Clean Code

Notes of Chapter 2: Meaningful Names

Introduction:

Names are everywhere in software. We name our variables, our functions, our arguments,
classes, and packages. We name our source files and the directories that contain them. We name our jar files and war files and ear files. We name and name and name. Because we do so much of it, we'd better do it well. What follows are some simple rules for creating
good names.

Use Intention-Revealing Names:

- take care with your names and change them when you find better ones.
- If a name requires a comment, then the name does not reveal its intent.
- Choosing names that reveal intent can make it much easier to understand and change code.
- Everyone who reads your code (including you) will be happier if you do.

Avoid Disinformation:

- avoid words whose entrenched meanings vary from our intended meaning.
- Beware of using names which vary in small ways.
- Spelling similar concepts similarly is *information*. Using inconsistent spellings is *disinformation*.

Make Meaningful Distinctions:

Programmers create problems for themselves when they write code solely to satisfy a compiler or interpreter.

Number-series naming (a1, a2,... aN) is the opposite of intentional naming. Such
names are not disinformative—they are noninformative; they provide no clue to the
author's intention. Consider:

- Noise words are another meaningless distinction.
- Noise words are redundant.

Use Pronounceable Names:

- Make your names pronounceable.
- Use Pronounceable Names because programming is a social activity.
- Intelligent conversation is now possible.

Use Searchable Names:

Single-letter names and numeric constants have a particular problem in that they are not
easy to locate across a body of text.

- longer names trump shorter names, and any searchable name trumps a constant in code.
- The length of a name should correspond to the size of its scope.

Avoid Encodings:

- Encoding type or scope information into names simply adds an extra burden of deciphering.

- **Member Prefixes**
  You also don't need to prefix member variables with m_ anymore. Your classes and functions
  should be small enough that you don't need them. And you should be using an editing
  environment that highlights or colorizes members to make them distinct.

```
public class Part {
private String m_dsc; // The textual description
void setName(String name) {
m_dsc = name;
}
}
```

```
public class Part {
String description;
void setDescription(String description) {
this.description = description;
}
       }
```

Avoid Mental Mapping:

- single-letter names for loop counters are traditional. However, in most other contexts a single-letter name is a poor choice.
- clarity is king.

Class Names

Classes and objects should have noun or noun phrase names like Customer, WikiPage, Account, and AddressParser. Avoid words like Manager, Processor, Data, or Info in the name
of a class. A class name should not be a verb.

Method Names

Methods should have verb or verb phrase names like `postPayment`, `deletePage`, or `save`.
Accessors, mutators, and predicates should be named for their value and prefixed with `get`,
`set`, and `is` according to the javabean standard.

```
string name = employee.getName();
customer.setName("mike");
if (paycheck.isPosted())...
```

When constructors are overloaded, use static factory methods with names that
describe the arguments. For example,

```
Complex fulcrumPoint = Complex.FromRealNumber(23.0);
```

is generally better than

```
Complex fulcrumPoint = new Complex(23.0);
```

Consider enforcing their use by making the corresponding constructors private.

Don't Be Cute:

- Choose clarity over entertainment value.
- Say what you mean. Mean what you say.

Pick One Word per Concept:

- Pick one word for one abstract concept and stick with it.

Don’t Pun:

- Avoid using the same word for two purposes.

Use Solution Domain Names:

- Remember that the people who read your code will be programmers. So go ahead and use computer science (CS) terms, algorithm names, pattern names, math terms, and so forth.
- Choosing technical names for those things is usually the most appropriate course.

Use Problem Domain Names:

When there is no “programmer-eese” for what you’re doing, use the name from the problem domain.

Separating solution and problem domain concepts is part of the job of a good programmer and designer. The code that has more to do with problem domain concepts
should have names drawn from the problem domain.

Add Meaningful Context:

- place names in context for your reader by enclosing them in well-named classes, functions, or namespaces.
- You can add context by using prefixes: addrFirstName, addrLastName, addrState, and so on.

Don’t Add Gratuitous Context:

- Shorter names are generally better than longer ones, so long as they are clear. Add no more context to a name than is necessary.

Final Words:

- choosing good names requires good descriptive skills and a shared cultural background.
- Programmers are actually grateful when names change (for the better).