Chapter 5: Formatting

**The Purpose of Formatting**

* Formatting is Important because it’s about communication, the most important part of being a professional developer.
* Making the code works is not the first order of business for a professional programmer as many have thought.
* Formatting for a professional programmer is more important the code because as time goes by code changes and evolve but the formatting and readability of the code will always stay.

**Vertical Formatting**

* Vertical Formatting Measures the file size in terms of how many lines that file has.
* It’s possible to build significant systems with only a minimum of 200 lines of code and a maximum of 500.
* It’s much more desirable to have short lines of code as its easier to understand than large files.

**The Newspaper Metaphor**

* Formatting a Code should like a well written newspaper article where the topmost contains the broad idea of what you are going to read and as you read downward you’ll find the details.
* The name of the source file should be simple that it can easily be understood and self-explanatory, enough to tell the reader if it’s the right module
* The topmost part of the source file should contain the most high level concepts and algorithm and as it go downwards the details increases.
* If the formatting of the source file has no parts and just a disorganized list of long lines then it would be harder to understand.

**Vertical Openness between Concepts**

* Vertical Openness is important because it separates each concepts making them more understandable to read and is pleasing to the eye.
* If the blank lines where to disappear the code will be much harder to read and understand because you won’t know where the next concept of the code starts and ends.

**Vertical Density**

* Vertical density implies close association meaning that Lines of codes that are related to each other should be vertically dense.
* Concepts that are vertically dense are much more easily understood just by one look.
* Avoid having Concepts separated by comments or lines which is hard to understand and comprehend

**Vertical Distance**

* Concepts that are so closely related to each other that they are on the same source file, should be closer together based on how important each is to the understandability of the other.

**Variable Declaration**

* Variable should be displayed as close as possible to their usage
* if the function is not long then the variables should be displayed on top of the function
* Control variables should be declared within loop statements

**Instance variables**

* This type of variable should be declared at the top of each class as they are used most common by methods of the class.
* Always follow the convention of declaring this variable in a well-known place to easily be located by everyone.

**Dependent Functions**

* If the functions are dependent to another then they should be vertically close to each other, the function caller should always be above the function callee.
* Having followed this writing convention the code will have a natural flow and readers will know what to expect.

**Conceptual affinity**

* Even if the code has no relation to one another as long as the code for example a function has correlation in a conceptual manner like having to perform a similar operation, basic task or naming convention, then such codes should be close to one another.

**Vertical Ordering**

* As mentioned earlier functions that are being called should be below the function that does the calling, so readers can expect low level details to come last and while high levels are first.
* This kind of ordering is important to have good flow and easy reading allowing readers to be able to skim through the code without being to immerse in the detail.

**Horizontal Formatting**

* A line should be kept short in a way that readers won’t have to scroll to the right to see the rest of the code.
* Be sure to set a limit, and don’t rely on wide monitors or shrinking the fonts to fit 200 characters across the screen

**Horizontal Openness and Density**

* Using horizontal white space has many uses that can make the code more readable and understandable
* Surrounding assignment operators with white spaces can make them more obvious as well as the two distinct and major elements of the assignment statements.
* This can also be used to make the precedence of operators more obvious and make equations more readable.

**Horizontal Alignment**

* Aligning code declarations and assignments in a horizontal straight line is not very useful as it can lead reader to miss important details and forces them to read down the list of codes.
* Unaligned declarations and assignments are much more preferable as its not time consuming to format and doesn’t force the reader to miss important details.

**Indentation**

* Without indentation codes will be virtually unreadable and can’t be understood by humans, that’s why programmers rely on the indentation scheme to make their codes readable.
* Indentation scheme relies on the hierarchy, making it visible by indenting code lines according to their position in the hierarchy.
* Methods within classes are indented one level to the right of the class and implementation of the methods are indented one level to the right of the method declaration.

**Breaking indentation**

* Not using indentation can be tempting specially to short if statement, while loops or short functions, but it’s just as important to use indentation for them.

Dummy Scopes

* The body of a while loop or for statement is sometimes a dummy, in which case if you can’t avoid this type of code make sure to indent them properly and surrounded by braces.

**Team Rules**

* Every programmer has their own formatting rules but if they work with a team then they should follow what the team agreed upon as the single formatting rule every member should follow.
* A good software system is one that is composed of a nice to read documents that has a consistent and smooth style, making the reader trust the formatting gesture of the whole file as something that is easily understandable and readable.

**Uncle Bobs Formatting Rule**

* The example shown has used and implemented every formatting rule that was mentioned in this chapter.

Chapter 6: Objects and Data Structures

**Data Abstraction**

* Simply using interfaces or setters and getters is not enough, programmers should consider other better ways to represent an object’s data.
* It’s more preferable to hide to codes implementation in an abstract manner rather than exposing it, as it allows the user to manipulate the essence of the data.

**Data/Object Anti-Symmetry**

* The difference between objects and data structures us that they are complimentary in nature meaning they virtual opposite of one another
* Objects tends to hide their code implementation and data behind abstractions and expose functions that operated on the data.
* Data Structure on the other hand exposes their data and have no meaningful functions to show.

**The Law of Demeter**

* The law of Demeter states that a module should not know about the inner details about the objects it operates.
* As mentioned, objects are designed to hide their data and expose operations, so they should not expose its internal structure through accessors.

**Train Wrecks**

* In the code example shown it’s considered as a train wreck or a bad practice because chain of codes like that are sloppy and would expose the inner structure of the multiple objects.
* The following snippet of code can’t be properly decided if it’s a violation of Demeter and would depend if the ctxt, Options and ScratchDir are objects or data structures.
* If they are objects then they violet the law of Demeter because object should hide their internal structure rather than exposed.
* If they are data structures then they do not violate the law of Demeter because data structures naturally exposes their internal structure.

**Hybrids**

* Mixing both objects and data structure creates a hybrid that can be difficult to add new functions and add new data structures.
* Always avoid using hybrid as they are indicative of a poor design and make it difficult to evolve, improve and upgrade the code.

**Hiding Structures**

* What if ctxt, options, and scratchDir are objects with real behavior then then neither of the two option is good because objects hides their internal structures so it’s not good to navigate through their internals.
* If ctxt is an object then it’s better to tell it to do something and not ask to navigate through its internals, this way the object would be able to hide its internal structure and not violate the law of Demeter.

**Data Transfer Objects**

* The most accurate representation of a data structure is a class with public variables and no function and is sometimes called a data transfer object
* Data transfer object are very useful structures when communicating with databases.
* They are often start the first stages of converting raw data in a database into object in the application code.

**Active Record**

* Active records are data structures with public variables and are typically direct translations from database tables.
* Programmers often makes mistakes of treating this data structures as if they were objects by putting methods in them creating a hybrid between data structure and object.
* It’s better to treat the active record solely as a data structure and separate the objects that would contain the method that hides their internal data.