1. 为了“复用”(reuse)目的而是用继承，结局并不完美。
2. Java接口(interface)不具有实现代码，所以继承接口无法达到代码的复用。
3. 设计原则：把会变化的部分取出并封装起来，以便以后可以轻易地改动或扩充此部分，而不影响不需要变化的其他部分。
4. 设计原则：针对接口编程，而不是针对实现编程。
5. 设计原则：多用组合(Has-A)，少用继承。
6. 没有所谓设计模式的库，是一些构造OO系统的隐含经验被收集整理出来，形成了一群“设计模式”。
7. 模式不是被发明，而是被发现。
8. 我们常把系统中会变化的部分抽出来封装。
9. 设计模式汇总

|  |  |  |
| --- | --- | --- |
| **创建型模式** | **结构型模式** | **行为型模式** |
| 简单工厂 | 适配器 | 责任链 |
| 抽象工厂 | 桥接 | 解释器 |
| 工厂方法 | 组合 | 模板方法 |
| 单例 | 装饰器 | 命令 |
| 建造者 | 外观 | 迭代器 |
| 原型 | 享元 | 中介者 |
|  | 代理 | 备忘录 |
|  |  | 观察者 |
|  |  | 状态 |
|  |  | 策略 |
|  |  | 访问者 |

1. 创建者模式汇总: 都是帮助我们创建对象的!
   1. 单例模式: 保证一个类只有一个实例，并且提供一个访问该实例的全局访问点
   2. 工厂模式模式
      1. 简单工厂: 用来生产同一等级结构中的任意产品。(对于增加新的产品，需要修改已有代码)
      2. 工厂方法模式: 用来生产同一等级结构中的固定产品。(支持增加任意产品)
      3. 抽象工厂模式: 用来生产不同产品族的全部产品。(对于增加新的产品，无能为力;支持增加产品族)
   3. 建造者模式: 分离了对象子组件的单独构造(由Builder来负责)和装配(由Director负责)。从而可以构造出复杂的对象
   4. 原型模式: 通过new产生一个对象需要非常繁琐的数据准备或访问权限，则可以使用原型模式
2. 结构型模式汇总: 关注对象和类的组织

|  |  |
| --- | --- |
| 代理模式 | 为真实对象提供一个代理, 从而控制对真实对象的访问 |
| 适配模式 | 使原本由于接口不兼容不能一起工作的类可以一起工作 |
| 桥接模式 | 处理多层继承结构, 处理多维度变化的场景, 将各个维度设计成独立的继  承结构, 使各个维度可以独立的扩展在抽象层建立关联 |
| 组合模式 | 将对象组合成树状结构以表示”部分和整体”层次结构, 使得客户可以统一 的调用叶子对象和容器对象 |
| 装饰模式 | 动态地给一个对象添加额外的功能, 比继承灵活 |
| 外观模式 | 为子系统提供统一的调用接口, 使得子系统更加容易使用 |
| 享元模式 | 运用共享技术有效的实现管理大量细粒度对象, 节省内存, 提高效率 |

1. 行为型模式汇总: 关注系统中对象之间的相互交互, 研究系统在运行时对象之间的相互通信和协作, 进一步明确对象的责任, 共有11种模式

|  |  |
| --- | --- |
| 职责链模式 | 避免请求发送者和接收者耦合, 让多个对象都有可能接收请求, 将这些对象连成一条链, 并且沿 着这条链传递请求, 直到有对象处理为止 |
| 命令模式 | 将一个请求封装为一个对象, 从而使得请求调用者和请求接收者解耦 |
| 解释器模式 | 描述如何为语言定义一个文法, 如何解析 |
| 迭代器模式 | 提供了一种方法来访问聚合对象 |
| 中介者模式 | 通过一个中介对象来封装一系列的对象交互, 使得各对象不需要相互引用 |
| 备忘录模式 | 捕获一个对象的内部状态, 并保存之; 需要时, 可以恢复到保存的状态 |
| 观察者模式 | 当一个对象状态发生改变时, 其相关依赖对象皆得到通知并被自动更新 |
| 状态模式 | 允许一个对象在其内部状态改变时改变它的行为 |
| 策略模式 | 定义一系列算法, 并将每个算法封装在一个类中 |
| 模板方法 | 定义一个操作的算法骨架, 将某些易变的步骤延迟到子类中实现 |
| 访问者模式 | 表示一个作用于某对象结构中的各元素的操作, 它使得用户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 |