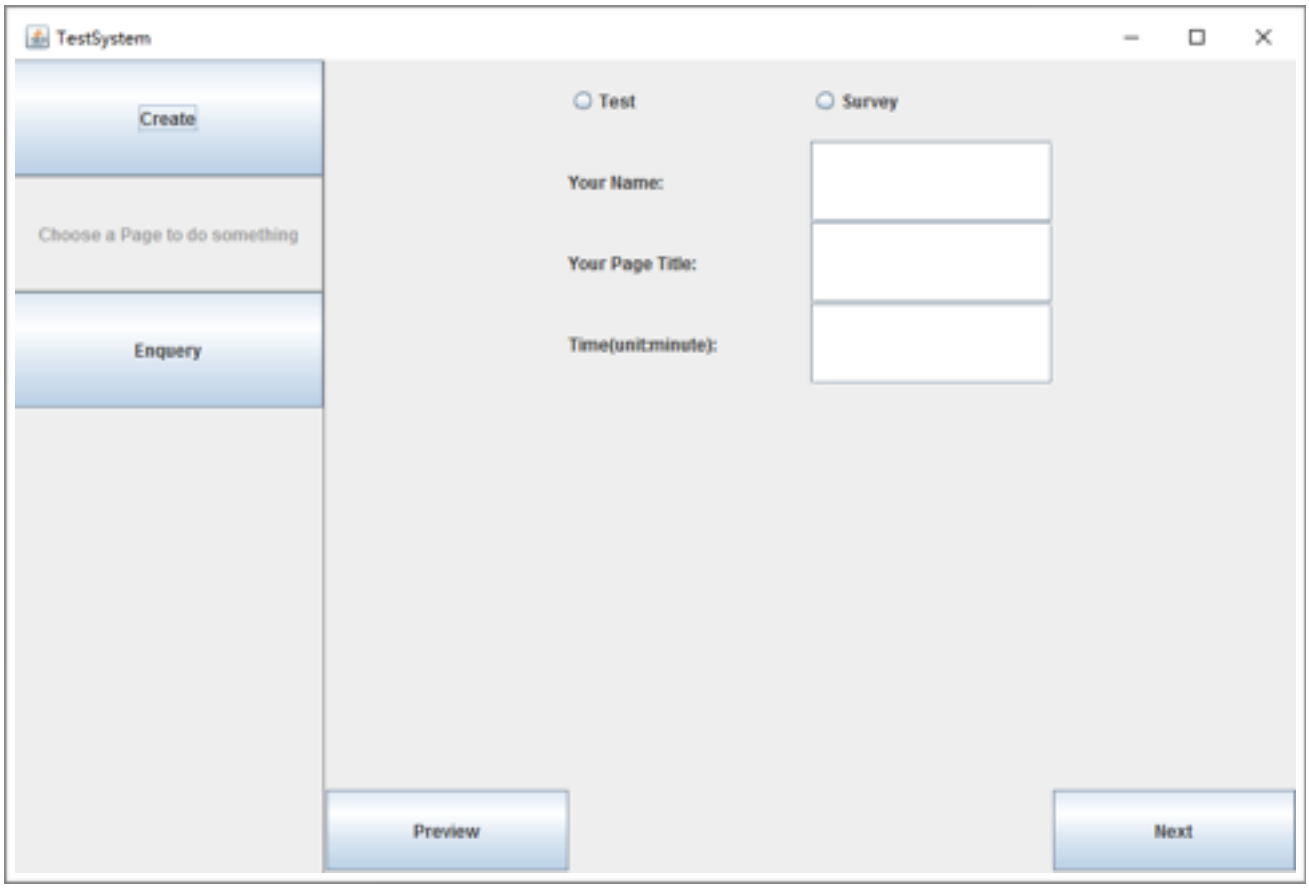
TestSystem

Lab3: 软件设计与重构

14ss505 - https://github.com/14ss505/lab3_1



组长：周俊颖　*14302010058*

组员：袁梦梦　*14302010063*

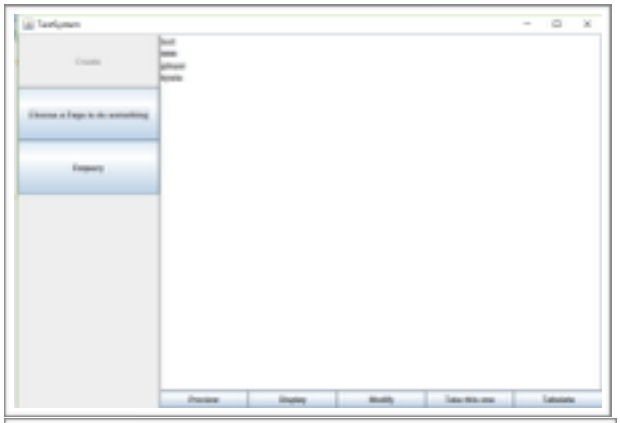
邢晓渝　*14302010053*

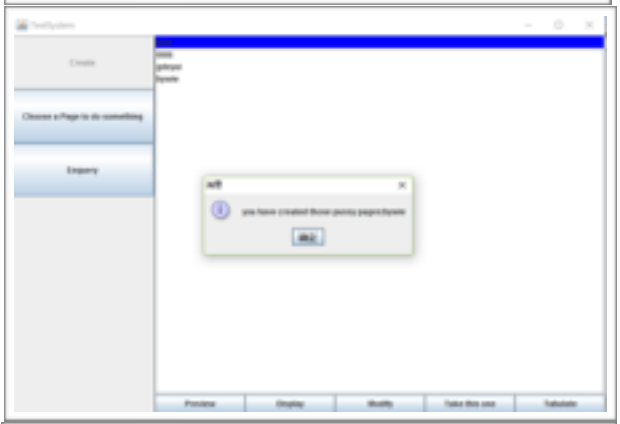
马叶舟　*14302010052*

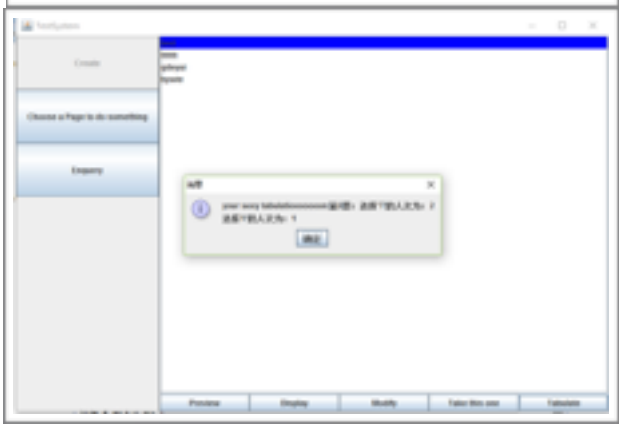
简介

从基本的Model逻辑进行重构，对于父子类的data field , methods进行合理调整，使得结构更为清晰，避免冗余代码和空指针报错；使用Command模式实现菜单的各种功能，完美将此模式融合于MVC模式中的C部分；对后台数据存储即IO类进行适当修改以方便我们读取xml数据。

组内成员分工		
周俊颖	GUI	
袁梦梦	数据存储	接口设计
邢晓渝	代码重构	设计类图
马叶舟	代码重构	文档





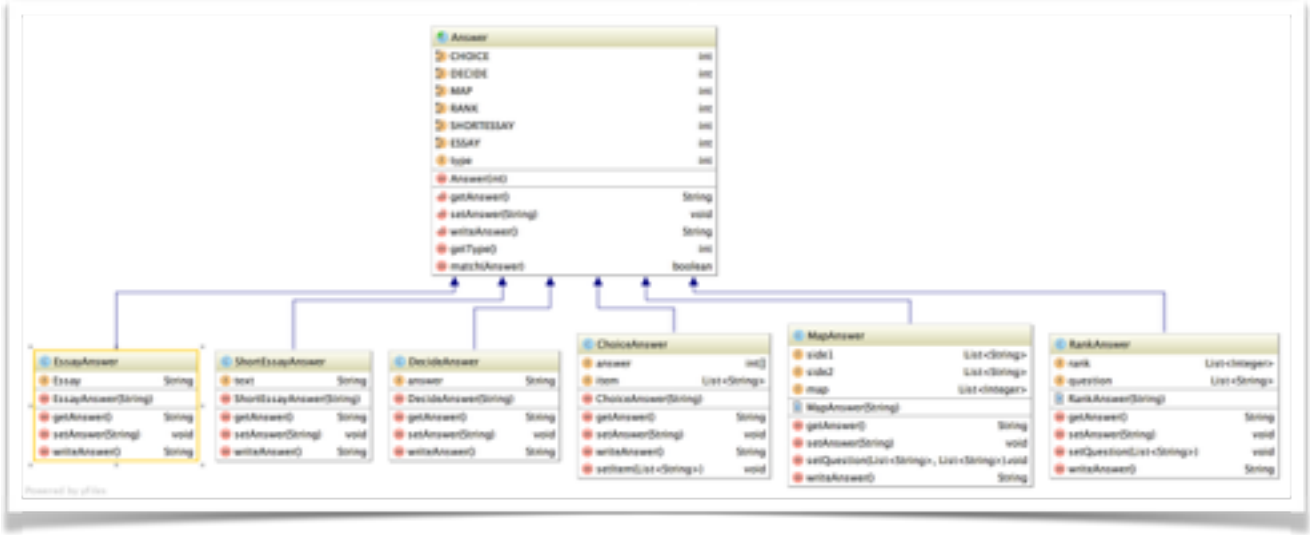


目录

一、重构

1. Model基础重构

- Answer



i. 在基类中采用常量标注各个子类类型，方便其他类（尤其是控制类）的使用

ii. 父类构造函数即传入类型值，该构造函数在子类构造函数中调用

iii. 子类构造函数显示传入String类型答案，调用setAnswer()设置属性

iv. 父类data field只存在type，剔除answer，因为各子类answer格式迥异

v. 统一除getAnswer(), setAnswer(), writeAnswer()需要根据不同题型进行不同实现外，其余方法均由父类实现，子类继承自父类。因此父类中此三者为抽象方法，强制子类必须实现自己的功能

vi. 除MapAnswer外其余子类均将答案转换为String进行判断，接口一致，故而可将match(Answer)统一实现，删除冗余代码

vii. MapAnswer考虑到答案内部可能存在空格，将导致setAnswer()内部逻辑中使用空格分隔出现错误，故而改为直接match数组，成为特殊类

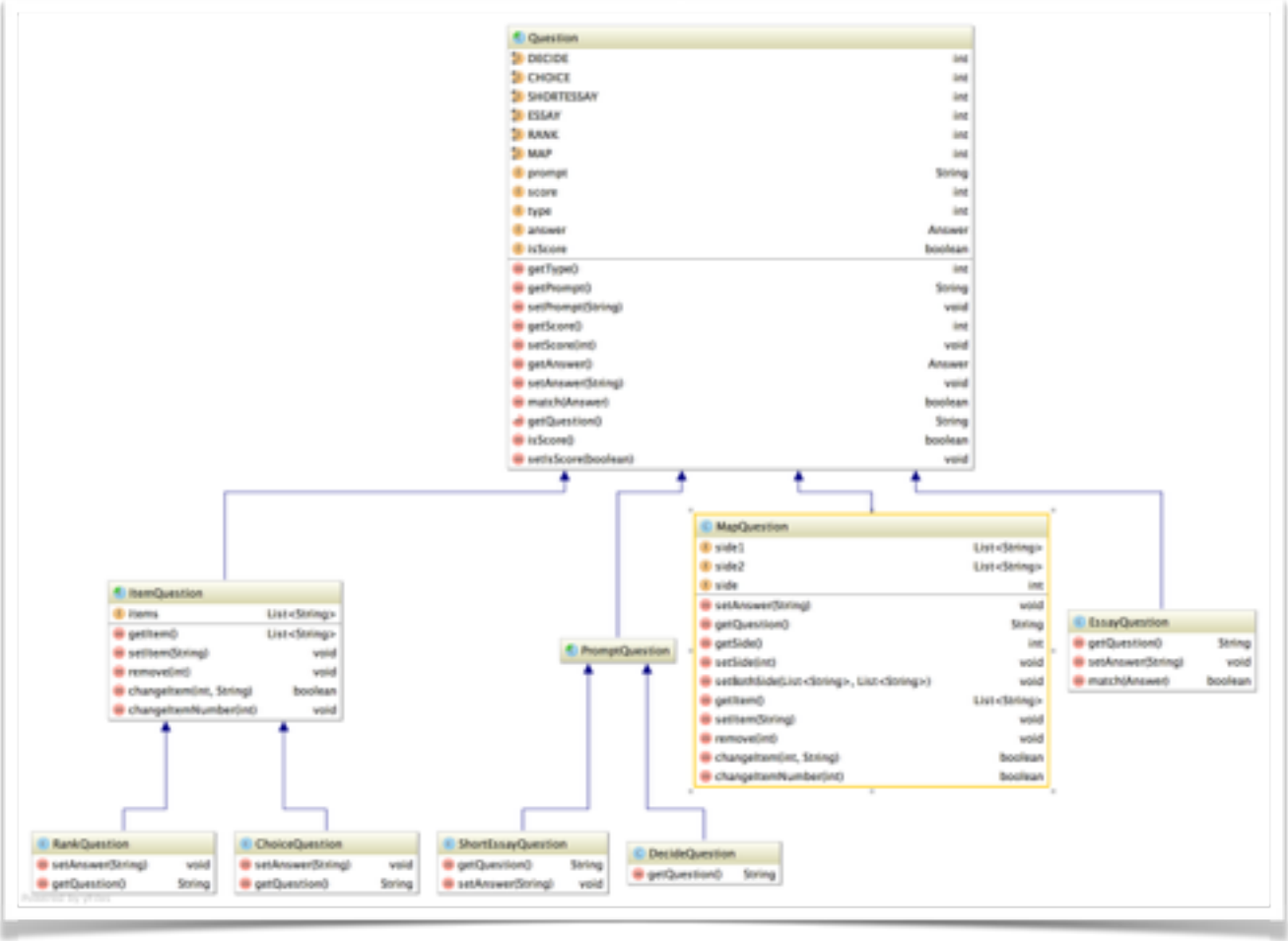
- Question

i. 同样在基类中采用常量标注各个子类类型，方便其他类（尤其是控制类）的使用

ii. 父类构造函数传入类型值，该构造函数在子类构造函数中调用，子类构造函数除此之外设置显示传入的各类参数为属性值

iii. 基类data field包含题干、类型、答案和分数，子类不再重复拥有以上属性

iv. 子类构造函数至少具有两种，一个为survey提供，不包含答案，另一个为test提供，需要传入标答



v. 除getQuestion需要各个子类实现自己的方法，其余getter & setter均有父类实现，删除冗余代码

vi. ItemQuestion中另外包含属性item为题目选项，对item的设置与获取也在此抽象类中全部实现，子类不再重复实现

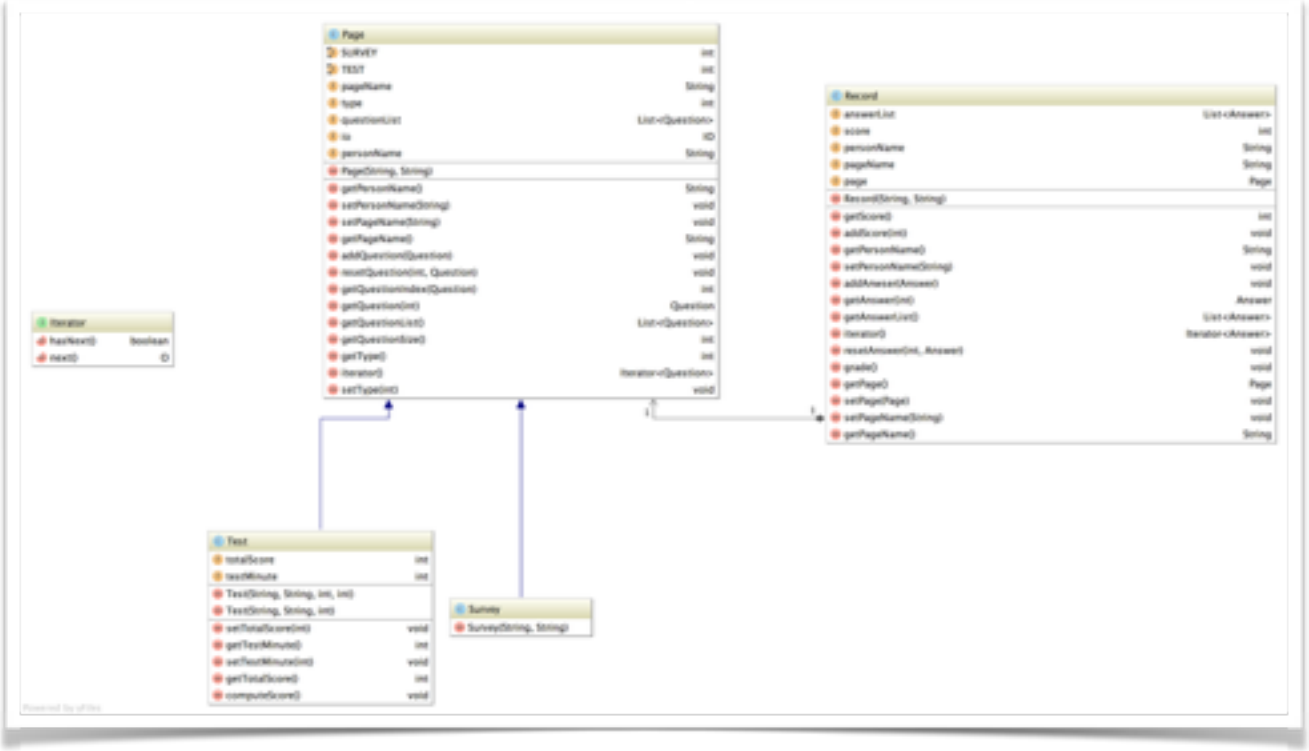
vii. 子类中除特殊连线题外不再多加属性，也只有getQuestion()和setAnswer()，仅少量代码

- Paper

i. 同样在基类中采用常量标注各个子类类型，且改变原代码中数据类型String为int，方便比较

ii. 父类中已包含各类属性及设置属性的方法，只需要Test类中增加分数和计时器，Survey中仅需要构造器，节省大量代码。主要原因为Test和Survey的差异仅在于表达和计分，在Question的依赖关系上均相同

iii. 为方便控制类的调用，父类中增加了相关接口，Page类实际就是Question的载体，主要功能也是对于Question的操作



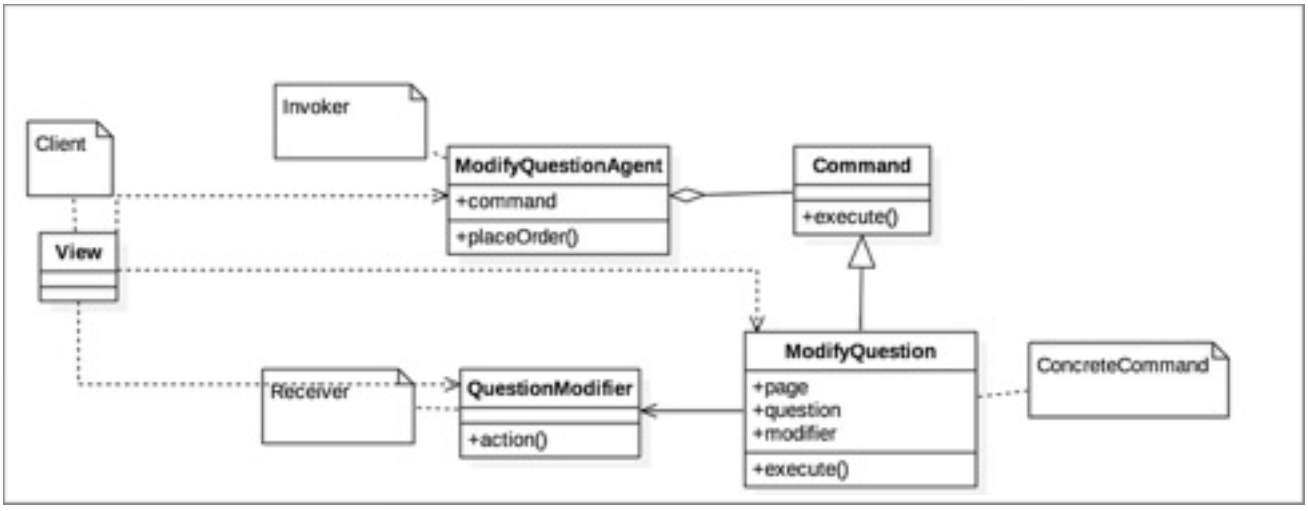
iv. Record中包含Page，强聚合一对一，但是Page独立，不包含Record

v. 计时功能放在Test中而非Record中作为Test的固有属性

vi. 分数属性在Test中为答卷总分，而在Record中为做题者得分，Record中完成批卷功能

vii. 分离清晰的结构使得后期增进变得容易，只需要在data field中增加计时器和做/出题者姓名即可

2. Control重构



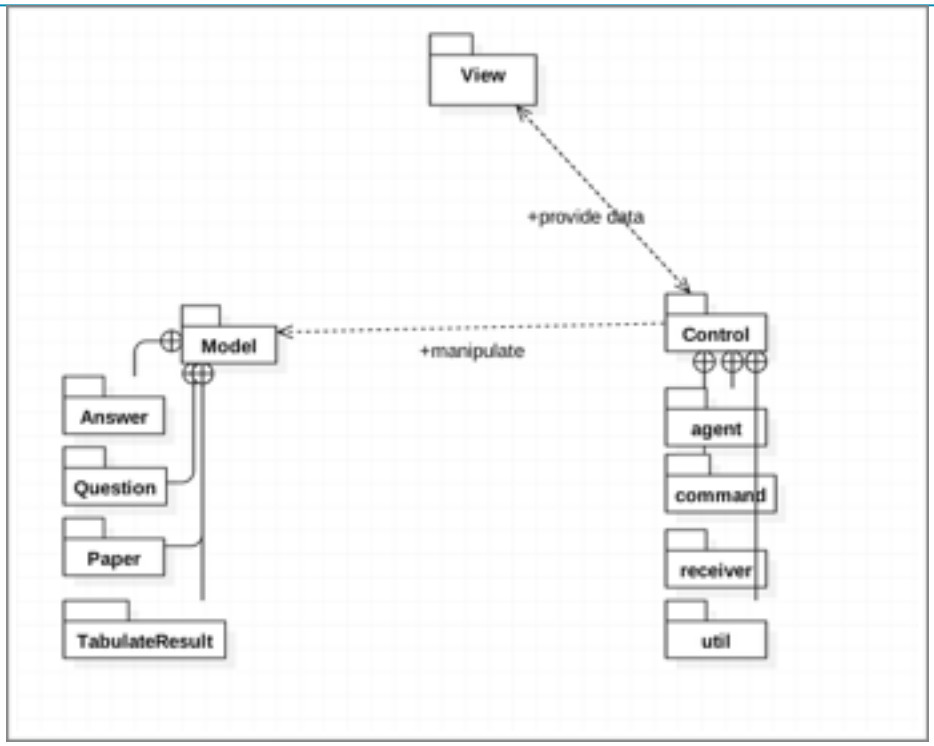
❖ Command模式[1]

- GUI中一旦某个button被点击Listener就会生成对应的invoker和receiver，再根据表单数据生成指令
- 指令中封装好了数据，将数据作为参数传入receiver的真正执行方法action()，既保持了Command接口的execute()零参数方法，使得invoker不用考虑细节，保持独立状态，又使得receiver可以多次复用
- 由于菜单指令过多，不同类型指令之间操作、封装数据迥异，基本不可复用，故而我们采用的Command模式并非对于所有指令进行统一复用“卡槽”，而是进行了基本分类

	AddQuestion	AddAnswer	ModifyQuestion
invoker	AddQuestionAgent	AddAnswerAgent	ModifyQuestionAgent
receiver	QuestionCreator	AnswerCreator	QuestionModifier
concrete command	AddChoiceQuestion	AddChoiceAnswer	ModifyChoiceQuestion
	AddDecideQuestion	AddDecideAnswer	ModifyDecideQuestion
	AddEssayQuestion	AddEssayAnswer	ModifyEssayQuestion
	AddMapQuestion	AddMapAnswer	ModifyMapQuestion
	AddRankQuestion	AddRankAnswer	ModifyRankQuestion
	AddShortEssayQuestion	AddShortEssayAnswer	ModifyShortEssayQuestion

- 如此针对不同题型invoker和receiver实际复用同一个，我们将所有invoker放在package agent中，所有receiver放在package receiver中，而command包括接口、基类、子类全部放在package command中，此包中又另分小包，结构清晰
- AddQuestion操作实际是在创建page的时候调用，CreateTest/Survey两个类实际只创建了承载问题的容器，之后一道题一道题地添加问题，采用这样的逻辑一是直接继承自原代码的模式，使GUI后的版本更具有兼容性；二是方便其他操作如修改问题，之后展开论述；最后则是方便GUI布局复用，回答问题和修改问题都是同样只针对一道题
- AddAnswer操作实际是基于Record的操作，每一次答题就会进行保存，保证了由于各种不可抗力中途退出程序也能够及时保存数据，不会数据丢失，从头回答
- ModifyQuestion操作基本舍弃原代码的结构，原代码CommandView中繁琐的操作无法与GUI版本相容，为了更加便利，我们实际每一次修改一道题时将原题从Page中用新题完全覆盖，而非具体修改题目的某个部分，如题干、某个选项内容、调整选项序号等，由于Model的独立性及各类间的结构清晰，代码实现比之之前的各类set反而更为容易。这样用户可以一下子修改题目的各个部分，用户体验也更好

[1] 以修改一道题为例



❖ **MVC模式**

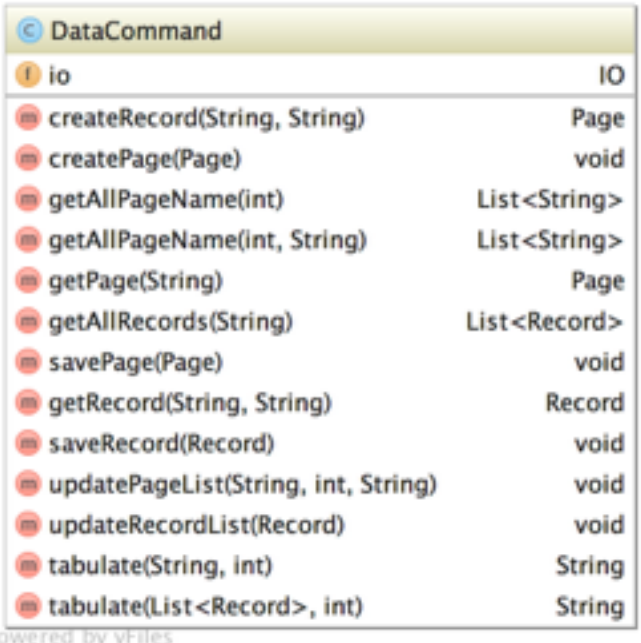
View得到用户输入，通过注册在各组件上的Listener生成相应操作control类，control类对Model进行操作，每一次操作进行了实时存储，故而View只需要读取后台存储数据（封装好的DataCommand控制类）即可，由于我们的特殊设计，Model不会直接对View产生操作，下文将展开叙述

3. 后台数据操作重构

- DataCommand

原代码中所有数据的操作均有IO类提供的接口直接调用进行，我们增加了DataCommand类作为控制类，视IO为底层实现，DataCommand才是封装好的接口方法，但凡读取数据均可通过该对象进行。

在上述各类receiver的方法中实际均采用DataCommand方法，使得IO对象只存在于该类



- XML文件

- pageInfo.xml：在每一次创建新Page时更新
- #pageName#.xml[2]：创建一个新的page时后台自动生成初始化文件，包含以下节点：personName,type,（totalScore,testMinute）[3]questions<question>

  在存储question时，根据questionType，跳至相应的save方法

  question节点包含以下子节点：questionType,pageType,isScore(是否可以自动打分)answer(String),score,其中answer,score是pageType为test时需要存储的
- #pageName#-recordInfo.xml：创建一个新的page时后台自动生成，在一个用户做page时自动更新，包含以下节点：recordName，子节点为pageName-personName
- #recordName#.xml：在用户开始回答卷子时自动创建初始化文件，包含以下节点：Score，answers

  经优化后pageName,personName本身可以从recordName中取得，不用另存

[2] ##之中表示被相应的string类型参数替换

[3] Test独有

二、困难与解决

前言：

最初分工确定为三人解耦合，一人GUI，两边并行开发，之所以这样分工的原因是GUI任务量大，等待重构、确定接口之后工作恐怕来不及，况且重构需要先熟悉所有原代码，学习各类模式，尝试进行初步解耦合，而一个人事先GUI可以先将基本界面画好，并保持风格一致。但过多的人进行解耦合在助教已经明确建议命令模式、MVC模式与观察者模式的情况下，不仅无法头脑风暴、集思广益（模式最多应该是这三种，过度设计应该避免），反倒造成前期意见难以统一，分工解耦合造成接口迥异，对Model的改动引起了git工作的merge冲突。而GUI和解耦合并线独立操作，没有实现完善沟通（预备会议时由于时间紧凑，大家采用敏捷开发，导致细节没有考虑，直接开始实现），导致解耦合针对的是CommandView，而命令行操作和GUI界面操作实际对于控制流的要求存在不小差异，逻辑需要改变。中期意识到这一问题后再度重构，导致重构实际消耗了两倍力气，且第二次重构虽然已经改善控制流逻辑，但是对GUI接口没有考虑完善，导致提供接口出现不符合MVC模式的错误，对方法进行重新修改，这一部分最为劳神伤民，负责GUI的同学被迫需要多次修改参数列表和相关操作，进度条被大幅度放缓。

1. 版本兼容

CommandView最大的特点即为同一时间只能进行特定某项操作，且操作颇为具体，而GUI界面丰富直观，直接点击即可控制逻辑的变化，不需要输入。

舍弃CommandView意味着相应的修改问题、新增问题、保存page操作均应进行修改，虽则Model的方法不变，但是control类的逻辑需要大改，这也是Git上显示工作流从master分支重开三条以“2”结尾的分支原因，此处值得庆幸的是GUI分支始终独立于解耦合的分支，没有进行过merge，否则GUI也需要另开一路。

另外CommandView可以直接舍弃而非前后版本保持兼容也节省了我们修改框架的时间，我们可以直接颠覆，不必考虑原本的模式

2. 接口的参数

首先在Page的传参中我们先后使用了直接在View中保存Model即Page对象，以及通过传参PageName等String类型参数，后在控制类中通过读取后台数据获得Page两种模式。前者符合MVC模式直观便利，后者保证了在断电等突如其来的不可抗力因素下数据的稳定性，两者各有利弊，但是团队合作最重要的因素是保持一致性，尽量减少返工，因此我们在历史情境的影响下选择了改动更少的情况。这里最为直观地体现了我们团队合作的弱点。

而最大的问题在于Answer、Question的操作，由于这两个对象属性复杂，给前台传参造成很大难度，前台在看到对外接口的时候由于不清楚后台Model具体实现，容易造成传参

的错误等潜在逻辑BUG。面对此类问题的有两种解决方法，一是修改Model以贴合View，使得View能够更为直接地传参，而是在View中采用Model后台处理模式对前台获得数据进行预处理。显然在时间充足而不需要考虑版本兼容性的情况下，前者更为推崇，因为这样的接口才是理想的不需要前端看到后台逻辑的接口，但是由于版本兼容的需求在中期之后才明确，重构过程中只进行了控制流的分离，没有改变Model逻辑设计，导致中后期才发现的此类问题只能折中解决。这里体现了现实因素（主要是时间）对程序的最大影响。

3. 多种模式的融合与解耦合

命令模式需要Client生成Invoker和Receiver，而GUI中Client就是前台View，如果在View中直接生成，是否意味着Control逻辑耦合入了View？最初出于这样的考虑，我们将这一部分的代码挪出View，但很快我们便发现这将出现类爆炸的现象。针对每一个Panel的Button，我们几乎都需要一个独立的控制类，这不仅在开发人员寻找类时劳神费力，也没有实现解耦合真正的意义，因为这只是纯粹地代码迁移罢了。

我们反观JAVASwing组件的Listener作用，为什么许多程序员往往采用匿名类的方式来给组件注册监听？实际Listener就是控制类，而且是更为便利的控制类，作为inner class，它可以直接获得组件，也明确是哪一个组件引发的事件，不需要再传参。前辈丰富的经验下采用的匿名Listener被我们舍弃，而另外创建大量控制类，实是焚琴煮鹤，因此我们做出一定妥协：在Listener中生成Invoker和Receiver以及Command，而将Command的实际操作采用命令模式在Control部分完成——实现MVC与命令模式的完美融合，即可复用大量代码，有将Model和View完全隔离，由Control进行控制。

另外，View和后台数据的直接传输，由于DataCommand的存在，数据读取已经被封装成简单的方法，前端可以直接读取，不需要再绕过Control。需要注意的是，后台数据和Model概念完全不同，View只是通过DataCommand读取数据，而若要对Model进行修改，则修改之后不需要调整View（因为原本修改就是在表单操作，表单保持原样即可），因此我们不需要观察者模式。