Clean Code 4

1.Explain Yourself in Code

- // Is employee eligible for full benefits?
- if (employee.flags & HOURLY_FLAG && employee.age > 65)
- if (employee.isEligibleForFullBenefits())

2. Classes should have a small number of instance variables

3.Don't add unneeded context

If your class/object name tells you something, don't repeat that in your variable name.

Bad:

```
class Car {
  public String carMake = "Honda";
  public String carModel = "Accord";
  public String carColor = "Blue";
}

void paintCar(Car car) {
  car.carColor = "Red";
}
```

Good:

```
class Car {
  public String make = "Honda";
  public String model = "Accord";
  public String color = "Blue";
}

void paintCar(Car car) {
  car.color = "Red";
}
```

4.Variables

It is recommended to declare variables as close as possible to the location where they are used. You don’t want to declare a variable on line 1 and use it only from line 15.

5.Functions

Functions that are dependent should be placed together. It is recommended to have the child function under the function that calls it. In this way you will be able to read the code very easily, without having to navigate too much between different places inside the code.

6.Use Nouns for Class Name & Use Pascal Case

Classes don't take things; they are the things. Class is mainly a blueprint for something. Don't use the verb in the class name.
Also, a class should contain Pascal case. Camel case is used for objects, so this won't be very clear if you use camel case for class.

```
//bad practice
class MakeCar = {
    //...
}

//good practice
class Car = {
    //...
}
```

7. Rule for class names
“The name of a class is inversely proportional to the size of the scope that contains it.”

- classes at the global scope have one word names
- derived classes have multiple word names
- inner classes have multiple word names
- as the scope shrinks, the name grows

```
protocol Store {}
protocol FeedStore: Store {}
final class CodableFeedStore: FeedStore {}
final class InMemoryFeedStore: FeedStore {}
```

8. Avoid negative conditionals.
Negatives are just a bit harder to understand than positives. So, when possible, conditionals should be expressed as positives. For example:

```
if (buffer.shouldCompact())
```

is preferable to

```
if (!buffer.shouldNotCompact())
```