Brief Notes on Java Documentation Guidelines for ENSF 409

By: M. Moussavi, PhD, P.Eng

ENSF 409 students are expected to use the following guidelines to document their Java code in ENSF 409. Java provides a tool called Javadoc that requires a specific format to comment and to document elements of a program such as classes, data fields, or methods

Class Comments:

Every class in your program should have brief description that should be commented right before the definition of the class confined `/** … */`.

Initial Description - Should be one sentence that describes the class clearly and concisely. Note that the initial description ends with a period followed by a whitespace (space, tab, or newline). The initial description is required. This discretion will appear in the description of the class and in the Summary Table of classes under the package documentation.

Follow Up Description(s) - After the initial description (above), you can add any additional description(s) in support of the first sentence. This description(s) will show up in the description for the class but not in the summary table.

Possible Class Tags – For the purpose of ENSF 409 lab assignments you should use the following tags in your class comments to address the author(s) name, the version of the code, and the date of creation:

@author
@version
@since

For more details on each tag, please see the given table in the following sections.

Data Field Comments:

For each data field add a very short comment. Please see the given example at the end of this document. A useful Javadoc tag that can be used in this type of comments is:

{@value}

Method Comments:

For each method in each class the following set of comments are required:

Initial Description: one sentence that starts with a verb and briefly but precisely describes the function of the method. This discretion will appear in the description of the method and in the summary table of methods.

Follow Up Description(s) - After the initial description, you can add any additional description in support of the first sentence. In other words any additional information that is useful to the user of the method

should be included here. Adding examples can be optional if it helps to describe the function of the method. This description(s) will show up in the description for the method but not in the summary table. Normally, this additional explanation is useful for constructors or implementer method and infrequently for getters and setters.

Useful Tags for Methods – The following tags are a set of useful tags for method. Use this tags in the same order that appears below:

@param
@return
@throw
@link.

In addition to above mentioned javadoc tags, the @override is also recommended for the cases that a method from a parent class is overridden. There are two benefits to it: First, you can take advantage of the compiler checking -- you will be warned if your method does not actually override as you think it does. Second, it helps the reader to know that the super class already has defined this method, or the definition of the method is required, because the class is implementing an interface.

See the given example at the end of this document and notice that @override tag is not confined between /** … */.

How to Generate a Java Document Using javadoc Command:

To create a java document from and IDE like Eclipse, under the 'File' pull-down menu select 'Export' and follow the necessary steps.

Or from command line simply use the following command:

```
Command line >>  javadoc –private yourProgram.java
```

List of some of the commonly tags for documenting Java program in ENSF 409 lab assignments:

Tag & Parameter	Usage	Applies to
@author *John Smith*	Describes an author.	Class, Interface, Enum
@version *version*	Provides software version entry. Max one per Class or Interface.	Class, Interface, Enum
@since *since-text*	Describes when this functionality has first existed.	Class, Interface, Enum, Field, Method
@see *reference*	Provides a link to other element of documentation.	Class, Interface, Enum, Field, Method
@param *name description*	Describes a method parameter.	Method
@return *description*	Describes the return value.	Method
@throws *classname description*	Describes an exception that may be thrown from this method.	Method
{**@link** *reference*}	Link to other symbol.	Class, Interface, Enum, Field, Method
{**@value** *#STATIC_FIELD*}	Return the value of a static field.	Static Field

Example:

```
import java.io.IOException;
/**
 * Provides data fields and methods to create a Java data-type, representing a
 * person in a Java application.
 * The overall purpose of this class exercise is to illustrate an example
 * of java documentation style that ENSF 409 students are expected to use in
 * their lab assignments.
 *
 * @author M. Moussavi
 * @version 1.0
 * @since January 16, 2015
 */
public class Person implements Cloneable{
	/**
	 * The first name and the last name of a Person
	 */
	protected String name;

	/**
	 * The date of birth of a Person
	 */
	protected String date;

	/**
	 * The id number of a person. The values for this data member requires to
	 * greater than or equal zero
	 */
	protected int id;

	/**
	 * A unique class id for class Person: {@value}.
	 */
	protected final static int ClassID = 10002;

	/**
	 * Constructs a Person object with the specified values for name, date and
	 * id. The values of the data fields are supplied by the given parameters.
	 * @param name the Person object's name
	 * @param date the Person object's date of birth
	 * @param id the Person object's id id number
	 */
	public Person(String name, String date, int id){
		this.name = name;
		this.date = date;
		this.id = id;
	}

	/**
	 * Constructs a Person object  with the default values.
	 * This constructor uses the other constructor and the default values are:
	 * - one blank space (means " ") for name
	 * - ##/##/#### for date
	 * - zero for id
	 */
	public Person(){
	this(" ", "##/##/####", 0);

	}
```

```
	/**
	 * Sets the value of id to the specified value.
	 * @param value is the new id for person. Needs to be greater than zero.
	 * @throws Exception
	 */
	void setId(int value) throws Exception{
		if(value < 0)
			throw new Exception("Error...");
		id = value;
	}

	/**
	 * {@inheritDoc}
	 */
	@Override
	public String toString(){
		String s = "A Person Information: ";
		// More Code
		return s;
	}

	/**
	 * Compares person's information with the specified supplied values.
	 * @param name the Person object's name
	 * @param date the Person object's date of birth
	 * @param id the Person object's  id number
	 * @return true if the Person name, date, and id matches the supplied values.
	 *         Otherwise returns false.
	 */
	boolean isTheSame(String name, String date, int id){
		if(name == this.name && this.date == date && this.id == id )
			return true;
		return false;
	}
	/**
	 * @throws IOException
	*/
	public static void fun() throws IOException{
		// TODO Auto-generated method stub

	}

	/**
	 * {@inheritDoc}
	 */
	@Override
	public void finalize()
	{

	}

	public static void main(String [] args){
		// TODO
	}
}
```