lecture #16 — wed nov 6, 2002

- news
  - homework #4 due friday
  - homework #5 posted later today (by midnight)
- today
  - software documentation
  - user documentation
- sources
  - http://java.sun.com/j2se/javadoc/writingdoccomments/index.html

software documentation.

- also called source code documentation
- is hard to write
- is harder to maintain
- must be up-to-date with the source code, otherwise it's useless
- think of it as a contract between the programmer and the subsequent programmers that will use the code
- two methodologies:
  - within source code
  - external to source code
- first way makes it easier to maintain and keep accurate

software documentation — bugs.

- two types:
  - specification bugs
  - code bugs
- differences between how code is specified and how it works
- only one is right (if any)

software documentation — javadoc.

- Java tool for generating the pretty API documentation you find on the java.sun.com web site
- easy to use
- good standard
- useful for conventions for other languages

software documentation — javadoc (2).

- handles four types of source code files:
  - class source code files (*.java)
  - package comment files
  - overview comment files
  - miscellaneous unprocessed files

software documentation — class source code files.

- documentation is in the form of source code comments
- comments contain javadoc tags and HTML tags
- javadoc tags are processed by javadoc
- HTML tags are processed when displayed by a browser
- must immediately precede a declaration of:
  - a class
  - a field
  - a constructor
  - a method
- consists of:
  - description (comes first)
  - block tags (come second)

software documentation — example.

```
/**
* Returns an Image object that can then be painted on the screen.
* The url argument must specify an absolute {@link URL}. The name
* argument is a specifier that is relative to the url argument.
* <p>
* This method always returns immediately, whether or not the
* image exists. When this applet attempts to draw the image on
* the screen, the data will be loaded. The graphics primitives
* that draw the image will incrementally paint on the screen.
*
* @param  url  an absolute URL giving the base location of the image
*         name the location of the image, relative to the url argument
* @return      the image at the specified URL
* @see         Image
*/
public Image getImage(URL url, String name) {
    try {
        return getImage(new URL(url, name));
    } catch (MalformedURLException e) {
        return null;
    }
}
```

software documentation — details.

- each line is indented to align with the code below the comment
- first line begins with `/**`
- last line ends with `*/`
- all intervening lines start with `*`
- first sentence is a short summary of the method
- `{@link URL}` converts to HTML and points to documentation for the `URL` class (this is an example of a javadoc tag)
- separate paragraphs with `<p>` HTML tag
- insert blank line between description and block tags
- description ends with first line that begins with `@`
- limit comment lines to 80 characters (or less)

software documentation — class source code files.

- documentation is in the form of source code comments
- comments contain javadoc tags and HTML tags
- javadoc tags are processed by javadoc
- HTML tags are processed when displayed by a browser
- class/interface tags:
  - `@see`
  - `@since`
  - `@deprecated`
  - `@serial`
  - `@author`
  - `@version`
  - `{@link}`
  - `{@linkplain}`
  - `{@docRoot}`

software documentation — class source code files (2).

- field tags:
  - `@see`
  - `@since`
  - `@deprecated`
  - `@serial`
  - `@serialField`
  - `{@link}`
  - `{@linkplain}`
  - `{@docRoot}`
  - `{@value}`

software documentation — class source code files (3).

- constructor and method tags:
  - `@see`
  - `@since`
  - `@deprecated`
  - `@param`
  - `@return`
  - `@throws` (same as `@exception`)
  - `@serialData`
  - `{@link}`
  - `{@linkplain}`
  - `{@inheritDoc}`
  - `{@docRoot}`

software documentation — package comment files.

- create a file named `package.html` and place it in the package directory
- contains one big documentation comment, **without the comment separators (`/** */` or leading `*`s)**
- **comment is in html**
- **make the first sentence a summary about the package**
- **do not put a title or any other text between the `<body>` tag and the first sentence**
- **package tags:**
  - `@see`
  - `@since`
  - `@deprecated`
  - `@serial`
  - `@author`
  - `@version`
  - `{@link}`
  - `{@linkplain}`
  - `{@docRoot}`

software documentation — overview comment files.

- contains overview documentation for an application or set of packages
- can be anywhere in source code tree and can have any name
- convention is to place it at top level and name it `overview.html`
- to process it, run

```
unix$ javadoc -overview <filename>
```

- make the first sentence a summary about the application or set of packages
- do not put a title or any other text between the `<body>` tag and the first sentence
- tags:
  - `@see`
  - `@since`
  - `@author`
  - `@version`
  - `{@link}`
  - `{@linkplain}`
  - `{@docRoot}`

software documentation — miscellaneous unprocessed files.

- contains hardcoded HTML documentation for files that don't fall into any of the other categories
- these files are *unprocessed* by javadoc
- but can be referenced by files that are
- place files in a directory called `doc-files`, anywhere in the source code tree
- you can have one doc-files directory per package

software documentation — tags.

- most frequently used tags (in Java API), in this order:
  - `@author` (classes and interfaces only)
  - `@version` (classes and interfaces only)
  - `@param` (methods and constructors only)
  - `@return` (methods only — except void)
  - `@throws` (same as `@exception`)
  - `@see`
  - `@since`
  - `@serial` (or `@serialField` or `@serialData`)
  - `@deprecated` (only if applicable)
- other tags:
  - `@link` and `@linkplain`
  - `@value`

software documentation — tag descriptions: @author.

- adds an “Author” entry with the specified name-text to the generated docs when the -author option is used
- syntax:

```
@author  name-text
```

- may contain multiple `@author` tags

software documentation — tag descriptions: @version.

- adds a “Version” subheading with the specified version-text to the generated docs when the -version option is used
- syntax:

```
@version  version-text
```

- a doc comment may contain at most one `@version` tag
- use RCS automated version identifier (`$Id$`) to fill in version-text

software documentation — tag descriptions: @param.

- adds a parameter to the “Parameters” section
- syntax:

```
@param  parameter-name description
```

- should describe the data type of the parameter
- should describe how the parameter is used within the method

software documentation — tag descriptions: @return.

- adds a “Returns” section with the description text
- syntax:

```
@return  description
```

- should specify the data type of the return value
- should describe the range of possible return values

software documentation — tag descriptions: @throws.

- adds a “Throws” subheading to the generated documentation, with the class-name and description text.
- syntax:

```
@throws  class-name  description
```

- class-name is the name of the exception that may be thrown by the method
- multiple `@throws` tags can be used
- documentation is copied from an overridden method to a subclass only when the exception is explicitly declared in the overridden method (or implemented)
- use `{@inheritDoc}` to force inheriting of documentation

software documentation — tag descriptions: @see.

- adds a “See Also” heading with a link or text entry that points to reference
- syntax:

```
@see reference
```

- three forms:

```
@see ‘‘string’’

@see <a href=‘‘URL#value’’>label</a>

@see package.class#member label
```

software documentation — tag descriptions: @since.

- adds a “Since” heading with the specified since-text to the generated documentation
- syntax:

```
@since  since-text
```

- indicates what was the current version of the class or package when this element was created

software documentation — tag descriptions: @link.

- inserts an in-line link with visible text label that points to the documentation for the specified package, class or member name of a referenced class
- syntax:

```
{@link  package.class#member  label}
```

- similar to `@see`, but generates an in-line link instead of placing a reference in the "See Also" section
- also —
- syntax:

```
{@linkplain  package.class#member  label}
```

- which is identical to `{@link}`, except the link's label is displayed in plain text than code font

software documentation — tag descriptions: @value.

- displays the value of the constant when used in a static field comment
- syntax:

```
{@value}
```

software documentation — tricks.

- sentences end with a period followed by a space
  - screws up things like “Prof. Sklar”
  - so do this: `Prof. Sklar`
- in Java, doc comments are inherited for:
  - methods in a class that override its superclass’ (javadoc automatically adds “overrides”)
  - methods in an interface that override its superinterface (javadoc automatically adds “overrides”)
  - methods in a class that implement a method in an interface (javadoc automatically adds “specified by”)

software documentation — style guide.

- use `<code>` style for:
  - Java keywords
  - package names
  - class names
  - method names
  - interface names
  - field names
  - argument names
  - code examples
- omit parentheses for the general form of methods and constructors
  - do this: The `add` method enables you to insert items
  - not this: The `add()` method enables you to insert items

software documentation — style guide (2).

- use a phrase for first “sentence” in description
- tag conventions
  - order of tags
  - ordering of multiple tags
  - required tags (`@param`, `@return`)
- be consistent!!

software documentation — the javadoc command.

- syntax:

```
javadoc <options> <packagenames> <sourcefiles> <@files>
```

- where:
  - options = command-line options, including:
    - -sourcepath
    - -tags
    - -overview
  - packagenames = series of names of packages, separated by spaces
  - sourcefiles = series of source file names, separated by spaces; can contain wildcard () to process all .java files
  - @files = files containing packagenames and sourcefiles
- do `man javadoc` for more information

user documentation.

- develop a plan of action
- decide on an approach
- design a useful format
- prepare first draft
- seek objective feedback
- revise as needed (edit!)

user documentation — plan of action.

- analyze your audience
- four types of audiences
  - lay (general public)
  - executive (the boss)
  - expert (peers)
  - technician (maintainers)
- or real world — a mix of all four

user documentation — interests of each audience.

- lay
  - overview
  - conclusions
  - personal implications — use “you”
  - use examples
- executive
  - solutions
  - budget, profits
  - recommandations
  - summary!

user documentation — interests of each audience (2).

- experts
  - data
  - theories
  - details
  - process
- technicians
  - how to fix it
  - be practical
  - they don’t like to *read*, they like to *do*

user documentation — approaches.

- narrative
  - report approach
  - emphasizes results and conclusions
- analytical
  - textbook
  - emphasizes processes
- instructional
  - directions
  - emphasizes procedure
  - takes reader's role

user documentation — formats.

- reports are not *read*, they are *used*
- need a high level of structure!
- general report format
  - front matter
  - internal matter
  - concluding matter

user documentation — front matter.

- letter of transmittal, preface, acknowledgements
- cover
- title page
- summary and/or abstract
- table of contents: headings, subheadings
- lists of tables and illustrations

user documentation — internal matter.

- introduction
  - write this last!
  - background
  - purpose
  - scope
  - be organized and brief
- body of report
  - remember audience analysis
  - conclusions
    - objective findings
    - results based on logical chain of reasoning
  - recommendations
    - subjective findings
    - opinions

user documentation — concluding matter.

- appendices
- glossary
- index
- bibliography

user documentation — summary/abstract.

- very important!
- (executive) summary
  - condensed version of longer document
  - should reflect style and organization of document
  - 5-10/
- abstract
  - brief description of document
  - very tight
  - conveys main idea
  - includes keywords
  - 75-200 words

user documentation — summary/abstract (2).

- present only critical information
- include controlling ideas (what is the thesis?)
- include major findings
- omit details
- should be self-contained
- exercise: take all topic sentences from each paragraph in the report and put them into an abstract/summary

user documentation — editing.

- revising
- systematic editing — do a limited number of things at a time
- distinguish between
  - content (ideas)
  - language
- condense
- consolidate
- simplify

user documentation — writing style.

- active versus passive voice
  - active voice (“Simon played the game.”)
    - more direct
    - natural speaking pattern
    - takes responsibility (“Our algorithm solved the problem.”)
  - passive voice (“The game was played by Simon.”)
    - can feel backwards
    - tends to lead to longer sentences
    - don’t use it too much
    - appears to remove blame (“The problem was not solved by our algorithm.”)

user documentation — writing style (2).

- use vigorous, descriptive verbs
- keep parallel thoughts parallel
  - use same structure in same situation
  - e.g., always start list elements with “-ing” words
- use periodic structure for emphasis and organization
  - i.e., “In the first step,... In the second step,...”
- structure useful paragraphs
  - like a sandwich
- writing should flow!

user documentation — rhetorical devices.

- use examples
- use definitions
- use comparisons
- use classifications and divisions
- use causal analysis
  - deduction (general $\rightarrow$ specific)
  - induction (specific $\rightarrow$ general)

user documentation — descriptions.

- identify with the reader
- only present relevant details
- describe overall object first
- then describe components in detail
- use rhetorical devices
- use graphics
- keep in mind reader's questions:
  - what is it?
  - what does it do?
  - what does it look like?
  - what is it made of?
  - how does it work?

user documentation — controlling jargon.

- two kinds of jargon
  - specialized terms and abbreviations
  - hidden jargon — giving common words a special twist
- easily abused
  - confuses and intimidates
  - can confer spurious legitimacy

user documentation — graphics: tables.

- typeset
- stay with text
- (typically) show exact quantities
- two types:
  - general, reference
  - summary

user documentation — graphics: illustrations.

- artwork
- divorced from text
- (typically) show approximate quantities
- demonstrate trends
- includes
  - graphs (e.g., line graphs, bar charts, ...)
  - diagrams (e.g., flowcharts, UML)
  - cartoons (artist's rendition)
  - maps
  - photographs

user documentation — graphics: design guidelines.

- reader's aid
- intelligent, efficient display of data
- limited symbols
- accurate
- honest scaling
- relatively self-contained
- numbered
- descriptive title (who? what? when? etc.)
- reference in text
- placement as close to first mention in text as possible

user documentation — graphics: accompanying text.

- explain significance of data
- interpret the data
- why is it interesting?
- explain outliers, anomalies

user documentation — good references.

- good dictionary (Random House, Oxford, etc)
- Roget's Thesaurus
- Strunk and White: "Elements of Style"
- Chicago Manual of Style
- "Elements of Programming Style"
- Brusaw, Alred and Oliu: "Handbook of Technical Writing"