Cohesion and Coupling

A class should represent a single concept. The public methods and constants that the public interfaces exposes should be cohesive. That is, all interface features should be closely related to the single concept that the class represents.

If you find that the public interface of a class refers to multiple concepts, then that is a good sign that is may be time to use separate classes instead. Consider, for example, the public interface of the Purse class:

```java
public class Purse {
    public Purse() { . . . }
    public void addNickels(int count) { . . . }
    public void addDimes(int count) { . . . }
    public void addQuarters(int count) { . . . }
    public double getTotal() { . . . }
    public static final double NICKEL_VALUE = 0.05;
    public static final double DIME_VALUE = 0.10;
    public static final double QUARTER_VALUE = 0.25;
    . . .
}
```

There are really two concepts here: a purse that holds coins and computes their total, and the value of the individual coins. It would make more sense to have a separate Coin class and have coins responsible for knowing their value.

```java
public class Coin {
    public Coin(double aValue, String aName) { . . . }
    public double getValue() { . . . }
    . . .
}
```

Then the Purse class can be simplified:

```java
public class Purse {
    public Purse() { . . . }
    public void addCoin(Coin aCoin) { . . . }
    public double getTotal() { . . . }
    . . .
}
```

This is clearly a better solution, because it separates the responsibilities of the purse and the coins.

Many classes need other classes to do their job. For example, the restructured Purse class now depends on the Coin class to determine the total value of the coins in the purse.

To visualize relationships such as dependence between classes, programmers draw class diagrams. UML ("Unified Modeling Language") is the notation for objects and classes. It is a notation for object-oriented analysis and design invented by Grady Booch, Ivar Jacobson, and James Rumbaugh. The UML notation distinguishes between object diagrams and class diagrams. In an object diagram the class names are
In a class diagram, you denote dependency by a dashed line with a >- shaped open arrow tip that points to the dependent class. For example:

![Class Diagram]

Note that the Coin class does not depend on the Purse class. Coins have no idea that they are being collected in purses, and they can carry out their work without ever calling any method in the Purse class.

If many classes of a program depend on each other, then we say that the coupling between classes is high. Conversely, if there are few dependencies between classes, then we say that the coupling is low.

Why does coupling matter? Of the Coin class changes in the next release of the program, all classes that depend on it may be affected. If the change is drastic, the coupled classes must all be updated. Furthermore, if we would like to use a class in another program, we have to take with it all the classes on which it depends. Thus, we want to remove unnecessary coupling between classes.

**Consistency**

So far you know of two criteria to analyze the quality of the public interface of a class. You should maximize cohesion and remove unnecessary coupling. There is another criterion that you should pay attention to - consistency. When you have a set of methods, follow a consistent scheme for their names and parameters. This is simply a sign of good craftsmanship.

Sadly, you can find any number of inconsistencies in the standard library. Here is an example. To show an input dialog, you call

```java
JOptionPane.showMessageDialog(null, messageString)
```

To show a message dialog, you call

```java
JOptionPane.showMessageDialog(null, messageString)
```

What's the `null` parameter? It turns out the the showMessageDialog method needs a parameter to specify the parent window, or null if no parent window is required. But the showInputDialog method requires no parent window. Why the inconsistency? There is no reason. It would have been easy to supply showMessageDialog method the exactly mirrors the showInputDialog method.

Inconsistencies such as these are not a fatal flaw, but they are an annoyance, particularly because they can be so easily avoided.