一、创建型模式

深入理解设计模式（一）：单例模式(Singleton pattern)：确保一个类只有一个实例，并提供全局访问点01 单例模式.note

深入理解设计模式（二）：简单工厂模式(factory method pattern)：实质是由一个工厂类根据传入的参数，动态决定应该创建哪一个产品类（这些产品类继承自一个父类或接口）的实例。
02 简单工厂模式.note

深入理解设计模式（四）：工厂方法模式(factory method pattern)：定义了一个创建对象的接口，但由子类决定要实例化的类是哪一个. 工厂方法让类把实例化推迟到子类

深入理解设计模式（五）：抽象工厂模式(Abstract factory pattern)：提供一个接口，用于创建相关或依赖对象的家族，而不需要指定具体类.

深入理解设计模式（六）：原型模式(prototype pattern)：当创建给定类的实例过程很昂贵或很复杂时，就使用原形模式.

深入理解设计模式（七）：建造者模式(Builder pattern)：使用生成器模式封装一个产品的构造过程，并允许按步骤构造. 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

二、行为型模式

深入理解设计模式（三）：策略模式(strategy pattern) ：定义了算法族，分别封闭起来，让它们之间可以互相替换，此模式让算法的变化独立于使用算法的客户.

深入理解设计模式（八）：观察者模式(observer pattern) ：在对象之间定义一对多的依赖，这样一来，当一个对象改变状态，依赖它的对象都会收到通知，并自动更新.
08观察者模式.note

深入理解设计模式（九）：模板方法模式(Template pattern) ：在一个方法中定义一个算法的骨架，而将一些步骤延迟到子类中. 模板方法使得子类可以在不改变算法结构的情况下，重新定义算法中的某些步骤

深入理解设计模式（十）：命令模式(Command pattern)：将"请求"封闭成对象，以便使用不同的请求,队列或者日志来参数化其他对象. 命令模式也支持可撤销的操作.

深入理解设计模式（11）：状态模式(State pattern) : 允许对象在内部状态改变时改变它的行为，对象看起来好象改了它的类

深入理解设计模式（12）：责任链模式(Chain of responsibility pattern)：通过责任链模式，你可以为某个请求创建一个对象链. 每个对象依序检查此请求并对其进行处理或者将它传给链中的下一个对象.

深入理解设计模式（13）：解释器模式(Interpreter pattern)：使用解释器模式为语言创建解释器.

深入理解设计模式（14）：中介者模式(Mediator pattern) : 使用中介者模式来集中相关对象之间复杂的沟通和控制方式.

深入理解设计模式（15）：访问者模式(visitor pattern) : 当你想要为一个对象的组合增加新的能力，且封装并不重要时，就使用访问者模式.

深入理解设计模式（16）：备忘录模式(Memento pattern)：当你需要让对象返回之前的状态时(例如，你的用户请求"撤销")，你使用备忘录模式.

深入理解设计模式（17）：迭代器模式(iterator pattern)：提供一种方法顺序访问一个聚合对象中的各个元素，而又不暴露其内部的表示

三、结构型模式

深入理解设计模式（18）：适配器模式(Adapter pattern) : 将一个类的接口，转换成客户期望的另一个接口. 适配器让原本接口不兼容的类可以合作无间. 对象适配器使用组合，类适配器使用多重继承.

深入理解设计模式（19）：装饰者模式(decorator pattern) : 动态地将责任附加到对象上，若要扩展功能，装饰者提供了比继承更有弹性的替代方案.

深入理解设计模式（20）：桥接模式(Bridge pattern) : 使用桥接模式通过将实现和抽象放在两个不同的类层次中而使它们可以独立改变.

深入理解设计模式（21）：组合模式(composite pattern) : 允许你将对象组合成树形结构来表现"整体/部分"层次结构. 组合能让客户以一致的方式处理个别对象以及对象组合.

深入理解设计模式（22）：享元模式(Flyweight Pattern): 如想让某个类的一个实例能用来提供许多"虚拟实例", 就使用蝇量模式.

深入理解设计模式（23）：代理模式(Proxy pattern): 为另一个对象提供一个替身或占位符以控制对这个对象的访问.

深入理解设计模式（24）：外观模式(facade pattern) : 提供了一个统一的接口, 用来访问子系统中的一群接口. 外观定义了一个高层接口, 让子系统更容易使用.

四、七大设计原则

深入理解设计模式（序）：常用的7大设计原则

单一职责原则（single responsibility principle，SPR）：一个类负责一项职责。
开闭原则（Open-Close Principe，OCP）：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。
里氏替换原则（Liskov Substitution Principe，LSP）：一个软件实体如果使用的是一个父类的话，那么一定适用于其他子类，二且他察觉不出父类对象和子类对象的区别。
依赖倒置原则（Dependence Inversion Principle）：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。即针对接口编程，不要针对实现编程。
接口隔离原则（Interface Segregation Principle, ISP）：建立单一接口，不要建立庞大臃肿的接口，尽量细化接口，接口中的方法尽量少。
迪米特法则（Law of Demeter,LoD）：最少知识原则，清掉了类之间的松耦合，降低了系统的耦合度。
组合/聚合复用原则（Composite Reuse Principe，CRP）：尽量使用组合和聚合少使用继承的关系来达到复用的原则