

Documentation (Javadoc)

Aaron Karper

February 29, 2012

Contents

1 Why write JavaDoc

2 Guidelines

3 Examples

Less talk, more code?

- Inform other coders how to use your code without having to read it.
- Understanding your code
 - Know what the intention is.
 - Know what the code does and does not need to handle.
- Specification
 - Reminder to yourself what you need to do.
 - Makes you think about your responsibilities.
- Only necessary for public interface.

What is Good Documentation?

- Concise
 - Do *not* repeat what the code says, don't explain *how*.
 - No fillers – *This method...* is not necessary.
 - Make the first sentence count – Javadoc assumes it to be the summary.
 - Link to other documentation – with `@see` or `@link`
- Complete
 - Responsibilities (pre- & post-conditions)
 - Corner cases. e.g. `null`? negative ints?

Class Comments

- What is the class responsible for? What information does it hold, what things can it do?
- Who uses this class? Does it for example generate itself?
- Does this class need special treatment, for example a lifetime?

Method Comments

- Use `@param` to
 - Define constraints
 - What are your preconditions?
- Use `@return` to
 - Offer more specific information.
 - What are your postconditions?

What not to do

```
public class ServerProxy implements IServer{
```

What not to do

```
/**
 * Constructor.
 */
public ServerProxy(String url, int port) throws
    NetworkConnectionException {
    // ...
}
```


What not to do

```
/**
 * Ends the connection.
 */
public void disconnect() throws DeadConnectionException {
    // ...
}

/**
 * Returns the number of jobs.
 */
public int getJobCount() throws DeadConnectionException {
    // ...
}
```

What not to do

```
/**  
 * Returns the url of the server.  
 */  
public String getUrl() {  
    return url;  
}
```

How to do it better

```
public class ServerProxy implements IServer{
```

How to do it better

```
/**  
 * Relays method calls to a remote {@see Server}.  
 * <p>  
 * The proxy is responsible for establishing and  
 * keeping a connection to the server. The caller  
 * must ensure that a connection is destroyed with  
 * the {@see #disconnect} method.  
 */  
public class ServerProxy implements IServer {
```

How to do it better

```
/**
 * Constructor.
 */
public ServerProxy(String url, int port) throws
    NetworkConnectionException {
    // ...
}
```

How to do it better

```
/**
 * Established a connection to a remote server.
 * Throws if it fails to do so.
 *
 * @param url address that can either be resolved
 * via hosts.conf or DNS or is an IP
 * address.
 *
 * @param port port to connect to on the server. A
 * positive integer, typically above 1024.
 * Must be the same as the {@see Server}
 * uses with its {@see Server#listenOn} method.
 *
 * @throws NetworkConnectionException if it was
 * not able to initiate a connection.
 */
public ServerProxy(String url, int port)
    throws NetworkConnectionException {
    // ...
}
```

How to do it better

```
/**
 * Ends the connection.
 */
public void disconnect() throws DeadConnectionException {
    // ...
}

/**
 * Returns the number of jobs.
 */
public int getJobCount() throws DeadConnectionException {
    // ...
}
```

How to do it better

```
/**
 * Ends the connection. After this call, no other
 * method call is valid, including this one. The
 * server is not affected by this.
 */
public void disconnect() throws DeadConnectionException {
    // ...
}
```

```
/**
 * Returns the number of jobs running on the server.
 *
 * @return a non-negative integer that is the
 *         number of jobs that are alive.
 */
public int getJobCount() throws DeadConnectionException {
    // ...
}
```


How to do it better

```
/**  
 * Returns the url of the server.  
 */  
public String getUrl() {  
    return url;  
}
```

How to do it better

```
public String getUrl() {  
    return url;  
}
```