CODING BEST PRACTICES

You should follow the existing coding style that's already used in the application and always follow the coding best practices.

HERE ARE SOME BEST PRACTICES YOU SHOULD ALWAYS HAVE IN MIND:

1. **Use consistent indentation**
   There is no right or wrong indentation that everyone should follow. The best style, is a consistent style. Once you start competing in large projects you'll immediately understand the importance of consistent code styling.
   2. Follow the DRY Principle
   DRY stands for "Don't Repeat Yourself.
   The same piece of code should not be repeated over and over again.

Don't Repeat Yourself (DRY) is a software development principle, the main aim of which is to reduce repetition of code.

Write Everything Twice (WET) is a cheeky abbreviation to mean the opposite i.e. code that doesn't adhere to DRY principle.

It is quite obvious which one of the two should all developers be aiming for. But hey!, someone out there might be proving me wrong at this very moment.

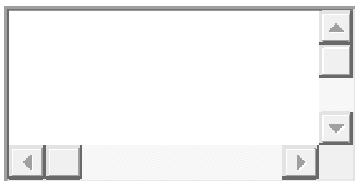
In this post, we look at the benefits of applying DRY principle to your code. Firstly, we will start with a simple example that illustrates the basic advantage of the DRY principle.

DRY — Simple example

Assume you have many places in your code that need to be executed based on the current user's role. For instance, createPage() can only be

executed if the user is an editor or an administrator, deletePage() only if the user is an administrator etc.

Instead of spreading the logic of checking for a user's role in both createPage and deletePage, we could use a single function isPermitted() as below.



```
1
//get the current Subject
2
Subject currentUser = context.getSubject();
3

4
if (isPermitted(currentUser)) {
5
    //allow execution of deletePage
6
} else {
7
    //block execution
8
}
```

By keeping the logic of isPermitted() to one place, you avoid duplication and also enable re-use of the code. The added advantage is separation of logic i.e. createPage() and deletePage() don't need to know how the permission is checked.

As always there is more than meets the eye.

Advantages of DRY

Maintainability

The biggest benefit of using DRY is maintainability. If the logic of checking permissions was repeated all over the code, it becomes difficult

to fix issues that arise in the repeated code. When you fix a problem in one, you could easily forget to fix the problem in other occurrences. Also, if you have to modify the logic, you have to copy-paste all over the place. By having non-repeated code, you only have to maintain the code in a single place. New logic and bug fixes can be made in one place instead of many. This leads to robust and dependable software.

Readability

More often than not, DRY code is more readable. This is not because of the DRY principle itself, but rather because of the extra effort the developer put into the code to make it follow certain principles such as DRY.

Reuse

DRY inherently promotes reuse of code because we are merging 2 or more instances of repeating code into a single block of code. Reusable code pays of in the long run as it speeds up development time.

Cost

If management needs to be convinced to spend more time on improving the quality of code, this is it. More code costs more. More code takes more people more time to maintain and to address bugs. More time to develop and more bugs leads to a very unhappy customer. Enough said!

Testing

We are talking about unit tests and integration tests here, not manual testing. The more paths and functions you have to cover using the tests, the more code you have to write for tests. If code is not repeated, you just have to test one main path. Of course, different behaviors still need to be tested.

Caution

With all the advantages of using DRY, there are some pitfalls though.

1. Not all code needs to be merged into one piece. Some times 2 pieces of code can look similar but with subtle differences. When to merge these prices of code into one and when to leave them separated needs to be thought over carefully.

2. If the code is “over dried”, it becomes difficult to read and understand. I have also seen developers trying to apply DRY principles even when there is only one instance of a block of code. While I appreciate their thinking and foresight into making the code better and reusable, I wouldn’t bother to make the code follow DRY principle until the situation to re-use it is needed.
3. Often missed, DRY is not to be limited to just the code. It is to be applied in equal measure to database design, documentation, testing code etc.

3. Avoid Deep Nesting
Too many levels of nesting can make code harder to read and follow.
For example:

```
if (a) {
    …
  if (b) {
      …
    if (c) {
        …
        …
        …
    }
  }
}
```

Can be written as:

```
    ```
    if (a) {
       return …
    }
    if (b) {
       return …
    }
    if (c) {
       return …
    }
    ```
```

4. Limit line length
Long lines are hard to read. It is a good practice to avoid writing horizontally long lines of code.
5. File and folder structure
You should avoid writing all of your code in one of 1-2 files. That won’t break your app but it would be a nightmare to read, debug and maintain your application later.

Keeping a clean folder structure will make the code a lot more readable and maintainable.

6. Naming conventions

Use of proper naming conventions is a well known best practice. Is a very common issue where developers use variables like X1, Y1 and forget to replace them with meaningful ones, causing confusion and making the code less readable.

7. Keep the code simple

The code should always be simple. Complicated logic for achieving simple tasks is something you want to avoid as the logic one programmer implemented a requirement may not make perfect sense to another. So, always keep the code as simple as possible.

For example:

```
1  ```
2  if (a < 0 && b > 0 && c == 0) {
3     return true;
4  } else {
5     return false;
   ```
6
```

Can be written as:

```
?
1 ```
2 return a < 0 && b > 0 && c == 0;
3
```
```