Meaningful Names

Avoid Disinformation:

- Programmers must avoid leaving false clues that obscure the meaning of code.
- Do not refer to a grouping of accounts(array) as an accountList unless it's actually a List.

Make Meaningful Distinctions:

- If names must be different, then they should also mean something different.
- Noise words are redundant.
- Noise words are another meaningless distinction.
- Distinguish names in such a way that the reader knows what the differences offer.
- Imagine finding one class named Customer and another named CustomerObject. What should you understand as the distinction? Which one will represent the best path to a customer's payment history?

Use Pronounceable Names:

If you can't pronounce it, you can't discuss it without sounding like an idiot. "Well, over here on the bee cee arr three cee enn tee we have a pee ess zee kyew int, see?" This matters because programming is a social activity.

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.
- My personal preference is that single-letter names can ONLY be used as local variables inside short methods. The length of a name should correspond to the size of its scope.
- If a variable or constant might be seen or used in multiple places in a body of code,it is imperative to give it a search-friendly name(no single letter names).

Hungarian Notation:

- Hungarian notation is a convention for naming and differentiating between data objects.
- **When Hungarian notation is used, a programmer adds an indicator prefix to each object name to easily and readily identify its type.**
- if the object YesOrNo were a boolean variable, its Systems Hungarian name would be bYesOrNo.

Meaningful Names

- So nowadays HN and other forms of type encoding are simply avoided.
- They make it harder to change the name or type of a variable, function, or class. They make it harder to read the code.

Interfaces and Implementations:(IMP)

- IshapeFactory ior ShapeFactory? I prefer to leave interfaces unadorned. The preceding I, so common in today's legacy wads, is a distraction at best and too much information at worst. I don't want my users knowing that I'm handing them an interface.
- Implementing class name should be InterfacenameImpl for eg: ShapeFactoryImpl.
- You also don't need to prefix member variables with m_anymore.

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 Infoin the name of a class. A class name should not be a verb.

Method Names:(IMP)

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 isaccording to the javabean standard.

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.

This is a class with one primary and two secondary constructors:

```
class Color {
    private final int hex;
    Color(String rgb) {
        this(Integer.parseInt(rgb, 16));
    }
    Color(int red, int green, int blue) {
        this(red << 16 + green << 8 + blue);
    }
    Color(int h) {
        this.hex = h;
    }
}
```

Meaningful Names

This is a similar class with three static factory methods:

```
class Color {
    private final int hex;
    static Color makeFromRGB(String rgb) {
        return new Color(Integer.parseInt(rgb, 16));
    }
    static Color makeFromPalette(int red, int green, int blue) {
        return new Color(red << 16 + green << 8 + blue);
    }
    static Color makeFromHex(int h) {
        return new Color(h);
    }
    private Color(int h) {
        return new Color(h);
    }
}
```

There are three basic advantages to using static factory methods instead of constructors

- They have names.
- They can cache.
- They can sub-type.

Pick One Word per Concept:

Pick one word for one abstract concept and stick with it. For instance, it's confusing to have fetch, retrieve,and getas equivalent methods of different classes.

Don't Pun:

Avoid using the same word for two purposes. Using the same term for two different ideas is essentially a pun.

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.

Meaningful Names