The Object-Oriented Thought Process

Chapter 05

Class Design Guidelines

Modeling Real World Systems

One of the primary goals of object-oriented (OO) programming is to model real-world systems in ways similar to the ways in which people actually think.

- Rather than using a structured, or *top-down, approach,* where data and behavior are logically separate entities.
- The OO approach encapsulates the data and behavior into objects that interact with each other.

Indentifying Public Interfaces

Perhaps the most important issue when designing a class is to keep the public interface to a minimum.

- The entire purpose of building a class is to provide something useful and concise
  - "the interface of a well-designed object describes the services that the client wants accomplished."

The Minimum Public Interface

If the public interface is not properly restricted, problems can result in the need for debugging, and even trouble with system integrity and security can surface.

- Creating a class is a business proposition, and it is very important that the users are involved with the design right from the start and throughout the testing phase.

Hiding the Implementation

The implementation should not involve the users at all.

- The implementation must provide the services that the user needs.
- But how these services are actually performed should not be made apparent to the user.

Robust Constructors

A constructor should put an object into an initial, safe state.

- This includes issues such as attribute initialization and memory management.
- You also need to make sure the object is constructed properly in the default condition.
- It is normally a good idea to provide a constructor to handle this default situation.

Error Handling Design

The general rule is that the application should never crash.

- It is not a good idea to ignore potential errors.
- When an error is encountered, the system should either fix itself and continue, or exit gracefully without losing any data that's important to the user.

Documenting a Class

One of the most crucial aspects of a good design, whether it's a design for a class or something else, is to carefully document the process.

- Too much documentation and/or commenting can become background noise and may defeat the purpose of the documentation in the first place.
- Make the documentation and comments straightforward and to the point.

Cooperating Objects

A class will service other classes; it will request the services of other classes, or both.

- When designing a class, make sure you are aware of how other objects will interact with it.

Designing with Reuse in Mind

Objects can be reused in different systems, and code should be written with reuse in mind.

- This is where much of the thought is required in the design process. Attempting to predict all the possible scenarios in which an object must operate is not a trivial task.
- In fact, it is virtually impossible.

Designing with Extensibility

Adding new features to a class might be as simple as extending an existing class, adding a few new methods, and modifying the behavior of others.

- It is not necessary to rewrite everything.
- Consider the future use of a class when designing it.

Descriptive Names

Make sure that a naming convention makes sense.

- People often go overboard and create conventions that are incomprehensible to others.
- Take care when forcing others to conform to a convention.
- Make sure that conventions are sensible.

Isolating Nonportable Code

If you are designing a system that must use nonportable (native) code (that is, the code will run only on a specific hardware platform), you should abstract this code out of the class.

- By abstracting out, we mean isolating the non-portable code in its own class or at least its own method (a method that can be overridden).

Copying and Comparing Objects

It is important to understand how objects are copied and compared.

- You must make sure that your class behaves as expected, and this means you have to spend some time designing how objects are copied and compared.

Minimizing Scope

Keeping the scope as small as possible goes hand-in-hand with abstraction and hiding the implementation.

- The idea is to localize attributes and behaviors as much as possible.
- In this way, maintaining, testing, and extending a class are much easier.

Class Responsibility

Classes should be responsible for their own behavior whenever possible.

- For example, if you have several objects of various shapes, each individual shape should be responsible for drawing itself.
- This design practice localizes functionality and make it easier to add new shapes.

Maintainability

Designing useful and concise classes promotes a high level of maintainability.

- Just as you design a class with extensibility in mind, you should also design with future maintenance in mind.
- One of the best ways to promote maintainability is to reduce interdependent code.
  - that is, changes in one class have no impact or minimal impact on other classes.

Using Iteration

Create the code in small increments and then build and test it at each step.

- A good testing plan quickly uncovers any areas where insufficient interfaces are provided.
- In this way, the process can iterate until the class has the appropriate interfaces.
  - Iterate through all phases of the software life cycle.

Testing the Interface

The minimal implementations of the interface are often called *stubs.*

- By using stubs, you can test the interfaces without writing any *real code.*
- Stubs also allow testing without having the entire system in place.

Object Persistence

Object persistence is the concept of maintaining the state of an object.

- When you run a program, if you don't save the object in some manner, the object dies, never to be recovered.
- In most business systems, the state of the object must be saved for later use.

Marshalling Objects

To send an object over a wire (for example, to a file, over a network), the system must deconstruct the object (flatten it out), send it over the wire, and then reconstruct it on the other end of the wire.

- This process is called *serializing an object.*
- *The act of sending the object across a* wire is called *marshaling an object.*