CSC 7437

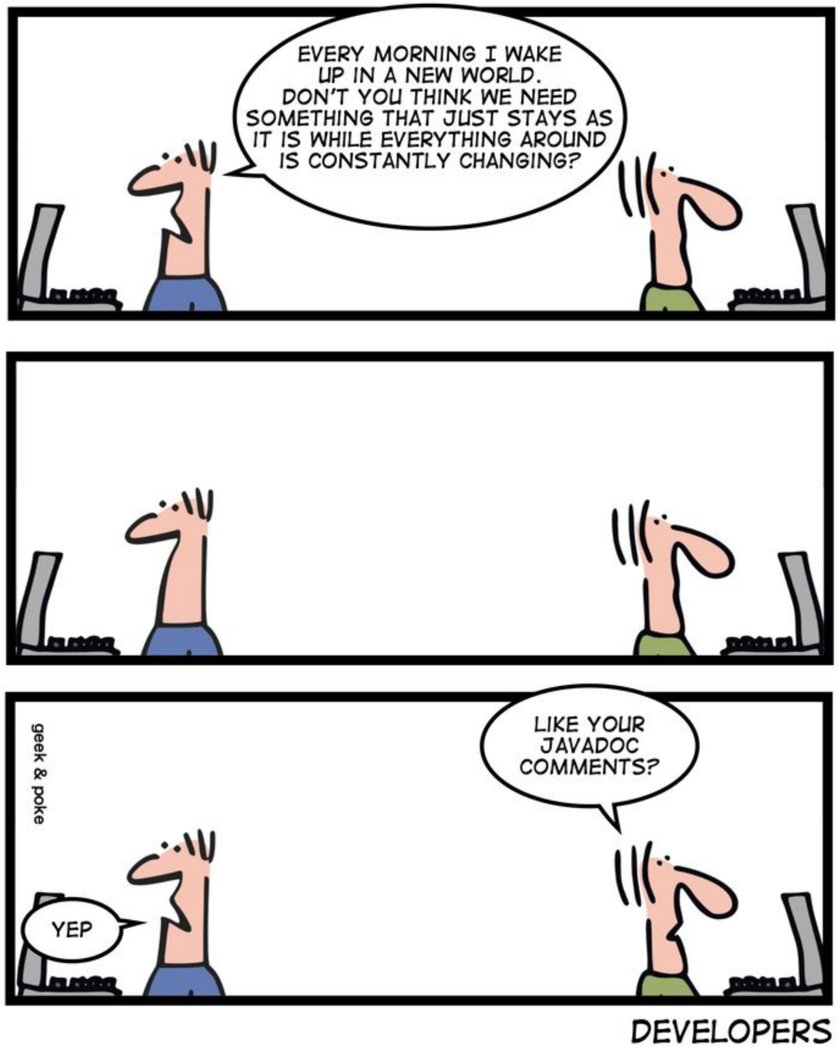
J Paul Gibson

paul.gibson@telecom-sudparis.eu

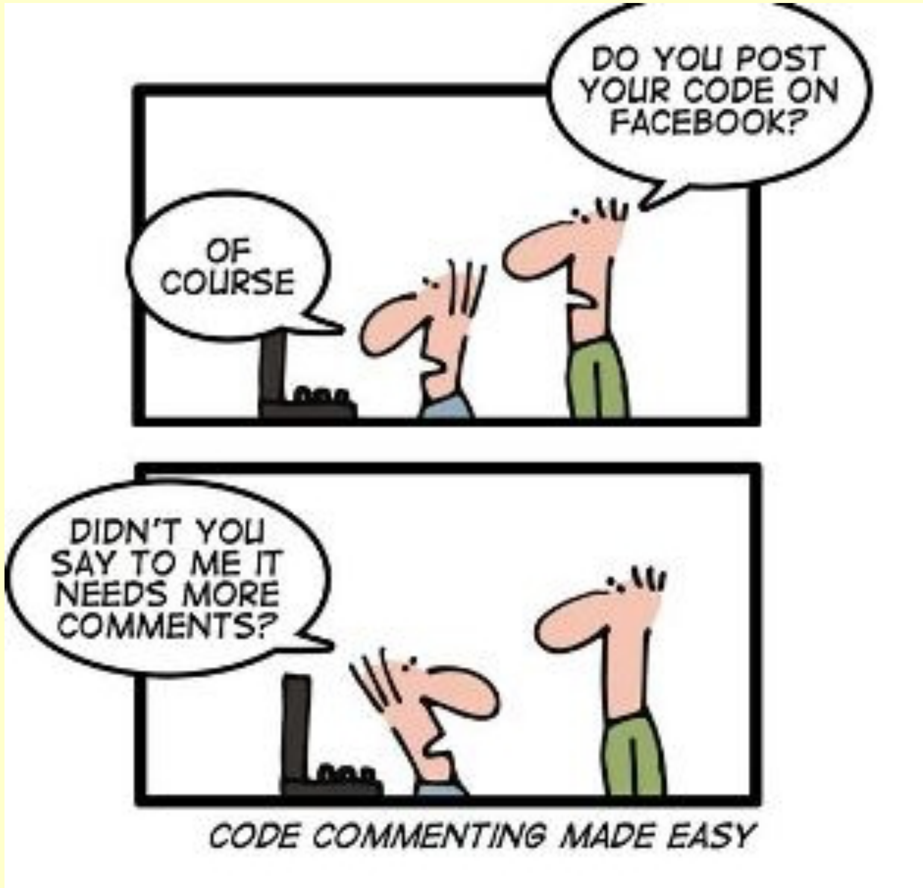
http://jpaulgibson.synology.me/~jpaulgibson/TSP/Teaching/CSC7437/

Code Documentation (with Javadoc)

Code Documentation (Comments) with Javadoc


EVERY MORNING I WAKE UP IN A NEW WORLD. DON'T YOU THINK WE NEED SOMETHING THAT JUST STAYS AS IT IS WHILE EVERYTHING AROUND IS CONSTANTLY CHANGING?
LIKE YOUR JAVADOC COMMENTS?
YEP
geek & poke
DEVELOPERS

https://www.datamation.com/trends/tech-comics-java-developers/


DO YOU POST YOUR CODE ON FACEBOOK?
OF COURSE
DIDN'T YOU SAY TO ME IT NEEDS MORE COMMENTS?
CODE COMMENTING MADE EASY

To Add Comments or Not to Add?

Andrey Akinshin

- Comments can spoil the code itself, make it less readable;
- Comments require time for writing and maintenance;
- Comments lie (starting from the improperly composed comments and ending with the obsolete ones)

THE PROGRAM I CODED HAS LOTS OF BUGS HOW DO I REMOVE THEM?
WHY DON'T YOU PUT ENTIRE CODE IN COMMENTS //
Brainstock.com

http://blogs.perpetuumsoft.com/dotnet/to-add-comments-or-not-to-add/

http://blogs.perpetuumsoft.com/dotnet/to-add-comments-or-not-to-add-part-2/

Types Of Documentation In Software Development

It is, traditionally, written text that complements software/models in order to clarify or explain:

- **Requirements**
- **Design**
- **Implementation**
- **Tests**
- **End User**
- **Marketing**

The implementation/code should be documented, and the most important aspect is that the documentation should clarify the consistency between the code and the other software/models in the system. It should highlight implementation decisions – *why not what*

Documenting Code: some reading material

•*Information Distribution Aspects of Design Methodology*, **David Lorge Parnas**, 1971

•*Literate Programming*, **Donald E. Knuth**, 1984

•*How To Comment Code***, Steve Drevok,** 1996

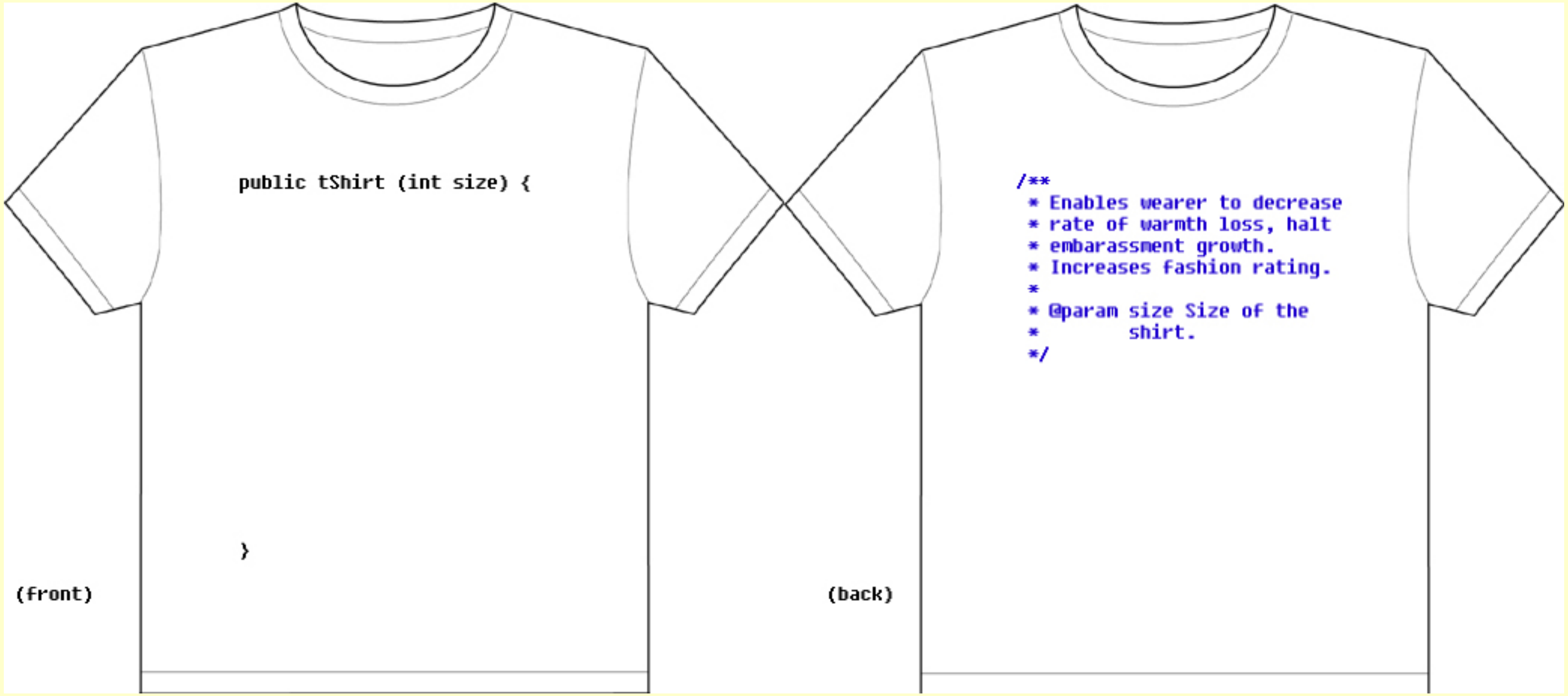
• *Comments Are More Important Than Code*, **Jef Raskin**, 2005

Clean code: a handbook of agile software craftsmanship, **Robert Martin**, 2008

•*Coding Guidelines: Finding the Art in the Science***, Robert Green and Henry Ledgard,** 2011

Documenting Code : 2 useful tools

Doxygen and JavaDoc: links on web site



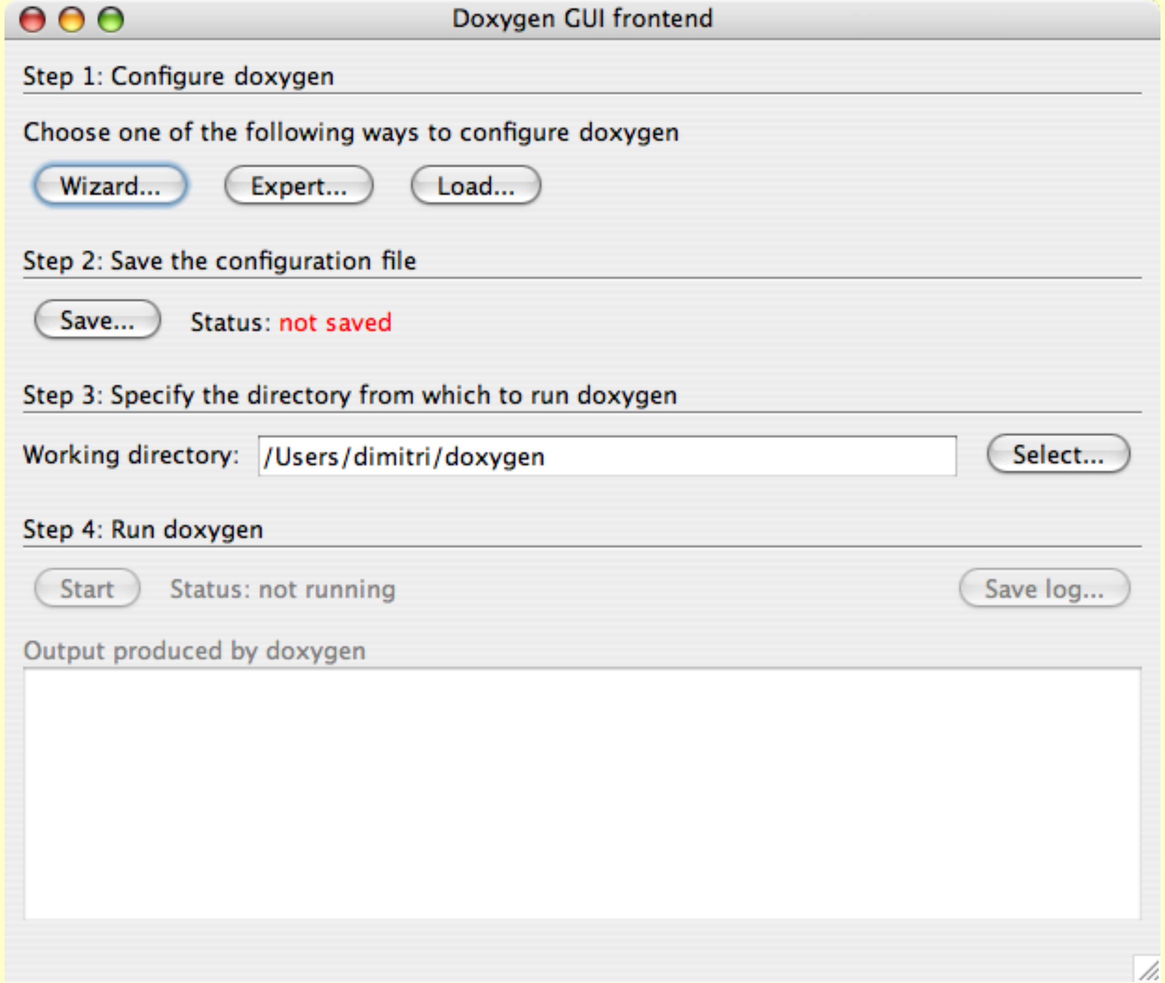
```
public tShirt (int size) {

}
```

```
/**
 * Enables wearer to decrease
 * rate of warmth loss, halt
 * embarassment growth.
 * Increases fashion rating.
 *
 * @param size Size of the
 *        shirt.
 */
```

Doxygen http://www.stack.nl/~dimitri/doxygen/

Doxywizard is a GUI front-end for configuring and running doxygen. When you start doxywizard it will display the main window (the actual look depends on the OS used).



Doxygen GUI frontend

Step 1: Configure doxygen

Choose one of the following ways to configure doxygen

Step 2: Save the configuration file

Status: not saved

Step 3: Specify the directory from which to run doxygen

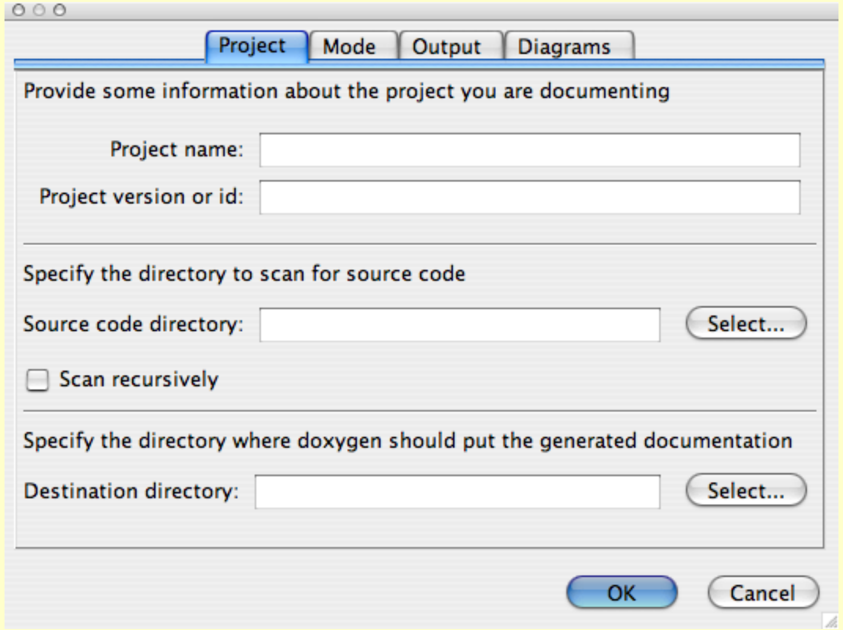
Working directory: /Users/dimitri/doxygen

Step 4: Run doxygen

Status: not running

Output produced by doxygen

Doxywizard



Provide some information about the project you are documenting

Project name:

Project version or id:

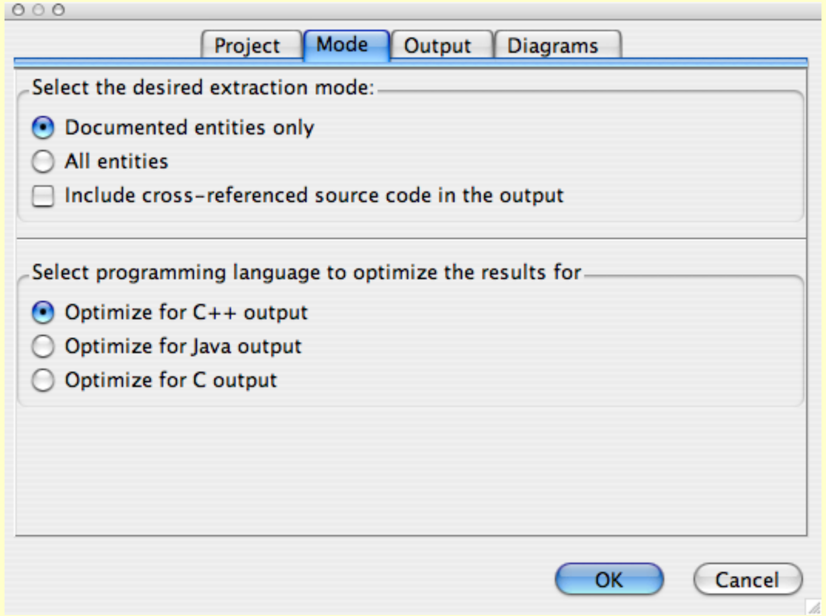
Specify the directory to scan for source code

Source code directory: Select...

□ Scan recursively

Specify the directory where doxygen should put the generated documentation

Destination directory: Select...

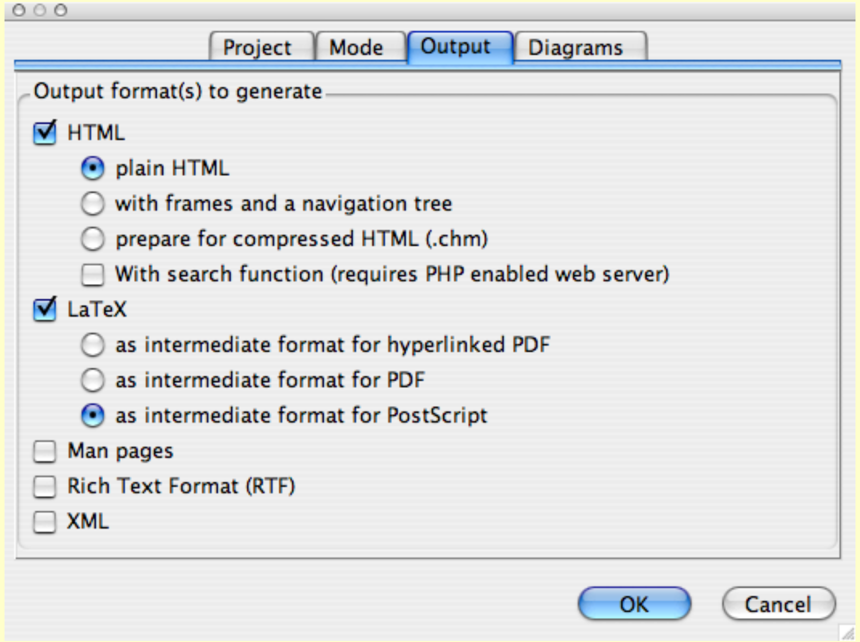


Select the desired extraction mode:

- ● Documented entities only
- ○ All entities
- □ Include cross-referenced source code in the output

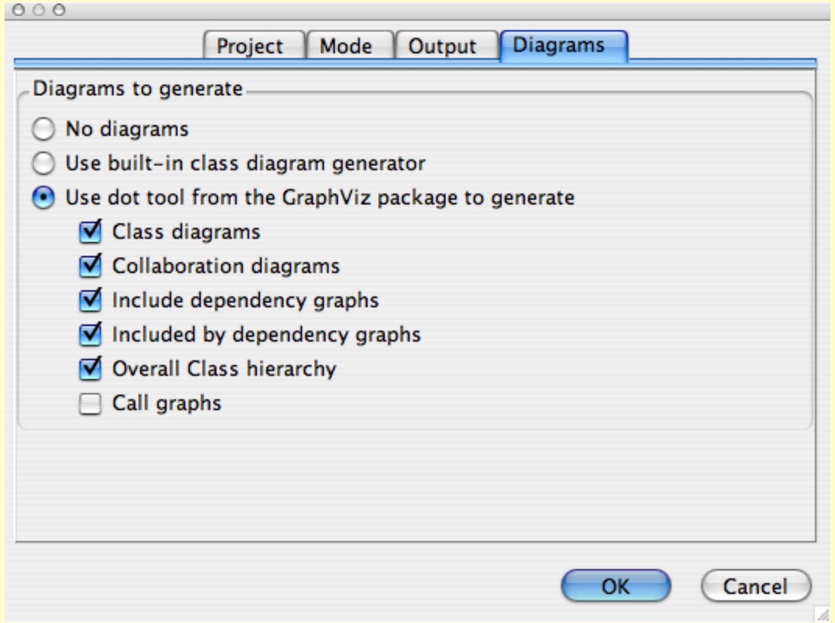
Select programming language to optimize the results for

- ● Optimize for C++ output
- ○ Optimize for Java output
- ○ Optimize for C output



Output format(s) to generate

- ☑ HTML
  - ● plain HTML
  - ○ with frames and a navigation tree
  - ○ prepare for compressed HTML (.chm)
  - □ With search function (requires PHP enabled web server)
- ☑ LaTeX
  - ○ as intermediate format for hyperlinked PDF
  - ○ as intermediate format for PDF
  - ● as intermediate format for PostScript
- □ Man pages
- □ Rich Text Format (RTF)
- □ XML

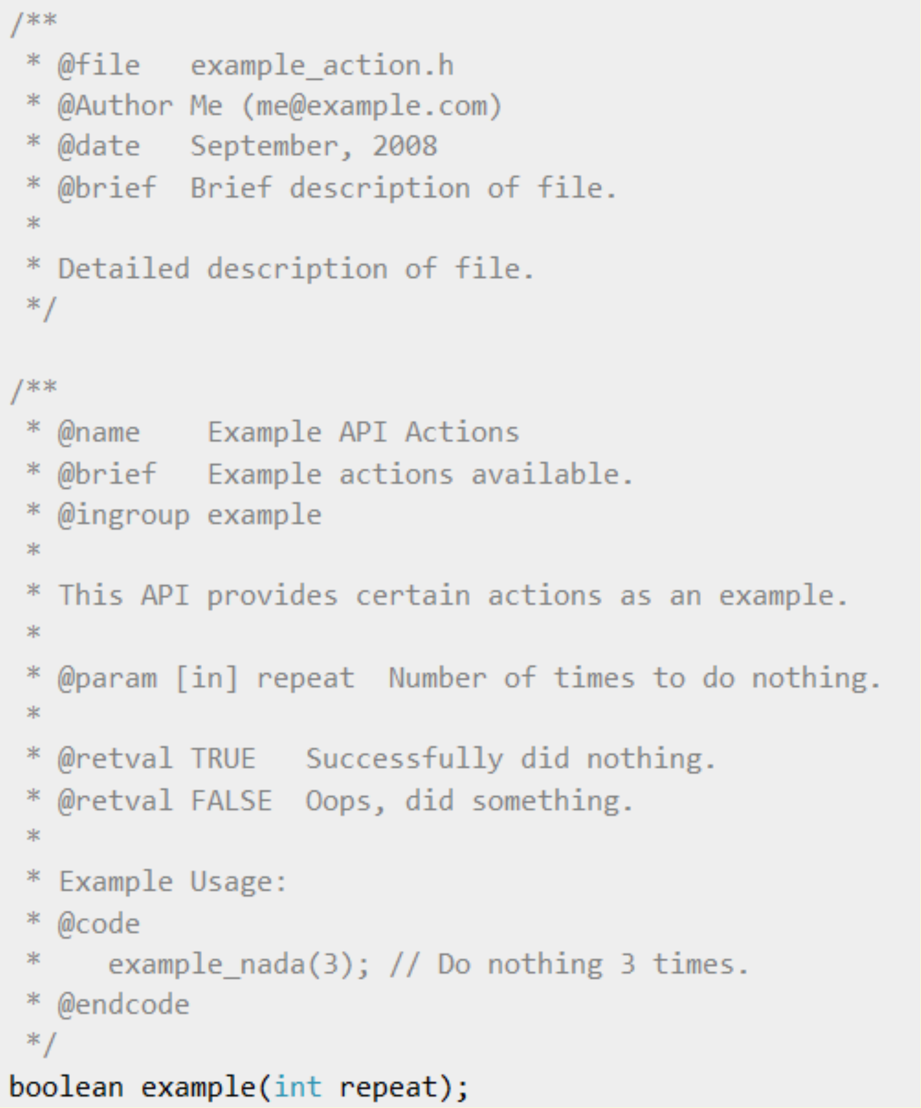


Diagrams to generate

- ○ No diagrams
- ○ Use built-in class diagram generator
- ● Use dot tool from the GraphViz package to generate
  - ☑ Class diagrams
  - ☑ Collaboration diagrams
  - ☑ Include dependency graphs
  - ☑ Included by dependency graphs
  - ☑ Overall Class hierarchy
  - □ Call graphs

Doxygen example C code

http://stackoverflow.com/questions/51667/best-tips-for-documenting-code-using-doxygen

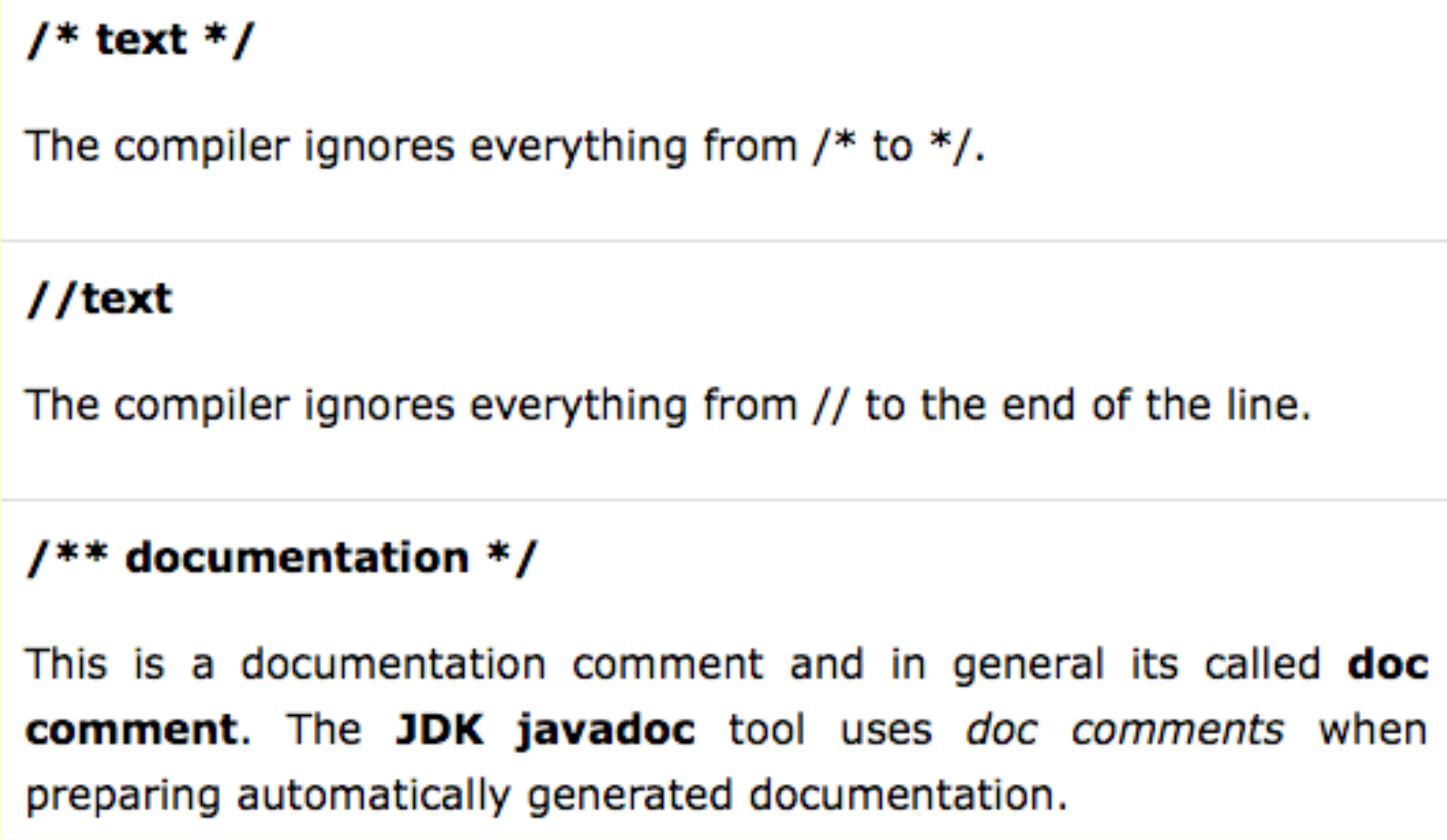


```
/**
 * @file   example_action.h
 * @Author Me (me@example.com)
 * @date   September, 2008
 * @brief  Brief description of file.
 *
 * Detailed description of file.
 */

/**
 * @name    Example API Actions
 * @brief   Example actions available.
 * @ingroup example
 *
 * This API provides certain actions as an example.
 *
 * @param [in] repeat  Number of times to do nothing.
 *
 * @retval TRUE   Successfully did nothing.
 * @retval FALSE  Oops, did something.
 *
 * Example Usage:
 * @code
 *    example_nada(3); // Do nothing 3 times.
 * @endcode
 */
boolean example(int repeat);
```

Full command list - http://www.stack.nl/~dimitri/doxygen/manual/commands.html

Java Comments



/* text */

The compiler ignores everything from /* to */.

//text

The compiler ignores everything from // to the end of the line.

/ documentation */**

This is a documentation comment and in general its called **doc comment**. The **JDK javadoc** tool uses *doc comments* when preparing automatically generated documentation.

Private Comments

Public Comments

What is Javadoc?

Javadoc (originally cased JavaDoc) is a documentation generator created by Sun Microsystems for the Java language (now owned by Oracle Corporation) for generating API documentation in HTML format from Java source code.

Javadoc comments are specific to the Java language and provide a means for a programmer to fully document his / her source code as well as providing a means to generate an Application Programmer Interface (API) for the code using the javadoc tool that is bundled with the JDK. These comments have a special format.

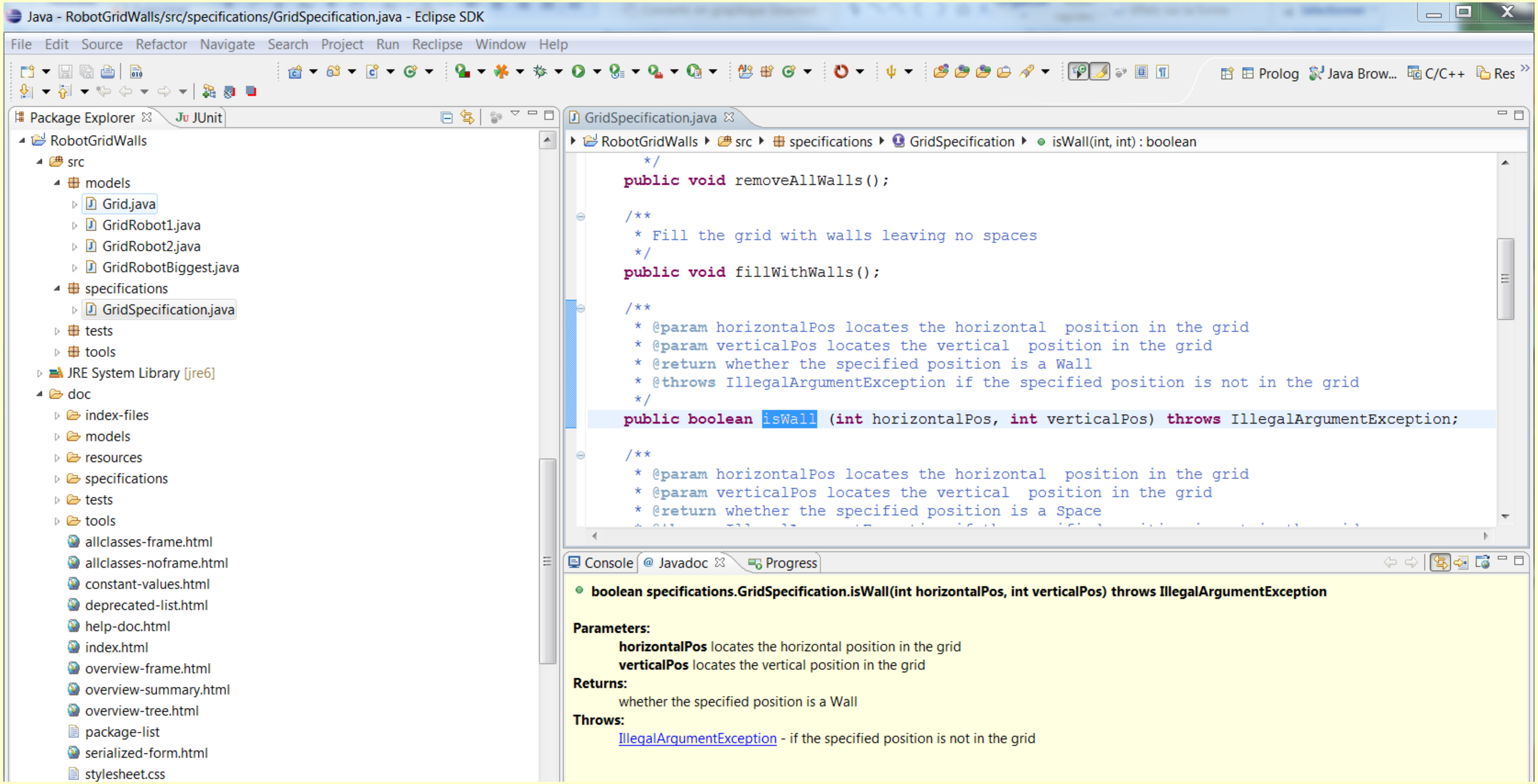
A Javadoc comment precedes any class, interface, method or field declaration and is similar to a multi-line comment except that it starts with a forward slash followed by two asterisks (/**). The basic format is a description followed by any number of predefined **tags**. The entire comment is indented to align with the source code directly beneath it and it may contain ***any valid HTML***.

Javadoc

Good on-line tutorial

http://www.cs.laurentian.ca/aaron/cosc1047/eclipse-tutorials/javadoc-tutorial.html

Example in Eclipse (Robot problem in Java)



```
     */
    public void removeAllWalls();

    /**
     * Fill the grid with walls leaving no spaces
     */
    public void fillWithWalls();

    /**
     * @param horizontalPos locates the horizontal  position in the grid
     * @param verticalPos locates the vertical  position in the grid
     * @return whether the specified position is a Wall
     * @throws IllegalArgumentException if the specified position is not in the grid
     */
    public boolean isWall (int horizontalPos, int verticalPos) throws IllegalArgumentException;

    /**
     * @param horizontalPos locates the horizontal  position in the grid
     * @param verticalPos locates the vertical  position in the grid
     * @return whether the specified position is a Space
```

boolean specifications.GridSpecification.isWall(int horizontalPos, int verticalPos) throws IllegalArgumentException

Parameters:
horizontalPos locates the horizontal position in the grid
verticalPos locates the vertical position in the grid
Returns:
whether the specified position is a Wall
Throws:
IllegalArgumentException - if the specified position is not in the grid

Tags come in two types:

- **Block tags** - Can be placed only in the tag section that follows the main description. Block tags are of the form: `@tag`.
- **Inline tags** - Can be placed anywhere in the main description or in the comments for block tags. Inline tags are denoted by curly braces: `{@tag}`.

General Order of **Tags**

http://javaworkshop.sourceforge.net/chapter4.html

The general order in which the **block tags** occur is as follows:

1. @author
2. @version
3. @param
4. @return
5. @throws
6. @see
7. @since
8. @deprecated

Useful inline tags

```
{@code text}
{@docRoot}
{@inheritDoc}
{@link package.class#member label}
{@linkplain package.class#member label}
```

Order of multiple repeated tags

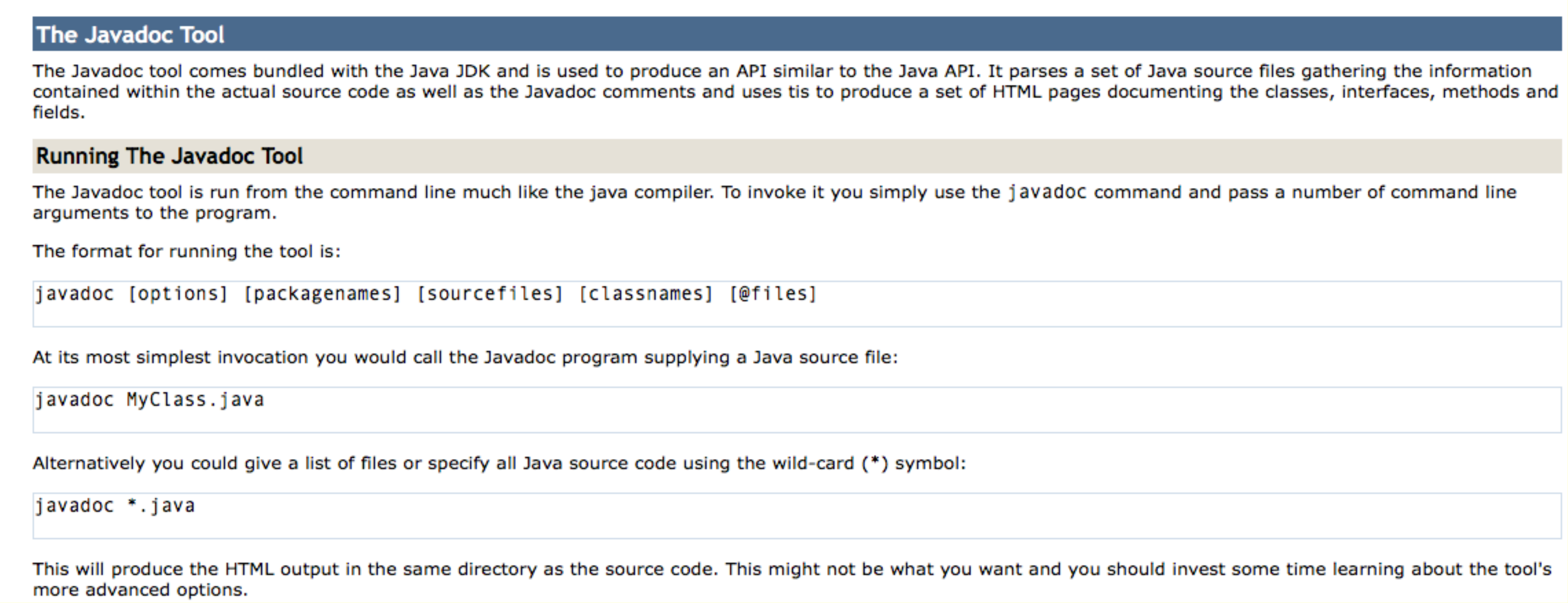
There are three block tags that may occur more than once, they are:

1. @author
2. @param
3. @throws

As mentioned above, the author tag should be listed in chronological order, with the creator of the class or interface listed first. This implies that the last person to work on the source code will have their name appended to the bottom of the list of author tags.

A method may have numerous parameters. In this case, the param tags should be defined in the exact same order as the parameters are declared in the method declaration.

A method may throw numerous exceptions, in such a case it is customary to list the exceptions in alphabetical order, although in some cases they may be listed according to severity, the most severe exception is listed first.



The Javadoc Tool

The Javadoc tool comes bundled with the Java JDK and is used to produce an API similar to the Java API. It parses a set of Java source files gathering the information contained within the actual source code as well as the Javadoc comments and uses tis to produce a set of HTML pages documenting the classes, interfaces, methods and fields.

Running The Javadoc Tool

The Javadoc tool is run from the command line much like the java compiler. To invoke it you simply use the `javadoc` command and pass a number of command line arguments to the program.

The format for running the tool is:

```
javadoc [options] [packagenames] [sourcefiles] [classnames] [@files]
```

At its most simplest invocation you would call the Javadoc program supplying a Java source file:

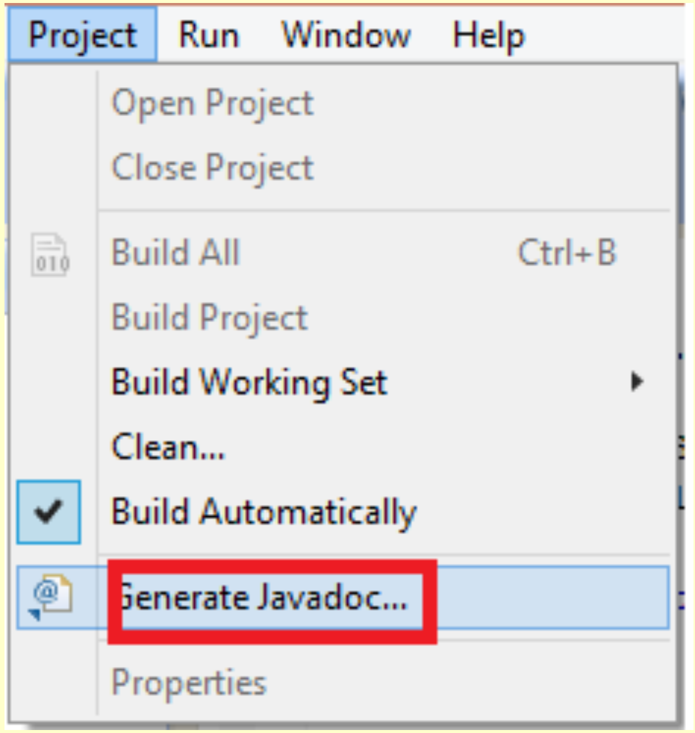
```
javadoc MyClass.java
```

Alternatively you could give a list of files or specify all Java source code using the wild-card (*) symbol:

```
javadoc *.java
```

This will produce the HTML output in the same directory as the source code. This might not be what you want and you should invest some time learning about the tool's more advanced options.

Eclipse includes Javadoc tool



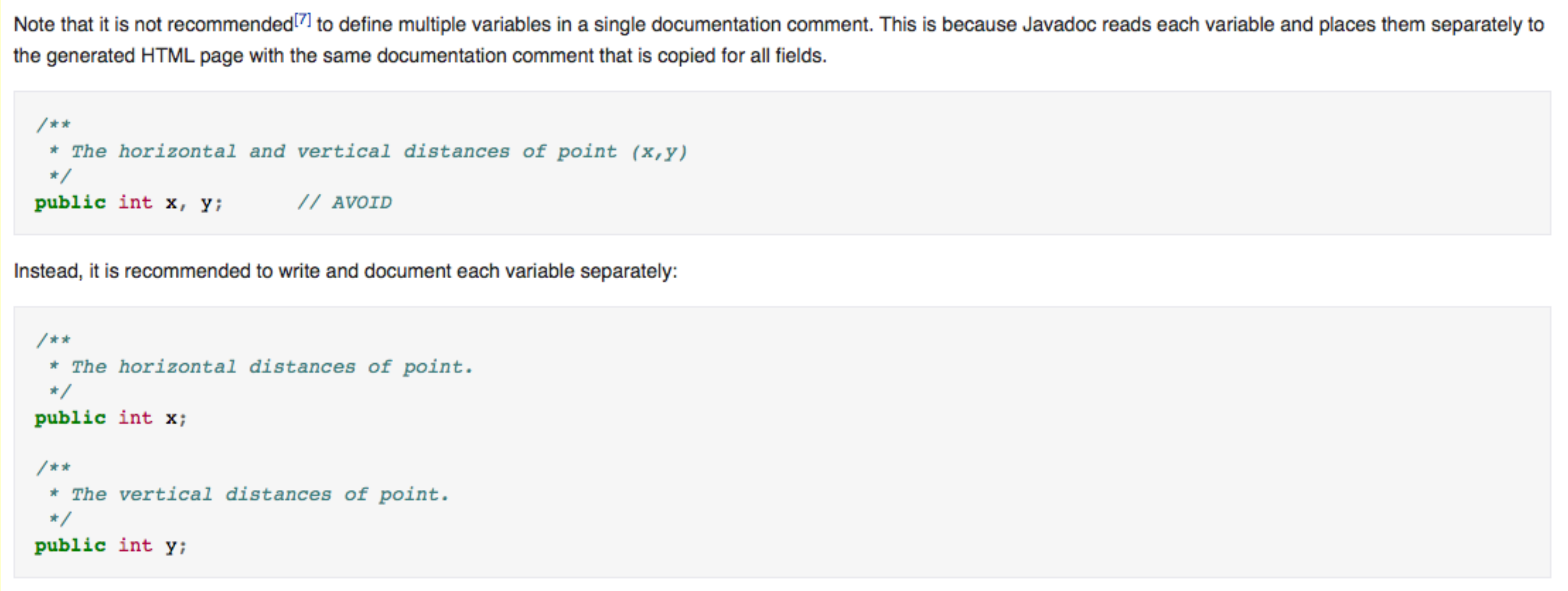
Can compile/generate documentation (html) for any Java project

Eclipse Javadoc tips and tricks

`Shift-Alt-J` is a useful keyboard shortcut in Eclipse for creating Javadoc comment templates.

At a place where you want javadoc, type in `/**`<NEWLINE> and it will create the template.

In Eclipse, see the Javadoc tab at the bottom of the screen to preview the Javadoc information included for the class you’re viewing. Hovering over code also pops up a preview window



Note that it is not recommended[7] to define multiple variables in a single documentation comment. This is because Javadoc reads each variable and places them separately to the generated HTML page with the same documentation comment that is copied for all fields.

```
/**
 * The horizontal and vertical distances of point (x,y)
 */
public int x, y;      // AVOID
```

Instead, it is recommended to write and document each variable separately:

```
/**
 * The horizontal distances of point.
 */
public int x;

/**
 * The vertical distances of point.
 */
public int y;
```

Documenting Code : Some Good Habits

- Cross reference test code with code being tested

- Link tests to requirements and design documentation

- Always keep documentation up to date as the rest of the system evolves

- Use version control on documentation

- Always use a documentation support tool/plugin

- Use 2 types of comments:
  1. External – explaining to users of the code how to (re)use it
  2. Internal – explain to maintainers of the code how it works and justify implementation decisions