Clean Code:

A Handbook of Agile Software Craftsmanship

Chapter 05: Formatting

What does formatting exactly mean?

Your code should follow some simple rules to have a stable format that is readable and easy to understand. The communication between different parts of the code, such as different methods and properties, is like the grammar of a language. It should have an integrated structure.

By formatting we refer to the horizontal and vertical length of a code file, the space between elements of code and the density of instructions.

Vertical Formatting

Vertical formatting

How large could a source file(class) ever be?

The shorter, the better. Smaller classes are understood easier.

We want our source code to be read like a newspaper. The names should be simple but descriptive. The module name tells the reader the story behind it. The upper parts of the file should talk about the general issues. The more we go down a file, the more details we see.

Vertical formatting

- ## Vertical Openness Between Concepts

Every line of code represents a condition or command and every group of lines represents an entire thought. The thoughts must be separated using empty line.

Vertical openness between concepts

```
private List<Flight> getFlightRows() {
    List<Flight> flightList = new LinkedList<>();
    for (Flight flight : flightListFromFlightDTOList(flightsDTOList))
        flightList.add(readFlightFromTable());
    flightList = flightList.sortFlightsByDepartureTime();
    return flightList;
}
```

```
private List<Flight> getFlightRows() {
    List<Flight> flightList = new LinkedList<>();
    for (Flight flight : flightListFromFlightDTOList(flightsDTOList))
        flightList.add(readFlightFromTable());

    flightList = flightList.sortFlightsByDepartureTime();
    return flightList;
}
```

Vertical formatting

- ## Vertical Density

Code lines that are related to the same concept should stick together and create a high density of instructions. Specially comments are not allowed to intervene between them.

Vertical density

```java
public class ReportConfig{
    /**
    * The class name of the reporter listener
    */
    private String m_className;

    /**
    * The properties of the reporter listener
    */
    private List<Property> m_properties = new List<>();

    public void addProperty(Property property){
        m_properties.add(property);
    }
}
```

```java
public class ReportConfig{
    private String m_className;
    private List<Property> m_properties = new List<>();

    public void addProperty(Property property){
        m_properties.add(property);
    }
}
```

Vertical formatting

- ## Vertical Distance

  Related concepts should be closer to each other.

  Methods: Similar and related methods should be implemented inside one class close to each other. If method A calls method B, B should be declared under A without distance. If we call B, C and D in A, they should be declared under A in the exact order.

  Class Properties: Should be declared at the top of the class.

  Local Variables: Their declaration and usage should be close to each other. Since our methods are simple and short, we declare the variables at the top the methods.

Vertical distance

```
public void A{
    B();
    C();
    if (condition)
        D();
}

private void B(){}
private void C(){}
private void D(){}
```

Vertical formatting

- ## Conceptual Dependency

  The elements of code with more conceptual dependency should be closer to each other. It keeps the structure of the code clean and continuous. For example methods with similar naming or logic better be declared underneath each other.

Conceptual Dependency

```
public class Assert{
   public static void assertTrue(String message, boolean condition){
      if(!condition)
         fail(message);
   }
   public static void assertTrue(boolean condition){
      assertTrue("", condition);
   }
   public static void assertFalse(String message, boolean condition){
      assertTrue(message, !condition);
   }
   public static void assertTrue(boolean condition){
      assertFalse("", condition);
   }
}
```

Vertical formatting

- ## Vertical Order

  The called method should be declared under the caller method. The details of the implementation belong to the lower parts of the class. At the top of the class we talk about the general abstractions.

Horizontal Formatting

Horizontal formatting

How long could a line of code be?

The line should not be so long, as it makes reading the code more difficult. The standard number of characters in most IDEs is 120 and there is a vertical line that indicated the line length limit.

Horizontal formatting

- ## Horizontal Openness And Density

Horizontal space is used for attaching related objects and detaching with a weak relation.

Assignment Operations: Variable Name + Space + = + Space + Value + ;

Method Arguments: Method Name + (+ First Argument + , + Space + Next Argument + ... +)

Mathematical Equations: The operations with the same priority stick together. The IDEs auto formatting systems usually don't observe this point.

Horizontal Openness And Density

```
int n = 10;

private void methodName(int number, String name){}

double delta = b*b - 4*a*c;
```

Horizontal formatting

- ## Horizontal Alignment

It's not recommended to use horizontal alignment. Because it emphasizes on a minor point and tempts the reader not to look at variable's type in the assignment operations.

Horizontal Alignment

```
private     String                  firstName;
private     String                  lastName;
private     Address                 address;
private      int                    age;
public      boolean                 isMarried;
protected EmploymentStatus employmentStatus;
```

```
private String firstName;
private String lastName;
private Address address;
private int age;
public boolean isMarried;
protected EmploymentStatus employmentStatus;
```

Horizontal formatting

- Indentation

One of most important points to consider is to have a recognized indentation. It makes the code far more readable. Usually the IDEs apply the correct indentation according to the {} and the scope automatically.

Indentation

```
private void setFirstName(String firstName) {this.firstName = firstName;}
```

```
private void setFirstName(String firstName) {
    this.firstName = firstName;
}
```

Formatting

- ## Team Rules

Sometimes the development team comes to an agreement about a self made format for the code. It's not a good idea to use it. The best way is to use the standard formatting.

But whether we use the standard format or the one we made ourselves, all team members should apply it to their code so that the code doesn't look like it was written by multiple teams.

A good software system is consisted of a readable document set and the best document is the code source. The reader should be guaranteed that all the files in a system go by the same formatting and they could read it like a well written document.