What makes for good comments?

Know your audience; generally experienced programmers. The important thing about comments is that you don't write them for yourself. You write them for future maintainers (possibly your own future self). They are not intended for programming novices (except for teaching demos). All comments should be spell-checked and clear in what they convey to the reader. In general, comments should appear near the code they describe, such as at the end of a line of code or in a separate line just above the code.

A goal of writing good documentation is to anticipate the readers' perspectives.

There are different types of comments:

- ***What* comments refer to the actual steps taken.** Use *what* comments to describe what a chunk (also called a code *fragment* or *snippet*) of code is doing. Simple, clear code needs fewer *what* comments. If during a code review you ask the author what a chunk of code is doing, it either means that their code is unclear and/or the *what* comments are deficient. Complex formulas, algorithms, Regular expressions, and formats (file formats, date/time formats, and others), all need *what* comments. When brain-storming using pseudocode, the steps you discover can become *what* comments which are place-holders for code you intend to add, and you fill in the code later. (These are often called ***TODO*** comments, and often you put that word in to find such spots easily. Once you added the code, if the result is short and clear you then remove the comment. (It is redundant "noise" at this point.)

  Unclear code should not have *what* comments added. Instead, fix up the code! (This is known as *refactoring*.)

  Here's an example of a good *what* comment:

  ```
  // Format standard email dates, for example
  // Sat, 13 Mar 2010 11:29:05 -0800:

  String RFC5322DateTimeRegExPattern =
  "^(?:\s*(Sun|Mon|Tue|Wed|Thu|Fri|Sat),\s*)?(0?[1-9]|[1-2][0-
  9]|3[01])\s+(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)\s
  +(19[0-9]{2}|[2-9][0-9]{3}|[0-9]{2})\s+(2[0-3]|[0-1][0-9]):([0-
  5][0-9])(?::(60|[0-5][0-9]))?\s+([-\+][0-9]{2}[0-5][0-
  9]|(?:UT|GMT|(?:E|C|M|P)(?:ST|DT)|[A-IK-
  Z]))(\s*\((\\\(|\\\)|(?<=[^\\])\((?<C>)|(?<=[^\\])\)(?<-
  C>)|[^\(\)]*)*(?(C)(?!))\))*\s*$";
  ```

- ***Why* comments refer to the reasons for writing the code in a particular way.** A *why* comment is for explaining a particular implementation decision or the programmer's intent, especially if it's not the "obvious" design choice. If the obvious choice is to use an `int` but you use a `double` or (even stranger) a `String`, a *why* comment is useful.

  There can be many reasons why some code was written in a non-obvious way. Some examples include formulas that were rearranged to reduce round-off error or to avoid overflow, code you are not allowed change, and code requiring non-intuitive design for security reasons (say to eliminate side-channel attacks).

  Why comments are also used to document the reasons for the code, often listing a reference to an issue (or ticket) in some issue-tracking system.

  Here's an example of a good *why* comment, adapted from Robert Martin's *Clean Code* (p. 59):

  ```
  // The following trim is required, since it
  // removes leading space that would cause the
  ```

```
   // the item to be recognized as another list:
   String listItemContent = match.group(3).trim();
```

- **A third type of comment is a *how* comment.** These are for libraries and reusable modules, and tell the reader how to use your code with their code. The Java "doc" (or "API") comments for Java SE is an example of this. (Such docs are usually extracted from comments in the source code.) *How* comments are used to describe how to use methods, fields, and classes of your code. Generally, only `public` methods and fields need these, but never hurt on private methods too. *How* comments might describe method argument types and ranges, return value type and range, method semantics, pre- and post- conditions, and use cases (example code). They can include warnings (such as *this method is not thread-safe*), examples, and references to other documentation. *What* and *why* comments are generally not included in such API docs.

  The *how* or *API* comments (also called *doc comments*) are specially treated in Java, and are converted to HTML. These will be discussed at a later time.

- **A fourth type of comment is a *required* comment.** These are the comments you must have at the top of each source code file. If your code is under any sort of license or copyright, you can either list the license in the comments or include a link to the license. (I've seen source code file with over 100 lines of such comments at the top!)

  What else is required depends on circumstances. You might need to identify the author(s) who wrote the code and purpose of the code. If any of the code came from others (taken from Stack Overflow for example), include references. For our class, you must include the purpose of the file ("project x, to do such-and-such"), the names of all authors, and any collaborations you used ("I found an example of this at URL xyz").

  Required comments include compliance information, such as what standards the code must meet. For example, security standards, financial standards, military standards, etc. There are standards for code that runs on airplanes and boats. A common example today is compliance with the EU's GDPR (a European privacy standard). Such comments may include a notice about who and when a compliance audit was performed.

  If the code is online or you use online systems to manage the software development process (code review, issue tracker, wiki), have links for those as well.

  Every organization will have different rules for required comments.

Bad comments should be eliminated. Some examples of bad/useless comments include "`//initialize variables`", "`x = 2; //set x to 2`", "`num = num + 1; // add 1 to num`", "`//Default constructor:`", etc.

Often you can eliminate such bad comments by choosing better names. For example:

```
p = n * e + t; // Compute price
```

can be written as:

```
price = quantity * priceEach + tax;
```

The same applies to method names.

Avoid ambiguous comments, for example:

```
// No file found means all defaults are loaded
```

That must have meant something to the author, but what? Is this a place-holder to remind the programmer to do something later, or a statement that some other code was supposed to load defaults, or something else?

Comments are a vital part of your code. Always keep them clear, correct, and useful. Do not forget to update comments whenever you update the related code.