原有代码的问题：

原有代码每一个answer都设置了一个类，然后我们发现这些类很大一部分是具有相同结构的，这样单独设置类会使代码过于冗杂；此外原有代码的Question 的设类，继承关系混乱，内部的方法存在重复,没有使用到的情况；IO 类内部的方法一旦添加一个Page，它采用了全部读入，全部写入的方式，效率不高,源代码将一个Page完全写入了xml，不仅过程比较复杂，而且读取也不方便，然后我们采用了object存入的方式，读写都很方便；源代码将两类page------survey 和 test 的info全部写入一个pageinfo.Xml文件内，不便于区分，判断效率不高；源代码用0~5数字表示题型，现在我们增加了QType的枚举类型，方便记忆和判断；源代码存在上帝类control类，由他处理一切与命令行有关的输入输出的，代码冗长，不方便理解和修改

项目分工：

|  |  |
| --- | --- |
| 姓名 | 工作 |
| 杨安逸 | 负责Command的构建和模式的应用、设计类图 |
| 吴晓岩 | 后台数据，基础类的搭建以及IO、设计类图 |
| 魏娜 | GUI和文档 |

.

模式的使用及理解：对源代码进行重构使，我们采用了命令模式。命令模式定义是将命令封装后进行传输。模式有Client,Invoker,Command,ConcreteCommand和Receiver这五个成员。整个模式中,Client是下令者，它知道自己的命令，知道完成命令的对象，Invoker是命令的传递者，它负责调用相应的命令，Receiver是真正执行命令的部分。Client根据自己的需求创建ConcreteCommand的实例并指定完成命令的Receiver(即创建Receiver的实例)，将两者绑定后，再让Invoker去调用ConcreteCommand完成命令。Command是一个接口，所有的ConcreteCommand都是继承这个接口的，Invoker中用Command代替所有的ConcreteCommand，使得新的命令很容易地被添加到系统当中。这是命令模式的优点之一，此外，命令模式把发出命令的责任和执行命令的责任分隔开，使得发出命令的一方不必知道执行命令的一方的接口，更不必知道命令是怎么被接收，以及命令是否被执行，何时被执行，怎么被执行的，让两者解耦。命令模式还能较容易地设计一个命令队列，但缺点是会导致有过多的ConcreteCommand类。在本次lab中，我们将GUI当作是Client，IO、Page、Question等类当作Receiver，用一系列ConcreteCommand将Control的功能分化，从而达到解耦以及消除上帝类的效果。

GUI实现：

首先是GUI图形界面代码：根据lab需求，首先是主页面提供11项选择，Create ,Display,Modify,Take,Outcome,Quit 前五个选项分为test和survey两部分，create之后会进入选择题型页面，提供了六种题型，点击每个选项后会分别进入各个题型页面，完成创建page，display首先会提供一个可供选择的列表，点击后进入显示页面；Modify同样提供了可选择列表，选择后一方面进入添加题型界面，另一方面进入修改单个原有题目的界面，提供题目，答案，选项，分数的修改；Take即进入考试界面，登记后选择page进行作答，test提交后会立刻给出分数；outcome界面则是对已经做好page数据的统计，答案的选择数量等。

遇到的问题和解决：

原有代码的理解上：首先源代码存在bug 无法正常运行，对流程的梳理上造成了困难，另外代码量巨大，不容易理解各个部分的关系。需要一定的时间进行理解。

代码交互：当各自成员分别完成代码，进行整合的时候，首先是调用方面的使用，进行添加，另外是数据传输上的格式一致性，这都需要进行统一，我们在这一部分花费了较多的时间和精力。

心得和体会：

首先这个lab工作量比前两个工作量要大，实现起来也比较复杂。要求多，进行一一实现，有时还会出现新的漏洞。在这个lab中，我们对GIT确实是有了极为深刻的理解，对于conflict的处理等问题，此外还有对分工合作上，比以往更加明确，更加的相对独立，达到很好的分工效果。