙ И VIEW

Должны использоваться следующие префиксы для имен хранимых процедур, функций и VIEW.

Префиксы для имен хранимых процедур, функций и VIEW

Префикс	Назначение
get_	служебные процедуры получения данных общего характера (например, выдачи списка подчиненных объектов)
app_	прикладные процедуры
r_	процедуры формирования данных для отчетов
imp_	процедуры импорта
view_	VIEW

Для хранимых процедур и функций других категорий по согласованию с руководителем разработки могут применяться иные префиксы имен.

Имена процедур рекомендуется задавать по возможности по компонентам или asp-страницам, их вызывающим. Если имя не совпадает, указывать вызывающий объект. Если в странице или компоненте вызывается несколько процедур, или надо подчеркнуть смысл операции, имя делается из двух частей, например: r_page_operation.

10.3.3 ПРЕФИКСЫ ПАРАМЕТРОВ И ПЕРЕМЕННЫХ В ХРАНИМЫХ ПРОЦЕДУРАХ И ФУНКЦИЯХ

Обязательно использовать префиксы типов для параметров и переменных/

Префиксы типов для параметров и переменных

Префикс	Тип параметра
N	Целое число
S	Varchar
C	Char
Dt	дата/время
B (f)	Булевское
I	Smallint

Bt	Tinyint
Ft	Float
Re	Real

Для параметров и переменных других типов по согласованию с руководителем разработки могут применяться иные префиксы.

10.3.4 ПРАВИЛА ФОРМИРОВАНИЯ ИМЕН ПАРАМЕТРОВ ХРАНИМЫХ ПРОЦЕДУР И ФУНКЦИЙ

Стандартные параметры хранимых процедур/

Стандартные параметры хранимых процедур

Параметр	Имя параметра	Пример
Идентификатор объекта	n<имя типа>ID	nAreaID
Идентификатор объекта в случае, если в хранимую процедуру или функцию передают несколько идентификаторов объекта одного типа	n<прилагательное><имя типа>ID	nSrcDistrictID, nDstDistrictID
Имя типа объекта	sType	
Идентификатор пользователя	sUserID	
Идентификатор процесса	nPID	
Название операции	sOP	

Для параметров других категорий по согласованию с руководителем разработки могут применяться иные ОПИСАТЕЛЬНЫЕ наименования.

10.3.5. ДОПОЛНИТЕЛЬНЫЕ СОГЛАШЕНИЯ

Все создаваемые ограничения (CONSTRAINT) должны быть поименованы. Например:

```
CREATE TABLE DAMN_TABLE
        (A CHAR CONSTRAINT I_DAMN_TABLE_PK PRIMARY KEY
    USING INDEX TABLESPACE
                &&index_tablespace);
```

а не:

```
CREATE TABLE DAMN_TABLE
        (A CHAR PRIMARY KEY); ,
```

т.к. во втором варианте название индекса генерируется автоматически и не связанно с именем таблицы.

11. ПРАВИЛА ПРОГРАММИРОВАНИЯ

11.1 ДОСТУП К ОБЪЕКТАМ И ПЕРЕМЕННЫМ КЛАССА

Не объявляйте объекты или переменные public без необходимости. Зачастую не требуется явно устанавливать или просматривать значения переменных объекта. Для этого обычно используются специальные методы, которые вернут или установят значение переменной.

Пример описывающий возможность использования `public` переменной, когда класс является структурой данных и не описывает поведение. Другими словами, если необходимо использовать `struct` вместо класса (если Java поддерживает `struct`), то необходимо описать переменную – объект класса как `public`.

11.2 ВЫЗОВЫ ПЕРЕМЕННЫХ И МЕТОДОВ КЛАССА

Необходимо избегать использования объектов для получения доступа к статическим переменным или методам. Пользуйтесь для этого методами класса. Например:

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


anObjectlass.classMethod(); //Следует избегать!
```

11.3 КОНСТАНТЫ

Числовые константы не должны объявляться в прямом виде, исключая -1,0,1 которые могут использоваться в циклах `for` как значения счетчика.

11.4 ПРИСВОЕНИЕ ЗНАЧЕНИЙ ПЕРЕМЕННЫМ

Необходимо избегать присвоения значения нескольким переменным одного значения в одном выражении, т.к. это затрудняет чтение кода. Например:

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

Не используйте операторы присвоения в местах, где они могут быть спутаны с оператором сравнения. Например:

```
if (c++ = d++) { // Следует избегать! (Java disallows)
     ...
}
```

Должно быть написано

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

ни выполнения. Так как такая структура очень сильно замедляет работу компилятора. Например:

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


Необходимо писать


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

11.5 MISCELLANEOUS PRACTICES

11.5.1 СКОБКИ

Необходимо использовать скобки при описании арифметических операций для избежания проблем с их обработкой. Даже если оператор кажется понятен, необходимо помнить, что он может быть непонятен другим программистам, которые будут работать с кодом.

```
if (a == b && c == d)       // Необходимо избегать!
if ((a == b) && (c == d)) // Следует использовать
```

11.5.2 ВОЗВАЩАЕМЫЕ ЗНАЧЕНИЯ

Старайтесь сделать структуру программы компактнее. Например:

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


Также выражение типа
if ( условие) {
     return x;
}
return y;
следует писать
return (condition ? x : y);
```

11.5.3 ВЫРАЖЕНИЯ ПОСЛЕ ‘?’ В ОПЕРАТОРАХ СРАВНЕНИЯ

Если выражение содержит бинарный оператор перед символом `?` в операторе `?:`, необходимо заключить это выражение в скобках. Например:

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

11.5.4 СПЕЦИАЛЬЫЕ КОММЕНТАРИИ

Рекомендуется использовать следующие комментарии

- XXX, чтобы указать на код который работает но который необходимо поправить.
- FIXME, чтобы указать на код который не работает и который необходимо исправить.

12. ПРИМЕРЫ ОФОРМЛЕНИЯ ИСХОДНОГО КОДА

12.1 ПРИМЕРЫ ОФОРМЛЕНИЯ ИСХОДНОГО КОДА JAVA

```
/*
 * @(#)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;

/**
 * Здесь идет описание класса.
 *
 * @version 1.82 12 Jan 2006
 * @author Имя Фамиляи
 */
public class Blah extends SomeClass {
       /* Здесь располагаются комментарии по реализации класса. */
       /** classVar1 документируемые комментарии */
       public static int classVar1;
       /**
        * classVar2 Документируемые комментарии длинной
        * более чем в одну строку
        */
```

```
        private static Object classVar2;

        /** instanceVar1 Документируемый комментарий */
        public Object instanceVar1;

        /** instanceVar2 Документируемый комментарий */
        protected int instanceVar2;

        /** instanceVar3 Документируемый комментарий */
        private Object[] instanceVar3;

        /**
         * ... Документируемый комментарий  конструктора Blah ...
         */
        public Blah() {
              // ...выполнение...
        }

        /**
         * ... Документируемый комментарий метода doSomething...
         */
         public void doSomething() {
              // ...implementation goes here...
         }

        /**
         * ... Документируемый комментарий метода doSomethingElse...
         * @param someParam описание
         */
         public void doSomethingElse(Object someParam) {
              // ...исполнение...
         }
    }
```

12.2 ПРИМЕР ОФОРМЛЕНИЯ ИСХОДНОГО КОДА JSP

```
<?xml version="1.0" encoding="UTF-8"?>
    <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:t="http://myfaces.apache.org/tomahawk">
      <jsp:text>
         <link href="../css/layerstyles.css" rel="stylesheet" type="text/css" />
```

```
            <ui:composition template="../templates/templateadmin.xhtml">
                <ui:define name="body">
                    <h:dataTable value="#{usersList.listOfUsers}" var="aUserBean"
                    styleClass="infotable"
                    rowClasses="infotablerow,infotablerow" cellpadding="0"
                    cellspacing="0" headerClass="infotableheader"
                    rowStyleClass="infotablerow">
                     <h:column>
                         <f:facet name="header">
                            Логин
                         </f:facet>
                         <h:commandLink action="#{aUserBean.viewAction}"
                          value="#{aUserBean.lnkEUser.login}">
                         </h:commandLink>
                     </h:column>
                     <h:column>
                        <f:facet name="header">
                                Имя
                         </f:facet>
                         <h:commandLink action="#{aUserBean.viewAction}"
                         value="#{aUserBean.lnkEUser.fullname}">
                         </h:commandLink>
                      </h:column>
                      <h:column>
                         <h:commandLink action="#{aUserBean.editAction}">
                           <h:graphicImage
                           value="../images/administration/Accountant2Edit2.gif"
                           styleClass="menuicon" />
                         </h:commandLink>
                      </h:column>
<!-- Начало блока комментированного кода
                            <h:column>
                               <f:facet name="header">
                                   <t:commandSortHeader columnName="name" arrow="true">
                                   </t:commandSortHeader>
                               </f:facet>
                            </h:column>
      Конец блока комментированного кода
-->
                </h:dataTable>
            </ui:define>
        </ui:composition>
    </jsp:text>
</jsp:root>
```

13. Правила для WEB-приложений

13.1 СТРУКТУРА КАТАЛОГОВ WEB-ПРИЛОЖЕНИЙ

На структуру каталогов WEB-приложения сильно влияют применяемые инструментальные средства и инфраструктура развертывания приложения (например, размещение на кластере с балансировкой загрузки). **Поэтому структура каталогов определяется руководителем разработки на основе требований применяемых инструментальных средств и требований инфраструктуры.**