山东大学 软件 学院

高级程序设计语言 课程实验报告

学号:202022300320	姓名： 时书桓	班级： 2020 级软件 7 班
实验题目：考试平台的设计与开发		
实验学时：32	实验日期： 2021.5.14	

实验目的：

锻炼学生对高级程序设计语言的应用开发能力，结合 Java 的 GUI 技术，数据库技术，网络技术，多线程技术，图形技术等实现一个综合性的应用系统。

硬件环境：

Intel Core i5-10300H + 8G RAM

软件环境：

Windows 10 20H2
JDK 14.0.2
IntelliJ IDEA 2020.2.2 (Community Edition)

实验步骤与内容：

注：虽然在这次课程设计中既负责了教师端的开发也负责了部分学生端的设计,但这份报告只讨论教师端内容。

这次课程设计主要分为以下几个阶段进行：

第一阶段：UI 设计

这一阶段主要以 UI 的设计为主要任务，同时兼以构建程序框架的任务，在此阶段中，主要完成了与窗口有关的程序包（详见技术报告中的“项目架构”部分）的开发，这部分的开发工作较为繁琐，因此占用了较多的时间。

第二阶段：事件预装载

此阶段主要对第一阶段中的各种事件进行了初步装载，使得窗口事件与中间层进行联系，同时借此预留中间层的相关方法的定义，明确中间层需要为应用交互层提供哪些服务。这部分开发较为简单，仅使用了极少的时间，但由于部分定义不清晰，对后续开发造成了一定影响。

第三阶段：数据库连接

这一阶段的主要任务是将程序与数据库进行连接，同时将中间层预先定义的与数据库相关的方法进行实现。这部分与数据库息息相关，因此相较于上几个部分较为复杂，不过所幸期间并未遇到较大障碍。

第四阶段：网络与文件 IO 连接

此阶段主要完成了程序的网络连接以及文件 IO 相关的功能，同时实现了中间层与之相关的方法。这一阶段由于制订学生端与教师端的网络通信规则时进行了几次修改，导致开发进程较为缓慢，不过由于中间层隔离机制的存在，并没有对已经完成的应用交互层程序进行修改。

第五阶段：双端联调

这一阶段主要进行了教师端和学生端的联合调试，调试主要集中在二者的网络通信方面，由于学生端进度远远慢于预期，导致这一阶段不得不进行推后，对开发日程造成了一定影响。

结论分析与体会：
由此次课程设计不难看出，对于中等及以上规模的程序开发工作，分阶段进行、制订合理的日程表，提前协调不同部分的通信规则等工作都是至关重要的。另一方面，程序规模扩大后不可避免的会出现更多的 Bug，因此在各个阶段进行合理的测试与修改都是不可或缺的。
同时，由于中型或以上规模的程序开发基本都是由团队进行的，因此在团队中进行合理的分工并随时进行沟通也是必要的，要尽量避免因为单个成员拖慢整体开发进程的情况出现。

附：技术报告

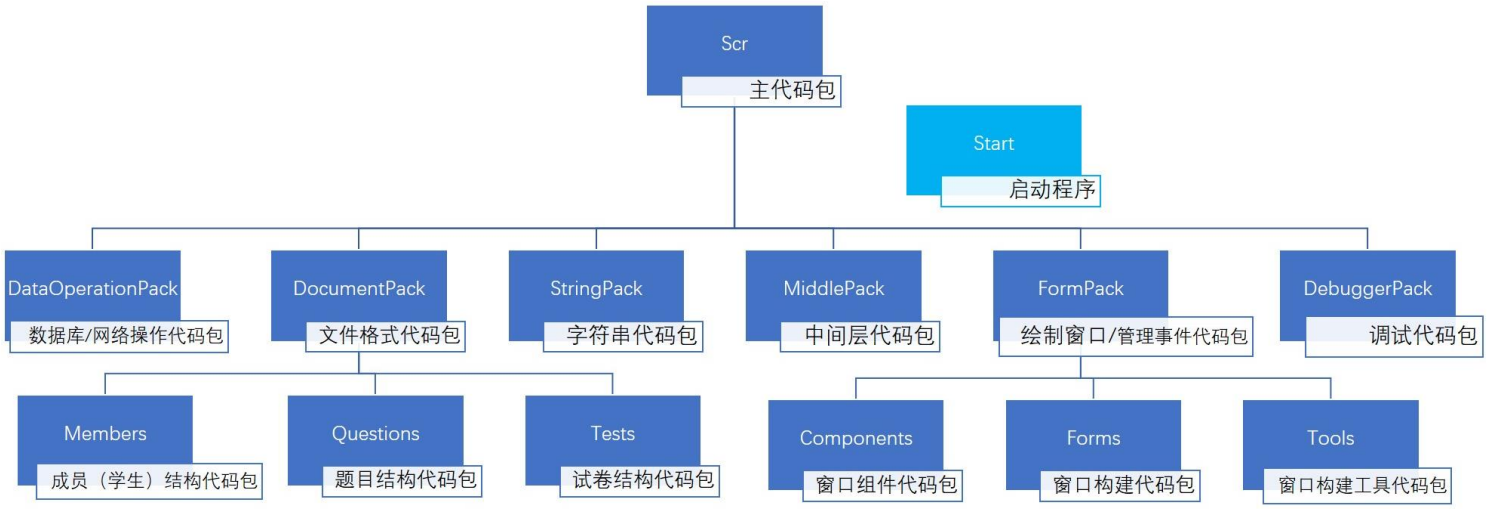
一. 系统模块架构

1. 宏观架构
此次教师端的设计使用了典型的分层型设计，分层结构如下图所示：

为了实现应用交互部分与底层的数据操作隔离，在进行教师端的设计时添加了一个中间层。这个中间层提供了供交互界面发送接收数据的一系列方法，保证了交互界面不会直接和图中的底层操作发生联系，避免了底层操作方法的修改对交互界面代码产生影响，这虽然对于实际使用程序没有影响，但对于提高程序的开发效率、降低程序开发协调难度起到了巨大的作用。
由于这个设计的存在，整个程序被分为了五个开发部分：交互界面部分、中间层部分、数据库部分、网络操作部分以及文件 IO 操作部分。除了中间层部分，其他几个部分都是互不相关、没有直接联系的（这里的“直接联系”指相互之间方法的调用或者对象的使用），与之相配合的，它们都是与中间层有着直接联系，并且所有的间接通信都是通过中间层进行的。因此，中间层是整个程序运行效率的瓶颈所在，这次课程设计由于目标场景数据流不会太大，因此并未对中间层进行特别的设计。事实上，在面对超大流量的数据时，中间层应当进行相关的优化。

2. 程序总架构

程序的总体架构如图所示：



每个代码包的作用都在图中进行了说明，这里不再赘述。需要注意的一点是，整个调试代码包在程序正式运行时都未被使用，因此下文中不对调试代码包进行介绍。

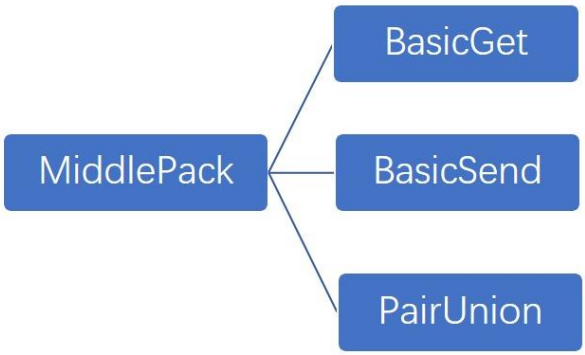
另外，按照先期设计，程序应当能够读取 xml 表并对相应的字符串进行配置，借此使得程序能够切换语言版本，但由于开发时间所限，这个功能被删除了，不过用以实现这个功能的字符串代码包保留了下来。程序通过这个代码包实现了字符串的统一管理，使得修改程序中的字符串（例如：窗口标题、按钮名称等）变得十分容易。

除此之外，由于这个程序对于文件 IO 的操作极少，对文件进行操作的部分被直接整合成了中间代码层中的一个方法。

3. 程序模块结构

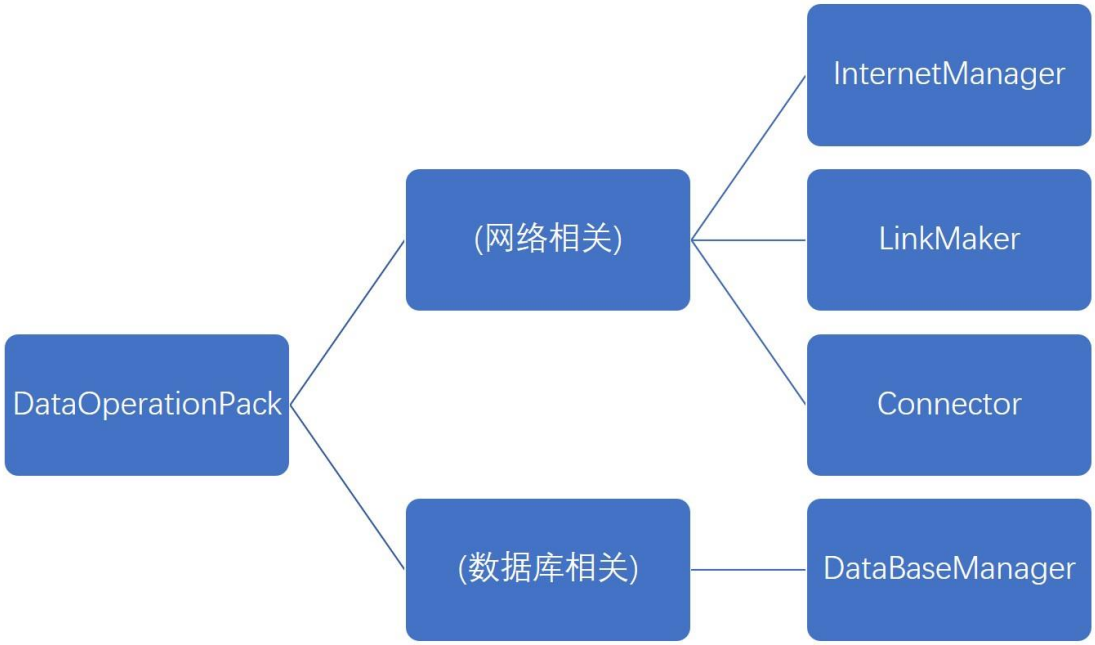
从上一部分的架构图中可以看出，除调试代码包外，程序由五个主要模块（代码包）组成。这一部分将对每个模块的内容进行简单的介绍。

中间层代码包结构如图：



可以看出，这个代码包的结构十分简单，其中的 Basic Get 类用以实现交互界面获取底层数据的功能，而 Basic Send 类则用以实现交互界面向底层发送数据的功能。Pair Union 类是一个单纯的工具类，它仅具有一个静态方法（synchronize Pair）用来使得一个 Basic Send 对象和一个 Basic Get 对象进行配对，同时使二者连接相应的数据库/网络管理对象。

数据库/网络操作代码包结构如下：

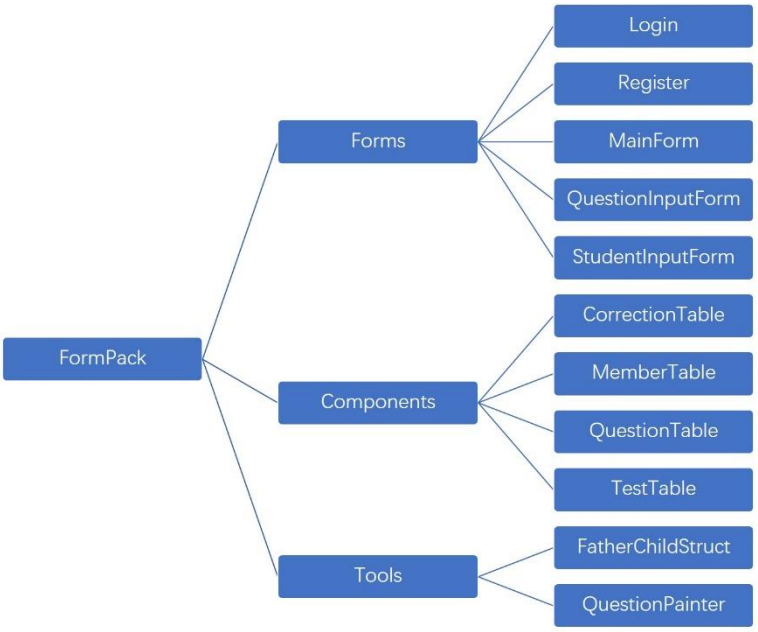


与网络相关的类共有三个，Internet Manger 类是用来进行网络资源管理的主要类；Link Maker 类用来监听端口并建立与学生端的连接；Connector 类用来进行与学生端的通信。一般情况下，一个 Internet Manager 对象对应一个 Link Maker 对象，和多个 Connector 对象。这三个类中，只有 Internet Manager 类与中间层进行直接的数据传输。

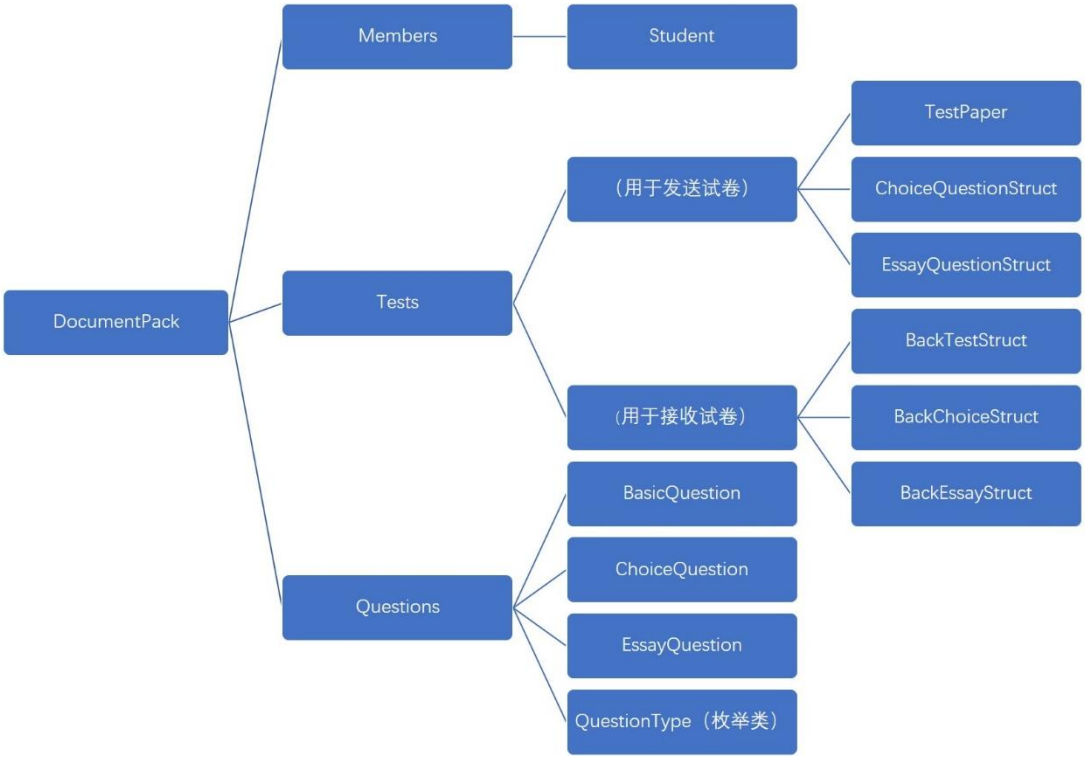
只有一个与数据库相关的类，即 Database Manager 类，这个类只能负责与单个数据库的所有通信，这意味着在连接多个数据库时需要相等数量的 Database Manager 对象。

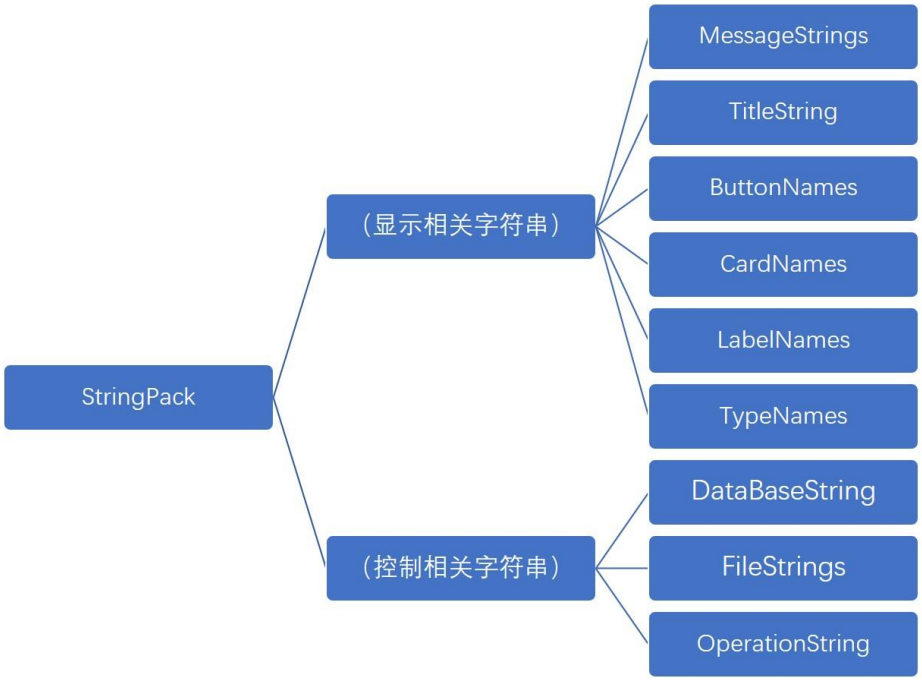
另外，对应中间层而言，一个 Basic Get -Basic Send 对只能同时连接一个 Internet Manager 对象和一个 Database Manager 对象。

绘制窗口/管理事件代码包结构如下：



Forms 子包为程序的 5 个主要窗口（不包括提示相关信息的对话框），它们按照顺序分别是登录窗口、注册窗口、主窗口、题目添加窗口和学生添加窗口。其中登录窗口、注册窗口和主窗口是互斥的（即不会同时出现），而题目添加窗口和学生添加窗口则是可以和主窗口同时出现的。
Components 子包为主窗口中四个主要面板，它们按照顺序分别是批改面板、学生管理面板、题目设计面板、试卷设计面板。
Tools 子包为两个工具类，它们均被题目设计面板所使用，这涉及到了本次课程设计的第一个技术难点，它们将会在下一部分被重点分析。
字符串代码包和文件格式代码包的结构分别如下：





这两个代码包均为数据存储结构，因此不进行详细介绍。

二．主要技术难点

这次的技术难点以窗口绘制中的题目设计面板绘制为例进行。

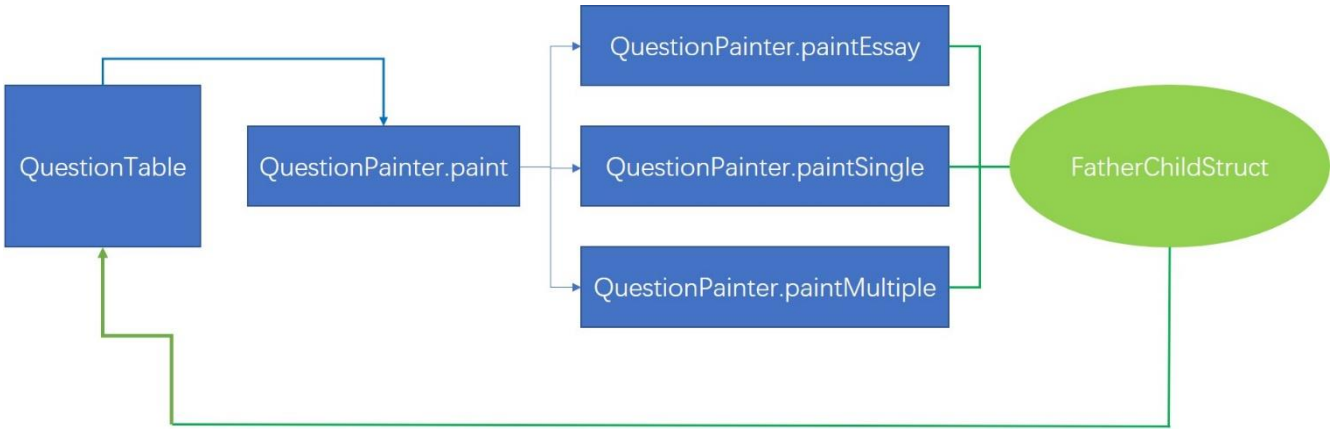
通常而言，一份试卷中一般会存在单选题、多选题以及简答题三种题目类型（可以将判断题看作只有两个选项的单选题），这三种题目所需要的绘制方式存在极大的差异，例如，单选题需要绘制单选按钮组件而简答题则不需要这一组件。

这种差异的存在使得题目设计面板的实现难度大幅提高，需要提供额外的工具来进行设计，而这两个工具就是 Father Child Struct 以及 Question Painter。

Father Child Struct 类是一个对象集，它在不同的情况下会存储不同的窗口组件对象。这个对象集由 Question Painter 类生成并提供给题目设计面板进行下一步的绘制。

Question Painter 类是一个只提供了多个静态方法的工具类，在进行题目绘制时，Question Painter 类的 paint 方法会根据提供的题目类型选择对应的方法进行组件的生成与设置，而对应的方法则会返回一个 Father Child Struct 对象给他们设计面板。

整个流程如下图所示：



具体的代码将在下一部分给出。

三．关键代码

1. Father Child Struct 类以及 Question Painter 类的代码：

```
1.  package FormPack.Tools;
2.
3.  import DocumentPack.Questions.QuestionType;
4.
5.  import javax.swing.*;
6.
7.  //This is a pack about QuestionPainter
8.  public class FatherChildStruct
9.  {
10.     private final QuestionType type;
11.
12.     public  JPanel mainPanel;
13.     public  JTextArea textArea;
14.     public  JTextArea answerArea;
15.     public  JRadioButton[] options;
```

```
16.     public  JCheckBox[] mulOptions;
17.
18.     public FatherChildStruct(JPanel theMainPanel,JTextArea theTextArea,QuestionType theType)
19.     {
20.         mainPanel=theMainPanel;
21.         textArea=theTextArea;
22.         type=theType;
23.         answerArea=null;
24.         options=null;
25.         mulOptions=null;
26.     }
27.
28.     public FatherChildStruct(JPanel theMainPanel,JTextArea theTextArea,JTextArea theAnswerArea)
29.     {
30.         mainPanel=theMainPanel;
31.         textArea=theTextArea;
32.         answerArea=theAnswerArea;
33.         options=null;
34.         mulOptions=null;
35.         type=QuestionType.Essay;
36.     }
37.     public FatherChildStruct(JPanel theMainPanel,JTextArea theTextArea,JRadioButton[] buttons)
38.     {
39.         mainPanel=theMainPanel;
40.         textArea=theTextArea;
41.         answerArea=null;
42.         options=buttons;
43.         mulOptions=null;
44.         type=QuestionType.Single;
45.     }
46.     public FatherChildStruct(JPanel theMainPanel,JTextArea theTextArea,JCheckBox[] buttons)
47.     {
48.         mainPanel=theMainPanel;
49.         textArea=theTextArea;
50.         answerArea=null;
51.         options=null;
52.         mulOptions=buttons;
53.         type=QuestionType.Multiple;
54.     }
55.
56.     public void setAnswerArea(JTextArea answerArea)
57.     {
58.         this.answerArea = answerArea;
59.     }
```

```
60.
61.     public void setMainPanel(JPanel mainPanel)
62.     {
63.         this.mainPanel = mainPanel;
64.     }
65.
66.     public void setMulOptions(JCheckBox[] mulOptions)
67.     {
68.         this.mulOptions = mulOptions;
69.     }
70.
71.     public void setOptions(JRadioButton[] options)
72.     {
73.         this.options = options;
74.     }
75.
76.     public void setTextArea(JTextArea textArea)
77.     {
78.         this.textArea = textArea;
79.     }
80. }
```

```
1.  package FormPack.Tools;
2.  import DocumentPack.Questions.BasicQuestion;
3.  import DocumentPack.Questions.ChoiceQuestion;
4.  import DocumentPack.Questions.EssayQuestion;
5.  import DocumentPack.Questions.QuestionType;
6.  import FormPack.Forms.QuestionInputForm;
7.  import StringPack.ButtonNames;
8.  import StringPack.LabelStrings;
9.  import javax.swing.*;
10. import java.awt.*;
11. public class QuestionPainter
12. {
13.  private static final int columns =86;
14.  private static final int rows =16;
15.  public static FatherChildStruct paint(BasicQuestion question)
16.  {
17.   if(question.getType() == QuestionType.Essay)return paintEssay((EssayQuestion) question);else
18.   if(question.getType() == QuestionType.Single)return paintSingle((ChoiceQuestion)question);else
19.   if(question.getType() == QuestionType.Multiple)return paintMultiple((ChoiceQuestion)question);
20.   return null;
21.  }
```

```
22.  private static FatherChildStruct paintEssay(EssayQuestion question)
23.  {
24.   JPanel mainPanel =new JPanel(new GridLayout(2,1));
25.   JPanel upArea=new JPanel(new BorderLayout());
26.   JPanel downArea=new JPanel(new FlowLayout(FlowLayout.LEFT));
27.   JPanel subUpArea=new JPanel(new FlowLayout(FlowLayout.LEFT));
28.   JPanel subDownArea=new JPanel(new FlowLayout(FlowLayout.LEFT));
29.   JLabel nameLabel=new JLabel(LabelStrings.QuestionName);
30.   JLabel textLabel=new JLabel(LabelStrings.QuestionText);
31.   JLabel answerLabel=new JLabel(LabelStrings.QuestionAnswer);
32.   JTextField nameArea=new JTextField(question.getName(), columns);
33.   JTextArea theTextArea=new JTextArea(question.getQuestionText(), rows, columns);
34.   JTextArea theAnswerArea=new JTextArea(question.getRightAnswer(), rows, columns);
35.   JScrollPane textArea=new JScrollPane(theTextArea);
36.   JScrollPane answerArea=new JScrollPane(theAnswerArea);
37.   nameArea.setEditable(false);
38.   theTextArea.setLineWrap(true);
39.   theAnswerArea.setLineWrap(true);
40.   subUpArea.add(nameLabel);
41.   subUpArea.add(nameArea);
42.   subDownArea.add(textLabel);
43.   subDownArea.add(textArea);
44.   upArea.add(subUpArea,BorderLayout.NORTH);
45.   upArea.add(subDownArea,BorderLayout.CENTER);
46.   downArea.add(answerLabel);
47.   downArea.add(answerArea);
48.   mainPanel.add(upArea);
49.   mainPanel.add(downArea);
50.  }
51.  private static FatherChildStruct paintSingle(ChoiceQuestion question)
52.  {
53.   JPanel mainPanel =new JPanel(new GridLayout(2,1));
54.   JPanel upArea=new JPanel(new BorderLayout());
55.   JPanel downArea=new JPanel(new FlowLayout(FlowLayout.LEFT));
56.   JPanel subUpArea=new JPanel(new FlowLayout(FlowLayout.LEFT));
57.   JPanel subDownArea=new JPanel(new FlowLayout(FlowLayout.LEFT));
58.   JLabel nameLabel=new JLabel(LabelStrings.QuestionName);
59.   JLabel textLabel=new JLabel(LabelStrings.QuestionText);
60.   JLabel choiceLabel=new JLabel(LabelStrings.QuestionChoice);
61.   JTextField nameArea=new JTextField(question.getName(), columns);
62.   JTextArea theTextArea=new JTextArea(question.getQuestionText(), rows, columns);
63.   JScrollPane textArea=new JScrollPane(theTextArea);
64.   JPanel choiceArea=new JPanel(new GridLayout(question.getOptionNumber()+1,1));
65.   ButtonGroup choiceGroup=new ButtonGroup();
```

```
66.     JPanel buttonArea=new JPanel(new FlowLayout());
67.     JButton addChoice=new JButton(ButtonNames.AddOption);
68.     JButton deleteChoice=new JButton(ButtonNames.DeleteOption);
69.     nameArea.setEditable(false);
70.     theTextArea.setLineWrap(true);
71.     FatherChildStruct retStruct=new FatherChildStruct(mainPanel,theTextArea,QuestionType.Single);
72.     //Build Choice Button
73.     addChoice.addActionListener(event->
74.     {
75.      QuestionInputForm inputForm=new QuestionInputForm(LabelStrings.InputOption);
76.      inputForm.inputNewOption(question,buttonArea,choiceArea,retStruct,choiceGroup);
77.     });
78.     deleteChoice.addActionListener(event->
79.     {
80.      question.removeOption();
81.      choiceArea.removeAll();
82.      choiceArea.setLayout(new GridLayout(question.getOptionNumber()+1, 1));
83.      //Same as "Build RadioButton"
84.      int optionNumber= question.getOptionNumber();
85.      JRadioButton[] options=new JRadioButton[optionNumber];
86.      for(int i=0;i<optionNumber;i++)
87.      {
88.       options[i]=new JRadioButton(question.getChoice(i),question.isRight(i));
89.       choiceGroup.add(options[i]);
90.       choiceArea.add(options[i]);
91.      }
92.      //
93.      retStruct.setOptions(options);
94.      choiceArea.add(buttonArea);
95.      choiceArea.updateUI();
96.     });
97.     buttonArea.add(addChoice);
98.     buttonArea.add(deleteChoice);
99.     //Build RadioButton
100.    int optionNumber= question.getOptionNumber();
101.    JRadioButton[] options=new JRadioButton[optionNumber];
102.    for(int i=0;i<optionNumber;i++)
103.    {
104.     options[i]=new JRadioButton(question.getChoice(i),question.isRight(i));
105.     choiceGroup.add(options[i]);
106.     choiceArea.add(options[i]);
107.    }
108.    retStruct.setOptions(options);
109.    //Add Button to ChoiceArea
```

```
110.    choiceArea.add(buttonArea);
111.    //Build Panel Area
112.    subUpArea.add(nameLabel);
113.    subUpArea.add(nameArea);
114.    subDownArea.add(textLabel);
115.    subDownArea.add(textArea);
116.    upArea.add(subUpArea,BorderLayout.NORTH);
117.    upArea.add(subDownArea,BorderLayout.CENTER);
118.    downArea.add(choiceLabel);
119.    downArea.add(choiceArea);
120.    mainPanel.add(upArea);
121.    mainPanel.add(downArea);
122.    return retStruct;
123.  }
124.  private static FatherChildStruct paintMultiple(ChoiceQuestion question)
125.  {
126.    JPanel mainPanel =new JPanel(new GridLayout(2,1));
127.    JPanel upArea=new JPanel(new BorderLayout());
128.    JPanel downArea=new JPanel(new FlowLayout(FlowLayout.LEFT));
129.    JPanel subUpArea=new JPanel(new FlowLayout(FlowLayout.LEFT));
130.    JPanel subDownArea=new JPanel(new FlowLayout(FlowLayout.LEFT));
131.    JLabel nameLabel=new JLabel(LabelStrings.QuestionName);
132.    JLabel textLabel=new JLabel(LabelStrings.QuestionText);
133.    JLabel choiceLabel=new JLabel(LabelStrings.QuestionChoice);
134.    JTextField nameArea=new JTextField(question.getName(), columns);
135.    JTextArea theTextArea=new JTextArea(question.getQuestionText(), rows, columns);
136.    JScrollPane textArea=new JScrollPane(theTextArea);
137.    JPanel choiceArea=new JPanel(new GridLayout(question.getOptionNumber()+1,1));
138.    JPanel buttonArea=new JPanel(new FlowLayout());
139.    JButton addChoice=new JButton(ButtonNames.AddOption);
140.    JButton deleteChoice=new JButton(ButtonNames.DeleteOption);
141.    nameArea.setEditable(false);
142.    theTextArea.setLineWrap(true);
143.    FatherChildStruct retStruct=new FatherChildStruct(mainPanel,theTextArea,QuestionType.Multiple);
144.    //Build Choice Button
145.    addChoice.addActionListener(event->
146.    {
147.     QuestionInputForm inputForm=new QuestionInputForm(LabelStrings.InputOption);
148.     inputForm.inputNewOption(question,buttonArea,choiceArea,retStruct);
149.    });
150.    deleteChoice.addActionListener(event->
151.    {
152.     question.removeOption();
153.     choiceArea.removeAll();
```

```
154.     choiceArea.setLayout(new GridLayout(question.getOptionNumber()+1, 1));
155.     //Same as "Build CheckBox"
156.     int optionNumber= question.getOptionNumber();
157.     JCheckBox[] options=new JCheckBox[optionNumber];
158.     for(int i=0;i<optionNumber;i++)
159.     {
160.      options[i]=new JCheckBox(question.getChoice(i),question.isRight(i));
161.      choiceArea.add(options[i]);
162.     }
163.     //
164.     retStruct.setMulOptions(options);
165.     choiceArea.add(buttonArea);
166.     choiceArea.updateUI();
167.    });
168.    buttonArea.add(addChoice);
169.    buttonArea.add(deleteChoice);
170.    //Build CheckBox
171.    int optionNumber= question.getOptionNumber();
172.    JCheckBox[] options=new JCheckBox[optionNumber];
173.    for(int i=0;i<optionNumber;i++)
174.    {
175.     options[i]=new JCheckBox(question.getChoice(i),question.isRight(i));
176.     choiceArea.add(options[i]);
177.    }
178.    retStruct.setMulOptions(options);
179.    //Add Button to ChoiceArea
180.    choiceArea.add(buttonArea);
181.    //Build Panel Area
182.    subUpArea.add(nameLabel);
183.    subUpArea.add(nameArea);
184.    subDownArea.add(textLabel);
185.    subDownArea.add(textArea);
186.    upArea.add(subUpArea,BorderLayout.NORTH);
187.    upArea.add(subDownArea,BorderLayout.CENTER);
188.    downArea.add(choiceLabel);
189.    downArea.add(choiceArea);
190.    mainPanel.add(upArea);
191.    mainPanel.add(downArea);
192.    return retStruct;
193.   }
194.  }
```

2. Internet Manager 以及 Connector 代码如下：

```
1.  package DataOperationPack;
2.
3.  import DocumentPack.Members.Student;
4.  import DocumentPack.Tests.BackTestStruct;
5.  import DocumentPack.Tests.TestPaper;
6.  import MiddlePack.BasicGet;
7.  import StringPack.MessageStrings;
8.
9.  import java.util.ArrayList;
10.
11. public class InternetManager
12. {
13.     public final static int port=3456;
14.
15.     private BasicGet getter;
16.
17.     private boolean listening;
18.     private LinkMaker linkMaker;
19.     private ArrayList<Connector> connectors;
20.
21.     private TestPaper paper;
22.     private ArrayList<BackTestStruct> backPapers;
23.
24.     public InternetManager(BasicGet Getter)
25.     {
26.         getter=Getter;
27.
28.         listening=false;
29.
30.         connectors=new ArrayList<>();
31.         backPapers=new ArrayList<>();
32.         paper=null;
33.     }
34.
35. //Start and Stop Link with Student
36.     public void linkStart()
37.     {
38.         listening=true;
39.         linkMaker=new LinkMaker(this);
40.         Thread linkThread=new Thread(linkMaker);
41.         linkThread.start();
42.     }
43.     public void linkStop()
```

```
44.     {
45.         listening=false;
46.     }
47.
48.   //Used by Link Maker
49.     public boolean isListening()
50.     {
51.         return listening;
52.     }
53.     public void catchConnection(Connector newConnector)
54.     {
55.         connectors.add(newConnector);
56.     }
57.
58.   //Used by Connector
59.     public Student giveStudent(String id)
60.     {
61.         return getter.getStudent(id);
62.     }
63.     public TestPaper givePaper()
64.     {
65.         return paper;
66.     }
67.     public void catchBackTest(BackTestStruct object)
68.     {
69.         backPapers.add(object);
70.     }
71.     public String givePoint(Student want)
72.     {
73.         String ret="-1";
74.         for(int i=0;i< backPapers.size();i++)
75.             if(want.equals(backPapers.get(i).getFromStudent()))
76.                 ret=Integer.toString(backPapers.get(i).getFinalPoint());
77.         if(ret.equals("-1"))return MessageStrings.NoInformation;
78.         return ret;
79.     }
80.     public void killConnector(Connector connector)
81.     {
82.         connectors.remove(connector);
83.     }
84.
85.   //Used by Sender&Getter
86.     public void setPaper(TestPaper thePaper)
87.     {
```

```
88.         paper=thePaper;
89.     }
90.     public BackTestStruct[] getBackPaper()
91.     {
92.         BackTestStruct[] ret=new BackTestStruct[0];
93.         if(backPapers.size()==0)return ret;
94.
95.         ret=new BackTestStruct[backPapers.size()];
96.         for(int i=0;i<backPapers.size();i++)
97.             ret[i]=backPapers.get(i);
98.         return ret;
99.     }
100.    public void correctOver(ArrayList<BackTestStruct> result)
101.    {
102.        backPapers=result;
103.    }
104. }
```

```
1.  package DataOperationPack;
2.
3.  import DocumentPack.Members.Student;
4.  import DocumentPack.Tests.BackTestStruct;
5.  import StringPack.MessageStrings;
6.
7.  import javax.swing.*;
8.  import java.io.ObjectInputStream;
9.  import java.io.ObjectOutputStream;
10. import java.net.Socket;
11.
12. public class Connector implements Runnable
13. {
14.     //Use their Class name
15.     private static final Student studentTag=new Student("","","");
16.     private static final Boolean booleanTag=true;
17.     private static final BackTestStruct testTag=new BackTestStruct(studentTag,0,0);
18.     //--------------------
19.     private Socket pipe;
20.
21.     private ObjectInputStream input;
22.     private ObjectOutputStream output;
23.
24.     private boolean linked;
25.     private boolean living;
```

```
26.     private Student whoLinked;
27.     private InternetManager manager;
28.
29.     public Connector(Socket socket,InternetManager theManager)
30.     {
31.         linked=false;
32.         living=false;
33.         whoLinked=null;
34.
35.         whoLinked=new Student(MessageStrings.UnknownStudent,"","");
36.
37.         pipe=socket;
38.         manager=theManager;
39.     }
40.
41.     public void run()
42.     {
43.         try
44.         {
45.             input=new ObjectInputStream(pipe.getInputStream());
46.             input.readObject();
47.             output=new ObjectOutputStream(pipe.getOutputStream());
48.             output.writeObject("linked");
49.             while(true)
50.             {
51.                 Object object=input.readObject();
52.                 if(object.getClass().getName().equals(studentTag.getClass().getName()))
53.                 {
54.                     if(manager.giveStudent(((Student)object).getId()).equals((Student)object))
55.                     {
56.                         output.writeObject(true);
57.                         whoLinked=(Student)object;
58.                         linked=true;
59.                     }else
60.                         output.writeObject(false);
61.                 }else if(object.getClass().getName().equals(booleanTag.getClass().getName()))
62.                 {
63.                     if(linked)
64.                     {
65.                         if((Boolean)object)
66.                         {
67.                             while(manager.givePaper()==null)
68.                                 Thread.sleep(100);
69.                             living=true;
```

```
70.                         output.writeObject(manager.givePaper());
71.                     }else
72.                     {
73.                         output.writeObject(manager.givePoint(whoLinked));
74.                     }
75.                 }else
76.                 {
77.                     input.close();
78.                     output.close();
79.                     return;
80.                 }
81.             }else if(object.getClass().getName().equals(testTag.getClass().getName()))
82.             {
83.                 if(linked&&living)
84.                 {
85.                     manager.catchBackTest((BackTestStruct)object);
86.                     living=false;
87.                 }else
88.                 {
89.                     input.close();
90.                     output.close();
91.                     return;
92.                 }
93.             }else
94.             {
95.                 input.close();
96.                 output.close();
97.                 return;
98.             }
99.           }
100.      }catch (Exception e)
101.      {
102.          JOptionPane.showMessageDialog
103.                  (null, whoLinked.getId()+MessageStrings.ConnectionBroken,
104.                   "",JOptionPane.INFORMATION_MESSAGE);
105.      }finally
106.      {
107.          this.killMe();
108.      }
109.  }
110.
111.  public void killMe()
112.  {
113.      manager.killConnector(this);
```

```
114.     }
115. }
```

3.Database Manager 代码：

```
1.   package DataOperationPack;
2.
3.   import StringPack.DataBaseStrings;
4.   import StringPack.MessageStrings;
5.   import StringPack.TitleStrings;
6.
7.   import javax.swing.*;
8.   import java.sql.*;
9.   import java.util.ArrayList;
10.
11.  public class DataBaseManager
12.  {
13.      private Connection link;
14.
15.      public DataBaseManager()
16.      {
17.          try
18.          {
19.              Class.forName(DataBaseStrings.DataBaseDrive);
20.              link=DriverManager.getConnection(DataBaseStrings.DataBaseUrl,
21.                                               DataBaseStrings.DataBaseUser,
22.                                               DataBaseStrings.DataBasePassword);
23.          }catch (Exception e)
24.          {
25.              this.dataBaseError();
26.          }
27.
28.      }
29.
30.  //Insert Part
31.      public void insertUser(String name,String password)
32.      {
33.          try
34.          {
35.              Statement state=link.createStatement();
36.              state.executeUpdate(DataBaseStrings.InsertUser+"(\""+name+"\",\""+password+"\")");
37.          }catch (Exception e)
38.          {
39.              dataBaseError();
40.          }
```

```
41.     }
42.     public void insertStudent(String id,String name,String password)
43.     {
44.         try
45.         {
46.             Statement state=link.createStatement();
47.             state.executeUpdate(DataBaseStrings.InsertStudent+"(\""+id+"\",\""+name+"\",\""+ password +"\")");
48.         }catch (Exception e)
49.         {
50.             dataBaseError();
51.         }
52.     }
53.     public void insertEssay(String name)
54.     {
55.         try
56.         {
57.             Statement state=link.createStatement();
58.             state.executeUpdate(DataBaseStrings.InsertEssay+"(\""+name+"\","+this.getNullString(2)+")");
59.         }catch (Exception e)
60.         {
61.             dataBaseError();
62.         }
63.     }
64.     public void insertChoice(String name,String type)
65.     {
66.         try
67.         {
68.             Statement state=link.createStatement();
69.             state.executeUpdate(DataBaseStrings.InsertChoice+
70.                                 "(\""+name+"\",\""+type+"\","+this.getNullString(11)+")");
71.         }catch (Exception e)
72.         {
73.             dataBaseError();
74.         }
75.     }
76.
77.   //Delete Part
78.     public void deleteStudent(String id)
79.     {
80.         try
81.         {
82.             PreparedStatement state=link.prepareStatement(DataBaseStrings.DeleteStudent);
83.             state.setString(1,id);
84.             state.executeUpdate();
```

```
85.         }catch (Exception e)
86.         {
87.             dataBaseError();
88.         }
89.     }
90.     public void deleteEssay(String name)
91.     {
92.         try
93.         {
94.             PreparedStatement state=link.prepareStatement(DataBaseStrings.DeleteEssay);
95.             state.setString(1,name);
96.             state.executeUpdate();
97.         }catch (Exception e)
98.         {
99.             dataBaseError();
100.        }
101.    }
102.    public void deleteChoice(String name)
103.    {
104.        try
105.        {
106.            PreparedStatement state=link.prepareStatement(DataBaseStrings.DeleteChoice);
107.            state.setString(1,name);
108.            state.executeUpdate();
109.        }catch (Exception e)
110.        {
111.            dataBaseError();
112.        }
113.    }
114.
115. //Update Part
116.    public void updateEssay(String name,String text,String answer)
117.    {
118.        try
119.        {
120.            PreparedStatement state= link.prepareStatement(DataBaseStrings.UpdateEssay);
121.            state.setString(3,name);
122.            state.setString(1,text);
123.            state.setString(2,answer);
124.            state.executeUpdate();
125.        }catch (Exception e)
126.        {
127.            dataBaseError();
128.        }
```

```
129.    }
130.    public void updateChoice(String name,String text,String opNumber,String answer,String[] options)
131.    {
132.        try
133.        {
134.            PreparedStatement state1=link.prepareStatement(DataBaseStrings.UpdateChoice);
135.            PreparedStatement state2=link.prepareStatement(DataBaseStrings.UpdateOption);
136.
137.            state1.setString(4,name);
138.            state1.setString(1,text);
139.            state1.setString(2,opNumber);
140.            state1.setString(3,answer);
141.
142.            state2.setString(9,name);
143.            for(int i=1;i<=8;i++)
144.             if(i<=options.length)state2.setString(i,options[i-1]);
145.              else state2.setString(i,"");
146.
147.            state1.executeUpdate();
148.            state2.executeUpdate();
149.        }catch (Exception e)
150.        {
151.            dataBaseError();
152.        }
153.    }
154.
155. //Search Part
156.    public String searchUser(String name)
157.    {
158.        String ret=null;
159.        try
160.        {
161.            PreparedStatement state=link.prepareStatement(DataBaseStrings.SearchUser);
162.            state.setString(1,name);
163.
164.            ResultSet result=state.executeQuery();
165.            if(result!=null&&result.next())
166.            {
167.                ret=result.getString(1);
168.            }
169.        }catch (Exception e)
170.        {
171.            dataBaseError();
172.        }
```

```
173.        return ret;
174.    }
175.    public String[] searchStudent(String id)
176.    {
177.        String[] ret=new String[2];
178.        try
179.        {
180.            PreparedStatement state=link.prepareStatement(DataBaseStrings.SearchStudent);
181.            state.setString(1,id);
182.
183.            ResultSet result=state.executeQuery();
184.            if(result!=null&&result.next())
185.            {
186.                ret[0]=result.getString(1);
187.                ret[1]=result.getString(2);
188.            }
189.        }catch (Exception e)
190.        {
191.            dataBaseError();
192.        }
193.        return  ret;
194.    }
195.    public String[] searchEssay(String name)
196.    {
197.        String[] ret=new String[3];
198.        try
199.        {
200.            PreparedStatement state=link.prepareStatement(DataBaseStrings.SearchEssay);
201.            state.setString(1,name);
202.
203.            ResultSet result=state.executeQuery();
204.            if(result!=null&&result.next())
205.            {
206.                ret[0]=result.getString(1);
207.                ret[1]=result.getString(2);
208.                ret[2]=result.getString(3);
209.            }
210.        }catch (Exception e)
211.        {
212.            dataBaseError();
213.        }
214.        return  ret;
215.    }
216.    public String[] searchChoice(String name)
```

```
217.    {
218.        String[] ret=new String[1];
219.        int number;
220.        try
221.        {
222.            PreparedStatement state1=link.prepareStatement(DataBaseStrings.SearchChoice);
223.            state1.setString(1,name);
224.            ResultSet result1=state1.executeQuery();
225.
226.            if(result1!=null&&result1.next())
227.            {
228.                String numberString=result1.getString(4);
229.                if(!numberString.equals(""))
230.                    number=Integer.parseInt(numberString);
231.                 else number=0;
232.                ret=new String[5+number];
233.                ret[0]=result1.getString(1);
234.                ret[1]=result1.getString(2);
235.                ret[2]=result1.getString(3);
236.                ret[3]=result1.getString(4);
237.                ret[4]=result1.getString(5);
238.
239.                if(number!=0)
240.                {
241.                    PreparedStatement state2=link.prepareStatement(DataBaseStrings.SearchOption[number]);
242.                    state2.setString(1,name);
243.                    ResultSet result2=state2.executeQuery();
244.                    if(result2!=null&&result2.next())
245.                    {
246.                        for(int i=5;i<5+number;i++)
247.                            ret[i]=result2.getString(i-4);
248.
249.                    }else throw (new Exception());
250.                }
251.            }else throw(new Exception());
252.        }catch (Exception e)
253.        {
254.            dataBaseError();
255.        }
256.        return  ret;
257.    }
258.
259.  //Sub Search Part
260.     public boolean hasUser(String name)
```

```
261.    {
262.        try
263.        {
264.            PreparedStatement state =link.prepareStatement(DataBaseStrings.SearchUser);
265.            state.setString(1,name);
266.            ResultSet result= state.executeQuery();
267.            return result.next();
268.        }catch (Exception e)
269.        {
270.            dataBaseError();
271.        }
272.        return false;
273.    }
274.    public boolean hasStudent(String id)
275.    {
276.        try
277.        {
278.            PreparedStatement state =link.prepareStatement(DataBaseStrings.SearchStudent);
279.            state.setString(1,id);
280.            ResultSet result= state.executeQuery();
281.            return result.next();
282.        }catch (Exception e)
283.        {
284.            dataBaseError();
285.        }
286.        return false;
287.    }
288.    public boolean hasEssay(String name)
289.    {
290.        try
291.        {
292.            PreparedStatement state =link.prepareStatement(DataBaseStrings.SearchEssay);
293.            state.setString(1,name);
294.            ResultSet result= state.executeQuery();
295.            return result.next();
296.        }catch (Exception e)
297.        {
298.            dataBaseError();
299.        }
300.        return false;
301.    }
302.    public boolean hasChoice(String name)
303.    {
304.        try
```

```
305.         {
306.             PreparedStatement state =link.prepareStatement(DataBaseStrings.SearchChoice);
307.             state.setString(1,name);
308.             ResultSet result= state.executeQuery();
309.             return result.next();
310.         }catch (Exception e)
311.         {
312.             dataBaseError();
313.         }
314.         return false;
315.     }
316.
317.  //Return List of Name
318.     public String[] getStudent()
319.     {
320.         ArrayList<String> stringBin =new ArrayList<>();
321.         try
322.         {
323.             Statement state=link.createStatement();
324.             ResultSet result=state.executeQuery(DataBaseStrings.StudentList);
325.             while(result!=null&&result.next())
326.             {
327.                 stringBin.add(result.getString(1));
328.             }
329.             if(stringBin.size()==0)
330.             {
331.                 String[] ret=new String[1];
332.                 ret[0]="";
333.                 return ret;
334.             }
335.         }catch (Exception e)
336.         {
337.             dataBaseError();
338.         }
339.
340.         String[] ret=new String[stringBin.size()];
341.         for(int i = 0; i< stringBin.size(); i++)
342.             ret[i]= stringBin.get(i);
343.         return ret;
344.     }
345.     public String[] getEssay()
346.     {
347.         ArrayList<String> stringBin =new ArrayList<>();
348.         try
```

```
        {
            Statement state=link.createStatement();
            ResultSet result=state.executeQuery(DataBaseStrings.EssayQuestionList);
            while(result!=null&&result.next())
            {
                stringBin.add(result.getString(1));
            }
            if(stringBin.size()==0)
            {
                String[] ret=new String[1];
                ret[0]="";
                return ret;
            }
        }catch (Exception e)
        {
            dataBaseError();
        }

        String[] ret=new String[stringBin.size()];
        for(int i = 0; i< stringBin.size(); i++)
            ret[i]= stringBin.get(i);
        return ret;
    }
    public String[] getChoice()
    {
        ArrayList<String> stringBin =new ArrayList<>();
        try
        {
            Statement state=link.createStatement();
            ResultSet result=state.executeQuery(DataBaseStrings.ChoiceQuestionList);
            while(result!=null&&result.next())
            {
                stringBin.add(result.getString(1));
            }
            if(stringBin.size()==0)
            {
                String[] ret=new String[1];
                ret[0]="";
                return ret;
            }
        }catch (Exception e)
        {
            dataBaseError();
        }
```

```
393.
394.        String[] ret=new String[stringBin.size()];
395.        for(int i = 0; i< stringBin.size(); i++)
396.            ret[i]= stringBin.get(i);
397.        return ret;
398.    }
399.
400.    private void dataBaseError()
401.    {
402.        JOptionPane.showMessageDialog
403.                (null, MessageStrings.DataBaseFailed,
404.                        TitleStrings.DataBaseError,JOptionPane.ERROR_MESSAGE);
405.    }
406.
407.    private String getNullString(int re)
408.    {
409.        String Null=",\"\"";
410.        String ret="\"\"";
411.        for(int i=1;i<re;i++)
412.            ret=ret+Null;
413.        return ret;
414.    }
415. }
```