Coding a Java Entity Class

Overview
This document outlines the “Best Practices” and IRS standards for defining an entity class in Java. The example code is the Artist class from the **Java Immersion Training**, and may not reflect the IRS package names, and copyright data.

Package Statement
- The package statement is the first line of code in the class
- The package is in the format “gov.irs.” followed by project specific sub-packages
- The package name is in lowercase.

Example:
```
package com.gallery.bizlogic;
```

Copyright Data
- The class must have a copyright statement as a comment near the top of the source file
- The IRS standard legal text:
```
/**
 *  United States of America - Official Use Only The US Government possesses the
 *  unlimited rights throughout the world for Government purposes to publish,
 *  translate, reproduce, deliver, perform, and dispose of the technical data,
 *  computer software, or computer firmware contained herein; and to authorize
 *  others to do so.
 */
```

Example:
```
/*
 *  Copyright 2014 - The Gallery
 *
 *  Licensed under the GNU General Public License (the "License"); you may not
 *  use this file except in compliance with the License. You may obtain a copy of
 *  the License at
 *
 *  http://www.gnu.org/licenses/gpl.html
 *
 *  Unless required by applicable law or agreed to in writing, software
 *  distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
 *  WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
 *  License for the specific language governing permissions and limitations under
 *  the License.
 */
```

Import Statements

- Each imported class should be define on a separate line
- Imports that use an "*" are not allowed

Example:

```
import java.io.Serializable;

import org.apache.log4j.Logger;

import com.gallery.bizlogic.util.Address;
import com.gallery.bizlogic.util.ExtendedContactDetails;
import com.gallery.bizlogic.util.GalleryException;
```

The Class Definition

- Has a meaningful name
- The name starts with an uppercase letter, if the name has more than one word, the words are concatenated together and the initial letter of each subsequent words are in upper case
- Implements java.io.Serializable
- Has a JavaDoc comment preceding the class definition, the JavaDocs include:
  - The first sentence of each doc comment should be a summary sentence, containing a concise but complete description of the class.
  - A meaningful description of the class
  - Any examples on how to use the class
  - Recommended; a listing of any dependency jars
  - Recommended; a history section listing (major) modifications
  - @author
  - @version
  - @since (if added after release 1.0)

Example:

```
/**
 * This is the Artist class for the Gallery business logic layer
 *
 * <p>
 *       <strong>Dependencies:</strong><br />
 *       <ul>
 *                <li>log4j-1.2.15.jar</li>
 *       </ul>
 * </p>
 *
 * @author Jonathan Earl
 * @version 1.0
```

```
 *
 */
public class Artist extends GalleryEntity implements Serializable
```

Class Scope Variables

- Have a meaningful name
- Includes a class version number:
  - private static final long serialVersionUID = 1L;
- Variables should be private, a final variable maybe declared public
- The variable starts with an lowercase letter, if the name has more than one word, the words are concatenated together and the initial letter of each subsequent words are in upper case
- Final variables should be all written in uppercase, with an underscore between words
- Variables should be organized by qualifier
  - Public
  - Static
  - Final
  - Private
- Any public variable must have JavaDocs attached to it

Example:

```
private static final long serialVersionUID = 1L;

private static final Logger LOG = Logger.getLogger(Artist.class);

private String fName = null;
private String lName = null;
private String profile = null;
private Address address = null;
private ExtendedContactDetails contactDetails = null;
```

Method Scope Variables

- Have a meaningful name
- Must have an initial value
- Method scope variables that do not change should be declared as final

Example:

```
final int prime = 31;
int result = super.hashCode();
```

Constructor(s)

- Every class should have a default (no argument) constructor
- Overloaded constructors maybe defined as needed
- All constructors must include Log
- Log statements should be included in all constructors
- The mutator methods should be called from within the constructor
- Log statements should check if logging is enabled prior to writing Trace/Debug level statements
- The JavaDocs must include:
    - The Initial values for each variable

Example:

```
/**
 * The default constructor for an Artist.
 * <p />
 *
 * <strong>The Artist's attributes are initialized to:</strong>
 * <dl>
 *   <dt>Id</dt><dd>1</dd>
 *   <dt>First Name</dt><dd>Dummy</dd>
 *   <dt>Last Name</dt><dd>Dummy</dd>
 *   <dt>Profile</dt><dd>null</dd>
 *   <dt>Address</dt><dd>new default Address</dd>
 *   <dt>ExtendedContactDetails</dt><dd>new default ExtendedContactDetails</dd>
 * </dl>
 */
public Artist()
{
        if (LOG.isDebugEnabled()) LOG.debug("Default Artist Constructor");
        setFName("Dummy");
        setLName("Dummy");
        setProfile(null);
        Address address = new Address();
        setAddress(address);
        ExtendedContactDetails contact = new ExtendedContactDetails();
        setContactDetails(contact);
        }
```

Getter/Setter Methods

- Getter methods follow this coding pattern:
  - public int getSupplierId()
- Setter methods follow this coding pattern:
  - public void setSupplierId(int id)
- Log statements should be included in all methods
- Log statements should check if logging is enabled prior to writing Trace/Debug level statements
- Setters must validate the incoming field against the business rules prior to setting the instance variable
- If a validation fails an appropriate exception must be thrown
- All methods must have a Unit Test, a Unit Test should be written for each business rule
- All public methods must have JavaDocs, the setter methods must document the business rules
- The method level JavaDocs must include:
  - The business rules being validated within the method
  - @param for all incoming method parameters
  - @return for any values returned from the method
  - @throws for any Exceptions thrown in the method

Example:

```
/**
 * Setter for the First Name.
 * <p />
 * Requirements for: First Name
 * <ul>
 *     <li>May not be null</li>
 *     <li>Must be greater than 2 character in length</li>
 *     <li>May not be empty (only whitespace characters)</li>
 *     <li>Must be 25 characters or less in length</li>
 * </ul>
 *
 * @param fName the fName to set
 * @throws GalleryException if the fName does not pass the requirements listed above
 */
public void setFName(String fName)
{
        if (LOG.isDebugEnabled()) LOG.debug("setting fName: " + fName);
        if (fName == null)
        {
                LOG.error("Null fName");
                throw new GalleryException("First Name is null");
        }
        fName = fName.trim();
        if (fName.isEmpty())
        {
```

```
            LOG.error("Empty First Name");
            throw new GalleryException("First Name is empty");
        }
        if (fName.length() < 2 || fName.length() > 25)
        {
            LOG.error("First Name wrong length");
            throw new GalleryException( "First Name length < 2 or > 25 characters");
        }
        this.fName = fName;
}
```

Other Required Methods

- Each class should include the ability to print itself via an override of the toString() method from java.lang.Object
- Each class should include the ability ot do an equality comparison via an override of the equals() method from java.lang.Object
- It is strongly recommend the class includes an override of java.lang.Object.hashCode()
- The JavaDoc comments for these methods should describe the conditions for printing, or equality

Example:

```
/**
 * The toString() for the Artist class.
 * <p />
 * Outputs: "Artist [fName=" + fName + ", lName=" + lName + "]"
 *
 * @see java.lang.Object#toString()
 */
@Override
public String toString()
{
        if (LOG.isDebugEnabled())
            LOG.debug("toString");
        return "Artist [fName=" + fName + ", lName=" + lName + "]";
}


/**
 * The hashCode() method of the Artist class.
 *
 * @see java.lang.Object#hashCode()
 */
@Override
public int hashCode()
```

```
{
	final int prime = 31;
	int result = super.hashCode();
	result = prime * result + ((address == null) ? 0 : address.hashCode());
	result = prime * result
			+ ((contactDetails == null) ? 0 : contactDetails.hashCode());
	result = prime * result + ((fName == null) ? 0 : fName.hashCode());
	result = prime * result + ((lName == null) ? 0 : lName.hashCode());
	result = prime * result + ((profile == null) ? 0 : profile.hashCode());
	return result;
}


/**
 * The equals() method of the Artist class.
 * <p />
 * <strong>This method compares:</strong>
 * <ul>
 *	<li>Id</li>
 *	<li>Address</li>
 *	<li>ContactDetails</li>
 *	<li>First Name</li>
 *	<li>Last Name</li>
 *	<li>Profile</li>
 * </ul>
 *
 * @see java.lang.Object#equals(java.lang.Object)
 */
@Override
public boolean equals(Object obj)
{
	if (this == obj)
		return true;
	if (!super.equals(obj))
		return false;
	if (getClass() != obj.getClass())
		return false;
	Artist other = (Artist) obj;
	if (! super.equals(obj))
	{
		return false;
	}
	if (address == null)
	{
		if (other.address != null)
			return false;
```

```
        }
        else
                if (!address.equals(other.address))
                        return false;
        if (contactDetails == null)
        {
                if (other.contactDetails != null)
                        return false;
        }
        else
                if (!contactDetails.equals(other.contactDetails))
                        return false;
        if (fName == null)
        {
                if (other.fName != null)
                        return false;
        }
        else
                if (!fName.equals(other.fName))
                        return false;
        if (lName == null)
        {
                if (other.lName != null)
                        return false;
        }
        else
                if (!lName.equals(other.lName))
                        return false;
        if (profile == null)
        {
                if (other.profile != null)
                        return false;
        }
        else
                if (!profile.equals(other.profile))
                        return false;
        return true;
}
```