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

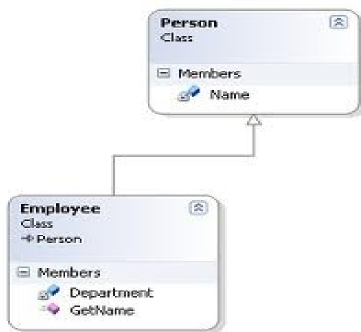
7. Mastering Inheritance and Composition

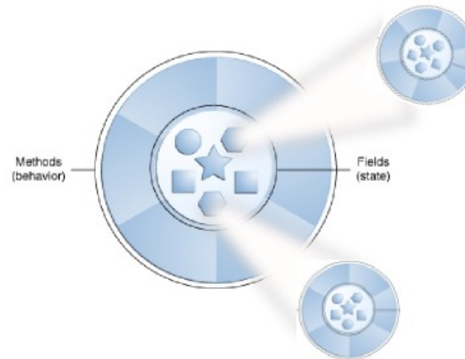
8. Frameworks and Reuse: Designing with interfaces and Abstract classes
9. Building objects
10. Creating Object Models with UML
11. Objects and Portable Data: XML
12. Persistent Objects: Serialization and Relational Databases
13. Objects and the Internet
14. Objects and Client/Server Aplications
15.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

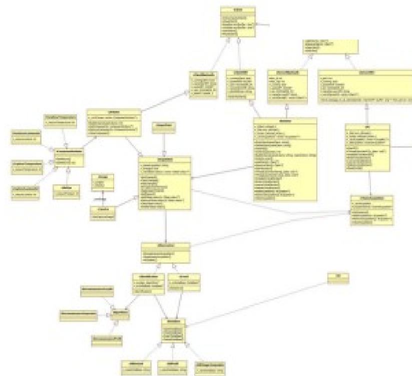
7. Mastering inheritance and composition

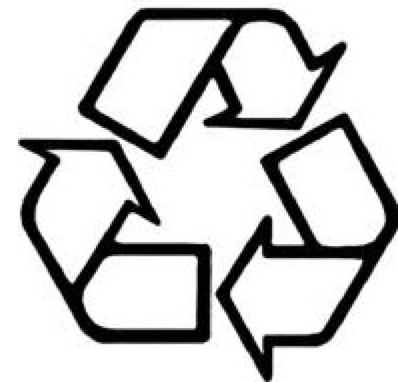
Reusing objects

Both inheritance and composition are mechanisms for reuse.





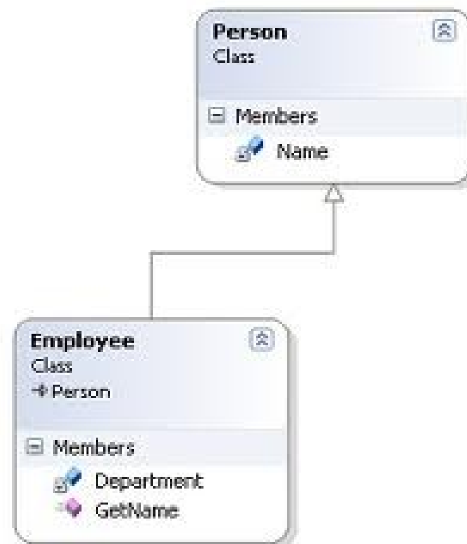




7. Mastering inheritance and composition

Reusing objects

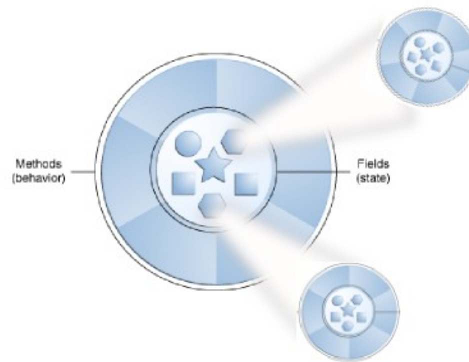
Inheritance, as its name implies, involves inheriting attributes and behaviors form other classes. In this case, there is a true parent/child relationship. The child (or subclass) inherits directly from the parent (superclass). Inheritance reprensents a « is-a » relationship.



7. Mastering inheritance and composition

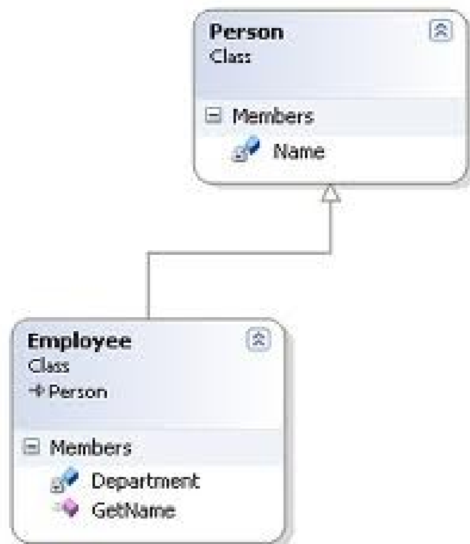
Reusing objects

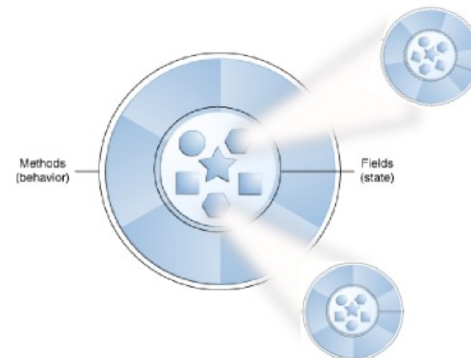
Composition, also as its name implies, involved building objects by using other objects, it's a sort of assembly. Composition represents a « has-a » relationship. For example a car has an engine.



Reusing objects

Inheritance or Composition?

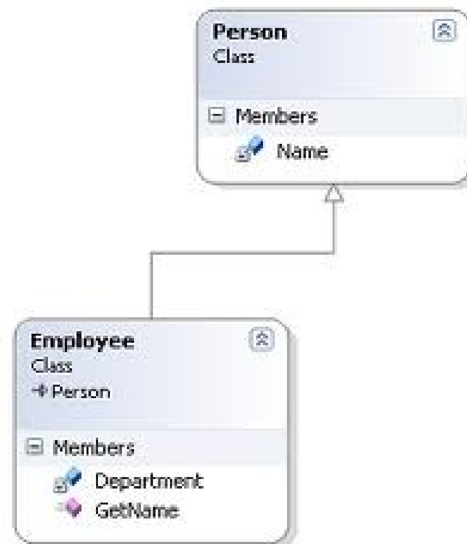




7. Mastering inheritance and composition

Inheritance

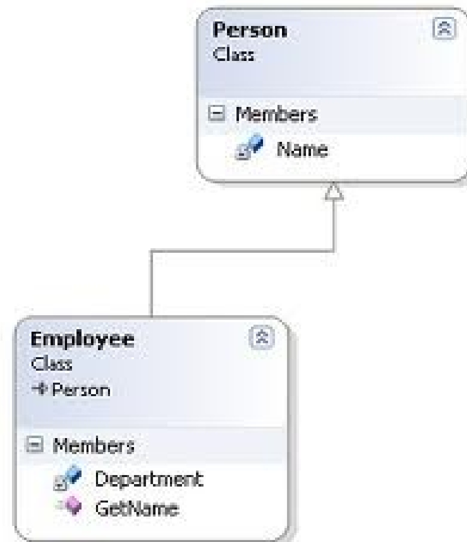
Child classes inherit attributes and behaviors from a parent class.
« If you can say Class B <u>is a</u> Class A, then this relationship is a good candidate for inheritance.



7. Mastering inheritance and composition

Inheritance, generalization and specialization

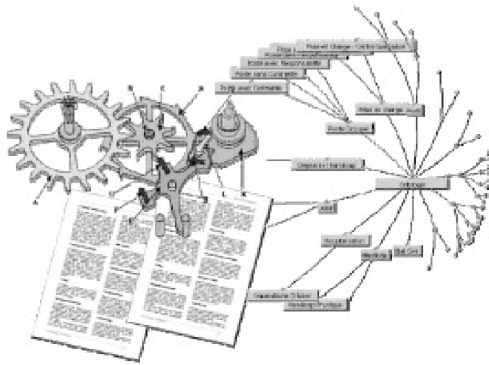
The idea is that as you make your way down the inheritance tree, things get more specific. The most general case is at the top of the tree.

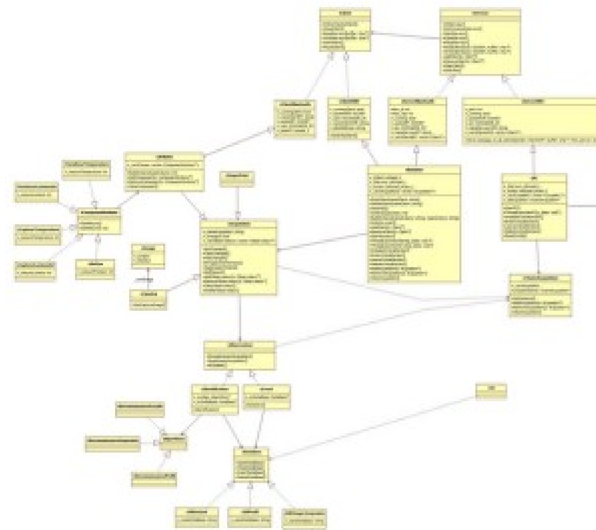


7. Mastering inheritance and composition

Model complexity

In larger systems, keepings as simple as possible is usually the best practice.

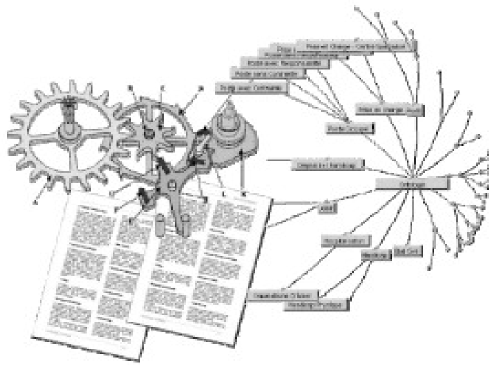




7. Mastering inheritance and composition

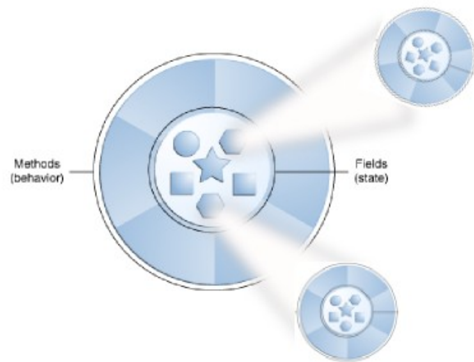
Making decision with the future in mind

You might as this point say, « Never say never ». If you do not design for the possibility now, it will be much more expensive to change the system later.



Composition

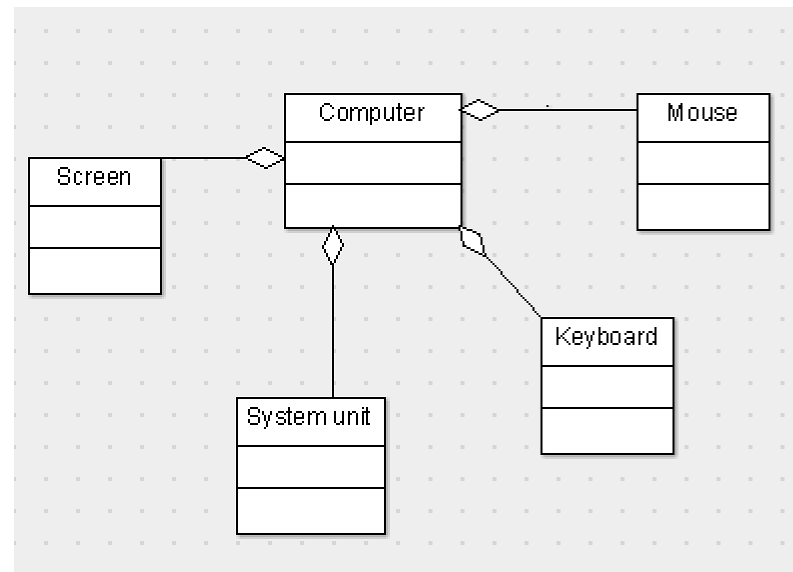
It is natural to think of objects as containing other objects. A computer contains video card, keyboard and drives. Each object can be a standalone object. In fact, you could take the hard drive to another computer and install it. The new object is known as a *compound*, an *aggregate*, or a *composite object*.



7. Mastering inheritance and composition

Representing Composition with UML

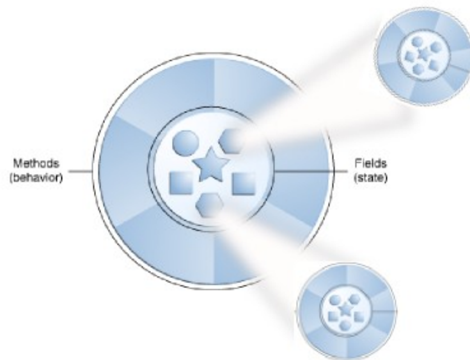
To model the fact that a computer contains a screen, a keyboard, a mouse, a system unit, UML uses the notation shown here:



7. Mastering inheritance and composition

Composition

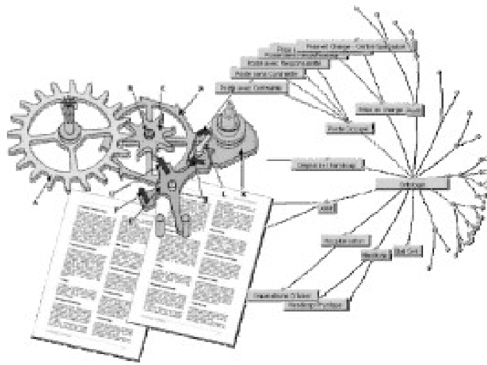
```
public class Cabbie{
  //Place name of Company here
  private static String companyName="Blue Cab Company";
  //Name of the Cabbie
  private String name;
  //Car assigned to Cabbie;
  private Cab myCab;
```

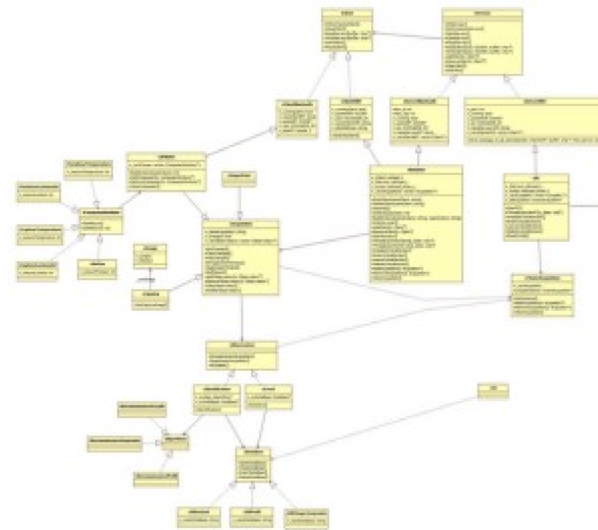


7. Mastering inheritance and composition

Model complexity

There is a fine line between creating an object model that contains enough granularity to be sufficiently expressive and a model that is so granular that is difficult to understand and maintain.

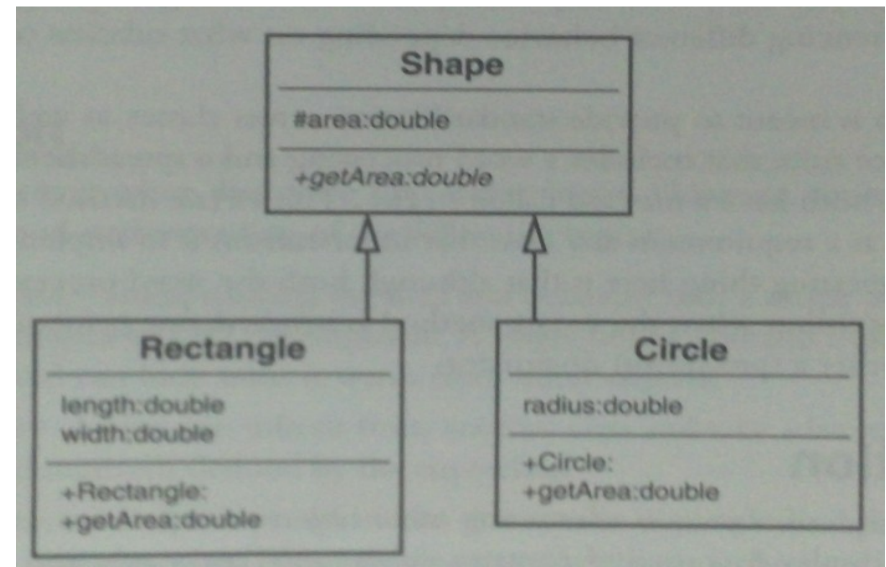




7. Mastering inheritance and composition

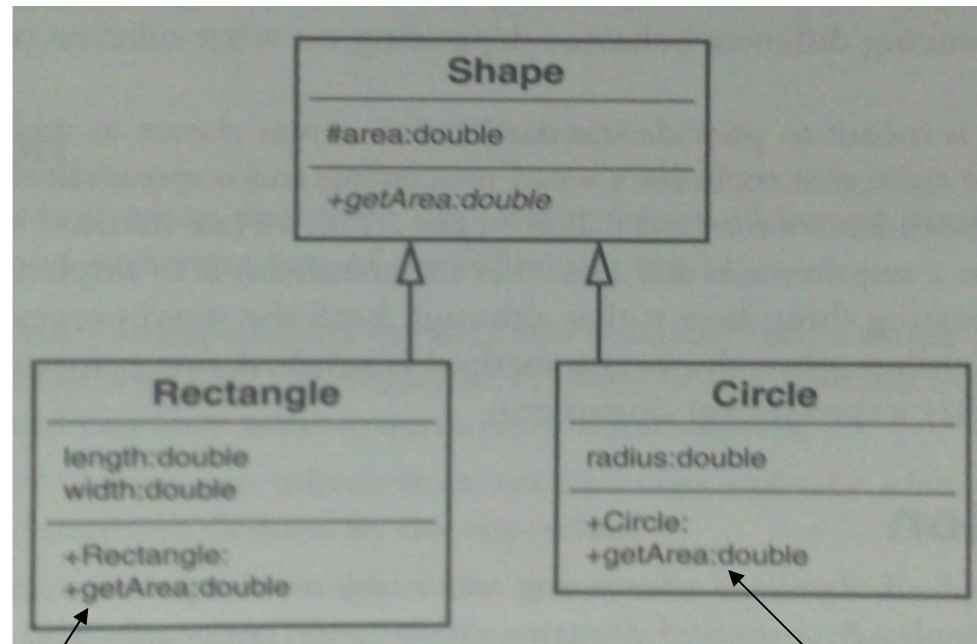
Polymorphism

In a well designed system, an object should be able to answer all the important questions about it. As a rule, an object should be responsible for itself. This independence is one of the primary mechanism of code reuse.



7. Mastering inheritance and composition

Polymorphism means that similar objects can respond to the same message in different ways.




```
public double getArea(){
  area=3.14*(radius*radius)
  return (area);
}
```

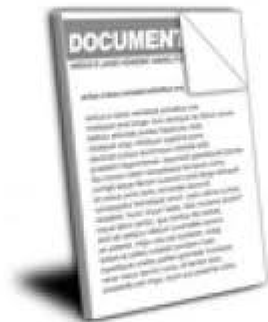
```
public double getArea() {
  area=length*width;
  return (area);
}
```

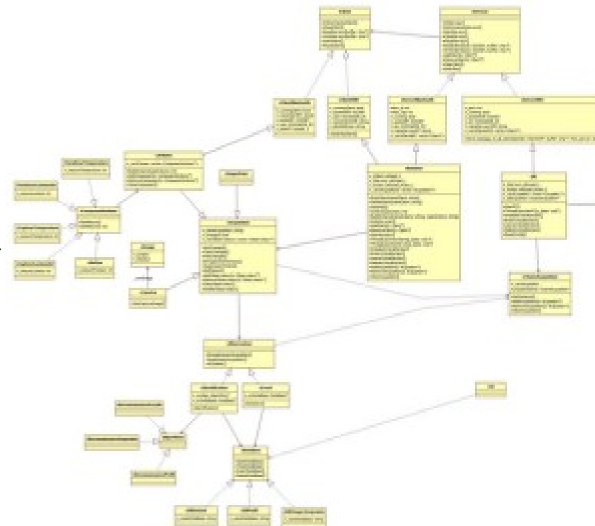
7. Mastering inheritance and composition

Example: zookeeper

Statement of work(SOW) / request-for-proposal (RFP) 需求方案说明书

The SOW contains everything that must be known about the system. Many customers create a request-for-proposal (RFP) for distribution, which is similar to the SOW. A customer creates a RFP that completely describes the system they want built.





7. Mastering inheritance and composition

Example: zookeeper 需求方案说明书：动物园管理员

The client hold a zoo and needs a system to manage all animals in the zoo. The zoo contains tigers, pandas, monkeys, birds, giraffes. All are living in zoo cages. Each animal of zoo has a unique name. The client needs to know for each animal:

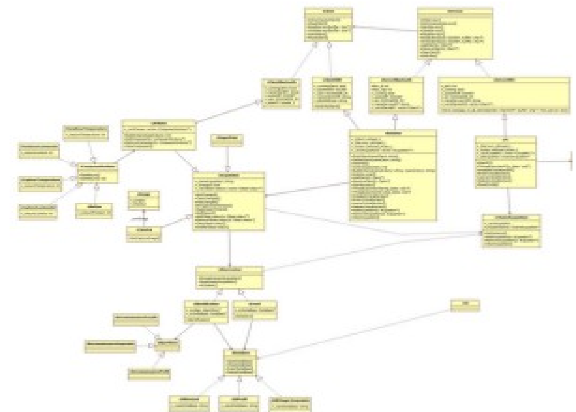
-The date of the last medical visit (就诊).

-If the animal got a meal for the current day.

-Which veterinary(兽医) is in charge of the animal.

He needs to know which guardian(监护人) is in charge of the cage and the number of animals for each cage(笼).

He needs also to know the number of visitors of the zoo.



7. Mastering inheritance and composition

Example: zookeeper

Zoo
NumberOfVisitors : Integer

Person

Visitors

Employee

Guardian

Veternary

Cage
numberOfAnimals Integer

AnimalInCaptivity
name : String
gotMealToday : Boolean
DateOfLastMedicalVisit : String

Animal

Panda

Bird

Dog

Giraffe

Monkey

Tiger

7. Mastering inheritance and composition

Example: zookeeper

Zoo
NumberOfVisitors : Integer

Person

Animal

Cage
numberOfAnimals : Integer

Panda

Bird

Dog

Visitors

Employee

Guardian

AnimalInCaptivity
name : String
gotMealToday : Boolean
DateOfLastMedicalVisit : String

Giraffe

Monkey

Tiger

Veternary

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