Documenting code, Grading & parsing JSON
Most important reason to comment

A) To summarize the code
B) To explain how the code works
C) To mark locations that need further work
D) To describe the code’s intent
E) To store non-code information with the code
Ideal Comment Density

- The book cites a study on optimal comment density; what was the ideal density?

A) 1 comment per 10 statements
B) 1 comment per 5 statements
C) 1 comment per 3 statements
D) 1 comment per 2 statements
E) 1 comment per statement
The best documentation is ...

code that doesn’t need comments to be understood

- Write straightforward code
- Use expressive variable and function names
- Follow common conventions
  - getSomething()
  - isEmpty()
What is wrong with this code?

/**
 * Test the primality of a number. See: https://en.wikipedia.org/wiki/Prime_number
 * @param candidate the number to be tested for primality
 * @return true if the candidate is prime, false otherwise
 */
public static boolean isPrime(int candidate) {
    // Negative numbers, 0, 1, and even numbers (other than 2) are not prime
    if (candidate < 2 ||
        ((candidate % 2 == 0) && candidate != 2)) { // Negative numbers, 0, 1, and even numbers (other than 2) are not prime
        return false;
    }

    // If a number can be evenly divided by a number other than 1 and itself,
    // then it is not prime. It is sufficient to test using only odd numbers (as
    // we've already eliminated even candidates) and to only test up to the square
    // root of the candidate, because if there is a divisor greater than the square
    // root, there must also be a divisor smaller than it.
    int sqrt = (int) Math.ceil(Math.sqrt(candidate));
    for (int divisor = 3; divisor <= sqrt; divisor += 2) {
        if (candidate % divisor == 0) { // If a number can be evenly divided by a number other than 1 and itself,
            return false;
        }
    }

    return true;
}
How could more comments be bad?

- harder to read / clutters
- harder to maintain

loose structure of the code
Pseudo-code approach to programming

1. write a series of comments outlining the steps
2. Implement each step in code, leaving the comment in place
Marking end of blocks with comments

if (condition) {
    callSomeFunction(with, some, arguments);
} // if

Is this appropriate commenting?
A) Yes
B) No
Survey

- How much serious programming activity have you had before this class? (serious programming = taking a high school or college class, programming independently for more than 5 hours/week for a prolonged period).

  A) 6 months or less
  B) 6 to 12 months
  C) 12 to 18 months
  D) 18 months to 3 years
  E) More than 3 years
Grading Philosophy

- Your grade in this course relates to your personal growth as a programmer.
  - Wide disparity of incoming experience
  - You are not competing with other students in the class

- Evaluate you relative to where you started
  - We’ll give you honest evaluations

- But there are standards-based expectations for passing
  - Demonstrate ability to write code using good design, style, test construction, etc. and get it to work to specification.
  - Will have multiple attempts to meet standard
Grading (subject to change)

- **Code Reviews:**
  - 70% based on rubric
    - Provided before the code review
    - Items often on a 3-point scale:
      - 3 = exceptional, 2 = good, 1 = needs improvement, 0 = poor
      - Must consistently get 2 or better to get an A.

- **Participation:**
  - 10% being present in class (3 free absences)
  - 10% for clicker question correctness

- **Assignments:**
  - 10% often based on completion
JSON  (www.json.org)

- JavaScript Object Notation

- A lightweight data-interchange format
  - Very commonly used by APIs
- It is easy for humans to read and write.
- It is easy for machines to parse and generate.
Example JSON object

{
   "name_of_a_string": "a string",
   "name_of_a_number": 2080.8827,
   "objects_can_be_values": { "here_is": "another object" },
   "an_array": [ 27, "word", { "objects_can": "be in arrays" } ]
}
Using APIs (e.g., https://newsapi.org)

- Get an API key

- Grab some JSON:
  - https://newsapi.org/v1/articles?source=associated-press&sortBy=top&apiKey=YOUR_API_KEY_HERE

- JSON formatter/pretty printer
  - https://jsonformatter.curiousconcept.com
  - There are a bunch of these, use your favorite
Parsing JSON in Java

- Use the GSON library from Google
  - Use Maven to add the library to your project
- Build classes with fields for the desired elements of the JSON
  - Use the same names and get the types right
- Instantiate a Gson object
  - Gson gson = new Gson();
- Use the fromJSON method to parse the JSON
  - Thing newThing = gson(jsonString, Thing.class);

- Extended example using
To Dos for Tuesday

- Read Ch. 10 (General Issues in Using Variables)
- Learn to use the GSON library
- Create a new IntelliJ project to upload to github
  - Get an API key for https://www.themoviedb.org
  - Save popular movies JSON to a file:
    https://api.themoviedb.org/3/movie/popular?api_key=API_KEY
  - Write Java classes to parse it into (write tests)
  - Write code to query/print titles based:
    - Films of a given genre (e.g., genre_ids includes a specific number)
    - Vote average > a specified threshold
    - Popularity > a specified threshold
- Try to avoid replicating code