Comments
Write for your audience

• Program documentation is for programmers, not end users
• There are two groups of programmers, and they need different kinds of documentation
  – Some programmers need to use your code
    • Do not explain to them how your code works – they don’t care and don’t want to know
    • Tell them what your methods do, how to call them, and what they return
    • Javadoc is the best way to document your code for users
  – Other programmers need to maintain and enhance your code
    • They need to know how your code works
    • Use internal comments for these programmers
• When you work on your program, you are in both groups
  – Document as though, the next time you see your program, you will have forgotten everything about it!
Internal comments

• Use internal comments to:
  – Explain the use of temporary variables
  – Label closing braces in deeply nested statements, or when many lines are between the open and close braces
    ```c
    while (i != j) { ... ... ... ... ... ... } // end while
    ```
  – Explain complex or confusing code
  – Explain what the next section of code does

• *Never* repeat the code!
  – `count = count + 1; // add one to count`
Good code requires few comments

• Explain the use of temporary variables
  – Better: Give them self-explanatory names

• Label closing braces in deeply nested statements, or when many lines are between the open and close braces
  – Better: Don’t nest statements that deeply
  – Better: Keep your methods short

• Explain complex or confusing code
  – Better: Rewrite the code
  – If it’s complex or confusing, it’s probably buggy as well

• Explain what the next section of the method does
  – Better: Make it a method with a self-explanatory name
Good uses of internal comments

• Use internal comments:
  – If you *really* can’t make the code simple and obvious
  – To reference a published algorithm
  – To mark places in the code that need work
    • Eclipse provides three tags for this purpose (you can add more):
      – **TODO** – Means: This code still needs to be written
      – **FIXME** – Means: This code has bugs
      – **XXX** – Means: I need to think about this some more
  – To indicate an intentional flow-through in a switch statement
  – To temporarily comment out code
javadoc

• **javadoc** is a separate program that comes with every Java installation

• **javadoc** reads your program, makes lists of all the classes, interfaces, methods, and variables, and creates HTML pages displaying its results
  – This means **javadoc**’s generated documentation is always accurate

• You can write special documentation (“doc”) comments
  – Your doc comments are integrated into **javadoc**’s HTML page
  – It’s your job to ensure these are also accurate

• **javadoc**’s output is very professional looking
  – This makes you look good
  – It also helps keep your manager from imposing bizarre documentation standards
javadoc

• Always use doc comments to describe the API, the Application Programming Interface
  – Describe all the classes, interfaces, fields, constructors, and methods that are available for use
• javadoc can be set to display:
  – only public elements
  – public and protected elements
  – public, protected, and package elements
  – everything--that is, public, protected, package, and private elements
• Remember, doc comments are for the programmer who uses your classes
  – Anything you want to make available outside the class should be documented
  – It is a good idea to describe, for your own use, private elements as well
javaxdoc is a contract

• In the “real world,” almost all programming is done in teams
  – Your Javadoc is a contract between you and the other members of your team
  – It specifies what you expect from them (parameters and preconditions)
  – It specifies what you promise to give them in return

• Do not be overly generous!
  – Provide what is really needed, but...
  – Remember that anything you provide, you are stuck with debugging, maintaining, and updating
  – Providing too much can really hamper your ability to replace it with something better someday
Know where to put comments!

- **javadoc** comments must be *immediately before*:
  - a class
  - an interface
  - a constructor
  - a method
  - a field
- Anywhere else, **javadoc** comments will be *ignored!*
javadoc comment style

• Use this format for all doc comments:

```java
/**
 * This is where the text starts. The asterisk lines
 * up with the first asterisk above; there is a space
 * after each asterisk. The first sentence is the most
 * important: it becomes the “summary.”
 *
 * @param n Describe the method parameter (don’t say its type).
 * @return Tell what value is being returned (don’t say its type).
 */
public void load(int n) { // p lines up with the / in /**
```

• Eclipse example: PezDispenser.java
Generating HTML javadoc

• You can use Eclipse to generate an HTML version of your javadoc

• This is very useful as it can be opened directly in your browser
  – It is well-formatted according to standard Java documenting conventions
  – Looks very professional!

• To generate: Project -> Generate Javadoc...

• Eclipse example: PezDispenser.java
Tags in documents

• Use the standard ordering for javadoc tags
• In class and interface descriptions, use:
  
  @author  your name
  @version  a version number or date

• In method descriptions, use:

  @param p  A description of parameter p.
  @return   A description of the value returned (unless the method returns void).
  @exception e  Describe any thrown exception.
Keep comments up to date

• Keep comments accurate
  – An incorrect comment is worse than no comment!
  – Any time you change the code, check whether you need to change the comment

• Write the doc comments before you write the code
  – It’s better to decide what to do, then do it than it is to
    *do something, then try to figure out what you did*
Document nearly everything

• If it’s available outside the class, document it!
• If it’s private to the class, it’s still a good idea to document it

• The class itself should be documented
  – In other words: Tell what your program does!
  – You would be surprised how quickly you can forget what the program does
The first sentence is special

• If your doc comment is more than one sentence long:
  – The *first sentence* should *summarize* the purpose of the element (class, method, etc.)
  – This first sentence should make sense when read alone
  – Javadoc uses the first sentence by itself, as a summary
  – Javadoc puts summaries near the top of each HTML page, with a link to the complete doc comment further down the page
Input and output conditions

• Document preconditions, postconditions, and invariant conditions.

• A **precondition** is something that must be true beforehand in order to use your method
  – Example: *The piece must be moveable*

• A **postcondition** is something that your method makes true
  – Example: *The piece is not against an edge*

• An **invariant** is something that must *always* be true about an object
  – Example: *The piece is in a valid row and column*
Bugs and missing features

• Document known problems
  – What? Admit my code isn’t perfect?
  – That might lower my grade, or get me in trouble with my boss!
  – But it will be worse if they discover it themselves

• Be kind to the poor user, struggling to find the bug in her code, when the bug is really in yours
When do you add comments?

• There is *always* time at the start of a project
• There is *never* time at the end of a project
• Remember the 90/90 rule:
  – The first 90% of a project takes the first 90% of the time; the remaining 10% of the project takes the remaining 90% of the time
• Do it right the first time
• Write the comments *before* you write the code.