SOEN6441: Advanced Programming Practices

Amin Ranj Bar

API documentation generation tools

Javadoc

Concordia University
Engineering and Computer Science

API DOCUMENTATION GENERATION TOOLS

API documentation generation tools

- Historically, manual documentation generation was used to write API documentation to help developers to understand how to use libraries or modules.
- Good API documentation is necessary for libraries to be widely accepted and used correctly and efficiently.
- Manual documentation has many disadvantages:
  - Very time-consuming to write.
  - Error-prone.
  - Requires dedication and time to update.
- Outdated or wrong API documentation may be worse than having none.

API documentation generation tools

- The goal of having API documentation is to make the software more understandable, decreasing the amount of time the programmers spend in learning how to use libraries/modules/classes.
- To be really useful and economically viable, the time to write/maintain API documentation must be less than the time it allows to save by its use.
- API documentation became much more useful with the advent of hypertext and automation tools.
  - Hypertext enables very efficient browsing through huge documentation.
  - Automated tools can be used to extract API documentation from code.
  - Lowers the cost of writing/maintaining API documentation though automation.
  - Many such tools now exist, e.g. Javadoc and Doxygen.
  - All of them can generate hypertext documents.

JAVADOC

What is Javadoc?

- JavaDoc is a software tool part of Java SDK for generating API documentation from Java source code augmented with special tags in the code's comments.
- Javadoc is an industry standard for documenting Java classes.

- How does JavaDoc work?
  - Instead of writing and maintaining separate documentation, the programmer writes specially-formatted comments in the Java code itself.
  - The JavaDoc tool is a compiler that reads these comments and generates an API documentation out of them**.**
  - It also gets information from the code itself, then merges both of these information sources together to create a structured, hyperlink-browsable document.

Other API documentation generation tools

- Many such systems exist that can be used for various programming languages:
  - Javadoc, Doxygen, …
- Many of these can output in different formats:
  - HTML, RTF, PDF, LaTeX, manpages, …
  - HTML has many advantages: portable, browsable, adaptable
- Doxygen is probably the most flexible of them all, as it can generate documentation for various programming languages and generate output in various formats.
- Most IDEs integrate some features to call API documentation tools.

Advantages and drawbacks

- Advantages:
  - Program documentation process is coupled with the programming process.
  - Automated generation of documentation: less error-prone.
  - Efficient generation of documentation.
  - Efficient update of documentation.
  - Can generate highly browsable documentation, accessible electronically over the web (HTML).

Advantages and drawbacks

- Disadvantages:
  - Learning curve to learn how to use the tool, though it is minimal.
  - Requires dedication, or else the documentation will be obsolete or incomplete.

Example

```
/**
 * This is the JINI Transport Agent implementation.
 * It is implemented as a JINI service.
 *
 * The initial tasks performed by the class are:
 * <ul>
 * <li>Sets a security manager.
 * <li>Runs a listener for discovering the Lookup Service.
 * <li>When LUS is discovered, registers with it - publishes the Proxy.
 * <li>Connects with the Demand Dispatcher
 * </ul>
 *
 * @author Your Name
 * @since 1.0.0
 */
public class JINITransportAgent implements Runnable
```

Javadoc comments

- A JavaDoc comment begins with the `/**` marker and ends with the `*/` marker. All the lines in the middle start with an asterisk lined up under the first asterisk in the first line.

```
/**
 * This is a <b>javadoc</b> comment.
 */
```

- Because JavaDoc generates HTML files, any valid HTML can be embedded. A JavaDoc comment may be composed of multiple lines, for example:

```
/**
 * This is line one.
 * This is line two.
 *
 * This is intended as a new paragraph.
 */
```

Javadoc comments

- Another useful HTML marker is **<code>**, which we can use to include a sample code in a JavaDoc comment. Any text between the **<code>** and **</code>** markers will appear in a Courier font.

```
/**
 * <p>
 * The constructor calls the constructor of the super class Activatable.
 * <p>
 * The constructor spawns a new thread.
 * <p>
 * <code>
 *   super(id, 0); <br>
 *   new Thread(this).start();
 * </code>
 */
public JTABackend(ActivationID id, MarshalledObject data)
    throws RemoteException
{
    super(id, 0);
    new Thread(this).start();
}
```

Javadoc comments

- Generates browsable HTML, where every identifier is a clickable link that leads you to its own documentation.

Constructor Detail

JINITransportAgent.JTABackend

```
public JINITransportAgent.JTABackend(java.rmi.activation.ActivationID id,
                                     java.rmi.MarshalledObject data)
                              throws java.rmi.RemoteException
```

The constructor calls the constructor of the super class Activatable.

The constructor spawns a new thread.

```
super(id, 0);
new Thread(this).start();
```

Javadoc comments

- For the JavaDoc comments to be recognized as such by the javadoc tool, they must appear immediately before the class, interface, constructor, method, or data member declarations.
  - For example, if you put the JavaDoc comment for the class before the import statements, it will be ignored.
- The first sentence is a “summary sentence”. This should be a short description of the element described by the comment.

- **Note:**
  - JavaDoc does not provide a format for commenting elements within methods, i.e. the local variables and the computing going on inside the methods.
    - But you still can use the regular comments marks `//` or `/*..*/`, to comment this part of your program.

Javadoc tags

- There are a number of special tags we can embed with the JavaDoc comments. These tags start with the “at” symbol @.
- JavaDoc tags must start at the beginning of a line.

- Example:

```
/**
 * Inner class to listen for discovery events.
 *
 * @author Your Name
 * @since 1.0.0
 */
class Listener implements DiscoveryListener
```

- Information provided in tags such as **`@author`** , **`@version`** and **`@since`** pertain to versioning, which is maintained by a versioning system.
- Some say it should not be used, as it is superfluous if using a versioning system.

Javadoc tags

- **`@author`**
  - Used to create an author entry. You can have multiple **`@author`** tags. This tag is meaningful only for the class/interface JavaDoc comment.

- **`@see`**
  - Used to add a hyperlinked "See Also" entry to the class.

Javadoc tags

- **`@version`**
  - Used to create a version entry.
  - A JavaDoc comment may contain at most one **`@version`** tag.
  - Version normally refers to the version of the software (such as the JDK) that contains this feature.
  - If you are using CVS, you can also use the following to have any CVS commit to fill in the version tag with the CVS revision number: **`@version $Revision $`**

Javadoc tags

- Example:

```
/**
 * This class implements the backend interface. It is activatable.
 * This is the class who is used by RMI to assure service-side execution.
 * The compilation process (see comiple_jta.bat) generates stubs
 * from this class, which are transported to the client.
 * Internally these stubs comunicates with the service JTABackend object.
 *
 * @author Your Name
 * @version 1.0.0
 * @see JTABackendProtocol
 *
 */
public static class JTABackend extends Activatable
```

Javadoc tags

- Generated browsable HTML:

public static class **JINITransportAgent.JTABackend**
extends java.rmi.activation.Activatable
implements JINITransportAgent.JTABackendProtocol, java.lang.Runnable

This class implements the backend interface. It is activatable. This is the class who is used by RMI to assure service-side execution. The compilation process (see comiple_jta.bat) generates stubs from this class, which are transported to the client. Internally these stubs comunicates with the service JTABackend object.

Version:
1.0.0
Author:
Your Name
See Also:
`JTABackendProtocol`, Serialized Form

Javadoc tags

- **`@param`**
  - Used to add a parameter description for a method.
  - This tag contains two parts: the first is the name of the parameter and the second is the description.
  - The description can be more than one line.

  - `@param size the length of the passed array`

- **`@return`**
  - Used to add a return type description for a method.
  - This tag is meaningful only if the method's return is non-void.

  - `@return true if the array is empty; otherwise return false`

Javadoc tags

- **`@throws`**
  - Used to describe an exception that may be thrown from this method. Note that if you have a throws clause, Javadoc will already automatically document the exceptions listed in the throws clause.

- **`{@inheritDoc}`**
  - Used to copy the description from an overridden method.

- **`{@link reference}`**
  - Used to link to another documented symbol, or to a URL external to the documentation.

Javadoc tags

- Example

```
/**
 * This method prints out the IP address of the client and the command granted to it.
 * In addition, the method sends the demand back to the client.
 *
 * @param idResult The ID of the result to be fetched from the demand space.
 * @param sSenderIP The IP address of the sender.
 * @return The result corresponding to the ID.
 */
public DispatcherEntry fetchResult(Uuid idResult, String sSenderIP)
    throws RemoteException, DemandDispatcherException
```

Javadoc tags

- Generated browsable documentation:

fetchResult

```
public DispatcherEntry fetchResult(Uuid idResult,
                                   java.lang.String sSenderIP)
                            throws java.rmi.RemoteException,
                                   DemandDispatcherException
```

This method prints out the IP address of the client and the command granted to it. In addition, the method sends the demand back to the client.

Specified by:

`fetchResult` in interface `JINITransportAgent.JTABackendProtocol`

Parameters:

`idResult` - The ID of the result to be fetched from the demand space.

`sSenderIP` - The IP address of the sender.

Returns:

The result corresponding to the ID.

Throws:

`java.rmi.RemoteException`

`DemandDispatcherException`

Generating Javadoc documentation

- After adding the JavaDoc comments to the source files, use the `javadoc` command to generate the documentation.
- Run the `javadoc` as you run `javac` or other Java tools.
- After the `javadoc` command, provide the relevant parameters. See the JavaDoc documentation for details.
- Most Java IDEs include functionalities to call Javadoc to generate the API documentation.

Generating Javadoc documentation

- Example:
  - In order to enforce JavaDoc to generate documentation for the complete GIPSY package, we write:

    ```
    javadoc gipsy
    ```

  - In order to enforce JavaDoc to generate documentation for the **`JINITransportAgent.java`** file, to include the author and version tag and to include all the classes, attributes and methods we write:

    ```
    javadoc -private -version -author
    JINITransportAgent.java
    ```

  - In order to check for missing Javadoc, we write:

    ```
    javadoc -Xdoclint:missing JINITransportAgent.java
    ```

Summary

- Having an API documentation aims at improving the productivity of programmers by increasing the browseability, readability and understandability of code.
- The Javadoc code itself also provides documentation within the code.
- Manual documentation is extremely tedious and error-prone.
- Automated API documentation generation tools exist that automate the generation of API documentation.
- Results in more efficiency in writing/maintaining the API documentation, thus more overall productivity.
- Requires dedication and rigor.

In the project

- You are required to use Javadoc
  - To document every class
  - To document every method
    - Every parameter using `@param`
    - Every returned value using `@return`

- You are required to integrate the Javadoc compilation in the continuous integration pipeline.

References

- Oracle Corporation. Javadoc Tool.
- Oracle Corporation. How to Write Doc Comments for the Javadoc Tool.
- Wikipedia. Comparison of document generators.