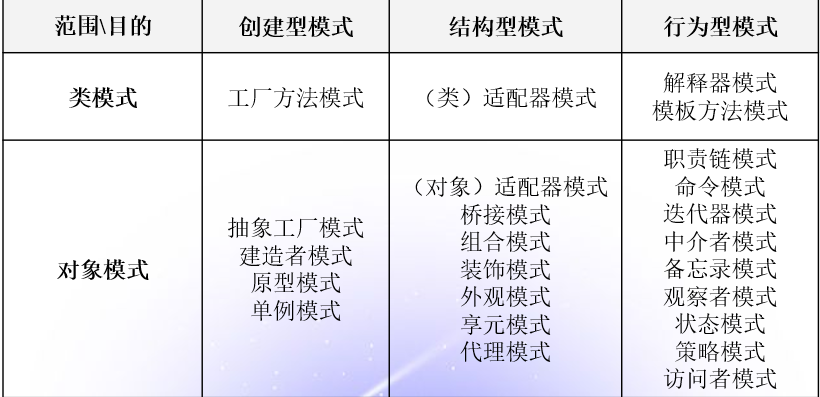
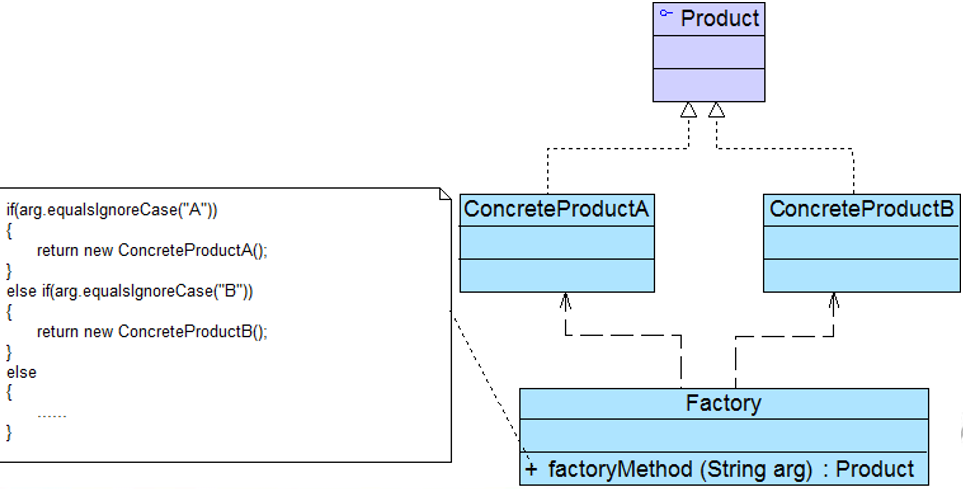
1. 面向对象程序设计主要有过程抽象和数据抽象
2. OOPL三大特性：封装性、继承性、多态性
3. 类中的成员有公有（public）、私有（private）和保护（protected）三种控制权限



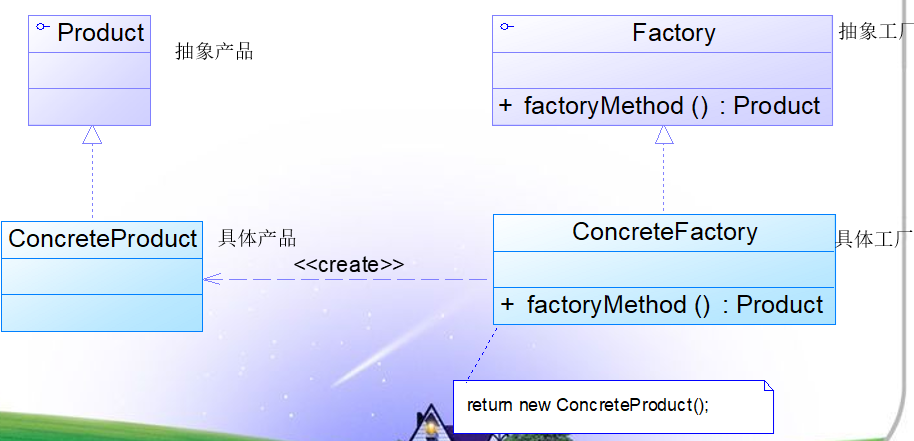
1. 设计模式一般有如下几个基本要素：模式名称、问题、目的、解决方案、效果



1. 简单工厂模式(Simple Factory Pattern)：又称为静态工厂方法(Static Factory Method)模式，它属于类创建型模式。在简单工厂模式中，可以根据参数的不同返回不同类的实例。简单工厂模式专门定义一个类来负责创建其他类的实例，被创建的实例通常都具有共同的父类。
   1. 考虑一个简单的软件应用场景，一个软件系统可以提供多个外观不同的按钮（如圆 形按钮、矩形按钮、菱形按钮等），这些按钮都源自同一个基类，不过在继承基类后不同的子类修改了部分属性从而使得它们可以呈现不同的外观，如果我们希望在使用这些按钮时，不需要知道这些具体按钮类的名字，只需要知道表示该按钮类的一个参数，并提供一个调用方便的方法，把该参数传入方法即可返回一个相应的按钮对象，此时，就可以使用简单工厂模式。

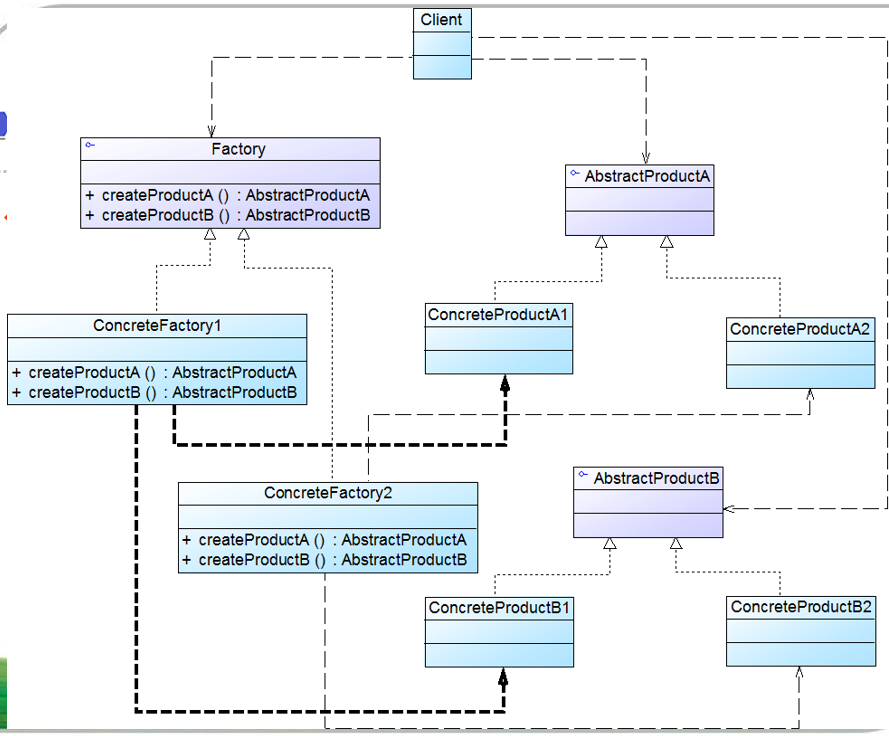
简单工厂模式包含如下角色：Factory：工厂角色；Product：抽象产品角色；ConcreteProduct：具体产品角色

1. 工厂方法模式(Factory Method Pattern)又称为工厂模式，也叫虚拟构造器(Virtual Constructor)模式或者多态工厂(Polymorphic Factory)模式，它属于类创建型模式。在工厂方法模式中，工厂父类负责定义创建产品对象的公共接口，而工厂子类则负责生成具体的产品对象，这样做的目的是将产品类的实例化操作延迟到工厂子类中完成，即通过工厂子类来确定究竟应该实例化哪一个具体产品类。

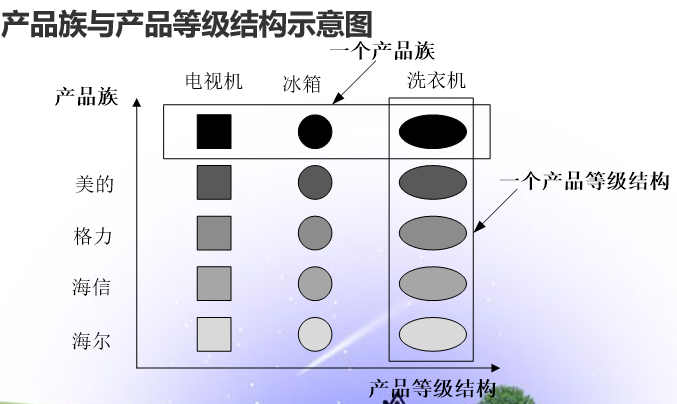
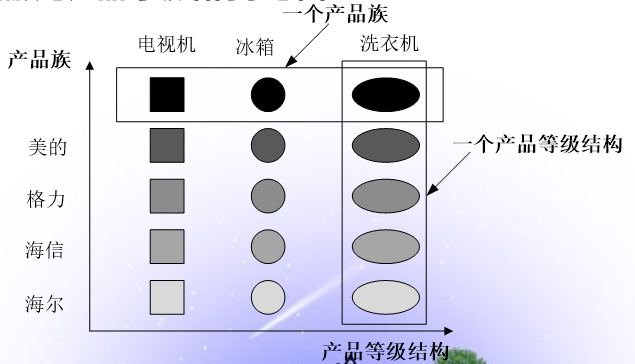


工厂方法模式包含如下角色：Factory：抽象工厂；ConcreteFactory：具体工厂；Product：抽象产品角色；ConcreteProduct：具体产品角色

1. 抽象工厂模式(Abstract Factory Pattern)：提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。抽象工厂模式又称为Kit模式，属于对象创建型模式。

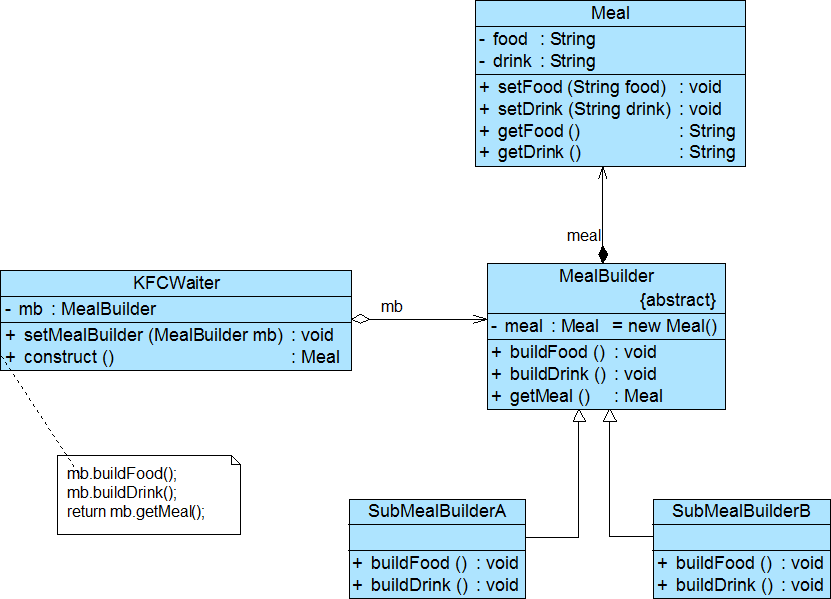


抽象工厂模式包含如下角色：AbstractFactory：抽象工厂；ConcreteFactory：具体工厂；AbstractProduct：抽象产品角色；ConcreteProduct：具体产品角色

1. 建造者模式(Builder Pattern)：将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。建造者模式是一步一步创建一个复杂的对象，它允许用户只通过指定复杂对象的类型和内容就可以构建它们，用户不需要知道内部的具体构建细节。建造者模式属于对象创建型模式。根据中文翻译的不同，建造者模式又可以称为生成器模式。





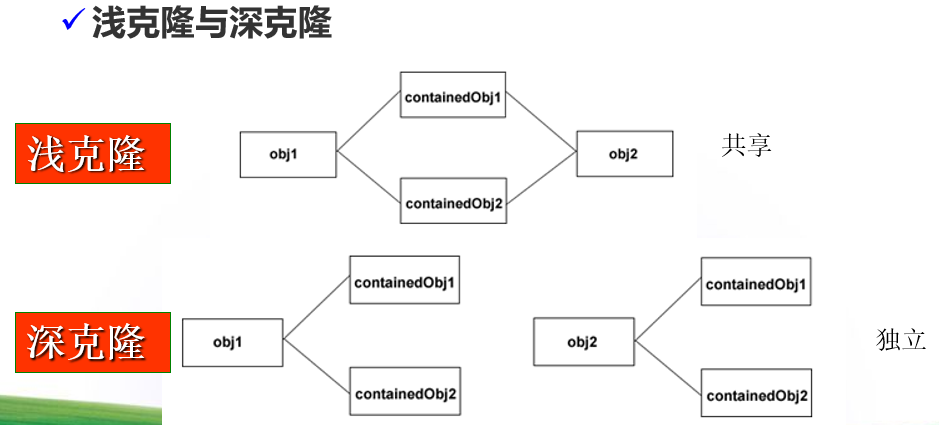
建造者模式包含如下角色：Builder：抽象建造者；ConcreteBuilder：具体建造者；Director：指挥者；Product：产品角色

1. 原型模式(Prototype Pattern)：原型模式是一种对象创建型模式，用原型实例指定创建对象的种类，并且通过复制这些原型创建新的对象。原型模式允许一个对象再创建另外一个可定制的对象，无须知道任何创建的细节。原型模式的基本工作原理是通过将一个原型对象传给那个要发动创建的对象，这个要发动创建的对象通过请求原型对象拷贝原型自己来实现创建过程。**就是复制对象/类**





原型模式包含如下角色：Prototype：抽象原型类；ConcretePrototype：具体原型类；Client：客户类



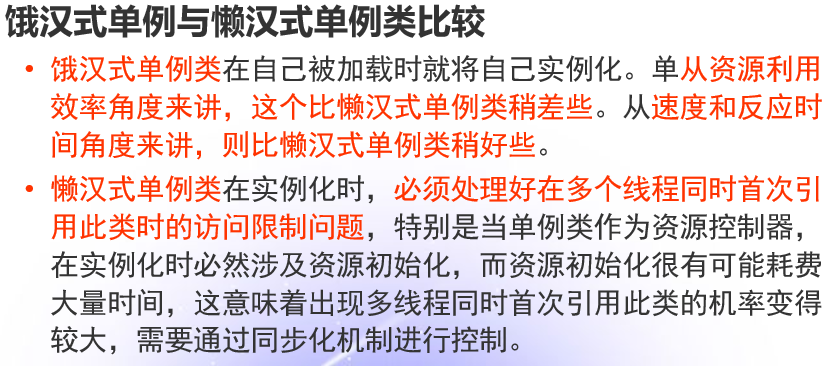
在浅克隆中，当对象被复制时它所包含的成员对象却没有被复制；在深克隆中，除了对象本身被复制外，对象包含的引用也被复制，也就是其中的成员对象也将复制。在Java语言中，通过覆盖Object类的clone()方法可以实现浅克隆；如果需要实现深克隆，可以通过序列化等方式来实现。

1. 单例模式(Singleton Pattern)：单例模式确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例，这个类称为单例类，它提供全局访问的方法。单例模式的要点有三个：一是某个类只能有一个实例；二是它必须自行创建这个实例；三是它必须自行向整个系统提供这个实例。单例模式是一种对象创建型模式。单例模式又名单件模式或单态模式。



单例模式包含如下角色：Singleton：单例；

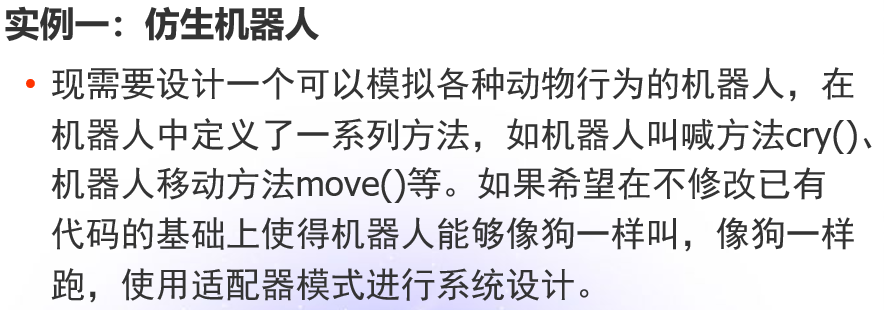
在单例模式的实现过程中，需要注意如下三点：单例类的构造函数为私有（用户无法通过new关键字直接实例化它）；提供一个自身的静态私有成员变量；提供一个公有的静态工厂方法



1. 适配器模式(Adapter Pattern) ：将一个接口转换成客户希望的另一个接口，适配器模式使接口不兼容的那些类可以一起工作，其别名为包装器(Wrapper)。适配器模式既可以作为类结构型模式，也可以作为对象结构型模式。

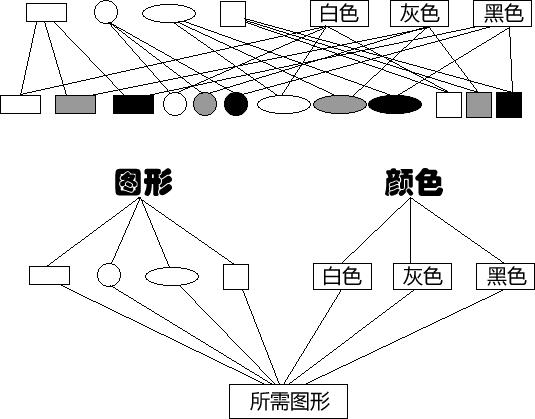


适配器模式包含如下角色：Target：目标抽象类；Adapter：适配器类；Adaptee：适配者类；Client：客户类



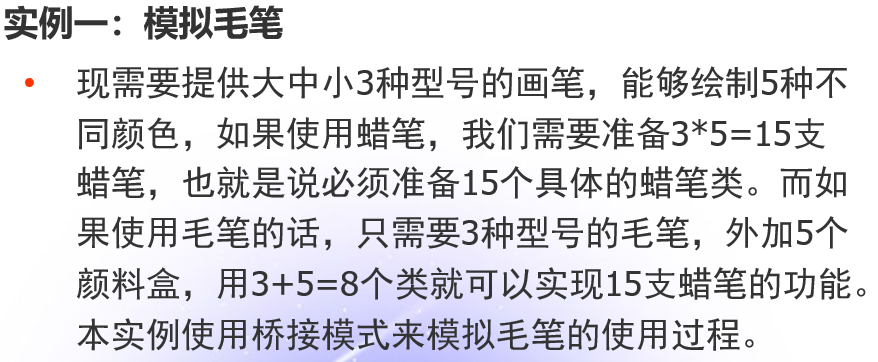


1. 桥接模式(Bridge Pattern)：将抽象部分与它的实现部分分离，使它们都可以独立地变化。它是一种对象结构型模式，又称为柄体(Handle and Body)模式或接口(Interface)模式。



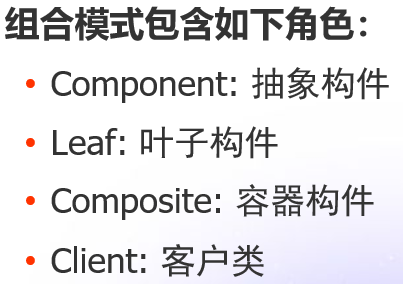
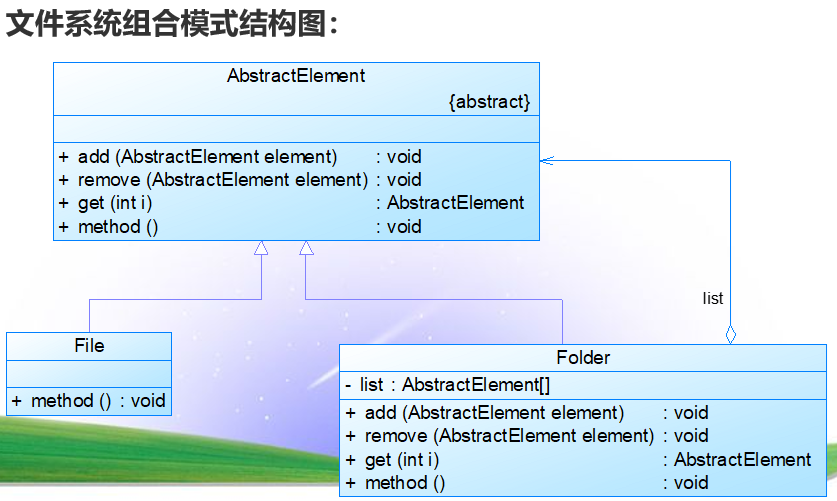


桥接模式包含如下角色：Abstraction：抽象类；RefinedAbstraction：扩充抽象类；Implementor：实现类接口；ConcreteImplementor：具体实现类

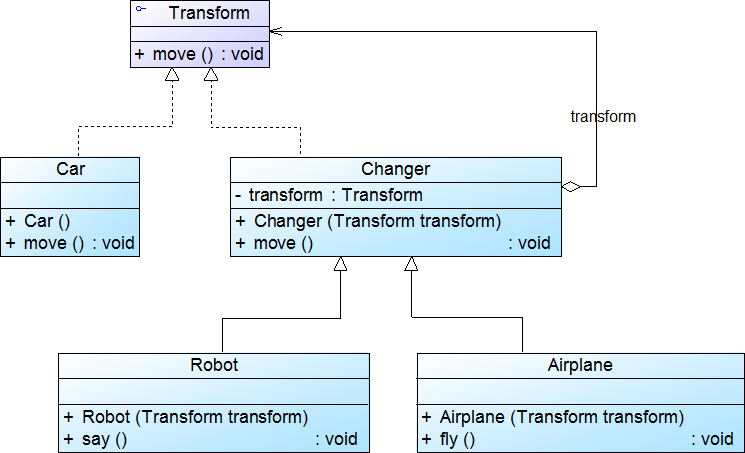
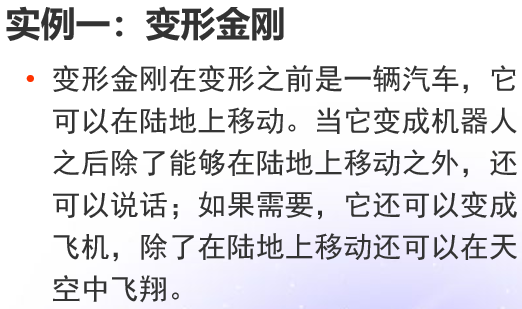
 

1. 组合模式(Composite Pattern)：组合多个对象形成树形结构以表示“整体-部分”的结构层次。组合模式对单个对象（即叶子对象）和组合对象（即容器对象）的使用具有一致性。组合模式又可以称为“整体-部分”(Part-Whole)模式，属于对象的结构模式，它将对象组织到树结构中，可以用来描述整体与部分的关系。

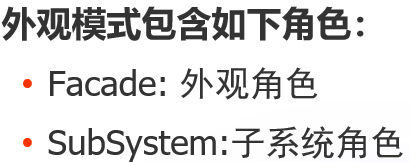


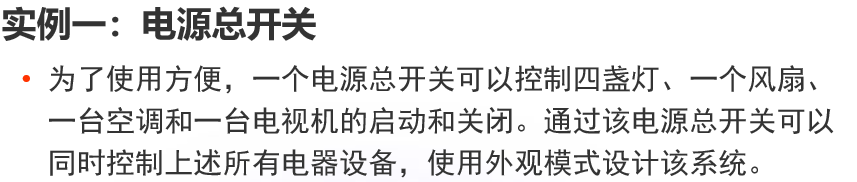
 

1. 装饰模式(Decorator Pattern) ：动态地给一个对象增加一些额外的职责(Responsibility)，就增加对象功能来说，装饰模式比生成子类实现更为灵活。其别名也可以称为包装器(Wrapper)，与适配器模式的别名相同，但它们适用于不同的场合。根据翻译的不同，装饰模式也有人称之为“油漆工模式”，它是一种对象结构型模式。

1. 外观模式(Facade Pattern)：外部与一个子系统的通信必须通过一个统一的外观对象进行，为子系统中的一组接口提供一个一致的界面，外观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。外观模式又称为门面模式，它是一种对象结构型模式









1. PPT\_C15\_P30—P34