

Software Documentation

Homework/Quiz Course Software

Draft 1.0

March 26, 2004

by Dr. Oakley E. Gordon
Department of Psychology

University of Utah

Salt Lake City, Ut 84112

(801) 58103258

oakley.gordon@psych.utah.edu

General Features

Application and Applet

The program can be run as an application or as an applet. The *init()* method is in class **Homework**, the *main()* method is in the class **Application**. Running the program as an application is mainly for development and debugging purposes (the applet uses a servlet to read and write data, and the servlet does not work when the applet is run locally).

Important Variables

The program keeps track of some important variables that influence how it is run:

The *mode* variable keeps track of whether the program is being run in *USER_MODE* (i.e. the mode when a student is using the program) or *EDIT-MODE* (the mode when an instructor is editing the contents of the course).

The *context* variable keeps track of whether the program is being run as an *APPLICATION* or as an *APPLET_SERVLET* (i.e. an applet using a servlet to read and write). Other contexts have been used in development as well, but are not currently supported.

The *cmsContext* is a new addition to the program. After I had basically finished the program the university offered an entryway into the program where the student name and password and course number would be passed as a parameter to the program (rather than having the program prompt for that information). To take advantage of this feature--without limiting the program to only working within the university context--the program now looks to see if those values are being passed as parameters, if they are, then the *cmsContext* variable is set to true, and the screens that prompt the student for that information are skipped. If those values are not passed then the *cmsContext* is set to false and the program operates as originally designed.

Display

The displays seen by the user are all extensions of the custom class **Card** which extends JPanel (not to be confused with the Java 'CardLayout'). When in edit mode, the cards are viewed within a larger panel that contains controls for the card (e.g. controls that allow the card to be edited).

Organization

Lesson is a term that applies to both homeworks and quizzes. A lesson consists of a series of cards, each card containing a question, a story problem, or a 'marker' card (a type of card that influences the presentation of subsequent questions but itself is not seen by the student).

The cards in a lesson are stored as a single String that can be parsed into individual cards. The lesson is written and read to file in the form of the serializable class **LessonData**, which contains several elements including; the list of cards as a String, the due date of the lesson, whether it has been assigned yet, and other such information.

Once a lesson has been read from file, the String of the cards is read, parsed into individual cards, and the Strings for the individual cards are put into a Vector, so that they can be displayed one at a time. This is all accomplished by the class **Lesson**, which is responsible for setting up a lesson after it has been selected by the student.

Read/Write

The reading and writing of information over the internet (i.e. when in APPLET-SERVER mode) is accomplished by a servlet. Communication with a servlet takes the form of sending it a **QueryObject** which contains one command and any information needed for that command (e.g. a command to write a file, the file name, and the content of the file), and getting back from the servlet a **ResponseObject** which contains the result of the command (e.g. the content of the file if the command was to read a file, and any error messages). To make the program simpler and consistent, reading and writing while in APPLICATION mode works exactly the same way, only it is carried out locally by the application rather than over the internet by a servlet.

<h2>Major Classes</h2>

Homework extends JApplet

This is the main class of the program. It has the **init()** method for running the program from an HTML call, and an **initFromApplication()** method that is called by class **Application** when the program is run as an application.

In addition to setting up the program the Homework class contains various methods and variables that are of general utility in the program. The variables are available through getter and setter methods.

The Homework class also sets up one static instance of each type of card available in the program. When it is time for a card to be displayed, the card is sent the information (content and various display options) it needs to display itself and is then made visible to the user.

HomeworkIO

All requests to read or write data go through this class. It has methods for adding a student, getting grades, getting lesson information, etc. It accomplishes this by setting up a QueryObject (described above), and then sending it to the servlet (if in APPLET_SERVLET mode) or to the local copy of the **DoFileIO** class (if in APPLICATION mode), and then interpreting the ResponseObject (also described above) that is returned.

DoFileIO

The actual nitty-gritty of carrying out the commands of a QueryObject and formulating a ResponseObject is carried out by this class. A copy of this class needs to be available locally for use when in APPLICATION mode and with the servlet on the server for use when in APPLET_SERVLET mode.

Lesson

This class is handed information about the lesson to be implemented (LessonData) and information about the student viewing the lesson (StudentGradeData). It uses the LessonData to set up a vector of the questions, story problems, and marker cards that constitute a lesson; and to determine whether the lesson is to be displayed as a homework or a quiz.

It uses the LessonData and StudentGradeData to determine whether the lesson is eligible to be handed in (using the due date from the LessonData and any individual homework deadline extensions from the StudentGradeData).

This class also implements various editing features related to lessons (e.g. the 'Vet Lesson' command from the 'Utilities' menu, or the 'Go to last card' command from the lesson control panel).

EditToolPanel extends JPanel

This class provides the larger panel within which the cards and the various control panels are displayed when in EDIT_MODE.

Card extends JPanel

This is the parent class of the various screens displayed for the user. It has an important abstract method: ***abstract public JPanel getCardControlPanel()***. Any extension of Card must provide a control panel that gives the instructor the ability to edit and set the various elements of that card . This control panel is displayed by the **EditToolPanel** class while in edit mode. The cardControlPanel can be simply a "new JPanel()" if no controls are needed for that card.

The hierarchy of Card and its children is given below, and described in more detail later in this document.

Card

Question

FillInTextQuestion

FillInNumberQuestion

MultipleChoiceQuestion

MultipleSelectQuestion

MultipleGraphicQuestion

StoryProblem

MarkerCard

BeginRandomMarker

EndRandomMarker

BeginSubsetMarker

EndSubsetMarker

SelectCourseCard

SignInCard

IndexCard

LessonIntroCard

EndLessonCard

Setting Things Up

HomeworkQuiz Directory: The directory structure is documented in class HomeworkIO, and the essential directories and files can be downloaded from the HomeworkQuiz open source site.

HomeworkQuiz (directory): contains the follow....

homeworkquiz.jar (file): the jar file of the classes

index.html (file): copy of the html code for the applet

password (file): the password for entering into edit mode, if this is missing (as it will be the first time you run the program) then you will be prompted to input the password. Deleting this file allows you to change passwords.

AppImages (directory): contains all of the images used in the running of the program (does not include images used in specific lessons).

Courses (directory): each course is a subdirectory of this directory. The program uses the subdirectories of this directory to create the list of available courses. Selecting 'Add a Course' in the program creates a course directory (along with required subdirectories and files) and places it in this directory. The contents of a course directory consists of:

lesson_download_directory (directory): the various lessons (homework assignments and quizzes) are stored here, there is a file for each lesson and a file containing information on all the lessons (called 'lesson_index'). The inconsistent use of all lower-case letters for some files and directories has to do with working for a while with a ftp program that refused to accept upper-case letters.

LessonImages (directory): contains the various gif and jpeg files for the lessons. These images need to be uploaded to this file before they can be accessed for a lesson. The image files must end in '.gif' or '.GIF', or '.jpeg' or '.JPEG' to be recognized by the program.

LessonStringsDirectory (directory): one option when editing a lesson is to store it as a serialized String, this is not how the program normally stores a lesson, but it is the easiest way to move lessons individually from one course to another. The String files all end in '.str'. All the lessons can be moved from one course to another by 'cloning' a course (see the users manual).

password (file): this is the password for gaining access to editing a specific course. If this file doesn't exist then the user will be prompted for a new password.

StudentDirectory (directory): contains the files for each student, and a file containing information on all of the students (called 'StudentList').

Retired_Courses (directory): selecting 'Retire a Course' in the program places the directory for that course into this directory, thus it no longer shows up in the list of courses but has not been deleted.

IntroLessonFiles (directory): this is not needed by the program, it contains some files that help you set up an introductory homework assignment that teaches the student how to run the software. This is described below under 'Getting Started'.

Servlet Directory: We have the servlet (responsible for reading and writing files) in a different directory than the HomeworkQuiz program. The location of the servlet must be coded in the html code and passed as a parameter so that HomeworkQuiz knows where to find it (see html applet code below). The servlet directory needs to have not only the servlet class, but also the copies of the classes that are serialized for reading and writing, as well as a the ServletConstants.class (an interface providing global constants).

Servlet Directory: (doesn't have to be called this) contains the following...

HomeworkServlet.class

StudentInfo.class

ResponseObject.class

ServletConstants.class

DoFileIO.class
HomeworkObjectMessenger.class
QueryObject.class
NavigateButton.class (?why here?)
J2EE.JAR (?necessary here?)

Things You Need to Set

- 1) In the code of the servlet you need to put the address of the HomeworkQuiz directory. Currently this is hard coded at the beginning of the code as String startingPath=...
- 2) In the html code:
 - a) ARCHIVE must be changed to reflect where you have put the code.
 - b) The parameter "path" must be changed to fit where you put the servlet.
 - c) The parameter "demoMode" is set to true if you want to demonstrate the program but don't want people to be able to add their name to the list of students in the class.

HTML Code and Parameters

```
<APPLET  
  CODE = "Homework.class"  
  ARCHIVE = "http://raiders.psych.utah.edu:8180/psych/3000_090/javaclasses.jar"  
  WIDTH = 720  
  HEIGHT = 680  
  HSPACE = 0  
  VSPACE = 0  
  ALIGN = middle  
>  
  <paramname="path" value="http://raiders.psych.utah.edu:8180/psych/servlet/HomeworkServlet">  
  <paramname="demoMode" value="true">  
</APPLET>
```

Starting Up

This assumes that you have read the documentation on how to run the software. When the program is up and running, and the opening screen appears, click on 'Edit (Password Required)'. The first time you will be asked to input a password. This password gives access to the editing functions, but additional passwords are required to edit any one particular class.

The next screen that appears has a blank area (because there aren't any courses yet) under the 'Select a Course' label. Click on the 'Edit Course List' and then on 'Add a Course'. When you have named your course and inputted a password for editing that particular course, click 'Stop Editing'. The course should now appear in the area under the 'Select a Course' label. Click on the course name. You will be asked for its password before you can proceed (this keeps other instructors from changing your course materials).

At this point the program will have created for the new course a directory with subdirectories and files within the 'Courses' directory. If you want to incorporate in the course the 'Introduction to the Software' homework assignment I provide you need to go into the HomeworkQuiz directory

and into its IntroLessonFiles directory, from there copy the 'Introduction to the Software.str' file into the LessonStringsDirectory of the course, and the gif files into the LessonImages directory of the course.

The next screen that appears lists the students who have inputted their names and passwords into the course. At this point it will be blank. You can either enter yourself as a student in the course or click the 'Enter as Guest' button (which is a bit faster as you don't have to enter a password).

The next screen lists the assignments for the course, which of course is also blank. Click on 'Edit Index'. Then click on 'Insert New First Row', an 'unnamed lesson' will appear. If you would like to use the Introduction to the Software assignment that I have written, then click on the 'unnamed lesson' and change its name to something like 'Introduction to the Software'. If you would like you can change the status, deadline, and type of the lesson by clicking on the appropriate part of the table. Then click 'Stop Editing' and save changes. Programming note, the program will always ask if you want to save changes, whether changes were made or not. Then click on the name of the lesson.

When the first card of the lesson appears (with its name and number of questions) click on 'Proceed'. You can now start to add questions through the 'Edit Menu...' or if you want to use the questions from my intro lesson then do the following (which assumes you've dragged the appropriate files into the course directory as described above):

- 1) Click on the 'Utilities' button, and select 'Work with Text' from the popup menu.
- 2) Click 'Import String File' button.
- 3) Select 'Introduction to the Software.str' and click 'OK'.
- 4) Click either 'Append' or 'Replace', the text representation of the lesson should appear in the text field.
- 5) Click 'Implement Text' which will create a lesson out of the text and return you to the lesson for editing purposes. Along the way you need to respond to the dialog box that warns you that the assignment has not been assigned yet (if this dialog box appears behind another window you'll need to find it before you can proceed).
- 6) You will be returned to the opening screen of the lesson, announcing the name of the lesson and stating that it has 7 questions. Click 'Proceed'.
- 7) You can now save the lesson and return to the index or edit the lesson if you wish.

Writing New Question Types

To write new question types you need:

- 1) To know how specific question types extend class Question (which in turn extends class Card). Information on this is given below.

- 2) Add the question type to the '**New Card**' menu item in the EditToolPanel so that the instructor can add that type of card when creating a lesson. See the *JMenu newCardMenu = new JMenu("New Card")* method in the EditToolPanel Class.
- 3) Write how the question is turned into a String so that later the String can be parsed back into that question. See the *toString (String s)* method description in class Card below.
- 4) Make some changes in class Homework
 - a) Find the list of public static declarations of various card types and add the new card type.
 - b) Add the card to the *createCards()* method.
 - c) Add the card to the *displayCardFromString(String cardText)* method.

abstract public class Card extends JPanel

- 1) Sets the size of all cards
- 2) Sets layout to GridBagLayout
- 3) *abstract public JPanel getCardControlPanel()*. Each extension of card needs to provide a control panel that allows the editor to set the various options of the card during editing mode. If the card needs no controls, then a *new JPanel()* can be returned.
- 4) *public void display()*. When the program begins, one static instance of each type of Card is created. When it is time to display that card this method is evoked. Most extensions of Card need information (in the form of text) to display correctly, and have a *display(String text)* method that puts information in various text fields, and then call *super.display()* to evoke this method.
- 5) *public String toString(String s)*. Each card needs to turn itself into a String that can then be parsed back into the card at a later time. For example, a MultipleChoiceQuestion extends Question extends Card. The *toString()* method of that question type fills in the String information that is unique to that question (e.g. the text of the question, what the correct answer is), then passes that String to the *toString(String s)* method of the parent class Question, which fills in the String information common to all questions, which then passes that String to the *toString(String s)* of class Card, which adds the String information common to all cards. An example of how a question is formatted into a String so that it can be parsed is given at the end of this document.
- 6) *public String newCardString(String s)*. This is the same as the *toString(String s)* described above, but as the question is turned into a String, default information is provided rather than examining the question for such things as the text of the question, what the correct answer is, etc.

7) *public String getType()*. This returns a String representation of the question type (e.g. "MultipleChoiceQuestion"). This information is obtained by parsing out the name of the question from the full String representation of the question.

abstract public class Question extends Card

1) Sets up almost everything having to do with a question (e.g. the title panel, the panel that displays the question, the panel that gives feedback, etc.). Various types of questions (i.e. classes that extend class Question) differ really only in terms of:

- a) How the student inputs an answer (e.g. fill in the blank or multiple choice).
- b) How much room the 'input answer' panel needs (e.g. multiple choice questions need more room than fill in the blank).
- c) And what type of controls are present in the card control panel (e.g. multiple choice questions have a 'random' button that allows the instructor to say that the various options should be shown in random order when the question is displayed). The Question class is abstract partly because the *abstract JPanel getCardControlPanel()* of the parent class Card is left up to specific question types to define.

2) *abstract public PlainDocument getCorrectAnswerDocument (JTextField field)*. This is a way to insure that the correct answer to the question provided by the instructor fits the type of question being asked. Some question types extend PlainDocument so that the correct answer must make sense given the question type. For example, in a fill-in-the-number question the correct answer provided by the instructor must be something that can be translated from a String into a number, in a multiple-choice question the correct answer must be a letter that fits one of the options provided in the question, etc.

3) *abstract public boolean evaluate()*. Each question type has to have a way to compare the user's answer with the correct answer

An Example Of How Questions Are Stored As Strings So That They Can Be Parsed Back Into Questions

The following is an example of how a multiple choice question is stored as a String.

```
[CARD.START]
[TYPE]MultipleChoice[E]
[RANDOMIZE>true[E]
[N.SELECTIONS]3[E]
[a]Ph.D.[E]
[b]Gordon[E]
[c]Oakley[E]
[STORY.PROBLEM.NAME][E]
```

```
[IMAGE.NAME][E]
[QUESTION]What is the last name of the professor of this course?[E]
[ANSWER]b[E]
[COMMENT]Do you have any idea how many people get this wrong?[E]
[CARD.END]
```

Class Homework has the following static method:

```
public static String parse(String s, String startString) {
    return parse(s,startString,"[E]");
}
```

If the String representation of the above question was named "questionText", then the following:

Homework.parse(questionText,"[TYPE]") would return "MultipleChoice".