JAVA CODING GUIDELINES

Dr. Saeed Rajput

1 General Concepts

1.0 Comments

There are 3 types of comments in Java.

1.1.1 Comment Type	1.1.2 Usage	1.1.2.1 Example
JavaDoc	Use JavaDoc immediately before declarations of classes, interfaces, methods, and attributes to document them.	/** * Product – represent a * product object. */
C style	Use C-style comments to document out code that is no longer applicable.	/* *The following code was *commented out because *the business rule is no *longer valid. */
Single line	Use single line comments internally within member functions to document business logic.	// Apply a discount to // Teachers with accounts // greater than 3 years.

2.0 Standards for Packages, Classes, Interfaces

2.1 Naming Packages

The prefix of a unique package name is always written in all-lowercase letters and should be one of the top-level domain names (com, edu, net, org, etc.)

Subsequent components of the package name should comply with internal corporate naming conventions.

Package names should be singular, not plural.

Package	edu.fau.gradingsystem.db

2.2 Naming Classes and Interfaces

The standard Java convention for naming Classes and Interfaces should be in mixed case with the first letter of each internal word capitalized. Try to keep the names simple and descriptive. Use whole names and avoid acronyms (unless the abbreviation is more frequently used than the whole name, such as HTML).

The preferred convention to name a class is to use a descriptive noun.

The preferred convention to name an interface is to use a descriptive adjective such as **Runnable** or **Cloneable**, although descriptive nouns are also common. The names should be preceded by an uppercase "I" letter e.g. ILearner.

Class	Student
Interface	IRunnable, ILearner

2.3 Class Visibility

Classes may be defined with one of two visibilities: Public or Package (default). Public classes are visible to all other classes whereas package classes are visible only to other classes in the same package. Public visibility is indicated with the keyword public. Package visibility is the default and has no keyword.

2.4 Testing Classes

For each class you create, include a **main()** method that contains code to test that class. You don't need to remove the test code to use the class in a project. This code also provides examples of how to use the class.

Each class should have at least one static test method that tests its functionality. It can be invoked from the *main()* method or directly from an external regression test class. The results can be output to the standard output or to a logging file. Comments for the test method should clearly state the procedure for determining the passing and failing of the test.

3.0 Standards for Methods

3.1 Naming Methods

Methods (member functions) should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.

Method	getName() setFullName(String fullName) isPersistent() setPersistent(boolean isPersistent)

3.2 Accessor Methods

3.2.1 Getters

Getters are methods that return the value of an attribute (property). You should prefix the name "get" to the attribute, unless it's a boolean attribute and then prefix "is" to the name of the attribute instead of "get".

3.2.2 Setters

Setters, also known as mutators, are methods that modify the values of an attribute. You should prefix the word "set" to the name of the attribute.

3.3 Constructor Methods

Constructors are methods that perform any necessary initialization when an object is first created. Constructors are always given the same name as their class.

Constructor	Teacher() Student()

3.4 Method Visibility

Good design dictates that you should reduce coupling between classes. A rule of thumb is to be restrictive as possible when setting the visibility of methods. Start out with **private** visibility and only move upwards in scope if required.

Visibility	Description
Public	A public method can be invoked by any other method in any other object or class.
Protected	A protected method can be invoked by any method in the class in which it is defined or any subclasses of that class.
Private	A private method can only be invoked by other methods in the class in which it is defined.
	No visibility is indicated. This is called default or package visibility. The method is effectively public to all other classes within the same package, but private to classes external to the package.

3.5 Method Layout

Each method length should not exceed a single display screen. Ideally it should be between 8-12 lines. In some cases, it may go more than that. However, if it exceeds two screen lengths, it should be broken down into multiple methods.

One way to improve the readability of a method is to indent your code within the scope of a code block. Any code within braces, the "{" and "}" characters, forms a block. The basic idea is that the code within a block should be uniformly indented one unit.

The default for this one unit will be the **four spaces (not TABS).**

3.6 Method Documentation

Methods should be describe their functionality in their documentation.

For multiple nested control structures, put a single line comment directly after the closing curly brace "}" telling which control structure it terminated.

4.0 Standards for Attributes

An attribute is a piece of data contained in an object or in a class. Attributes may be a base data type such as a string or a float, or may be an object such as a Teacher or bank account.

4.1 Naming Attributes

Attributes should be named in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.

Attributes that are collections, such as arrays or vectors, should be given names that are plural to indicate that they represent multiple values.

Attributes	firstName zipCode sqlDatabase orderItems ← a collection

4.2 Naming Components (Widgets)

For names of components (interface widgets) you should use a full descriptor postfixed by the widget type.

Components	okButton cancelButton TeacherListBox zipCode{{??}} fileMenu

4.3 Naming Constants

In Java, constants, values that do not change, are typically implemented at *static final* attributes of classes. The recognized convention is to use full descriptive words in all uppercase with words separated by underscores.

Under no condition should any constant be used directly in the code.

Constants	MININUM_BALANCE MAX_VALUE

4.4 Attribute Visibility

Attributes should be declared **private** for reasons of encapsulation. Attributes should never be accessed directly; instead accessor methods should be used.

4.5 Attribute Documentation

Attribute should describe their functionality in a single line comment in front of their declaration stating their objective.

5.0 Standards for Parameters (Arguments) to Methods

5.1 Naming Parameters

Parameters should be named following the rules for attributes

6.0 Standards for File Organization

Each Java source file contains a single public class or interface. Additionally, private classes and interfaces that are associated to a public class or interface can be part of the same source file. The public class or interface should be the first definition in the file.

Java source files should have the following order:

- Beginning comments

- Package and import statements
- Class and interface declarations

6.1 Beginning Comments

All source files should begin with a javadoc comment that lists the class name, version information, date, and copyright notice.

```
/**
*  Catalog.java        1.0  8 Jan 2008
*
*  Copyright © 2008 MST, FCAS, NSU
*  3301 College Avenue, Fort Lauderdale-Davie, Florida 33314-7796
*  All Rights Reserved.
*
*  This software is the confidential and proprietary information
*  of MST, FCAS, NSU. ("Confidential Information").  You shall not
*  disclose such Confidential Information.
*/
```

6.2 Package and Import Statements

The package keyword should be the first non-comment line in your Java file. Following that should be all import statements.

6.3 Class/Interface Declarations

The next sub-sections list the parts of class or interface declaration. These are listed in the order they should appear in the Java file.

6.3.1 Class/Interface Documentation Comments

In JavaDoc format, describe the purpose of the class/interface. List the author and version (as JavaDoc parameters). Document the development/maintenance history of the class.

```
/**
*  Course represents a collection of subject that is
*  taught repeatedly in different semesters.
*
*  @version       1.0      8 Jan 2008
*  @author        S Rajput
*
*  8  Aug 2008     Rajput     Initial Development of Class.
```

```
*   15 Aug 2008    Cobo  Rewrite Constructor
*
*/
```

6.3.2 Class/Interface Statement

```
public class Catalog extends DatabaseRecord implements ICourse{
```

6.3.3 Class/Interface Implementation Comment

This JavaDoc comment should contain any class-wide or interface-wide information that wasn't appropriate for the class/interface documentation comment.

6.3.4 Class (static) Variables

First list the public class variables, then the protected, then the package level, and finally private variables.

6.3.5 Instance Variables

First list the public instance variables, then the protected, then the package level, and finally private variables. Public instance variables should be avoided whenever possible. Instance variable names should be descriptive of the functionality they provide and contain a single line comment in front of their declaration stating their objective.

6.3.6 Constructors

6.3.7 Public Methods

Include a Javadoc comment just prior to method declaration. Include in it a description of the method, what the method must be passed as parameters, what the method returns, and any exceptions thrown.

```
/**
*   getCourseName() returns the name of the specified course.
*
*   @param     courseName   The name of the specified course.
*
*   @return    A String representing a Catalog name.
*
*   @throws    java.rmi.RemoteException
*
*/
```

6.3.8 Protected Methods

6.3.9 Private Methods

7.0 Code Example

```
/**
*   Catalog.java        1.0  8 Jan 2008
*
*   Copyright © 2008 MST, FCAS, NSU
*   3301 College Avenue, Fort Lauderdale-Davie, Florida 33314-7796
*   All Rights Reserved.
*
*   This software is the confidential and proprietary information
*   of MST, FCAS, NSU. (“Confidential Information”).  You shall not
*   disclose such Confidential Information.
*/

package edu.fau.gradingsystem;

import edu.fau.gradingsystem.db.*;
import java.util.Date;
import java.sql.*;
import java.util.Vector;
import java.rmi.*;
import java.rmi.server.*;

/**
*   Represents a Catalog of Items for sale by a Seller (Entity)
*
*   @version        1.1       15 Jan 2008
*   @author         S Rajput
*
*   18  Jan 2008    Rajput   Initial Development of Class.
*   15 Jan 2008   Cobo Rewrite Constructor/New business rule
*
*/

public class Course extends DatabaseRecord implements ICourse{

/**
*   A class implementation comment can go here
*/

    public static final int ACTIVE = 1;     // Class Variable Comment
    public static final int INACTIVE = 0;
    public static final int PENDING = 2;

    private int catalogNumber;   // Instance Variable Comment
    private int sellerNumber;
    private int status;
    private Entity seller;
    private String name;
    private Date lastUpdate;
```

```
/**
*   A constructor comment can go here
*/

   public Course(DBInfo dbinfo) throws RemoteException {
          super(dbinfo,"Courses");
   }

/**
*   The main() methods MUST be here
**/
   public static main() throws SomeException {
          //At laest call Test Function.
   }
/**
*   At Least one test() method MUST be here.
*   Document what is success and failure conditions of this test
**/
   public static test() throws SomeException {
          // At least call all public Functions,in this or other
    // test functions
   }


   // Accessors
/**
*   getCourseNumber() returns the number of the current course
*   @return   A integer representing the catalog number
*   @throws   java.rmi.RemoteException
*/
   public int getCourseNumber() throws RemoteException {
          return courseNumber;
   }

/**
*   getStudentNumber() returns the number of the current student
*   @return   A integer representing the seller number
*   @throws   java.rmi.RemoteException
*/
   public int getStudentNumber() throws RemoteException {
          return sellerNumber;
   }

   //Mutators
/**
*   setStudentNumber() changes the seller number
*   @param    studentNumber   A integer representing the seller
*                                                    number
*   @throws   java.rmi.RemoteException
*/
   public void setStudentNumber(int studentNumber)  throws
            RemoteException{
          this.studentNumber = studentNumber;
```

```
            setChanged();
    }

    ...
    ...
)
```