Build Configuration

v. 0.1.9-SNAPSHOT

Project Documentation

Table Of Content

1 Java Coding standard

Java Coding Standard

1.1 File organization

A file consists of sections that should be separated by blank lines.

Files longer than 2000 lines are cumbersome and should be avoided.

Each java source file contains a single `public` class or interface. When `private` classes and interfaces are associated with a public class they can be put in the same source file as the `public` class. The `public` class should be the first class or interface in the file.

Java source files should have the following ordering:

- Beginning comments
- Package and import statements
- Class and interface declarations

1.1.1 Beginning comments

All source files should begin with a comment block that list the copyright notice. For example:

```
/**
 * Copyright (C) 2014 UniKnow. All rights reserved.
 *
 * This resource is subject of the following restrictions:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of  source code must  retain the above copyright  notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. The end-user documentation included with the redistribution, if any, must
 *    include  the following  acknowledgment:  "This product includes  software
 *    developed  by UniKnow."
 *    Alternately, this  acknowledgment may  appear in the software itself,  if
 *    and wherever such third-party acknowledgments normally appear.
 *
 * 4. The name ''UniKnow'' must  not  be used to  endorse or promote
 *    products derived from  this software without prior written permission.
 *
 * 5. Products  derived from this software may not  be called ''UniKnow'',
 *    nor may ''UniKnow'' appear  in their name,  without prior written
 *    permission of UniKnow.
 *
 * THIS SOFTWARE IS PROVIDED ''AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
 * FITNESS  FOR A PARTICULAR  PURPOSE ARE  DISCLAIMED.  IN NO  EVENT SHALL  WWS
 * OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY,  OR  CONSEQUENTIAL  DAMAGES  (INCLUDING, BUT  NOT  LIMITED  TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOS  OF USE, DATA, OR  PROFITS;
 * OR BUSINESS  INTERRUPTION)  HOWEVER CAUSED AND ON ANY  THEORY  OF LIABILITY,
 * WHETHER  IN CONTRACT,  STRICT LIABILITY,  OR TORT (INCLUDING  NEGLIGENCE  OR
 * OTHERWISE) ARISING IN  ANY WAY OUT OF THE  USE  OF THIS  SOFTWARE,  EVEN  IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

1.1.2 Package and import statements

The first non-comment line is a `package` statement. After that `import` statements can follow. For example:

```
package org.uniknow.agiledev

import org.uniknow.agiledev.*;
```

1.1.3 Class and interface declarations

The following table describes the parts of a class or interface declaration in the order that they should appear.

Part of class/interface declaration	Notes
Class/interface documentation comment ()	See Documentation comments for information on what should be in this comment
`class` or `interface` statement	
`static` variables	First the `public static` variables, then the `protected`, and then the `private`
Instance variables	First `public`, then `protected`, and then `private`
Constructors	
Methods	These methods should be grouped by functionality rather than by scope or accessibility. For example, a `private` method can be in between two public instance methods. The goal is to make reading and understanding the code easier.

1.2 Indentation

Four spaces should be used as the unit of indentation.

1.2.1 Line length

Avoid lines longer than 80 characters.

1.2.2 Wrapping lines

When an expression does not fit on a single line, break it according these general principles:

- Break after a comma.
- Break before an operator.
- Align the new line with the beginning of the expression at the same level on the previous line.
- If the above rules lead to confusion code or to code that’s squished against the right margin, just indent 8 spaces instead.

Here are some examples of breaking method calls:

```
function(expression1, expression2, expression3,
         expression4, expression5);

var = function(expression1,
               function2(expression1,
                         expression2));
```

Following is an example of breaking an arithmetic expression.

```
name1 = name2 * (name3 + name4 - name5)
        + 4 * name6;
```

Following are 2 examples of indenting method declarations. The first is the conventional case. The second would shift the second and third lines to the far right if it is used conventional indentation, so instead it indents only 8 spaces:

```
// CONVENTIONAL INDENTATION
someMethod(int arg1, Object otherArg, String yetAnotherArg,
           Object andStillAnother);

// INDENT 8 SPACES TO AVOID VERY DEEP INDENTS
private static synchronized verryLongMethodName(int arg1,
        Object anotherArg, String yetAnotherArg,
        Object andStillAnother);
```

Line wrapping for `if` statements should generally use the 8 space rule, since conventional (4 spaces) indentation makes seeing the body difficult.

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

Following are examples to format a ternary expression:

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

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