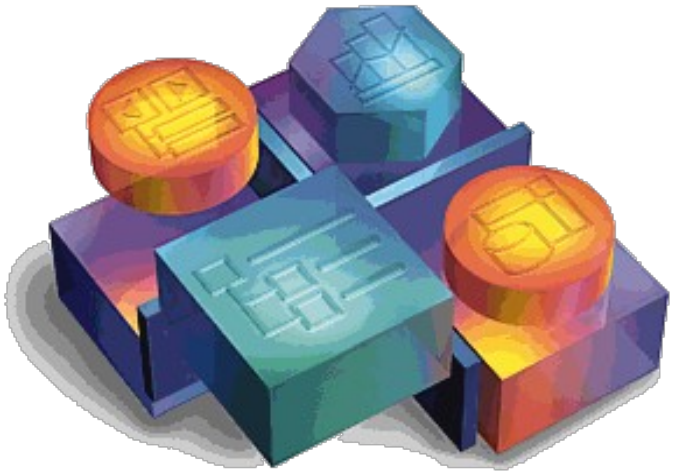
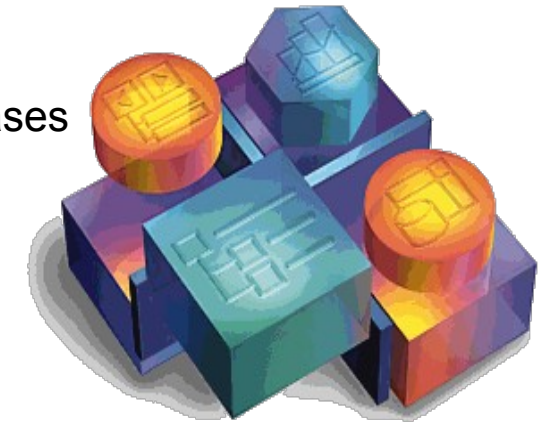
Object-Oriented Programming Design



1. Introduction to Object-Oriented Concepts
2. How to think in Terms of Objects
3. Advanced Object-Oriented Concepts
4. The anatomy of a Class

5. Class Design guidelines

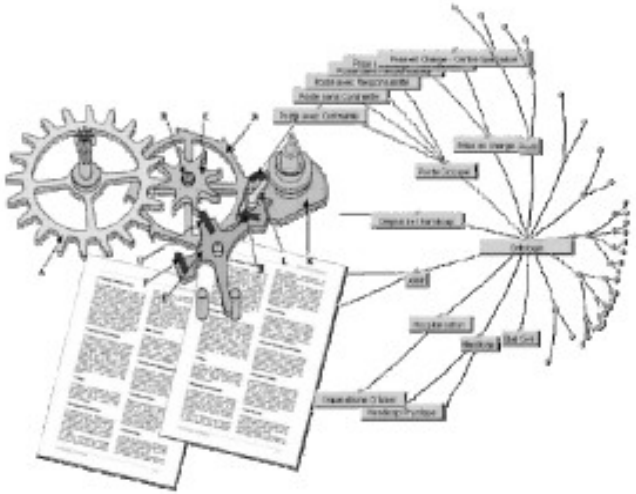
6. Designing with objects
7. Mastering Inheritance and Composition
8. Frameworks and Reuse: Designing with interfaces and Abstract classes
9. Building objects
10. Creating Object Models with UML
11. Objects and Portable Data: XML
12. Persistent Objects: Serialization and Relational Databases
13. Objects and the Internet
14. Objects and Client/Server Aplications
15. Design Patterns



5. Class Design Guidelines

Modeling Real World Systems

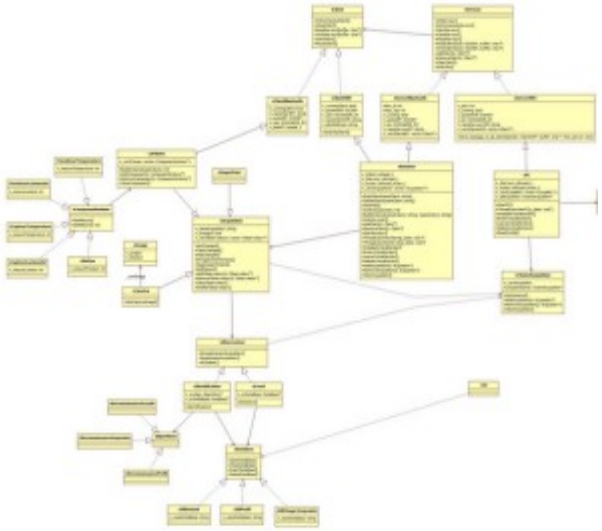
►One of the primary goals of object-oriented (OO) programming is to model real-world systems in ways similar to the ways in wich people actually think.



5. Class Design Guidelines

Modeling Real World Systems

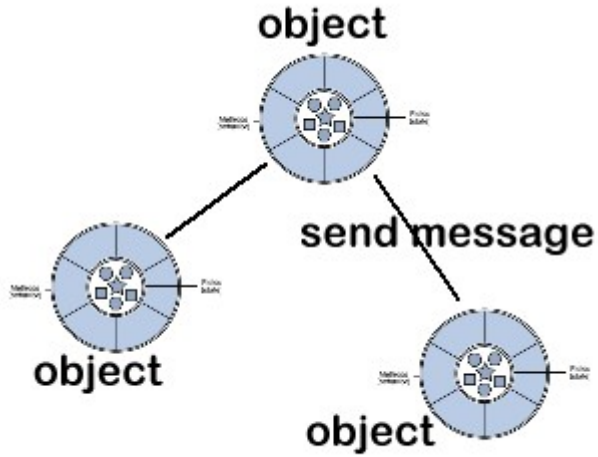
►Designing classes is the object-oriented way to create these models.



5. Class Design Guidelines

Modeling Real World Systems

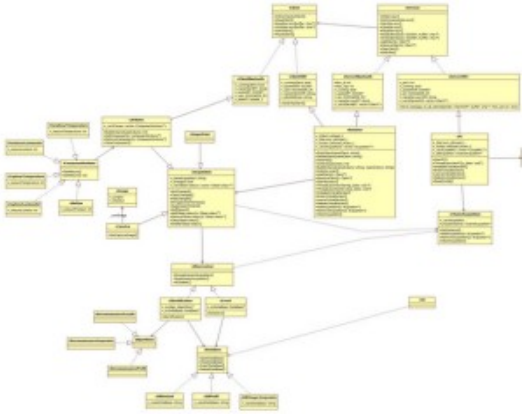
►Rather than using a structured approach where data and behavior are logically separate entities, the OO aproach encapsulates the data and behavior into objects that interact



5. Class Design Guidelines

Modeling Real World Systems

►Classes literally model the real-world objects and how these objects interact with other real-world objects. These interactions occur in a way similar to the interactions between real-world objects, such as people.

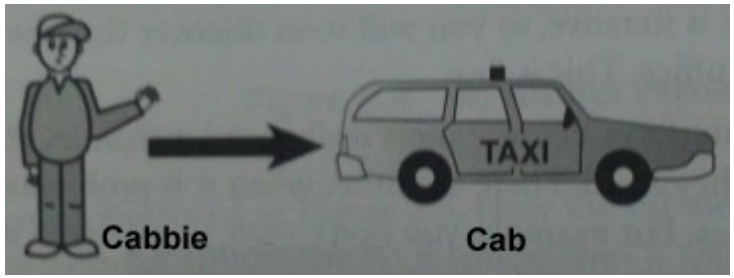


►When creating classes, you should design them in a way that represents the true behavior of the object.

5. Class Design Guidelines

Modeling Real World Systems

Let's use the cabbie example from previous chapters. The *Cab* Class and the *Cabbie* class model a real-world entity.



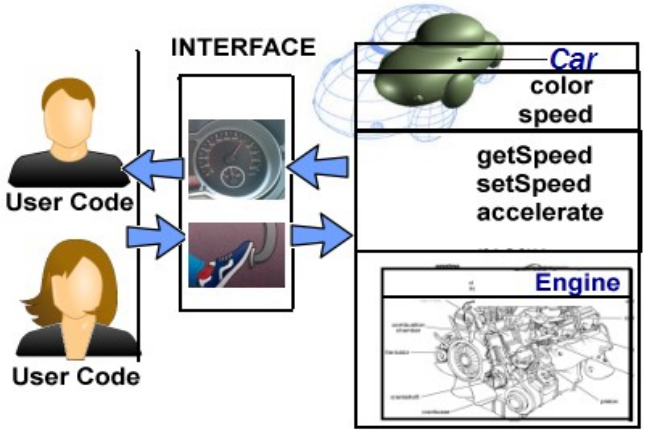
The *cab* and the *cabbie* objects encapsulate their data and behavior, and they interact through each other's public interface.

5. Class Design Guidelines

The Minimum Public Interface

►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.

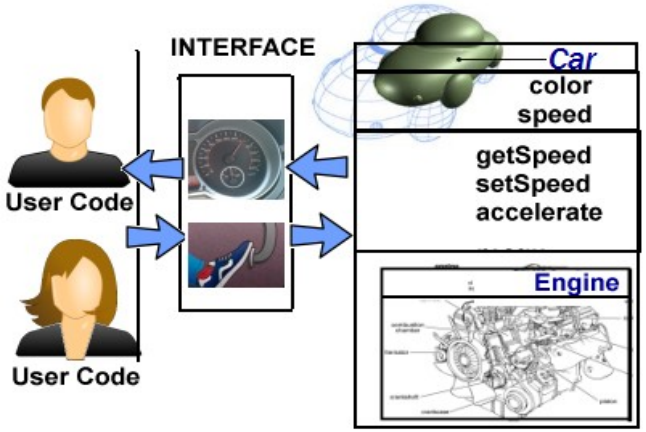


5. Class Design Guidelines

The Minimum Public Interface

►If the public interface is incomplete (missing behavior) the user will not be able to do the complete job.

►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.

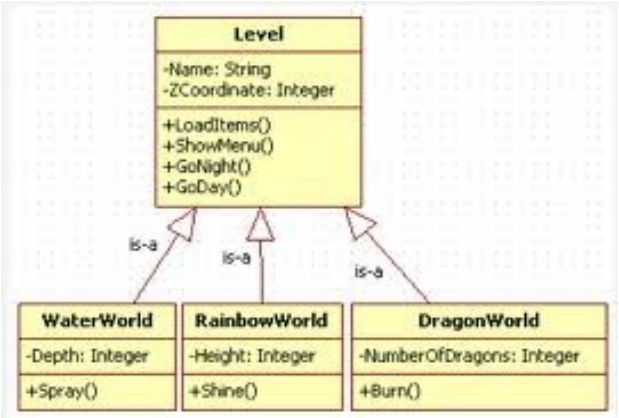


5. Class Design Guidelines

Extending the interface

Even if the public interface of a class is insufficient for a certain application, object technology easily allows the capability to extend and adapt this interface by means of inheritance.

If designed with inheritance in mind, a new class can inherit from an existing class and create a new class with an extended interface.

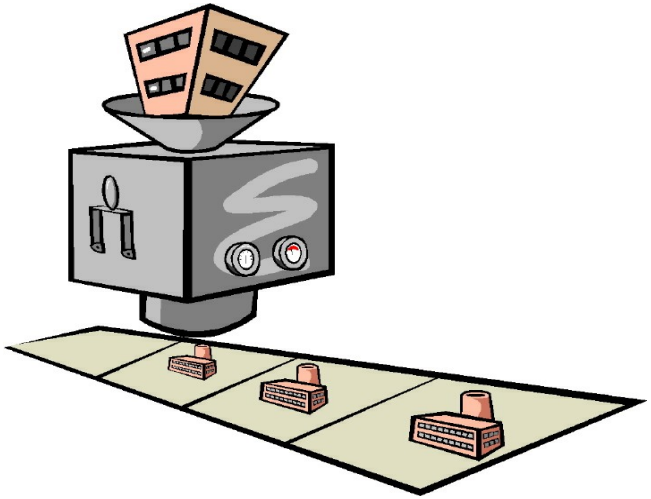


5. Class Design Guidelines

Designing Robust Constructors (and destructors)

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

(attribute initialization and memory management)



5. Class Design Guidelines

Designing Robust Constructors (and destructors)

►In languages that includes destructors, it is of vital importance that the destructors include proper clean-up functions.

Java and .Net reclaim memory automatically via a garbage collection mechanism.

In languages such as C++, the developer must include code in the destructor to properly free up the memory that the object acquired its in existence.

►If this function is ignored, a memory leak will result.

5. Class Design Guidelines

Designing Error Handling into a Class

►The developer anticipates potential errors and includes code to handle these conditions when they are encountered.

►When an error is encountered, the system should either fix itself and continue, or exit gracefully without losing any data important to the user.

5. Class Design Guidelines

Designing with Reuse in Mind

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

For example, when a *Cabbie* class is developed and tested, it can be used anywhere you need a cabbie.

►To make a class usable in various systems, the class must be designed with reuse in mind.

5. Class Design Guidelines

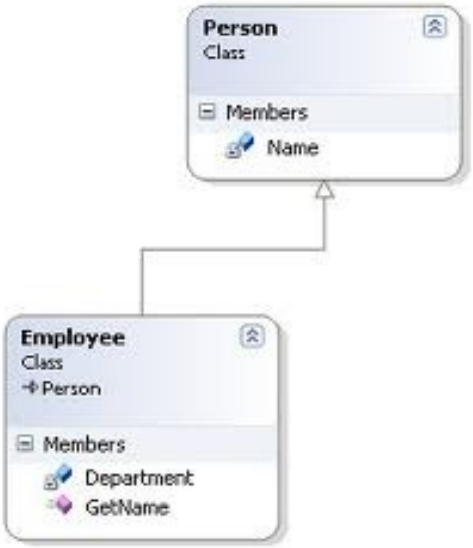
Designing with Extensibility in Mind

►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. This where inheritance comes into play.

5. Class Design Guidelines

Designing with Extensibility in Mind

►If you have just written a *Person* class, you must consider the fact that you might later want to write an *Employee* class. Having Employee inherit from Person might be the best strategy. In this case, the *Person* class is said **Extensible.**



5. Class Design Guidelines

Designing with Maintainability in Mind

►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.

5. Class Design Guidelines

Designing with Maintainability in Mind

►The process of designing classes forces you to organize your code into many manageable pieces. Seperate pieces of code tend to be more maintenable than larger pieces of code.

5. Class Design Guidelines

Testing the interface

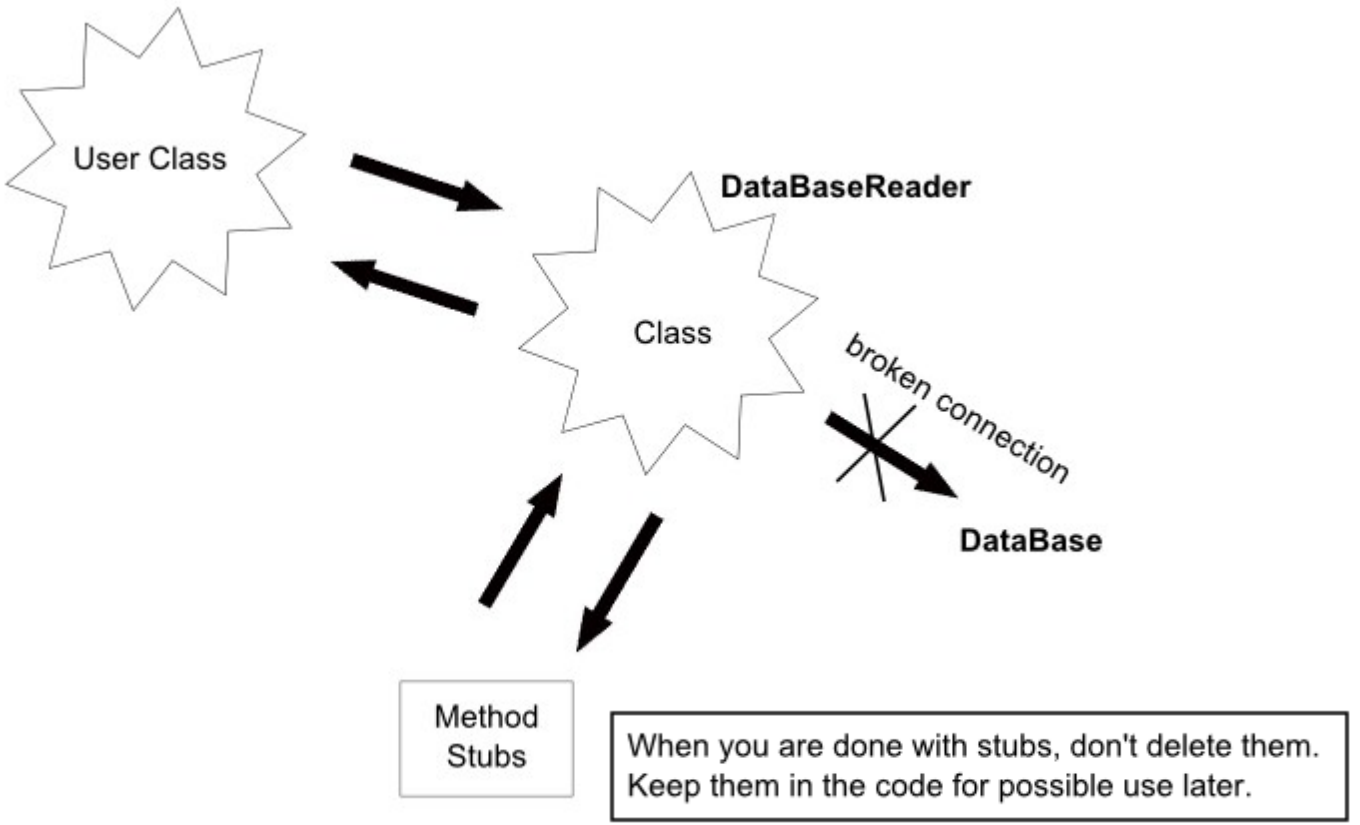
►The minimal implementations of the interface are often call *stubs.* By using stubs, you can test the interface without writing any *real* code.

►As you find problems with the interface design, make changes and repeat the process until you are satisfied with the result.

5. Class Design Guidelines

Testing the interface

In the following example, rather than connect to an actual database, stubs are used to verify that the interfaces are working properly.



1. Introduction to Object-Oriented Concepts
2. How to think in Terms of Objects
3. Advanced Object-Oriented Concepts
4. The anatomy of a Class
5. Class Design guidelines

6. Designing with objects

7. Mastering Inheritance and Composition
8. Frameworks and Reuse: Designing with interfaces and Abstract classes
9. Building objects
10. Creating Object Models with UML
11. Objects and Portable Data: XML
12. Persistent Objects: Serialization and Relational Databases
13. Objects and the Internet
14. Objects and Client/Server Aplications
15. Design Patterns