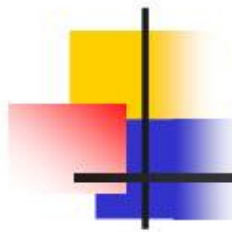


Software Documentation



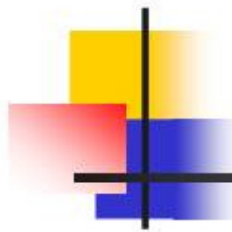
Source Code Documentation

- HTML Basics:

http://www.w3schools.com/html/html_primary.asp

- Ubiquitous problems

- Writing API documentation for a system is one of the most unpleasant jobs a developer will ever face
 - Application programming interface
 - The kind of job that could drive you to despair
- No documentation → no code
- “Informal” documentation isn’t standard
- As software evolves, “informal” documentation and code become out of sync!
 - Eventually, documentation becomes unusable making code hard to understand and update!



JavaDoc

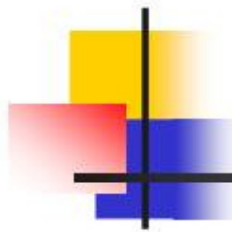
- `javadoc` utility makes writing and maintaining code documentation up-to-date easier!
 - Ships with JDK
- Defines a set of specially formatted comments
 - Can be added to document each
 - package,
 - class (& interfaces),
 - method, and
 - field
- Used to generate HTML documentation of classes or packages after parsing the comments
 - HTML documentation of the API



JavaDoc comments

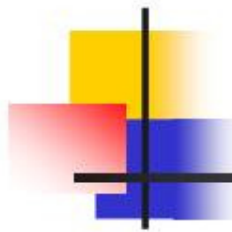
- IMMEDIATELY precede the feature it describes
- Consist of
 - Description of the feature
 - Copied as is to the documentation page
 - List of tags
 - Formatted by `javadoc` in a consistent style
 - Used in classes, interfaces, methods and variables
- Have the following format:

```
/**  
 * Summary sentence.  
 * More general information about the  
 * program, class, method or variable which  
 * follows the comment, using as many lines  
 * as necessary.  
 * <SPACE>  
 * zero or more tags to specify more specific kinds  
 * of information, such as parameters and return  
 * values for a method  
 */
```



JavaDoc comments

- Must be provided for every **public class** or interface
- Must be provided for each **public method** or **constructor**



JavaDoc Tags

■ **@author** *name*

- Name one of the authors of this class
- Use multiple tags if there are multiple authors
- Used in: *Class, Interface, Method*
- E.g. **@author Jane Smith, lab X**

■ **@version** *release*

- Indicate the version of the software containing this class
- Used in: *Class, Interface*



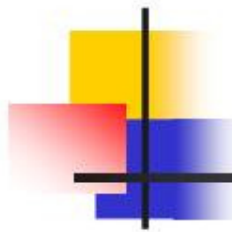
JavaDoc Tags

■ `@see target`

- Point the reader at something else relevant to read, like another class or a specific method
- Inserts a link pointing to the target
- Used in: *A//*

■ `@deprecated text`

- Marks the entity as deprecated and points the reader to what they should use instead via an `@see` tag
- Used in: *A//*



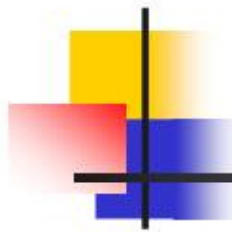
@see tag

- Points to another package, class, method, field or even URL
 - javadoc creates a link
- The syntax for the text after @see depends upon what you're pointing to:
 - @see *package* → Link to *package*
 - @see *classname* → Link to *classname* in the current package
 - @see *package.classname* → Link to *classname* in another package
 - @see #*method* → Link to *method* in the current class
 - @see #*method*(*type*) → Link to *method* with argument
 - @see *classname*#*method* → Link to *method* in another class
 - **Method in a class in a different package?**
 - @see text → Link to *URL*



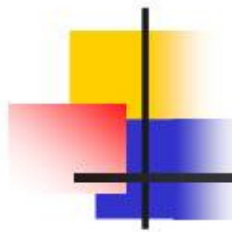
Examples

- `//same class entities`
- `@see #field`
- `@see#Constructor(Type,...)`
- `@see #method(Type,Type,...)`
- `//same package entities`
- `@see Class`
- `@see Class#field`
- `@seeClass#Constructor(Type,...)`
- `@see Class#method(Type,...)`
- `//different package entities`
- `@see package`
- `@see package.Class`
- `@see package.Class#field`
- `@see package.Class#Constructor(Type,...)`
- `@see package.Class#method(Type,Type,...)`
- `//URL`
- `@see <a href="spec.html#section">Java Spec</a>`



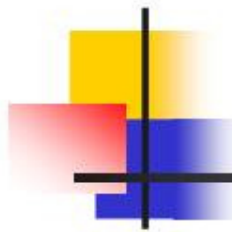
JavaDoc Tags

- **@param** *<name of parameter> short description of parameter*
 - Describe the named parameter to this method
 - Skip this tag if the method has no parameters
 - Used in: *Method*
 - E.g. @param **size** number of elements in the array
- **@return** *text*
 - Describe the value returned by this method
 - Skip this tag if the method has no return value
 - **Appears after** @param tag(s)
 - Used in: *Method*



Example

```
■ /**  
■  * Validates a chess move.  
■  *  
■  * @author John Doofus  
■  * @param theFromLoc Location of piece being moved  
■  * @param theRank Rank of piece being moved  
■  * @param theToLoc Location of destination square  
■  * @return true if a valid chess move or false if invalid  
■  */  
■ boolean isValidMove(int theFromLoc, int theRank, int  
   theToLoc) { ... }
```



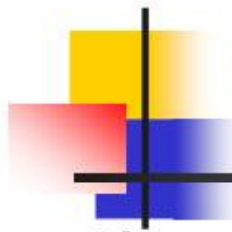
JavaDoc Tags

- *@throws <name of exception> description of circumstances under which exception is thrown*
 - Describes the named uncaught “checked” or explicitly thrown “checked”/“unchecked” exception
 - Skip this tag if the method throws no exceptions
 - Should follow @param and @return tags
 - If method throws more than one exception
 - they should appear in alphabetical order by exception name
 - *Used in*
 - *Method*
 - @throws FileNotFoundException raised if the user does not specify a valid file name



Checked vs. Unchecked

- You only advertise (via `throws` on method header) (or catch) and include `@throws` for
 - (P.S. `throws` on method header → exception is not handled in the method but forwarded to the invoker)
 - checked exceptions
 - explicitly thrown unchecked ones
- Unchecked exceptions :
 - beyond your control (`Error`) or result from a condition that you should not have allowed in the first place (`RuntimeException`)
 - are subclasses of
 - `RuntimeException` (e.g. `ArrayIndexOutOfBoundsException`)
 - `Error` (e.g. `OutOfMemoryError`)
 - The Java compiler checks all exception handling, except exceptions represented by (subclasses of) `Error` and `RuntimeException`



Checked vs. Unchecked

■ Checked exceptions:

- represent invalid conditions in areas *outside the immediate control of the program*
 - E.g. database problems, network outages, or absent files
- are subclasses of `Exception` (except `RuntimeException`)
- the compiler will confirm at compile time that the method includes code that might **throw** an exception
- must be caught or forwarded (advertised)
 - This can be done in a `try ... catch` statement or by defining the exception in the method definition (via `throws`)



Class Hierarchy

- [<http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Throwable.html>]
- `java.lang.Object`

+--`java.lang.Throwable`

+--`java.lang.EXCEPTION`

+--`java.lang.ClassNotFoundException`

+--`java.io.IOException`

+--`java.io.FileNotFoundException`

+--`java.lang.RUNTIMEEXCEPTION`

+--`java.lang.NullPointerException`

+--`java.lang.ArithmeticException`

+--`java.lang.IllegalArgumentException`

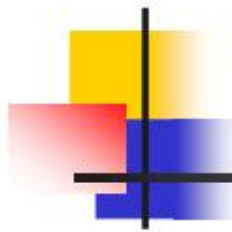
+--`java.lang.IndexOutOfBoundsException`

+--`java.lang.ArrayIndexOutOfBoundsException`

+--`java.lang.ERROR`

+--`java.lang.VirtualMachineError`

+--`java.lang.OutOfMemoryError`



Tag summary

- <http://java.sun.com/j2se/1.3/docs/tooldocs/win32/javadoc.html>
- <http://java.sun.com/j2se/javadoc/writingdoccomments/>



Minimum Requirements

- Class (or package)

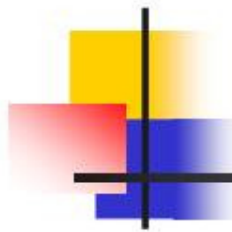
- `@author`
- `@version`
- `@see`
- `@deprecated`

- Field

- `@see`
- `@deprecated` (followed by another `@see` sometimes)

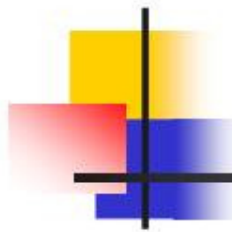
- Method/Constructor

- `<ordered>`
- `@param`
- `@return`
- `@throws`
- `</ordered>`
- `@see`
- `@deprecated` (followed by another `@see` sometimes)



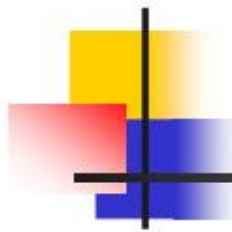
Running Javadoc

- `javadoc [options] [packages]`
`[filenames]`
- Can list one or packages
 - No wildcards (i.e. *)
- Can list one or more java files
 - Can use wildcards



javadoc Command-line Options

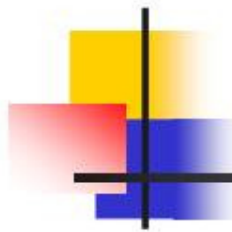
- `-public`
 - Only public classes, members and methods are documented
 - For API users
- `-protected`
 - Public and protected
 - Most common (default)
- `-package`
 - Public, protected and package
- `-private`
 - All
 - For internal use
- `-version`
 - Causes the `@version` tag to be included in the documentation
- `-d directory`
 - Location of output HTML documentation
 - Same directory as source by default
- `-author`
 - Causes the `@author` tag to be included in the documentation
- `-nodeprecated`
 - Deprecated methods and classes won't be documented



Result Documentation

■ Class Stack

- `javadoc *.java`
- `javadoc -version -author -private -
nodeprecated *.java`



JavaDoc using Eclipse

- Code formatter
 - Highlight code
 - `Source > Format`
- JavaDoc comment generator
 - Select the project or file in Package Explorer
 - `Project > Generate Javadoc ...`
 - Follow wizard