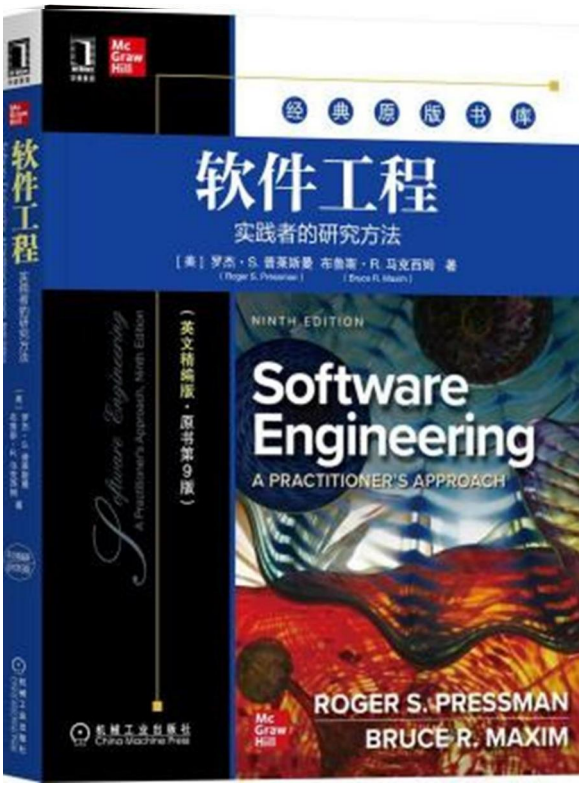
UNIT 1

软件 & 软件工程

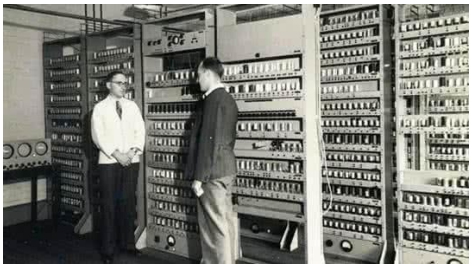
Software & Software Engineering

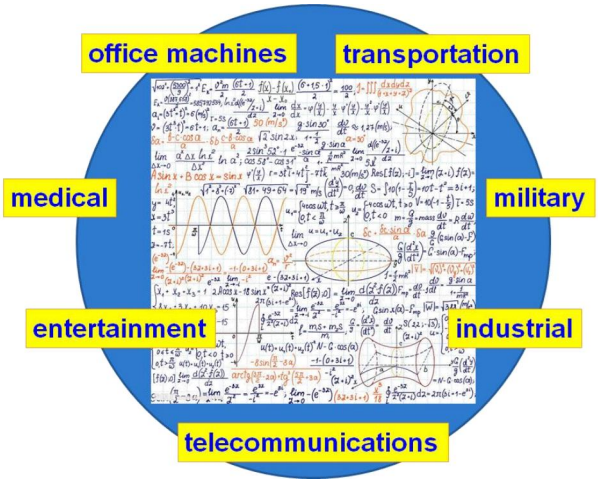
王传栋



the law of unintended consequences

- Software is ubiquitous and exists in various fields such as business, culture, and social lives.





Nature of Software

How does software differ from the artifacts produced by other engineering disciplines ?

- Software is both a product and a vehicle for delivering a product.
- As a product, software is an information transformer.
- As a vehicle for delivering a product, software serves as a basis for computer control, communication, and creation of other programs.

Software Application Domains

- System software.
- Application software.
- Engineering/Scientific software.
- Embedded software.
- Product-line software.
- Web/Mobile applications.
- AI software (robotics, neural nets, game playing).

What is Software?

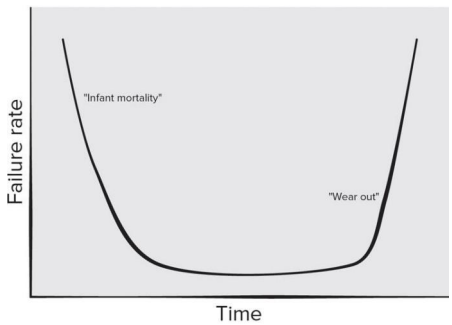
Software is :

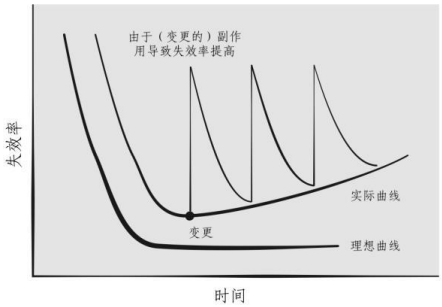
1) Instructions (computer programs) that when executed provide desired features, function, and performance;
2) Data structures that enable the programs to adequately manipulate information.
3) Documentation that describes the operation and use of the programs.

Software & Hardware

How do software characteristics differ from hardware characteristics ?

- Software is developed or engineered it is not manufactured in the classical sense.
- Software doesn't "wear out" but is does deteriorate.
- Although the industry is moving toward component-based construction, most software continues to be custom-built.





- **理想曲线**为双曲线，失效率随时间 x 增加，y 值降低接近于恒定值
- **实际曲线**，失效率降低并未达到最小值，随时间增加斜率减小并再次升高
- **实施变更时**，曲线突然上升，造成更高失效率，随后随时间增加，失效率回落到实际曲线
- 最后，因副作用致失效率增加

Legacy Software

The **proliferation** of such systems is causing … costly to maintain and risky to evolve.

- **longevity and business criticality & poor quality**

Why must change ?（What is wrong with the notion that software does not need to evolve over time ?）

- The software must be adapted to meet the needs of new computing environments or technology.
- The software must be enhanced to implement new business requirements.
- The software must be extended to make it work with other more modern systems or databases.
- The software must be re-architected to make it viable within an evolving computing environment.

How it All Starts

What are the reasons for developing the software product?

Every project is precipitated by some business need ——

- the need to correct a defect in an existing application;
- the need to adapt a “legacy system” to a changing business environment;
- the need to extend the functions and features of an existing application; or
- the need to create a new product, service, or system.

software crisis

A series of serious problems encountered during software development and maintenance.

- Why does it take so long to get software finished?
- Why are development costs so high?
- Why can't we find all errors before we give the software to our customers?
- Why do we spend so much time and effort maintaining existing programs?
- Why do we continue to have difficulty in measuring progress as software is being developed and maintained?

How to solve it ?

- Technology + Management ⟹ Software Engineering

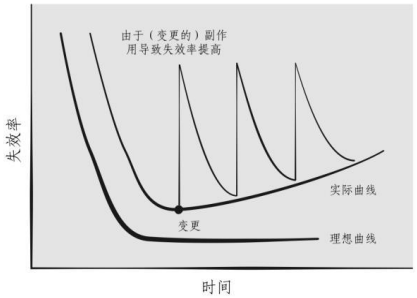
Software Engineering

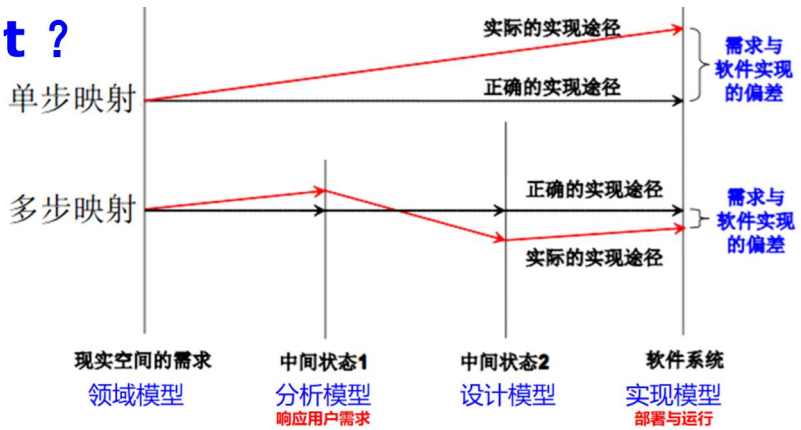
The IEEE definition — Software Engineering :

① The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

② The study of approaches as in ①.

How to understand it ?

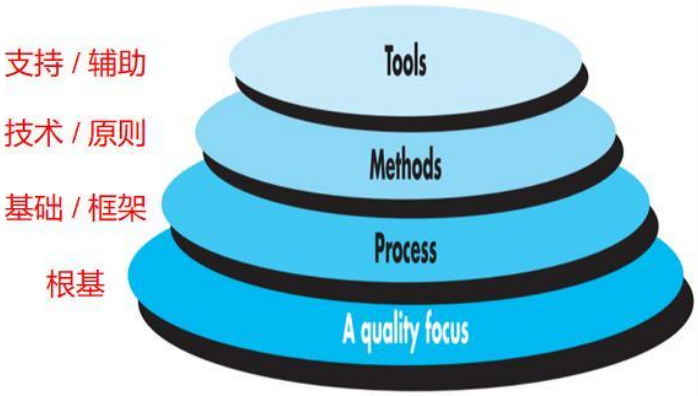


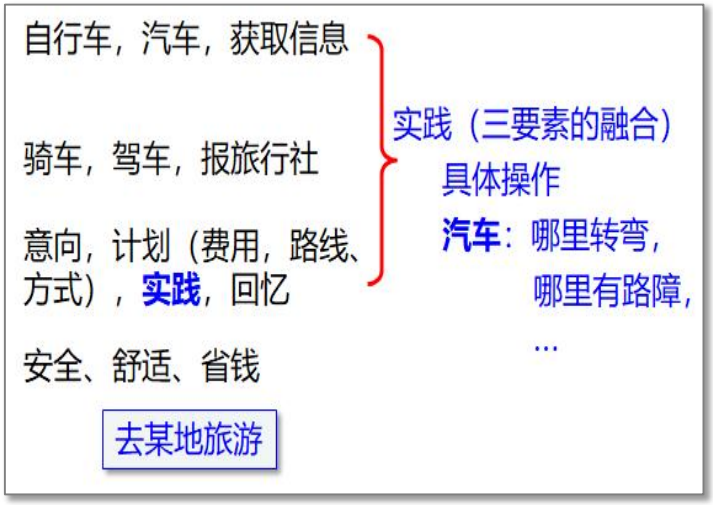


SE is a layered technology

The **bedrock** that supports **SE** is a **quality focus**.

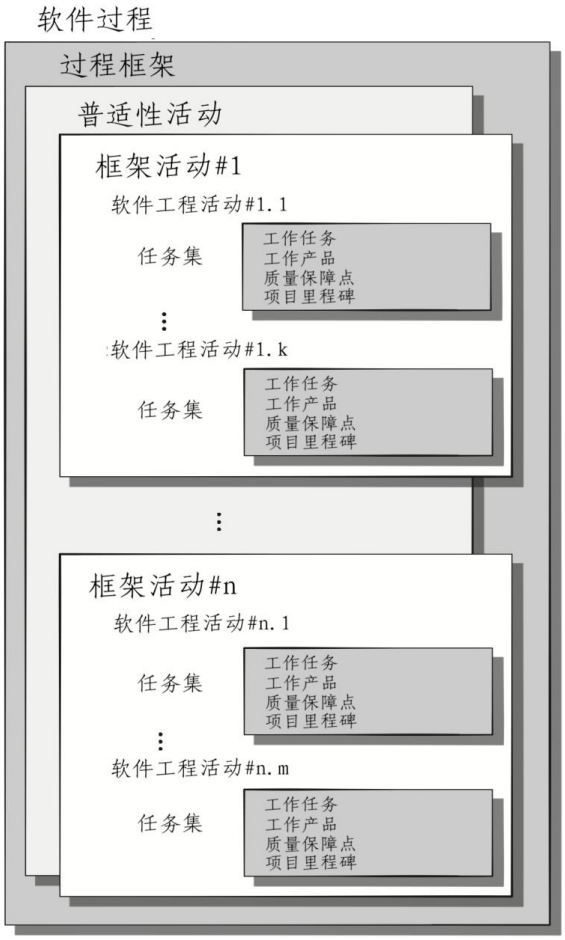
- **process** ： framework & basis, to hold the technology
- **methods** ： provide the technical (principles) to build software
- **tools** ： support (aided) for the process and the methods



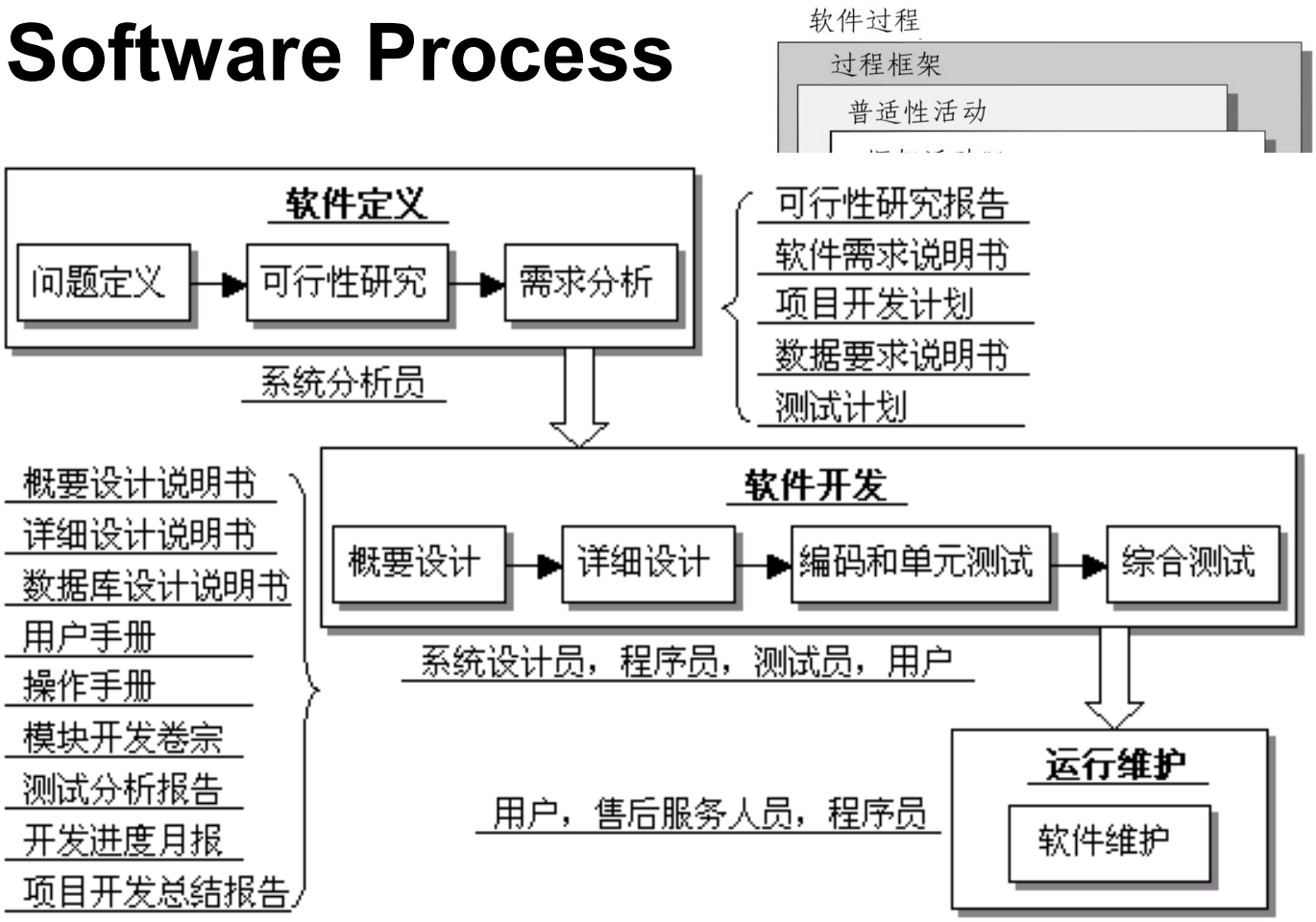


Software Process

- A **process framework** includes five **framework activities** and a set of **umbrella activities**
- An **activity** achieve a broad objective and contains multiple **actions**， each **action** is defined as one **task set.**
- each **task set** encompasses a series of “the work tasks, the work products, the quality assurance points and the milestones



Software Process



Framework Activities

Communication.

Planning.

Modeling.

- Analysis of requirements.
- Design.

Construction:

- Code generation.
- Testing.

Deployment.

不同案例，过程细节差别很大，但框架活动是一致的

- ✧ 无论是简单小程序，还是 WebApp 及大型复杂系统工程
- ✧ 项目开展中，框架活动可以多次迭代应用，5个框架活动不断重复
- ✧ 每次迭代产生一个实现部分特性和功能的增量（software increment）
- ✧ 随着每一次增量的产生，软件逐渐趋于完善

Umbrella Activities

- Software project tracking and control.
- Risk management.
- Software quality assurance.
- Technical reviews.
- Measurement.
- Software configuration management.
- Reusability management.
- Work product preparation and production.

Process Adaptation

What are the possible differences in process models for different projects? (What factors need to be considered when selecting a process model for a project?)

- Overall flow of activities, actions, and tasks and the interdependencies among them.
- Degree to which actions and tasks are defined within each framework activity.
- Degree to which work products are identified and required.
- Manner which quality assurance activities are applied.
- Manner in which project tracking and control activities are applied.
- Overall degree of detail and rigor with which the process is described.
- Degree to which the customer and other stakeholders are involved with the project.
- Level of autonomy given to the software team.
- Degree to which team organization and roles are prescribed.

Essence of SE Practice

How to integrate the engineering project practice into the process framework? (How Polya's problem solving principles describe the essence of engineering practice?)

The essence of the problem: How to combine the **process** with the **product** in a specific project ?

George Polya 《 How to Solve it 》 suggests:

1. *Understand the problem* (communication and analysis).
2. *Plan a solution* (modeling and software design).
3. *Carry out the plan* (code generation).
4. *Examine result for accuracy* (testing & quality assurance).

Essence of SE Practice

项目实践的常识性步骤，引发一系列问题

实践精髓		
1	理解问题 沟通和分析	✧ 谁将从问题解决中获益？即：谁是利益相关者 ✧ 哪些是未知的？哪些数据、功能和特性是解决问题所必须的？ ✧ 问题可以划分吗？是否可以描述为更小、更易理解的问题？ ✧ 问题可以图形化描述吗？可以建立分析模型吗？
2	策划解决方案 建模和软件设计	✧ 以前曾见过类似的问题吗？在可能的解决方案中，是否可以识别出一些模式？是否有软件已经实现了所需要的数据、功能和特性 ✧ 类似的问题是否已解决过？如果是，解决方案所包含的元素是否可以复用？ ✧ 可以定义子问题吗？如果可以，子问题是否有解决方案？ ✧ 能用一种可以很快实现的方式来描述解决方案吗？能构建出设计模型吗？
3	实施计划 代码生成	✧ 解决方案和计划一致吗？源码是否可追溯到设计模型 ✧ 解决方案的每个组成部分是否可以证明正确？设计和代码是否经过评审，算法是否经过正确性证明？
4	检查结果 测试和质量保证	✧ 能否测试解决方案的每个部分？是否实现了合理的测试策略？ ✧ 解决方案是否产生了与所要求的数据、功能和特性一致的结果？是否按照项目利益相关者的需求进行了确认？

General Principles

The dictionary defines the word principle as

- "an important underlying law or assumption required in a system of thought."

Regardless of their level of focus, principles help you establish a mind-set for solid SE practice.

The principles at different levels of abstraction have different focuses:

1. *Some focus on SE as a **whole**,*
2. *others consider a specific generic framework **activity** (e.g., communication), and*
3. *still others focus on SE **actions** (e.g., architectural design) or technical **tasks** (e.g., creating a usage scenario).*

Hooker's General Principles

1. The Reason It All Exists – *provide value to users*.
2. KISS (Keep It Simple, Stupid!) – *design simple as it can be*.
3. Maintain the Vision – *clear vision is essential*.
4. What You Produce, Others Will Consume.
5. **Be Open to the Future** - *do not design yourself into a corner*.
6. **Plan Ahead for Reuse** – *reduces cost and increases value*.
7. Think! – *placing thought before action produce results*.

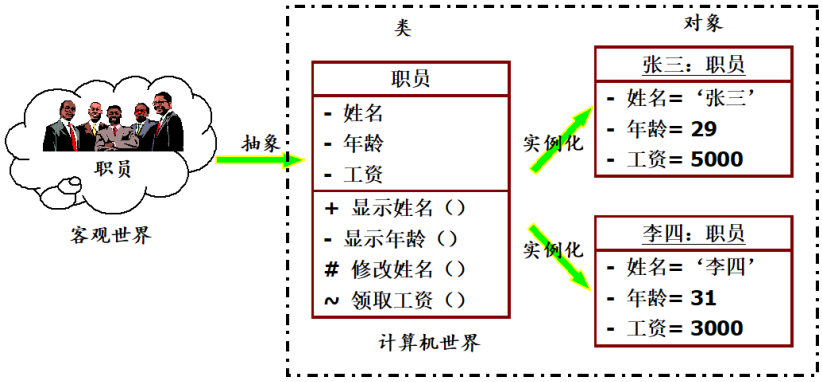
How to
"Be Open to the Future"
and
"Plan Ahead for Reuse" ?

OO 思想

- 类（具体类与抽象类）、接口、对象、实例
- 继承、实现、类等级
- 封装、信息隐藏
- 消息、协议、服务
- 重载、多态

类、对象、实例

类：抽象定义一组具有**相同数据**和**相同操作**的相似对象




```
class staff {
    name
    age
    salary
    public void staff(xm,nl,gz) {
        this.name = xm
        this.age = nl
        this.salary = gz
    }
    …… ……
}
```

对象：一个统称

```
staff zhangsan = new staff(张三,20,5000)
staff lisi = new staff(李四,31,3000)
```

实例：以某个特定的类为“样板”建立的一个具体对象

抽象类

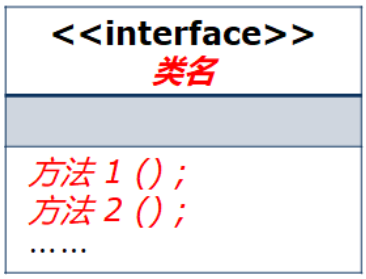
CombinatoricsKnife.class

```
package net.paoding.analysis.knife;

import java.util.HashSet;
import net.paoding.analysis.dictionary.Dictionary;
import net.paoding.analysis.dictionary.Hit;
import net.paoding.analysis.dictionary.Word;

public abstract class CombinatoricsKnife
  implements Knife, DictionariesWare
{
  protected Dictionary combinatoricsDictionary;
  protected HashSet noiseTable;

  public CombinatoricsKnife()
  {
```

- 成员变量可以是变量，也可以是常量
- 有构造方法，但是**不能实例化**（只能借助多态实现实例化），构造方法用于子类访问父类数据的初始化
- 通过抽象方法，限定子类必须完成某些动作，也可以有非抽象方法，提高代码的复用性
- 若类中有抽象方法，类必须定义为抽象类，抽象类可以没有抽象方法

接口

- 成员变量只能是常量，默认被static修饰
- 无构造方法和非抽象方法，通过定义抽象方法，实现具体类的行为约束
- 接口不能实例化，只能通过实现具体类的对象进行实例化（接口多态）



Knife.class

```
package net.paoding.analysis.knife;

public abstract interface Knife
{
  public static final int ASSIGNED = 1;
  public static final int POINT = 0;
  public static final int LIMIT = -1;

  public abstract int assignable(Beef paramBeef, int paramInt1, int paramInt2);

  public abstract int dissect(Collector paramCollector, Beef paramBeef, int paramInt);
}
```

继承 & 实现 & 类等级

- 子类继承抽象类，需重写父类的抽象方法

  不重写，子类也是抽象类

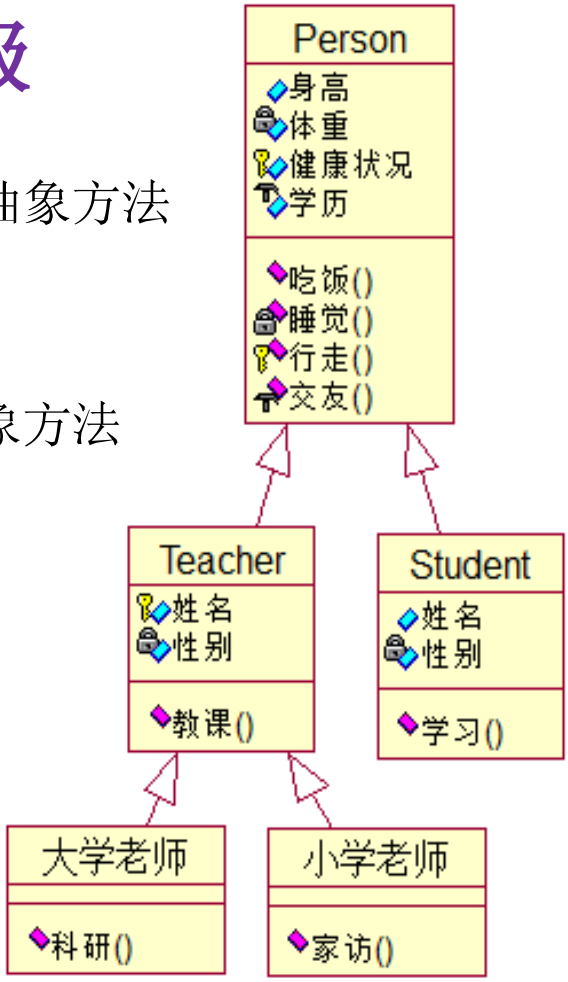
  父类的非抽象方法也可以重写
- 实现类，要重写接口中的所有抽象方法

  否则，实现类就是抽象类
- Teacher 类的对象有哪些属性 ？

  哪些操作 ？
- 大学老师类的对象有哪些属性 ？

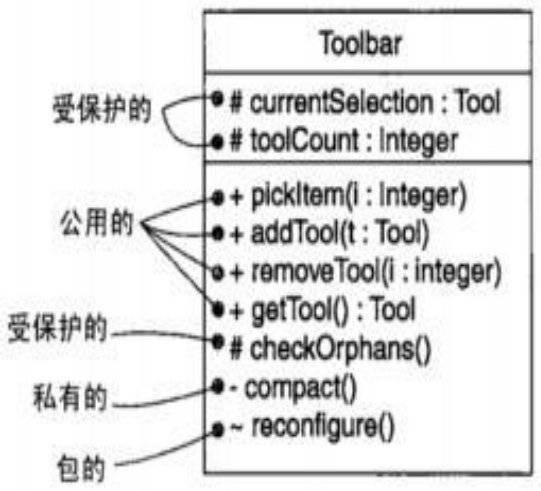
  哪些操作 ？
- 小学老师类的对象有哪些属性 ？

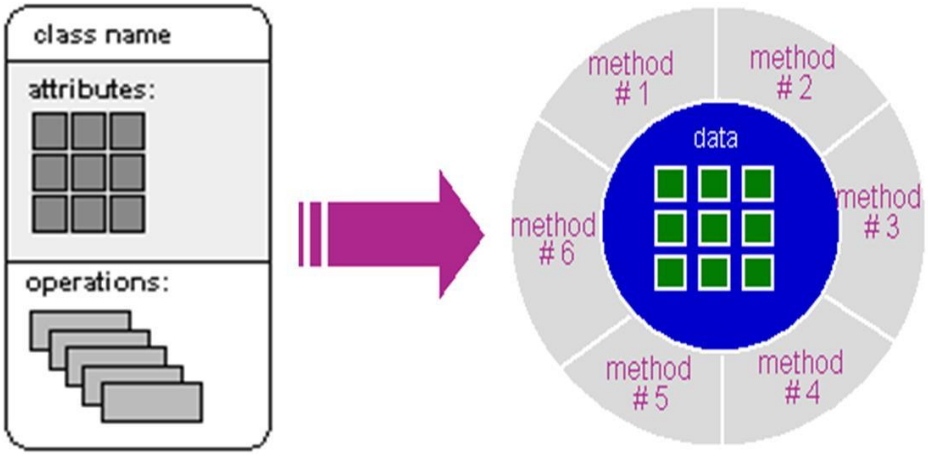
  哪些操作 ？



封装 & 信息隐藏

		同 包		不同包	
修饰符	同 类	子 类	非子类	子 类	非子类
public	Yes	Yes	Yes	Yes	Yes
private	Yes	No	No	No	No
protected	Yes	Yes	Yes	Yes	No
默认	Yes	Yes	Yes	No	No





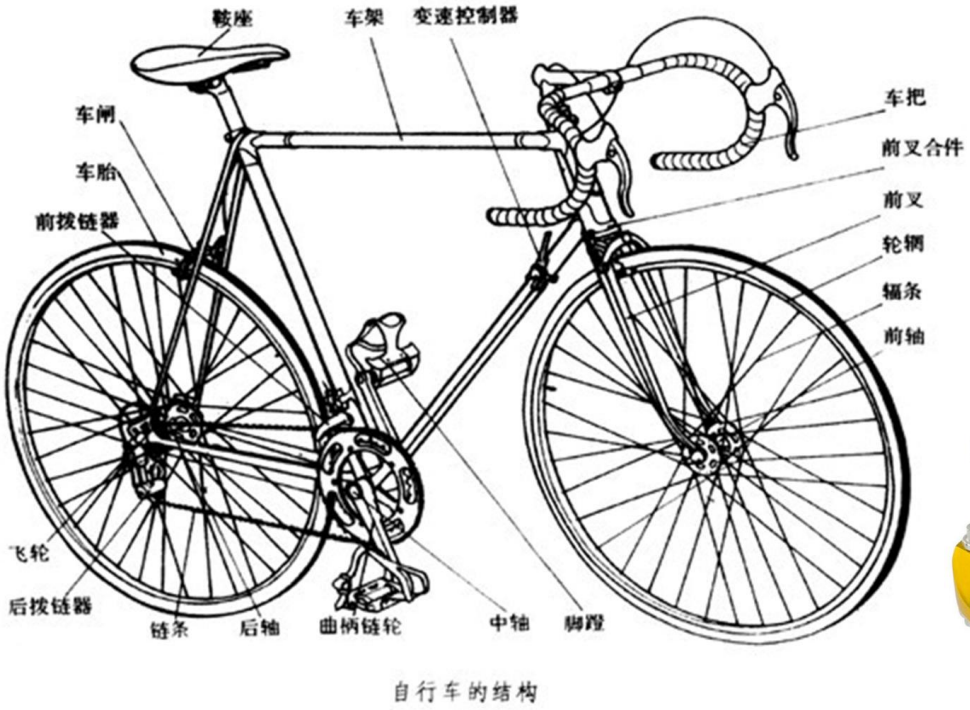
封装 & 信息隐藏

可见性	uml 符号	java 关键字	描述	推荐用法
公共	+	public	任何其它对象或类中的成员函数，都可以调用的公共成员函数	当定义该成员函数的**类层次结构之外**的对象和类必须访问它时
受保护	#	protected	可由其**定义类中**或其**任何子类中**的成员函数调用	当成员函数提供在**类层次结构内**，而非外部所需行为时
专用	-	private	仅可以由**与它在同一类中定义的其它成员函数**调用，而在子类中的成员函数不能调用	提供特定于某个类的行为 通常为了**封装一个特定行为**，实现再加工（也称重组）类中其它成员函数的行为结果时
缺省	~	空白	对**同一包中**的其它所有类实际上都是公共的，但对该包外的类是不可用的 有时，称为**包可见性**或**友好可见性**	一个有趣的功能，要小心使用 在构建实现诸如"客户"等具有凝聚力的商业概念的域组件、类集合时，用它将**访问权限定在**仅组件 / 软件包内的类

接口&协议

接口定义一组协议

《Interface》 螺纹
+ 顺时针拧紧 () + 逆时针放松 ()



自行车的结构


钛合金轴心

接口&协议

接口定义一组协议

Knife.class

```
package net.paoding.analysis.knife;

public abstract interface Knife
{
  public static final int ASSIGNED = 1;
  public static final int POINT = 0;
```

```
1, int paramInt2);

paramBeef, int paramInt);
```

FakeKnife.class

```
package net.paoding.analysis.knife;

import org.apache.commons.logging.Log;

public class FakeKnife
  implements Knife, DictionariesWare
{
  private Log log = LogFactory.getLog(
  private String name;
  private int intParam;
  private Inner inner = new Inner();

  public int assignable(Beef beef, int
    return -1;
  }

  public int dissect(Collector collect
    throw new Error("this knife doesn'
  }
```

```
class Client {
    int x, y ;
    ……
    FakeKnife fk = new FakeKnife() ;
    x = fk.assignable(…, …, …) ;
    ……
    y = fk.dissect(…, …, …) ;
    ……
}
```

消息、协议和服务

```
public class Driver {
    private String name;
    public Driver(String name) {
        super();
        this.name = name;}
    public String getName() {
        return name;    }
    public void setName(String name) {
        this.name = name; }
    public void drive(Car c) {
        c.go(new Address("东北"));   }
    public void drive(Car c, Address dest)
    {
        c.go(dest);
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Driver d = new Driver("老张");
        Car c = new Car();
        Address ad = new Address("北京");
        System.out.println(d.getName());
        d.drive(c, ad);
    }
}
```

协议的一部分

消息

服务

重载

```
public class Driver {
    private String name;
    public Driver(String name) {
        super();
        this.name = name;}
    public String getName() {
        return name;    }
    public void setName(String name) {
        this.name = name; }
    public void drive(Car c) {
        c.go(new Address("东北"));   }
    public void drive(Car c, Address dest)
    {
        c.go(dest);
    }
}
```

- 函数重载

  同一个作用域，若干参数特征不同的函数使用相同的函数名

- 运算符重载

  同一个运算符可以施加于不同类型的操作数

- 重载机制

  静态联编（static binding）

  在程序编译时，根据函数变元的个数和类型，决定使用同名函数的哪个实现代码

  提高OO程序灵活性和可读性

多态

- 通过重写（override）父类的同名操作

  使不同子类对象和父类对象接受同一消息，却提供不同服务

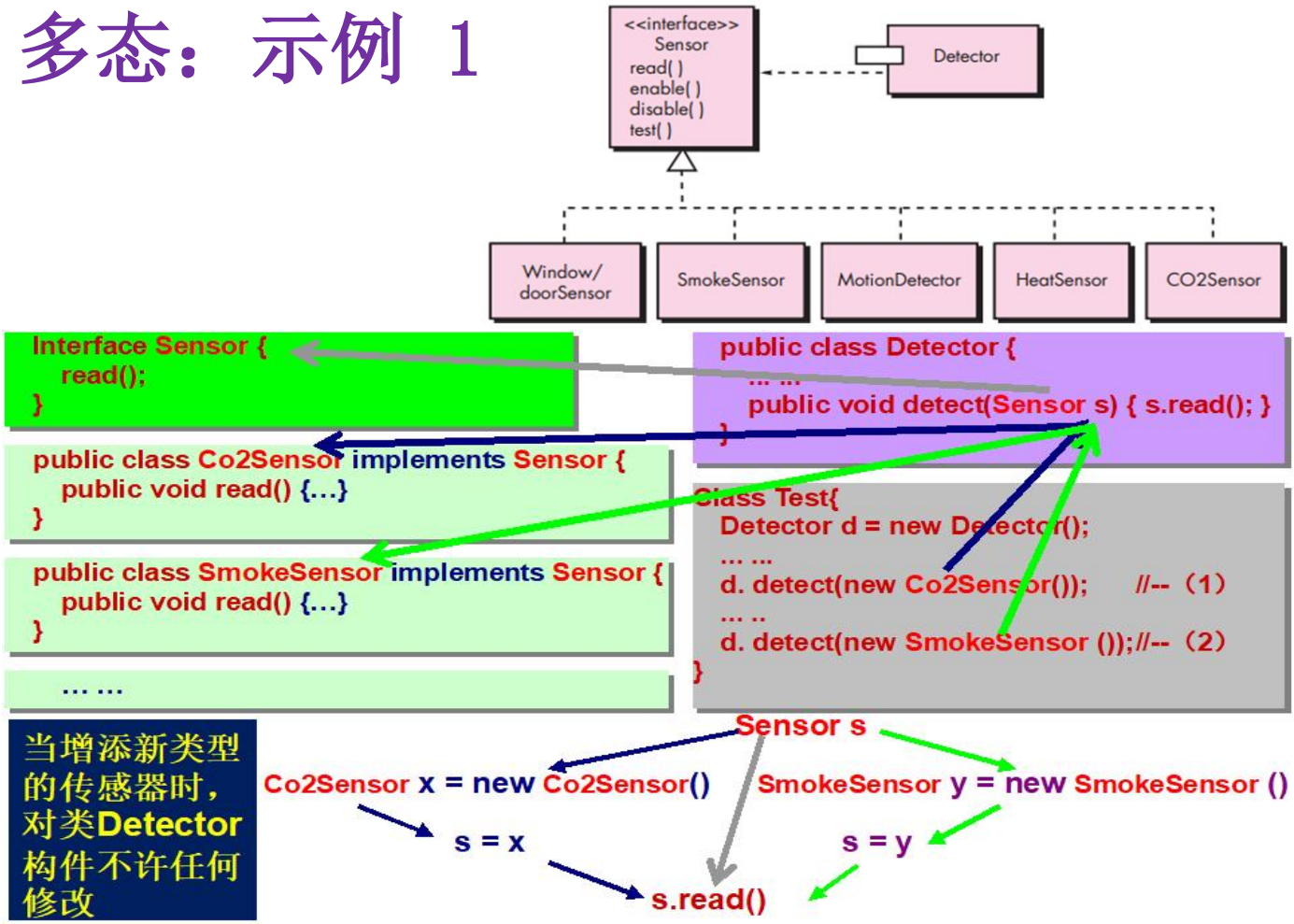
  不同层次的类共享一个行为（函数名、参数和返回值类型都相同），但行为实现却不同

- 机制：动态联编（dynamic binding）或滞后联编（late binging）
- 表现

	多态的 2 种 形式	
3个条件	继承	实现
	子类中重写父类的方法	具体类重写接口的方法
	父类引用指向子类对象	接口引用指向具体类对象

- 增强 OO 灵活性减少信息冗余，提高可重用性和可扩充性

多态：示例 1

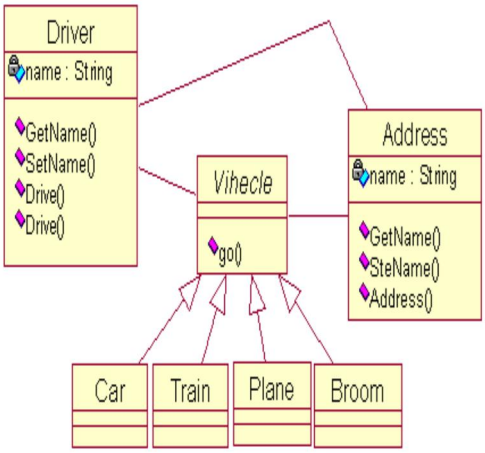


多态：示例 2

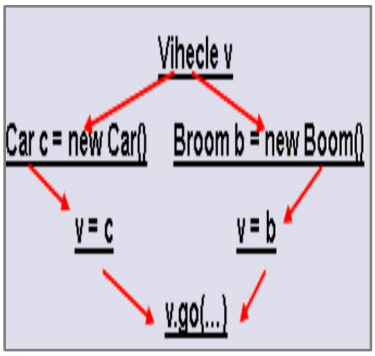
```
public class Driver {
    private String name;

    public String getName() {
        return name;  }

    public void setName(String name) {
        this.name = name; }

    public void drive(Vihecle v) {
        v.go(new Address("东北" )); }

    public void drive(Vihecle v, Address dest) {
        v.go(dest); }
}
```

```
public class Address {
    private String name;

    public String getName() {
        return name;  }

    public void setName(String name) {
        this.name = name;  }

    public Address(String name) {
        super();
        this.name = name; }
}
```

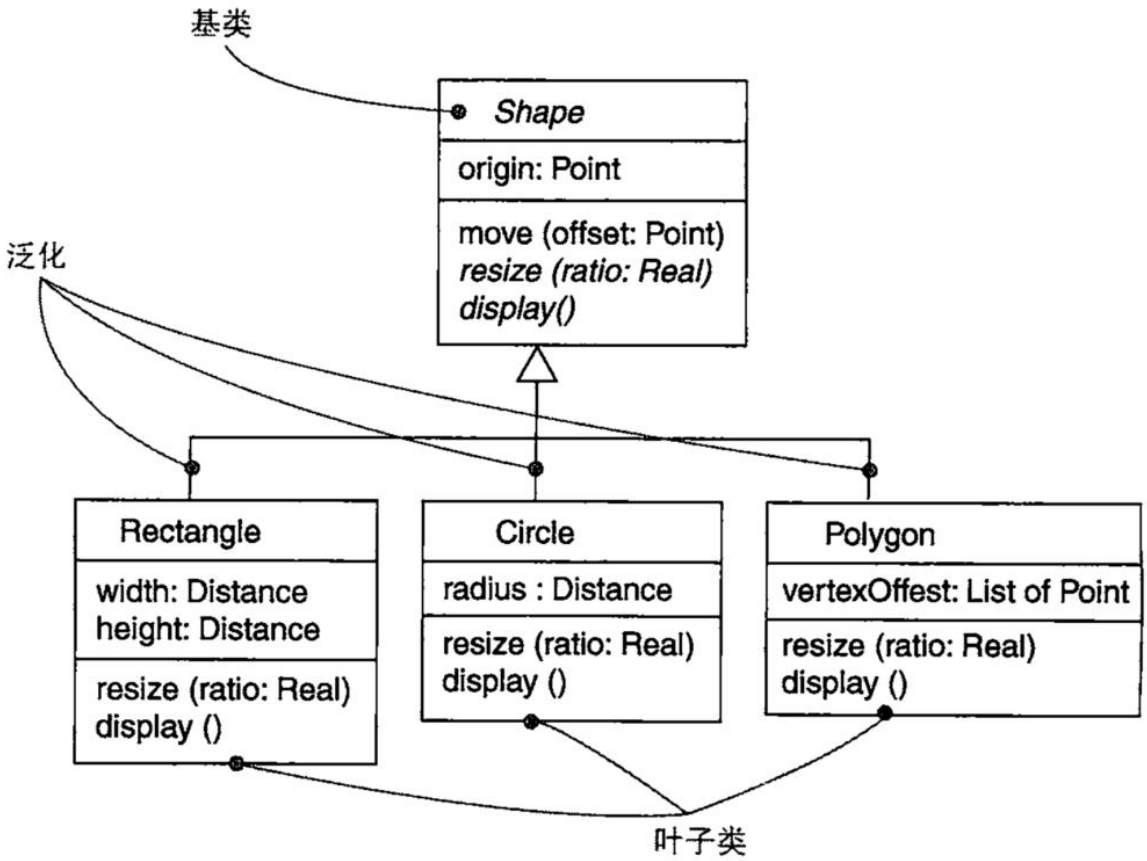
```
public abstract class Vihecle {
    public abstract void go(Address dest);
}

public class Car extends Vihecle {
    public void go(Address dest) {
        System.out.println("一路哼着歌，冒
            着烟，去了" + dest.getName()) ; }
}

public class SubClass extends Vihecle {
    public void go(Address dest) {
        … … }
}

public class Broom extends Vihecle {
    public void go(Address dest) {
        System.out.println("一路扫着土，笑
            呵呵，去了" + dest.getName()) ; }
}
```




```
public class Test {
    public static void main(String[] args) {
        Driver d = new Driver();
        Address ad = new Address("北京");
        d.setName("老张");
        System.out.println(d.getName());
        d.drive(new Car());
        d.drive(new Car(), ad);
    }

}
```



多态：示例 3



多态：示例 4

- 消息发送对象不知谁将接收消息，依赖接收对象以一种恰当方式解释消息
- 根据接收对象所属的类，决定提供什么样的行为，发送对象享受什么样的服务

How to

“Plan Ahead for Reuse”？

何为复用 ？

在构造新的软件系统时，重复使用已存在的软件产品（设计结构、源代码、文档等）技术

复用的 3 个层次：知识复用、方法复用、软件成分复用

软件成分复用的三个级别

1）分析结果复用（分析模型）

2）设计结果复用（设计模型）

3）代码复用：复制粘贴，包含导入（**关联**），继承

组件技术的软件工程：**CBSE，构件复用**

如何“提前计划复用”？

分析、设计、编码：

1）尽量复用已有的 ……

2）若没有可复的 ……，尽量为了**将来可复用**而做设计

以分析模式复用为例

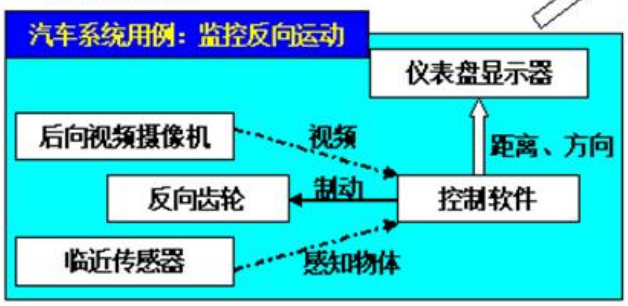
以接口复用为例

以“组合设计模式”复用为例

以分析模式复用为例

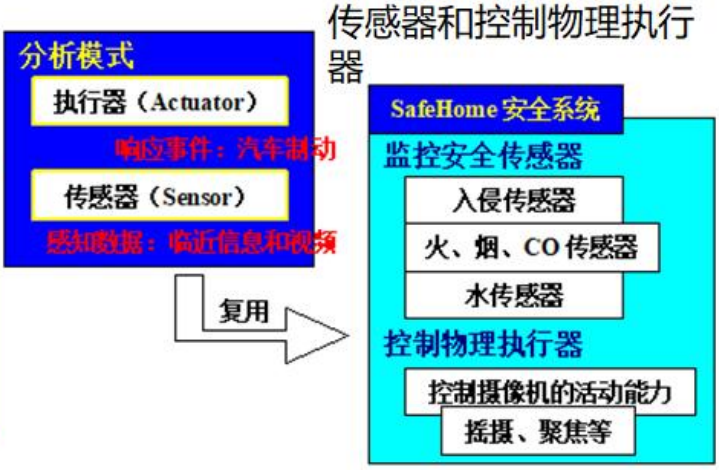
汽车控制中的用例：**监控反向运动**

- 控制和监控“实时摄像机”及汽车临近传感器

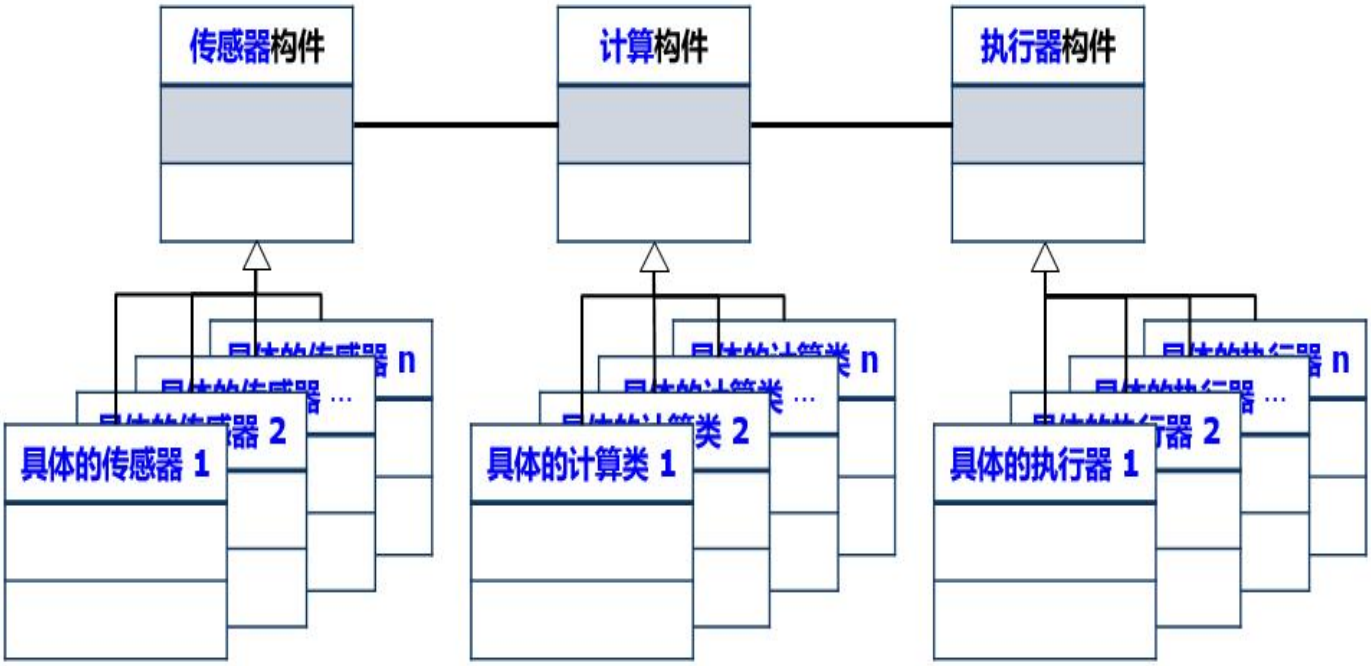


模式复用：

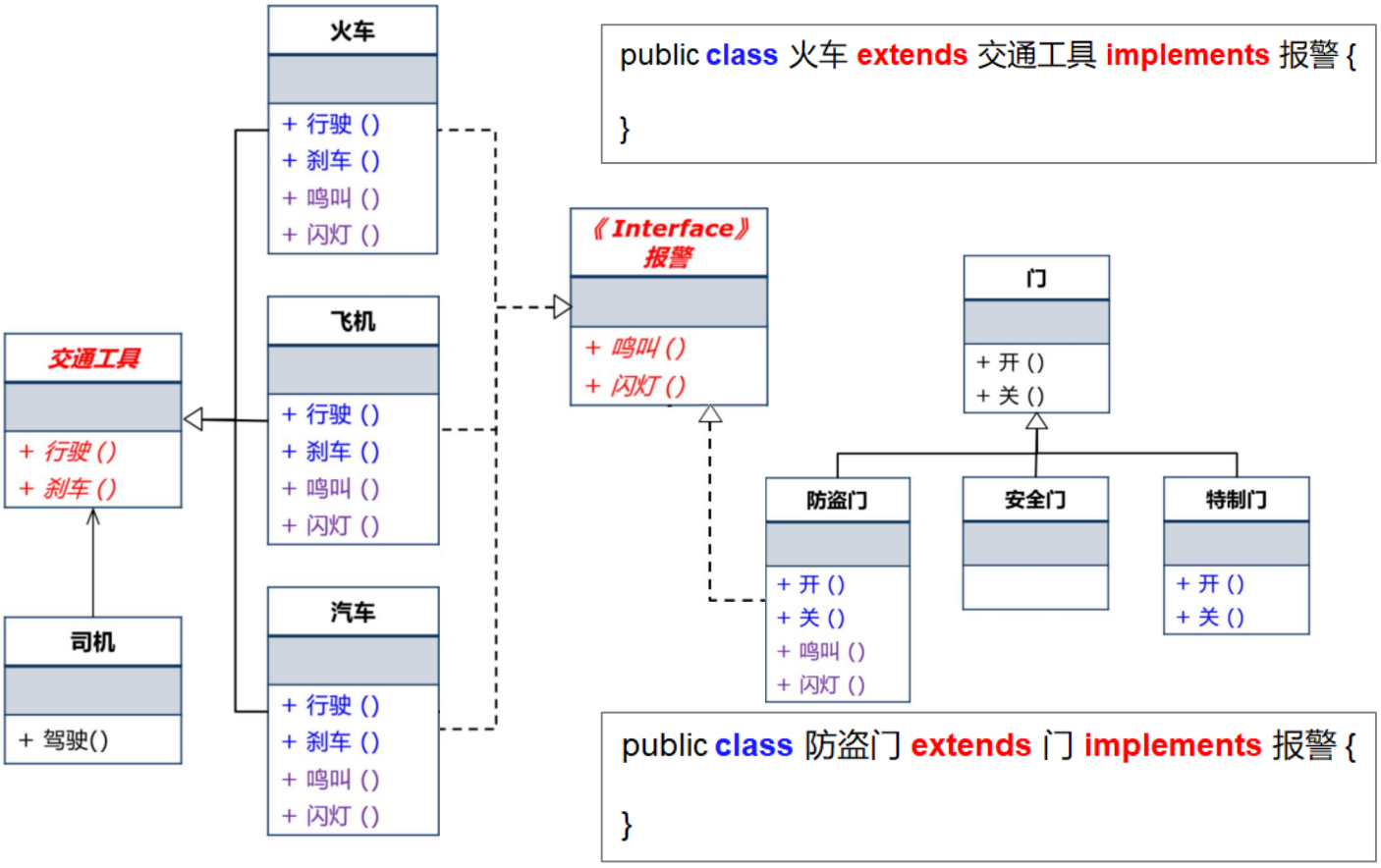
- 不同领域的软件需监控传感器和控制物理执行器



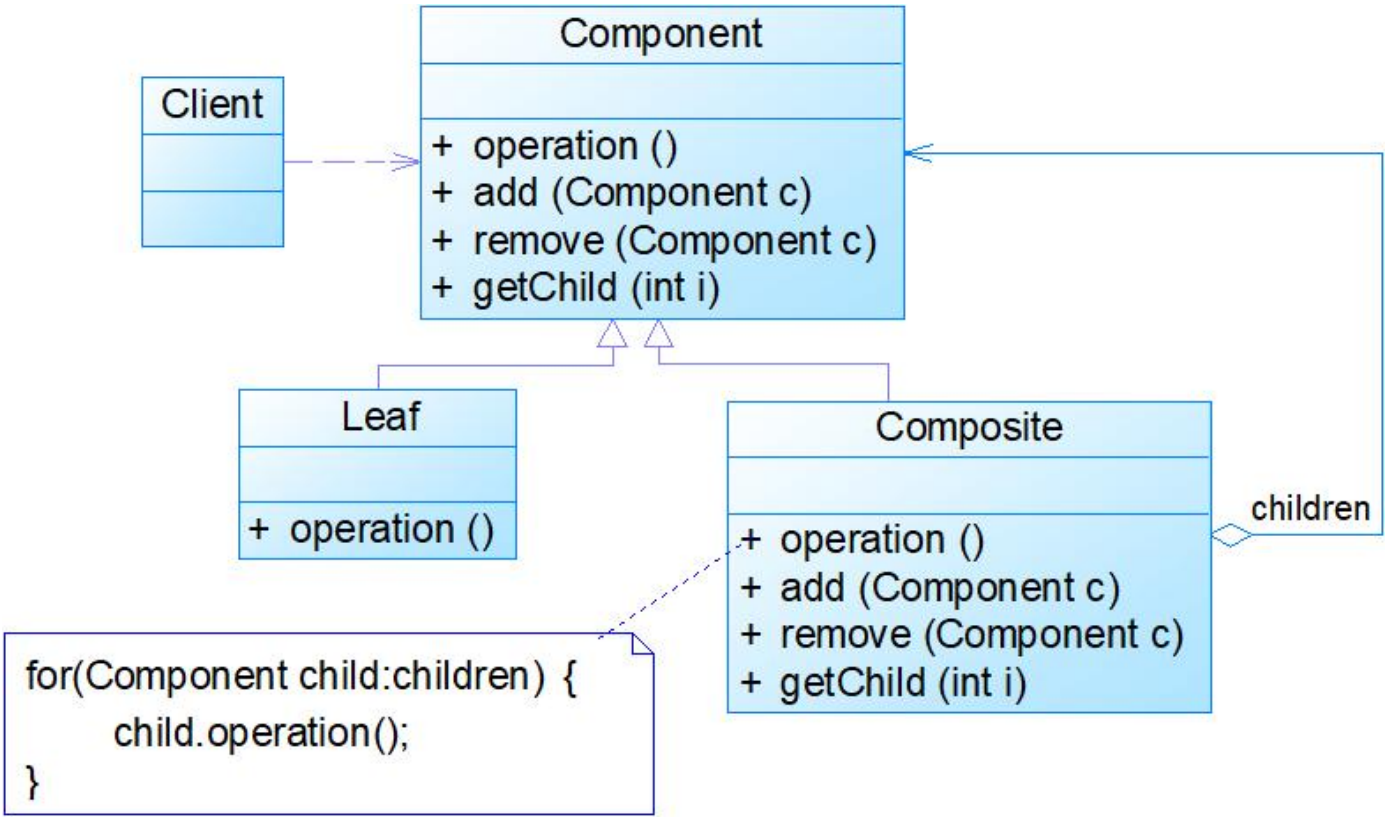
以分析模式复用为例



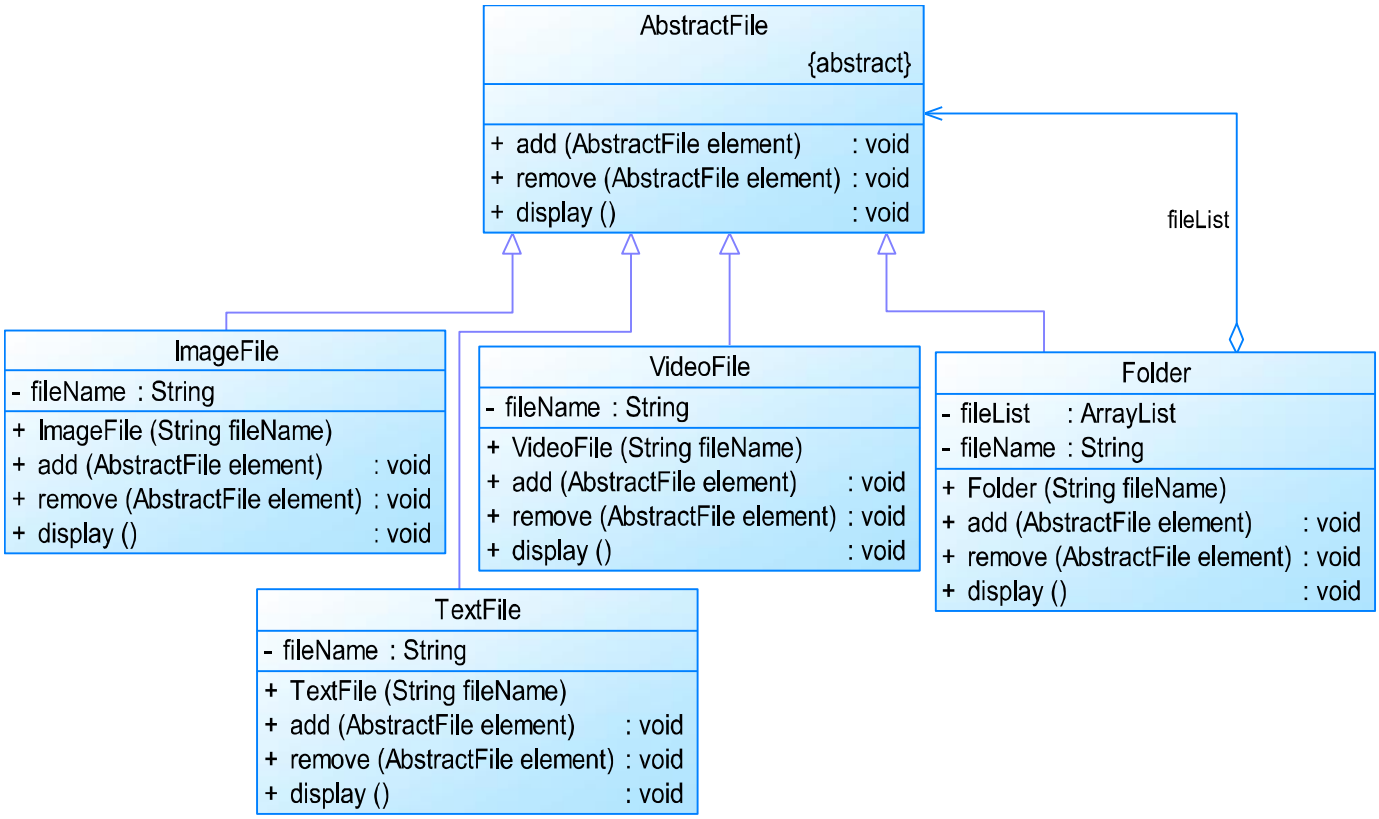
以接口复用为例



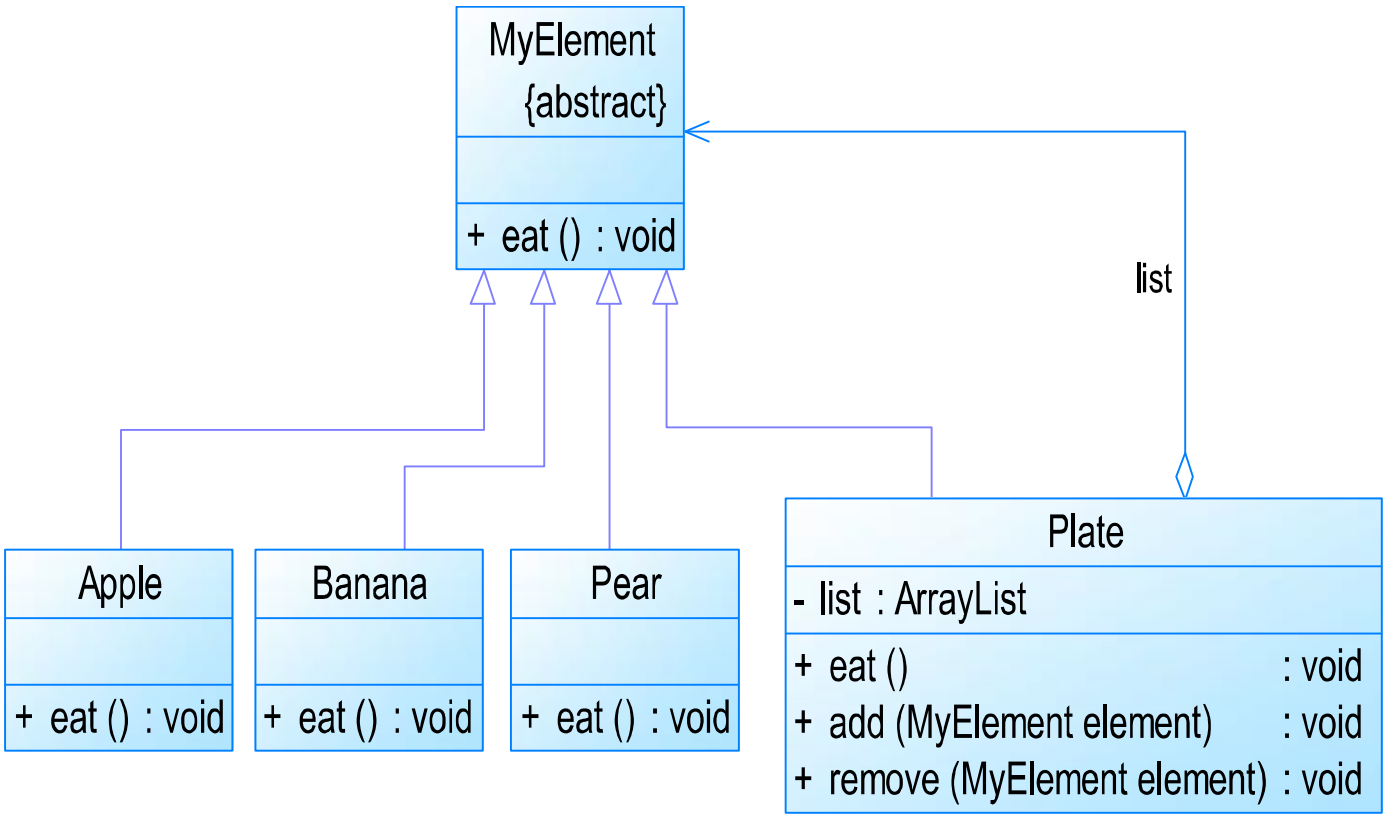
以“组合设计模式”复用为例



以“组合设计模式”复用为例



以“组合设计模式”复用为例



How to

“Be Open to the Future”？

如何“面向未来”？

分析、设计、编码：

1）尽量做到“复用”

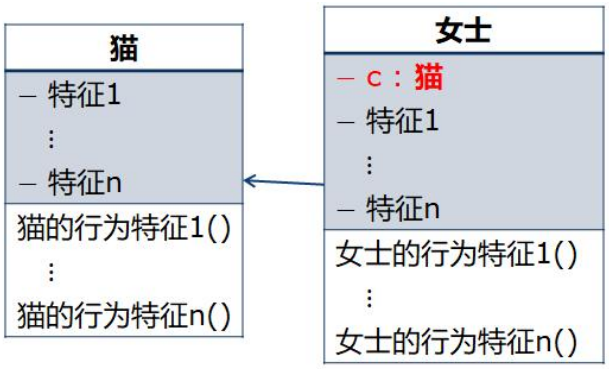
2）设计好的体系结构，面对需求的改变，做到可扩展性、可维护性

3）做设计时，心中要想着可测试性

4） ……，考虑非功能性需求

如何“面向未来”？

```
class Cat {
    特征1
      ⋮
    特征n
    cat() { …… }
    猫的行为特征1() {
        ……
    }
      ⋮
    猫的行为特征n() {
        ……
    }
}
```

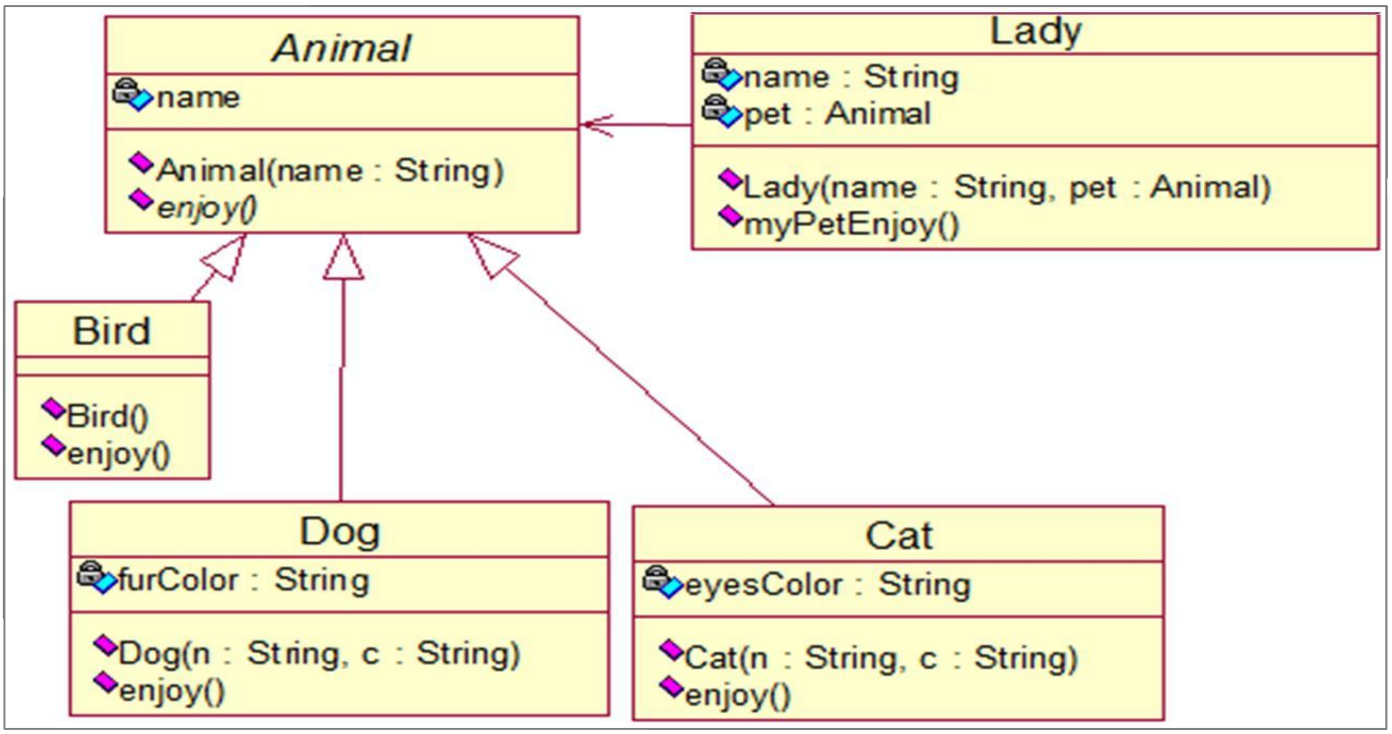



```
class Client {
    public static void main(String args[]) {
        Cat c = new Cat() ;
        Lady ℓ = new Lady(c) ;
        ℓ.行为特征1() ;
        ……
    }
}
```

```
class Lady {
    private Cat c ;
    特征1
      ⋮
    特征n
    Lady(Cat c) { this.c = c }
    女士的行为特征1() {
        ……
        this.c.猫的行为特征()
        ……
    }
      ⋮
    女士的行为特征n() {
        ……
    }
}
```

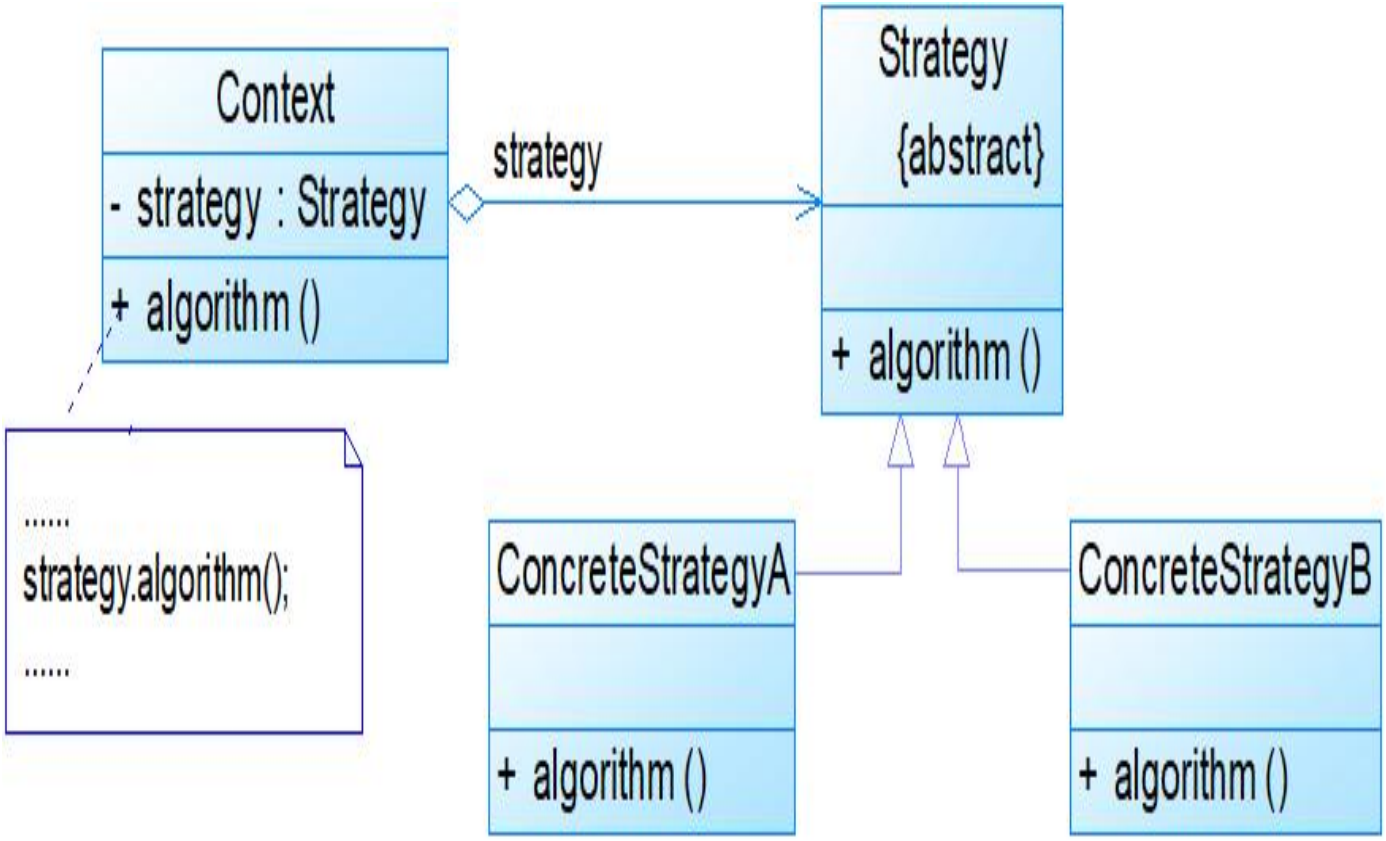
问题：可扩展性、可维护性比较差

如何“面向未来”？



优点：可扩展性、可维护性较好

扩展：策略模式



策略模式：分离责任与行为

封装、分割算法的责任和行为，委派给不同的对象管理

由客户端自己决定，在什么情况下使用什么具体策略

通用结构：

Context：环境类，表达一个问题环境

- 环境中可用多种策略解决问题，在环境类中维护一个对抽象策略类的引用实例

Strategy：抽象策略类，定义算法，保证策略一致性

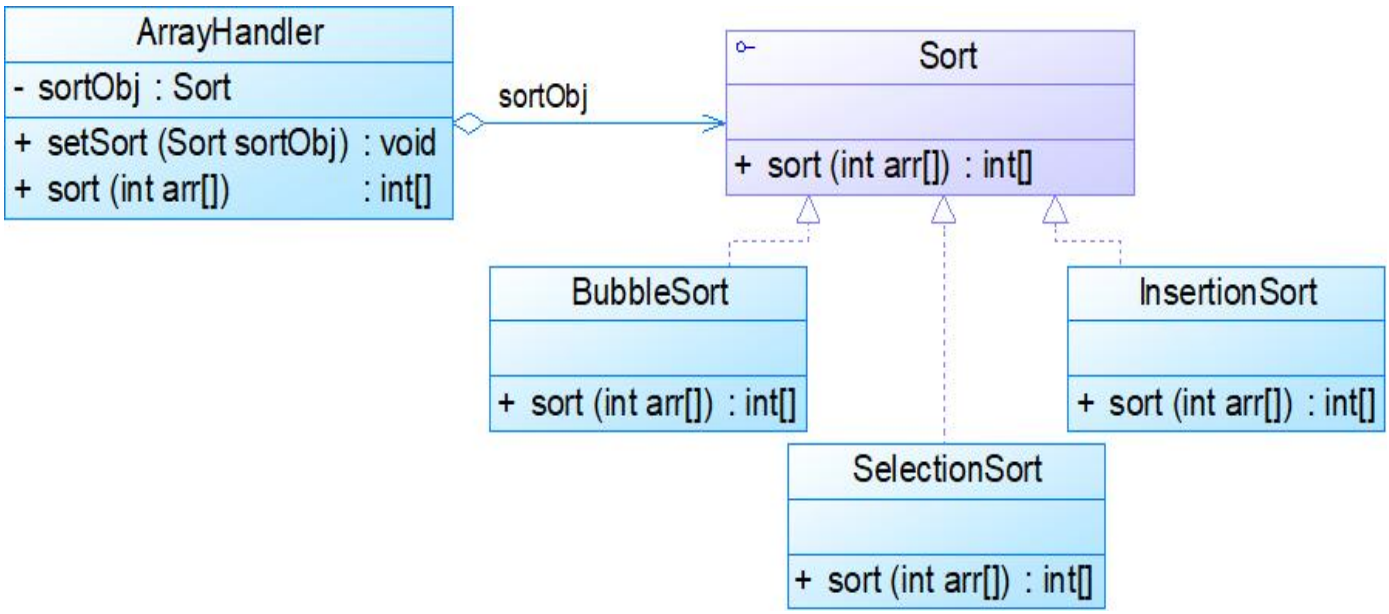
- 为所支持的算法声明抽象方法，是所有策略类的父类

ConcreteStrategy：具体策略类实现抽象策略类定义的算法

- 封装不同算法，一个类封装一个具体算法
- 每个算法独立于其他算法而变化，可以灵活相互替换

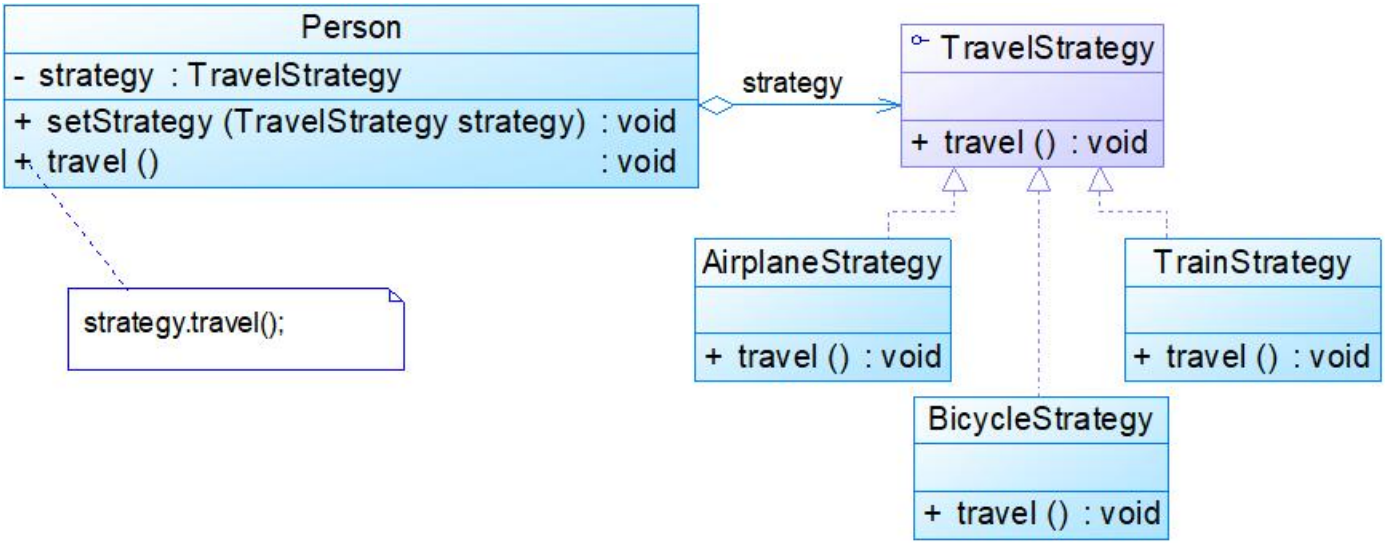
应用示例：1

用户需求：

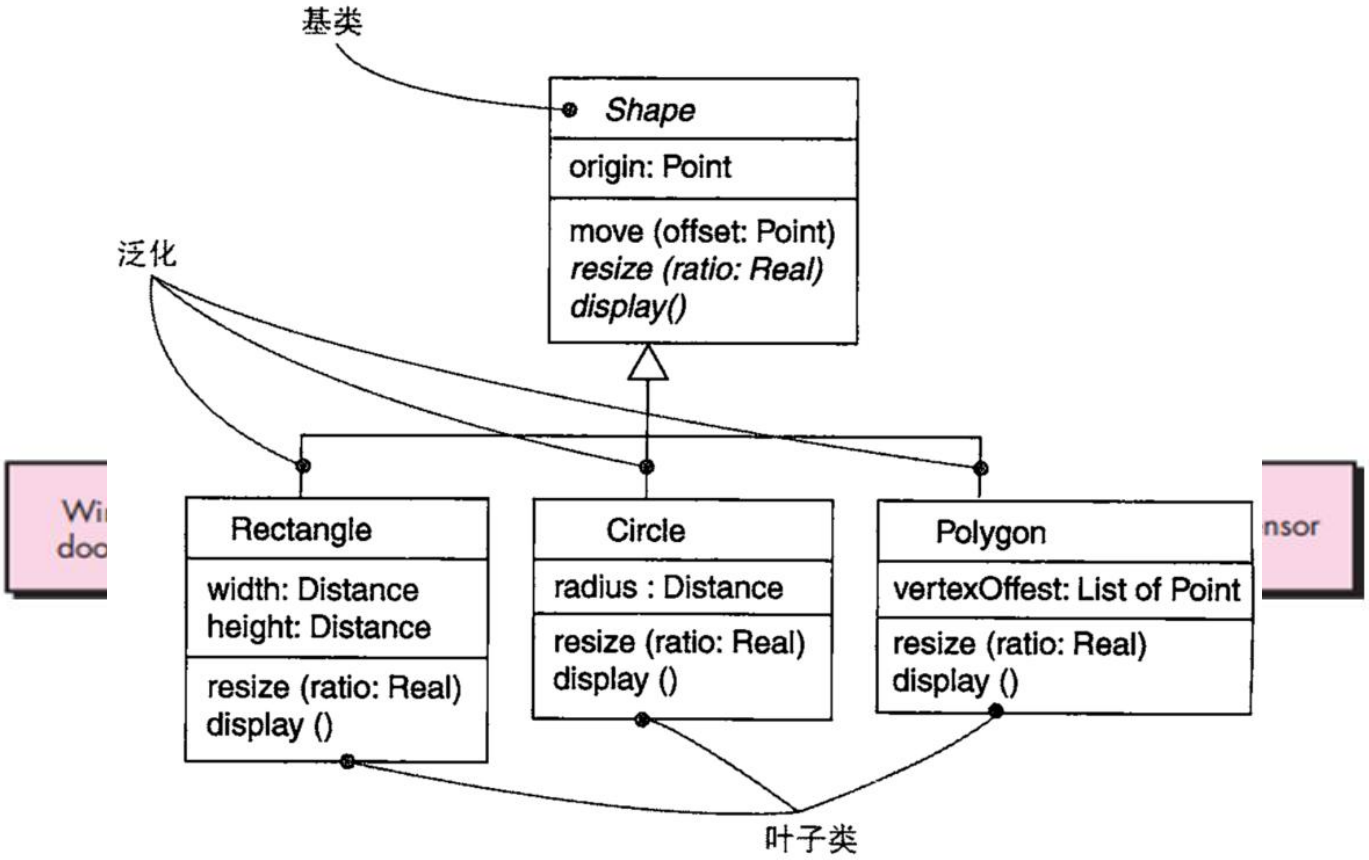


应用示例：2

用户需求：



应用示例：3


基类
Shape
origin: Point
move (offset: Point)
resize (ratio: Real)
display()
泛化
Rectangle
width: Distance
height: Distance
resize (ratio: Real)
display ()
Circle
radius : Distance
resize (ratio: Real)
display ()
Polygon
vertexOffest: List of Point
resize (ratio: Real)
display ()
叶子类