可复用性

1. 软件可复用性含义：软件复用是使用现有软件组件实现或更新软件系统的过程。
2. 软件可复用性流程：设计出可复用性的软件->利用设计的可复用的软件进行应用开发.
3. 两种基于软件可复用性实现：
   1. 面向复用编程：开发出可复用的软件。构建可复用的软件,实现复杂；分析各个实现的基本共性操作,实现共性操作。
   2. 基于复用编程：利用已有的可复用软件搭建应用系统。根据具体实现利用已实现的可复用软件实现自己的具体实现。
4. 面向复用的软件构造技术：
   1. 继承与重写( 行为子类型Liskov替换原则)：继承就是子类继承父类。继承要满足行为子类型和 Liskov替换原则。行为子类型: 就是子类继承父类,父类的行为子类一定要实现。在可以使用父类的场景，都可以用子类代替而不会有任何问题；
   2. 重载：方法名相同,参数不同；
   3. 参数多态与泛型编程：因List类型擦除，引出了泛型List<?>—这是一个未知类型的列表 <? super A>接收A的父类 <? extends A> 接收A的子类；
   4. 组合与委托：将功能实现利用接口抛出去,利用别的类实现具体功能。不同的接口定义不同的功能。通过接口组合完成各种操作实现。将接口作为该类的一个属性,使用接口属性使用功能（委托）。
5. 面向复用的设计模式
   1. 创建型模式：

a.工厂方法模式：当client不知道要创建哪个具体类的实例，或者不想在client代码中指明要具体创建的实例时，用工厂方法。定义一个用于创建对象的接口，让其子类来决定实例化哪一个类，从而使一个类的实例化延迟到其子类。相当于工厂和产品，产品不直接使用产品的构造方法实例，通过工厂提供的方法实例化；

b.抽象工厂模式：提供接口以创建一组相关/相互依赖的对象， 但不需要指明其具体类。创建的不是一个完整产品，而是“产品族”（遵循 固定搭配规则的多类产品的实例），得到的结果是：多个不同产品的object，各产品创建过程对client可见，但“搭配”不能改变。本质上，Abstract Factory是把多类产品的factory method组合在一起；

* 1. 结构型模式：

a.适配器模式（委托实现）：将某个类/接口转换为client期望的其他形式。通过增加一个接口，将已存在的子类封装起来，client面向接口编程，从而隐藏了具体子类。将已实现的功能通过接口满足client实现；

b.装饰器模式：抽象接口,共性操作。用每个子类实现不同的特性。如果需要特性的任意组合,对每一个特性构造子类，通过委派机制增加到对象上；

c.外观模式：提供一个统一的接口来取代一系列小接口调用，相当于对复杂统做了一个封装，简化客户端使用。各个小接口抛出，具体功能子类实现，外观模式提供统一的接口，根据参数选择不同接口的操作；

* 1. 行为类模式：

a.策略模式（委托实现）：为不同的实现算法构造抽象接口，利用delegation，运行时动态传入client倾向的算法类实例；

b.模板模式：两种实现：（a）黑盒框架：用户不知道内部实现，总体框架已经实现，将需要实现的方法通过接口抛出，子类具体实现，再通过委托传给总体框架；（b）白盒框架：直接继承父类，继承重写需要实现的功能；

c.迭代器模式：客户端希望遍历被放入容器/集合类的一组ADT对象，无需关心容器的具体类型。Iterator pattern：让自己的集合类实现Iterable接口，并实现自己的独特Iterator迭代器(hasNext, next, remove)，允许客户端利用这 个迭代器进行显式或隐式的迭代遍历；

1. 实验中的可复用性实例

在学生信息管理系统的实验中，我们几乎将所有函数模块化，经过层层调用来实现某些功能。

一旦要修改某些基础功能时，仅通过修改最终调用的函数就可以改变与这个函数相关的所有功能。

当我们相信加一个新的功能时，可以通过调用已有函数和新加少量函数，并将它们组织起来，就可以实现我们想要的功能。