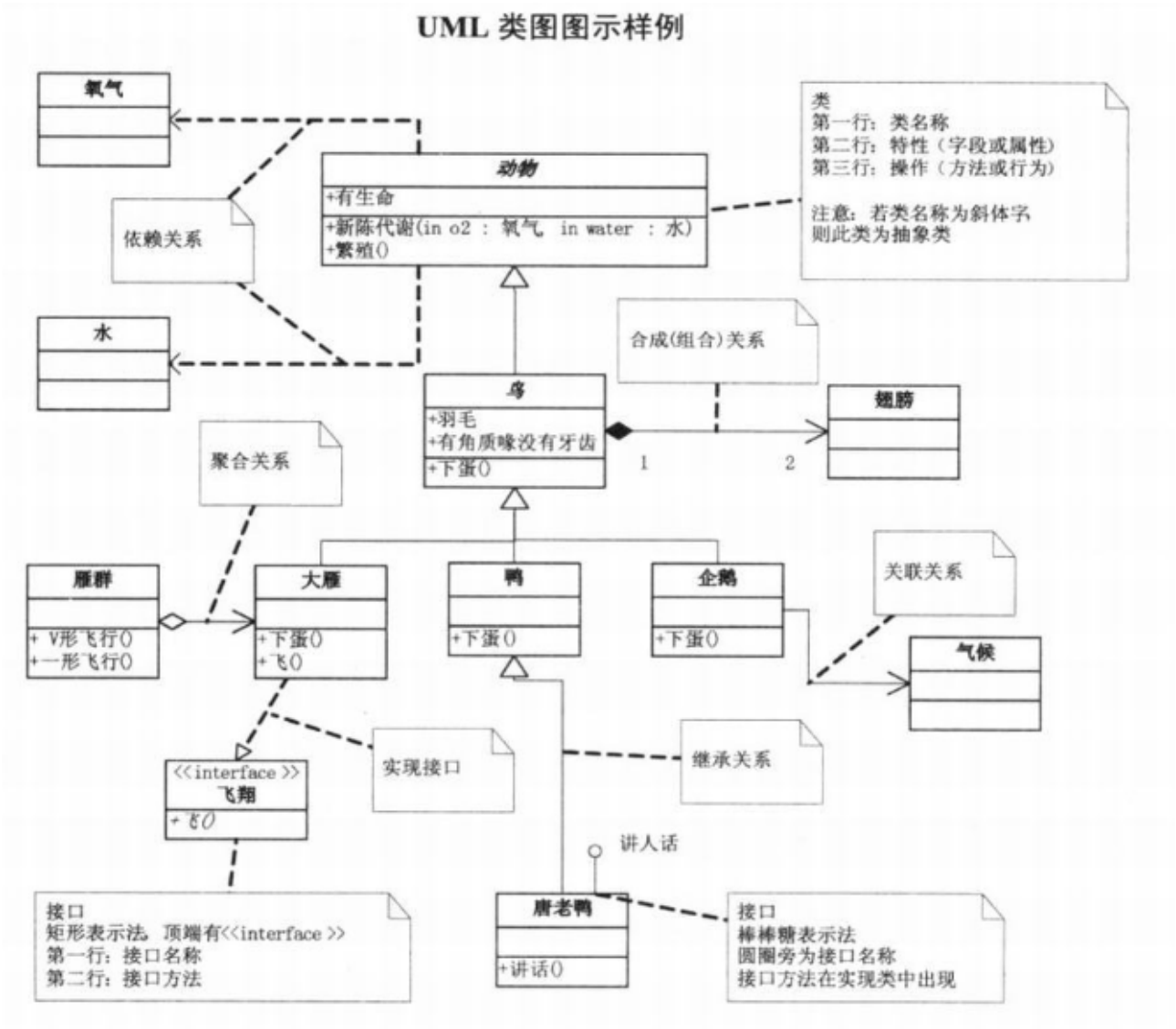
第 1 章

1、 编程原则

- 原则一：业务逻辑和界面逻辑分开。例如，将计算器的计算逻辑和显示逻辑分成两个类。
- 原则二：修改其中部分功能不能影响其余功能。这个需要将各个功能，比如加减乘除单独写成一个类。

2、 简单工厂模式解决的问题是如何去实例化一个合适的对象。简单工厂模式的核心思想就是：有一个专门的类来负责创建实例的过程。具体来说，把产品看着是一系列的类的集合，这些类是由某个抽象类或者接口派生出来的一个对象树。而工厂类用来产生一个合适的对象来满足客户的要求。

3、 UML 类图

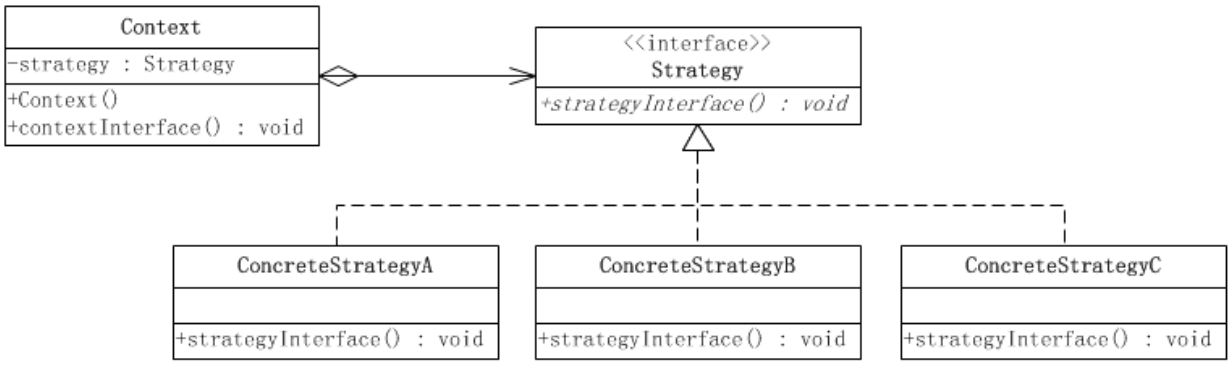


- 矩形框代表一个类，类图分三层，第一层为类名，如果是抽象类，用斜体表示。第二层是类的特性，通常就是字段和属性，第三层是类的操作，注意前面的符号，“+”表示 public ,“-”表示 private，“#”表示 protected。
- 继承关系用空心三角形+实线表示，实现接口用空心三角形+虚线表示，关联关系用实线箭头表示，聚合关系用空心菱形+实线箭头表示（A 包含 B，但 B 不是 A 的一部分），合成关系用实心的菱形+实线箭头表示（部分组合成整体），依赖关系用虚线箭头表示。（继空实，实空虚，关联实箭，聚合空菱实箭，合成实菱实箭，依赖虚箭）

第 2 章 策略模式

1、 面向对象的编程，并不是类越多越好，类的划分是为了封装，但分类的基础是抽象，具有相同属性和功能的对象的抽象集合才是类。

2、 策略模式：它定义了算法家族，分别封装起来，让它们之间可以互相替换，此模式让算法的变化，不会影响到使用算法的客户。



这个模式涉及到三个角色：

- 环境(Context)角色：持有一个 Strategy 的引用。
- 抽象策略(Strategy)角色：这是一个抽象角色，通常由一个接口或抽象类实现。此角色给出所有的具体策略类所需的接口。
- 具体策略(ConcreteStrategy)角色：包装了相关的算法或行为。

3、 策略模式和工厂模式的区别：

- 用途不一样，工厂是创建型模式,它的作用就是创建对象；策略是行为型模式,它的作用是让一个对象在许多行为中选择一种行为;

- 关注点不一样，一个关注对象创建，一个关注行为的封装。

第 3 章 单一职责原则

1、 面向对象基本原则之单一职责原则：规定一个类应该只有一个发生变化的原因。所谓职责是指类变化的原因。如果一个类有多于一个的动机被改变，那么这个类就具有多于一个的职责。而单一职责原则就是指一个类或者模块应该有且只有一个改变的原因。

2、 如果一个类承担的职责过多，就等于把这些职责耦合在一起，一个职责的变化可能会削弱或者抑制这个类完成其他职责的能力，这种耦合会导致脆弱的设计，当变化时，设计会受到意想不到的破坏。

3、 软件设计真正要做的许多内容，就是发现职责并把这些职责相互分离。

第 4 章 开放—封闭原则

1、 开放封闭原则核心的思想是：软件实体应该是可扩展，而不可修改的。也就是说，对扩展是开放的，而对修改是封闭的。（开放扩展，封闭修改）

第 5 章 依赖倒转原则

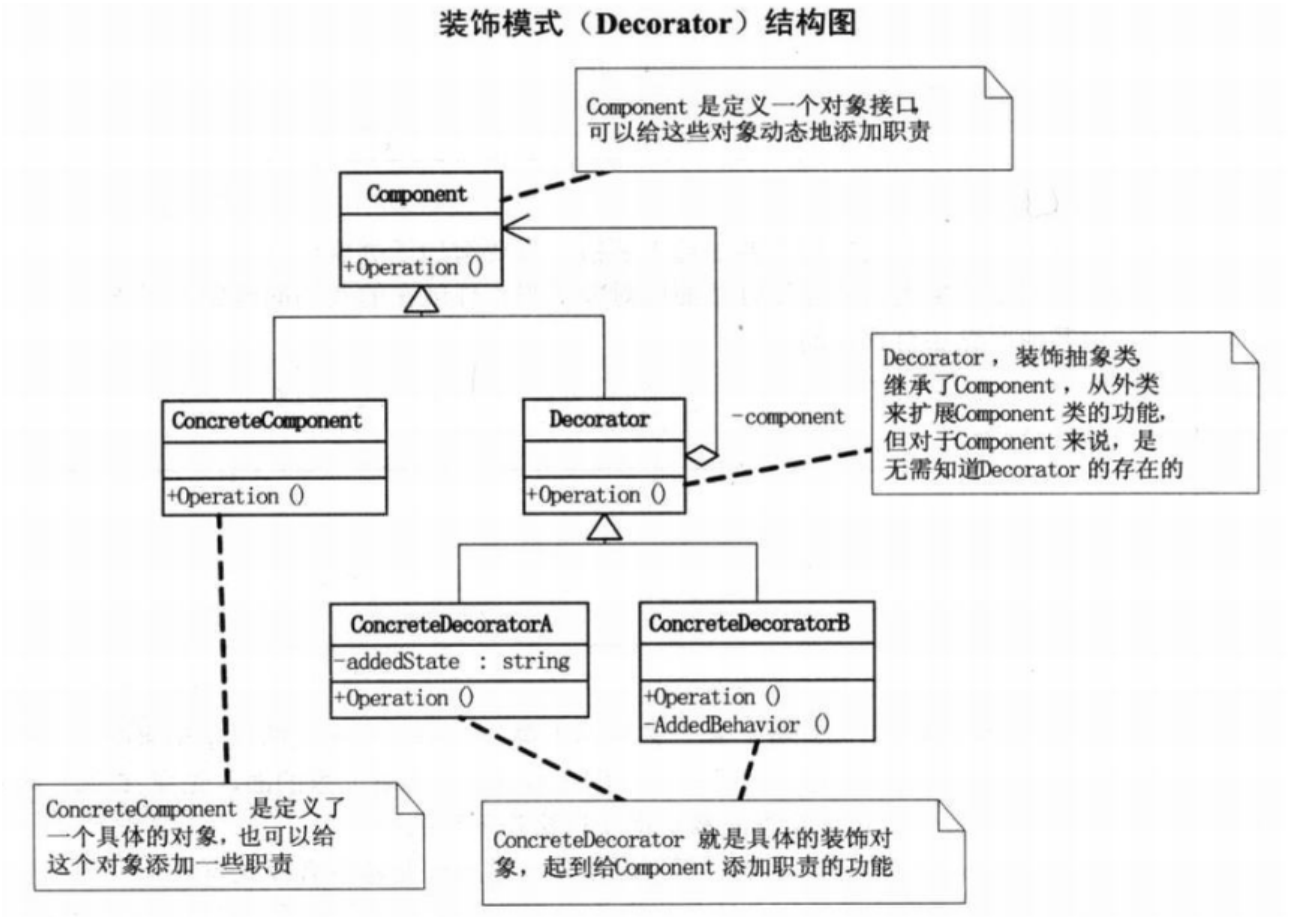
1、 依赖倒转原则：

- 高层次的模块不应该依赖于低层次的模块，他们都应该依赖于抽象。
- 抽象不应该依赖于具体实现，具体实现应该依赖于抽象。说白了，就是针对接口编程，而不是针对实现编程。

2、 里氏代换原则：子类必须能够替换掉它们的父类。只有当子类可以替换掉父类，软件单位的功能不受影响时，父类才能真正被复用，而子类也能够在父类的基础上增加新的行为。正是由于子类型的可替换性才使得使用父类类型的模块在无需修改的情况下就可以扩展。

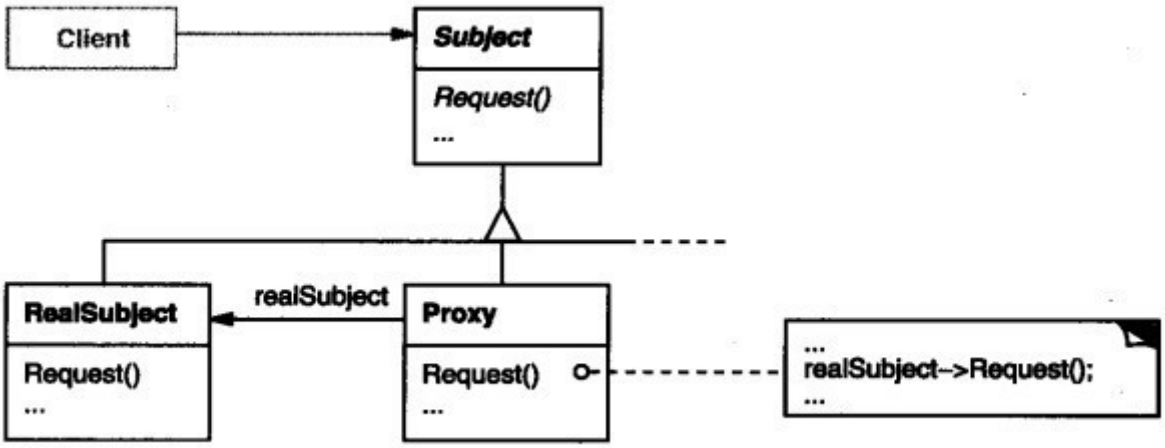
第 6 章 装饰者模式

1、 装饰者模式：动态地给一个对象添加一些额外的职责，就增加功能来说，装饰者模式比生成子类更为灵活。



第 7 章 代理模式

1、 代理模式：为其他对象提供一种代理以控制对这个对象的访问。



第 8 章 工厂方法模式

1、 工厂方法模式：定义一个用于创建对象的接口，让子类决定实例化哪一个类，工厂方法使一个类的实例化延迟到其子类。

2、 简单工厂模式的最大优点在于工厂类中包含了必要的逻辑判断，根据客户端的选择条件动态实例化相关的类，对于客户端来说，去除了与具体产品的依赖，但违反了开放-封闭原则。

3、 工厂方法模式实现时，客户端需要决定实例化哪一个工厂来实现运算类，选择判断的问题还是存在的，也就是说，工厂方法把简单工厂的内部逻辑判断移到了客户端代码来来进行。你想要加功能，本来是改工厂类的，而现在是修改客户端。

第 9 章 原型模式

1、 原型模式：用原型实例指定创建对象的各类，并且通过挎贝这些原型创建新的对象。

2、 浅拷贝与深拷贝：简单的来说就是，在有指针的情况下，浅拷贝只是增加了一个指针指向已经存在的内存，而深拷贝就是增加一个指针并且申请一个新的内存，使这个增加的指针指向这个新的内存，采用深拷贝的情况下，释放内存的时候就不会出现在浅拷贝时重复释放同一内存的错误！

第 10 章 模板方法模式

1、 定义一个操作中的算法骨架，而将一些步骤推迟到子类中（如使用虚函数可推迟）。模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。

第 11 章 迪米特法则

1、 迪米特法则：如果两个类不必彼此直接通信，那么这两个类就不应当发生直接的相互作用。如果其中一个类需要调用另一个类的某一个方法的活，可以通过第三者转发这个调用。

第 12 章 外观模式

1、 外观模式：为子系统中的一组接口提供一个一致的界面，此模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。

第 13 章 建造者模式

1、 建造者模式：将一个复杂对象的构建与它的表示分分离，使得同样的构建过程可以创建不同的表示。

说明：将对象的构建定义成抽象类，其具体的构建由其子类完成。

第 14 章 观察者模式

1、 观察者模式定义了一种一对多的依赖关系，让多个观察者对象同时监听某个主题对象。这个主题对象在状态发生变化时，会通知所有观察者对象，使它们能够自动更新自己。

2、 委托(delegation)是一种组合方法，它使组合具有与继承同样的复用能力 。在委托方式下，有两个对象参与处理一个请求，接受请求的对象将操作委托给它的代理者。使用继承时，被继承的操作总能引用接受请求的对象， C++中通过 this 成员变量，Smalltalk 中则通过 self。委托方式为了得到同样的效果，接受请求的对象将自己传给被委托者(代理人)，使被委托的操作可以引用接受请求的对象。举例来说，我们可以在窗口类中保存一个矩形类的实例变量来 代理 矩形类的特定操作，这样窗口类可以复用矩形类的操作，而不必像继承时那样定义成矩形类的子类。也就是说，一个窗口拥有一个矩形，而不是一个窗口就是一个矩形。窗口现在必须显式的将请求转发给它的矩形实例，而不是像以前它必须继承矩形的操作。

第 15 章 抽象工厂模式

1、 抽象工厂模式：

意 图：提供一个创建一系列相关或相互依赖对象的接口,而无需指定它们具体的类。

心 得：工厂方法把生产产品的方式封装起来了，但是一个工厂只能生产一类对象，当一个工厂需要生产多类产品的时候，就需要使用抽象工厂了。抽象工厂类定义了一组标准的实现接口，这些接口一般都是和具体的产品类继承层次对应的。如 createProductA 接口只能生产抽象产品类的子类产品，因此抽象工厂的具体实现类之间的关系就是个生产了一批不同产品族的组合。这样通过抽象工厂模式可以方便的更换产品族，代码修改的代价只需要更换一个具体的工厂对象就可以了。因此直观上可以把抽象工厂看作一组工厂方法，它的每一个接口都可以提取出一个单独的工厂方法。

解 说：抽象工厂类封装了几个类方法，而这些方法又是另外一个类 A 的方法，抽象工厂类的子类调用定义时根据需要调用类 A 的子类。

第 16 章 状态模式

1、 状态模式：当一个对象的内在状态改变时允许改变其行为，这个对象看起来像是改变了其类。

2、 状态模式的目的是为了消除庞大的条件分支语句，即 if 语句。定义一个抽象状态类，将各种状态定义成一个独立的抽象类的子类，当满足一定条件时，从这个状态转入下个状态。

第 17 章 适配器模式

1、 适配器模式：将一个类的接口转换成客户希望的另外一个接口。这种模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

第 18 章 备忘录模式

1、 备忘录：在不破坏封闭性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态。这样以后就可将该对象恢复到原先保存的状态。

第 19 章 组合模式

1、 组合模式：将对象组合成树形结构以表示“部分-整体”层次结构。组合模式使得用户对单个对象和组合对象的使用具有一致性。

第 20 章 迭代器模式

1、 迭代器模式：提供一种方法顺序访问一个聚合对象中各个元素，而又不暴露该对象的内部表示。

第 21 章 单例模式

1、 单例模式：保证一个类仅有一个实例，并提供一个访问它的全局访问点。

第 22 章 桥接模式

1、 合成/聚合复用原则：尽量使用合成/聚合，尽量不要使用类继承。聚合表示一种弱的拥有关系，体现的是 A 对象可以包含 B，但 B 不是 A 的一部分。合成则是一种强拥有关系，体现了严格的部分和整体的关系。部分和整体的生命周期一样。如小鸟和翅膀就是合成关系。

2、 桥接模式，将抽象部分与它的实现部分分离，使它们都可以独立地变化。这里的抽象与实现分离，并不是让抽象类与其派生类分离。实现指的是抽象类和它的派生类用来实现自己的对象。

第 23 章 命令模式

1、 命令模式：将一个请求封装为一个对象，从而使你可用不同的请求对客户进行参数化；对请求排队或者记录请求日志，以及支持可撤销的操作。

第 24 章 职责链模式

1、 职责链模式：使多个对象都有机会处理请求，从而避免请求的发送者和接收者之间的救命关系。将这个对象连成一条链，并沿着这条链传递该请求，直到有一个对象处理它为止。

第 25 章 中介模式

1、 中介者模式：用一个中介对象来封装一系列的对象交互中介者使各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。

第 26 章 享元模式

1、 享元模式：运用共享技术有效地支持大量细粒度的对象。

第 27 章 解释器模式

1、 解释器模式：给定一个语言，定义它的文法的一种表示，并定义一个解释器，这个解释器使用该表示来解释语言中的句子。

第 28 章 访问者模式

1、 访问者模式：表示一个作用于某个对象结构中的各元素的操作。它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。

2、 访问者模式适用于数据结构相对稳定的系统。

第 28 章 总结

1、 以下代码尽量使用面向对象编程避免：

- 大量的 if…else 语句
- 大量重复代码

2、 设计模式六大原则

- 单一职责原则：不要存在多于一个导致类变更的原因。通俗的说，即一个类只负责一项职责。
- 里氏替换原则：如果对每一个类型为 T1 的对象 o1，都有类型为 T2 的对象 o2，使得以 T1 定义的所有程序 P 在所有的对象 o1 都代换成 o2 时，程序 P 的行为没有发生变化，那么类型 T2 是类型 T1 的子类型（定义 1）。所有引用基类的地方必须能透明地使用其子类的对象（定义 2）。
- 依赖倒置原则：高层模块不应该依赖低层模块，二者都应该依赖其抽象；抽象不应该依赖细节；细节应该依赖抽象。
- 接口隔离原则：客户端不应该依赖它不需要的接口；一个类对另一个类的依赖应该建立在最小的接口上。 将臃肿的接口 I 拆分为独立的几个接口，类 A 和类 C 分别与他们需要的接口建立依赖关系。也就是采用接口隔离原则。
- 迪米特法则：一个对象应该对其他对象保持最少的了解。如果两个类不必彼此直接通信，那么这两个类就不应当发生直接的相互作用。如果其中一个类需要调用另一个类的某一个方法的活，可以通过第三者转发这个调用。
- 开闭原则 ： 一个软件实体如类、模块和函数应该对扩展开放，对修改关闭。