




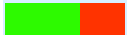



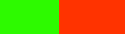
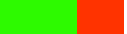
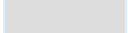
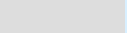
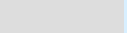



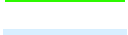
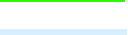
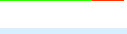



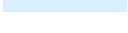
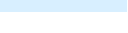
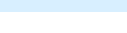




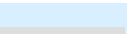
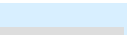

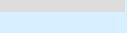
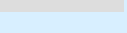

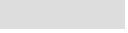
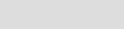
Coverage Report




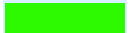
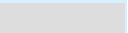
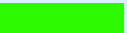









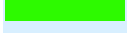
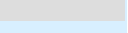
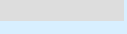

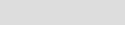
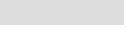

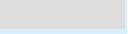
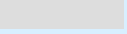



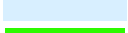
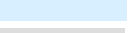
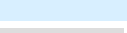
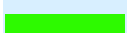
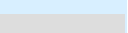
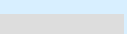


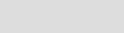









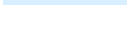
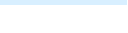
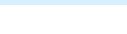

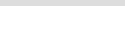

8-Mar-2014 12:47:39 PM


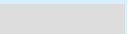
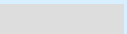

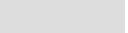
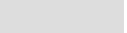

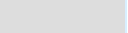
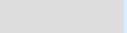



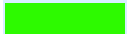
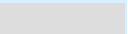
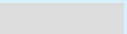




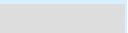
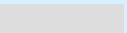




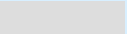
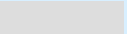




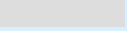
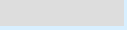

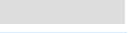
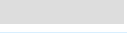

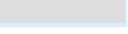
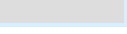

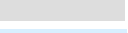
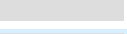

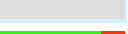
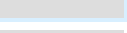


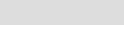
measured on March 8, 2014 12:47:39 PM MST

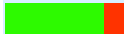
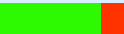
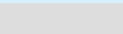




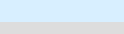
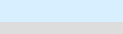
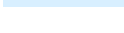
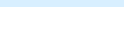
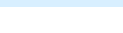
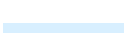
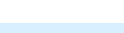
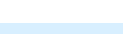
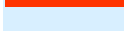
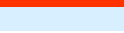
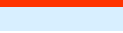

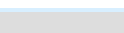
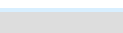



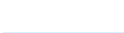
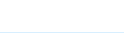
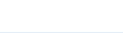

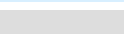
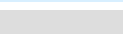







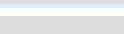
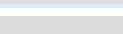

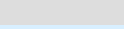
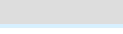
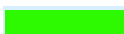
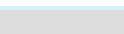
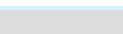

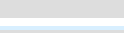
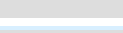


















Description:


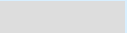
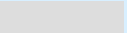

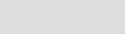
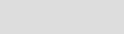
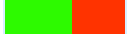
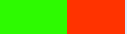


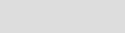
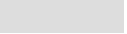

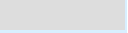
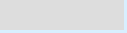




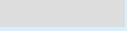
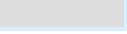




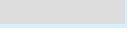
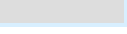

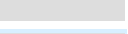
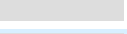

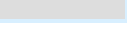
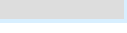




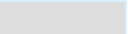
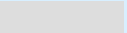






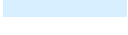
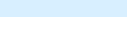
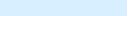



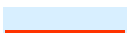
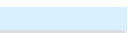
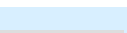

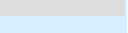
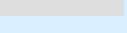

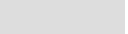
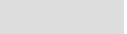

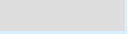
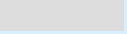


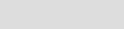

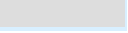
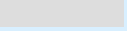


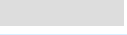
Name	Amount
default package	1
package	1
class	7
method	167
anonymous class	4
enumeration	1

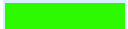
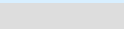
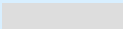




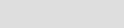
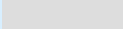




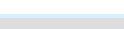
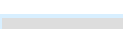

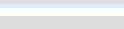
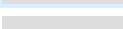

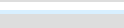
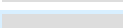
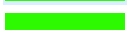
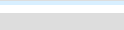
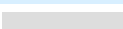
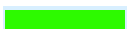
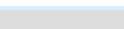
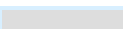




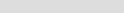
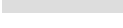

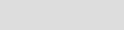
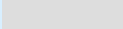

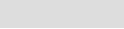
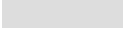

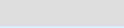
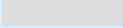




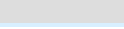
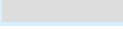



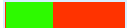







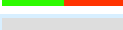

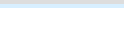
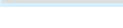










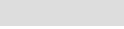
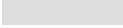

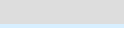
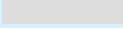

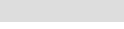
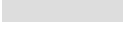

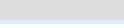
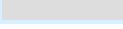
	Statement Coverage				Branch Coverage				Term Coverage			
default package	325 / 507	64 %			75 / 159	47 %			65 / 142	45 %		
cabra	325 / 507	64 %			75 / 159	47 %			65 / 142	45 %		
Utils	89 / 136	65 %			14 / 29	48 %			19 / 30	63 %		
private Utils()	0 / 0	---			0 / 0	---			0 / 0	---		
public static String getExtension(File file)	5 / 5	100 %			2 / 2	100 %			3 / 4	75 %		
public static boolean endsWith(String test, String... ends)	2 / 2	100 %			2 / 2	100 %			2 / 2	100 %		
public static String sanitizeURL(String url)	1 / 1	100 %			0 / 0	---			0 / 0	---		
public static boolean pushLuck(double chance)	0 / 1	0 %			0 / 0	---			0 / 0	---		
public static int arraySum(int[] nums)	3 / 3	100 %			0 / 0	---			0 / 0	---		
public static int average(int... nums)	5 / 5	100 %			0 / 0	---			0 / 0	---		
public static String toPercent(int chosen, int total)	3 / 3	100 %			0 / 0	---			0 / 0	---		
public static int percent(int chosen, int	3 / 4	75 %			1 / 2	50 %			1 / 2	50 %		





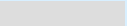
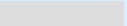



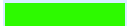
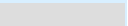
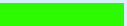








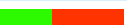




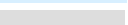
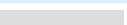

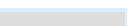
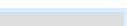
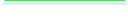
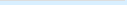
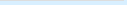






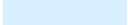
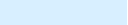
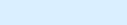

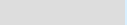
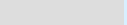
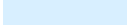
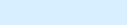
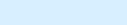






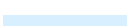
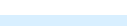
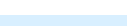
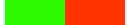
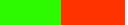
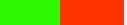
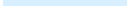
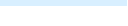
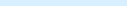



total)									
public static long daysToMillis(int days)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public static Integer[] toIntegerArray(int[] a)	3 / 3	100 %		0 / 0	---		2 / 2	100 %	
public static String stringWithPlural(String string, int quantity)	2 / 2	100 %		2 / 2	100 %		2 / 2	100 %	
public static int numDigits(int x)	2 / 2	100 %		2 / 2	100 %		2 / 2	100 %	
public static String padWithLeadingZeroes(int x, int digits)	4 / 4	100 %		0 / 0	---		2 / 2	100 %	
public static String stringFromArray(String[] array)	5 / 5	100 %		0 / 0	---		0 / 0	---	
public static String[] arrayFromString(String stringified)	3 / 3	100 %		0 / 0	---		0 / 0	---	
public static void purgeFolder(File folder)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public static void obliterate(File file)	5 / 5	100 %		3 / 4	75 %		3 / 4	75 %	
public static JPanel createAdvicePanel(String text)	8 / 8	100 %		0 / 0	---		0 / 0	---	
JPanel	2 / 2	100 %		0 / 0	---		0 / 0	---	
@Override public void paintComponent(Graphics g)	2 / 2	100 %		0 / 0	---		0 / 0	---	
public static void browse(String url)	0 / 7	0 %		0 / 3	0 %		0 / 0	---	
public static JPanel createEditorPane(String pageURL, int width, int height)	0 / 16	0 %		0 / 8	0 %		0 / 4	0 %	
HyperlinkListener	0 / 1	0 %		0 / 2	0 %		0 / 2	0 %	
public void hyperlinkUpdate(HyperlinkEvent evt)	0 / 1	0 %		0 / 2	0 %		0 / 2	0 %	
public static void openURLinDialog(String url, String title, JFrame frame, boolean visible)	0 / 1	0 %		0 / 0	---		0 / 0	---	


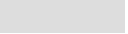
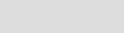

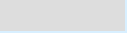
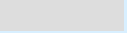



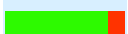
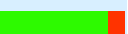
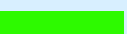
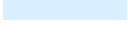
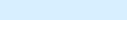
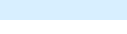




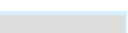
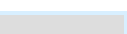

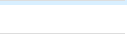
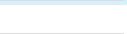




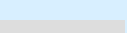
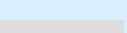

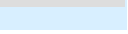
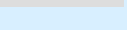



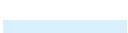
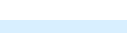
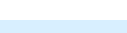

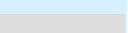
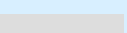

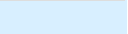
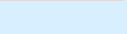












public static void openURLInDialog(String url, String title, String iconPath, JFrame frame, boolean visible)	0 / 12	0 %		0 / 0	---		0 / 0	---	
public static void changeFrameLocation(Component frame, int X, int Y)	3 / 3	100 %		0 / 0	---		0 / 0	---	
public static void centerComponent(Component component, Component owner)	7 / 7	100 %		0 / 0	---		0 / 0	---	
public static void centerOnScreen(Component component)	9 / 9	100 %		0 / 0	---		0 / 0	---	
public static void drawEmblem(JComponent component, Graphics g)	7 / 7	100 %		0 / 0	---		0 / 0	---	
public static JDialog putPanelInDialog(JPanel panel, JFrame owner, String dialogTitle, String iconPath, int width, int height)	7 / 9	77 %		2 / 4	50 %		2 / 6	33 %	
public static void showDialog(JFrame frame, String whatToSay, String title)	0 / 1	0 %		0 / 0	---		0 / 0	---	
public static void showDialog(JFrame frame, String whatToSay, String title, String iconPath)	0 / 1	0 %		0 / 0	---		0 / 0	---	
public static void debug(Exception e, String title)	0 / 5	0 %		0 / 0	---		0 / 0	---	
Status	21 / 32	65 %		10 / 22	45 %		0 / 8	0 %	
public int getReps()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public Color getColor()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public String getToolTipText()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public ImageIcon getImageIcon()	0 / 1	0 %		0 / 0	---		0 / 0	---	
@Override public String toString()	2 / 2	100 %		0 / 0	---		0 / 0	---	
public Status nextRank()	5 / 6	83 %		5 / 6	83 %		0 / 0	---	

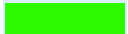
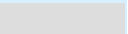
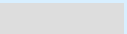




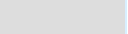
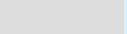

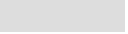
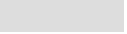



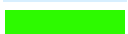


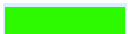
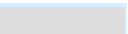
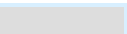










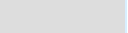
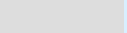

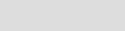
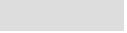
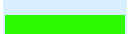
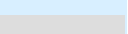
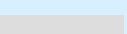

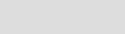
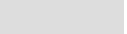
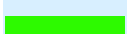
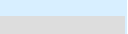
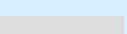








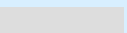

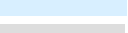
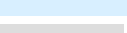

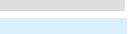
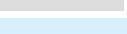
public Status previousRank()	5 / 6	83 %		5 / 6	83 %		0 / 0	---	
Status(char rank, int defaultReps, String hexCode, String toolTipText)	1 / 1	100 %		0 / 0	---		0 / 0	---	
Status(char rank, int defaultReps, Color color, String toolTipText)	4 / 4	100 %		0 / 0	---		0 / 0	---	
public static Status getStatus(String statusName)	0 / 4	0 %		0 / 4	0 %		0 / 2	0 %	
public static String importFromPast(String past)	0 / 4	0 %		0 / 6	0 %		0 / 6	0 %	
Session	31 / 36	86 %		7 / 9	77 %		6 / 8	75 %	
public Session(Project project)	4 / 4	100 %		0 / 0	---		0 / 0	---	
private void setupSession(ArrayList<Card> allCards)	6 / 6	100 %		2 / 2	100 %		5 / 6	83 %	
public boolean update()	0 / 1	0 %		0 / 0	---		0 / 0	---	
public void end()	0 / 1	0 %		0 / 0	---		0 / 0	---	
public Card getCard()	3 / 4	75 %		1 / 2	50 %		1 / 2	50 %	
public void putResult(KnowPanel.Choices choice)	7 / 7	100 %		4 / 5	80 %		0 / 0	---	
public Card reloadCard()	0 / 1	0 %		0 / 0	---		0 / 0	---	
public void cardSkipped()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public int getCurrentIndex()	0 / 1	0 %		0 / 0	---		0 / 0	---	
public int getNumCards()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public int numCards()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public int[] getCardStats()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public boolean isFinished()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public boolean isEmpty()	1 / 1	100 %		0 / 0	---		0 / 0	---	
Sanitizer	3 / 11	27 %		0 / 6	0 %		0 / 8	0 %	
private Sanitizer()	0 / 0	---		0 / 0	---		0 / 0	---	
public static boolean hasDisallowedChar(String string)	0 / 2	0 %		0 / 2	0 %		0 / 2	0 %	
public static String sanitize(String	0 / 6	0 %		0 / 4	0 %		0 / 6	0 %	

string)									
public static String removeSpaces(String string)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public static String removeUnderscores(String string)	1 / 1	100 %		0 / 0	---		0 / 0	---	
Project	49 / 87	56 %		10 / 18	55 %		7 / 16	43 %	
public Project(String name)	3 / 3	100 %		0 / 0	---		0 / 0	---	
public void setSession(Session session)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void newSession()	0 / 1	0 %		0 / 0	---		0 / 2	0 %	
public Session getSession()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void addNote(Note note)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void removeNote(Note note)	3 / 3	100 %		0 / 0	---		0 / 0	---	
public ArrayList<Note> getNotes()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public int numNotes()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void addCard(Card card,Status status)	3 / 7	42 %		2 / 4	50 %		2 / 4	50 %	
public void addCard(Card card)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void addCards(ArrayList<Card> givenCards)	2 / 2	100 %		0 / 0	---		0 / 0	---	
public void removeCard(Card cardToRemove)	2 / 6	33 %		2 / 4	50 %		2 / 4	50 %	
public File copyPictureFile(File pictureFile)	0 / 4	0 %		0 / 0	---		0 / 0	---	
public ImageIcon getImageIcon(String imageName)	0 / 1	0 %		0 / 0	---		0 / 0	---	
public String getPathTo(String thing)	0 / 3	0 %		0 / 0	---		0 / 0	---	
public void save()	0 / 2	0 %		0 / 0	---		0 / 0	---	
public void saveCards()	3 / 5	60 %		2 / 2	100 %		0 / 0	---	
public void saveNotes()	4 / 4	100 %		0 / 0	---		0 / 0	---	
Runnable	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void run()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void loadNotes()	1 / 1	100 %		0 / 0	---		0 / 0	---	

public void shuffle()	2 / 2	100 %		0 / 0	---		0 / 0	---	
public String getName()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void setName(String newName)	3 / 3	100 %		0 / 0	---		0 / 0	---	
public void print(Controller controller)	0 / 2	0 %		0 / 0	---		0 / 0	---	
public void resetAllCards()	0 / 2	0 %		0 / 0	---		0 / 0	---	
public void skipAll()	0 / 1	0 %		0 / 0	---		0 / 0	---	
public ArrayList<Card> getCards()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public boolean isEmpty()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public int numCards()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public int numMatchingCards(Status status)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public int[] cardStatuses()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public Card nextCard()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public File getFolder()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public Card getCurrentCard()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public int getCurrentIndex()	1 / 1	100 %		0 / 0	---		0 / 0	---	
@Override public String toString()	1 / 1	100 %		0 / 0	---		0 / 0	---	
@Override public boolean equals(Object aProject)	2 / 5	40 %		3 / 6	50 %		2 / 4	50 %	
public int compareTo(Project other)	3 / 4	75 %		1 / 2	50 %		1 / 2	50 %	
@Override public int hashCode()	0 / 3	0 %		0 / 0	---		0 / 0	---	
public static void createSampleProject(Controller controller)	0 / 5	0 %		0 / 0	---		0 / 0	---	
Deck	26 / 30	86 %		6 / 8	75 %		7 / 8	87 %	
public Deck()	2 / 2	100 %		0 / 0	---		0 / 0	---	
public ArrayList<Card> getCards()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public Card getCurrentCard()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void makeCurrentCardNull()	0 / 1	0 %		0 / 0	---		0 / 0	---	
public int getCurrentIndex()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public int numCards()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public int numMatchingCards(Status	2 / 2	100 %		2 / 2	100 %		2 / 2	100 %	

status)	3 / 3	100 %		2 / 2	100 %		2 / 2	100 %	
public void add(Card card)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void remove(Card card)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void shuffle()	7 / 7	100 %		0 / 0	---		2 / 2	100 %	
private Card nextCard()	4 / 7	57 %		2 / 4	50 %		1 / 2	50 %	
public Card getCard()	3 / 3	100 %		2 / 2	100 %		2 / 2	100 %	
Controller	72 / 114	63 %		23 / 48	47 %		21 / 48	43 %	
public Controller()	20 / 35	57 %		10 / 20	50 %		9 / 22	40 %	
public GUI getGUI()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public int getPoints()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void gainPoints(PointEnums.Activity activity, boolean refresh)	3 / 3	100 %		1 / 2	50 %		1 / 2	50 %	
public void gainPoints(PointEnums.Activity activity)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public VaultManager createVaultManager(JLabel pointLabel)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void updatePreferredFont(String fontName, int fontSize)	1 / 2	50 %		1 / 2	50 %		1 / 2	50 %	
private ArrayList<Project> loadProjectsFromFile()	3 / 10	30 %		1 / 4	25 %		1 / 4	25 %	
public NotePanel addNoteToActiveProject(NoteTabPane tabPane, Note note)	0 / 3	0 %		0 / 0	---		0 / 0	---	
public void setTheme(Themes theme)	4 / 4	100 %		0 / 0	---		0 / 0	---	
public Project getActiveProject()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public ArrayList<Project> getAllProjects()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public int getNumberOfProjects()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void refreshNow()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void refresh()	1 / 1	100 %		0 / 0	---		0 / 0	---	

public void refreshHomePage()	0 / 3	0 %		0 / 0	---		0 / 0	---	
Runnable	0 / 1	0 %		0 / 0	---		0 / 0	---	
public void run()	0 / 1	0 %		0 / 0	---		0 / 0	---	
public void setActiveProject(Project project, boolean shouldSave)	7 / 8	87 %		7 / 8	87 %		6 / 6	100 %	
public void setActiveProject(String projectName, boolean shouldSave)	3 / 3	100 %		2 / 2	100 %		2 / 2	100 %	
public void setNoActiveProject()	4 / 4	100 %		0 / 0	---		0 / 0	---	
public Project addProject(String projectName, boolean shouldSave)	4 / 4	100 %		0 / 0	---		0 / 0	---	
private Project addProject(Project project, boolean shouldSave)	6 / 6	100 %		0 / 0	---		0 / 0	---	
public void renameProject(Project project, String newName)	2 / 2	100 %		0 / 0	---		0 / 0	---	
public void createProjectFromExistingFile(String projectName, File projectFolder)	0 / 4	0 %		0 / 0	---		0 / 0	---	
public void removeProject(Project project)	5 / 9	55 %		1 / 6	16 %		1 / 6	16 %	
public void addCardToActiveProject(Card card)	0 / 4	0 %		0 / 4	0 %		0 / 4	0 %	
Card	34 / 61	55 %		5 / 19	26 %		5 / 16	31 %	
public Card(Status status, int sessionsLeft, String question, String answer, String pictureName)	5 / 5	100 %		0 / 0	---		0 / 0	---	
public Card(Status status, String question, String answer, String pictureName)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public Card(String question, String answer, String pictureName)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public Card(String question, String answer)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void trimPictureFile()	0 / 2	0 %		0 / 0	---		0 / 0	---	
public boolean hasPicture()	1 / 1	100 %		0 / 0	---		0 / 0	---	

public void setStatus(Status status)	2 / 2	100 %		0 / 0	---		0 / 0	---	
public Status getStatus()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public int sessionsLeft()	0 / 1	0 %		0 / 0	---		0 / 0	---	
public boolean isDueForStudying()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void study(KnowPanel.Choices result)	0 / 7	0 %		0 / 5	0 %		0 / 0	---	
public void skip()	1 / 1	100 %		1 / 2	50 %		1 / 2	50 %	
public String getQuestion()	2 / 2	100 %		0 / 0	---		0 / 0	---	
public String getAnswer()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void setQuestion(String text)	0 / 1	0 %		0 / 2	0 %		0 / 4	0 %	
public void setAnswer(String text)	0 / 1	0 %		0 / 2	0 %		0 / 4	0 %	
public String getPictureName()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public File getPictureFile()	0 / 1	0 %		0 / 0	---		0 / 0	---	
public void setPictureName(String name)	1 / 1	100 %		0 / 0	---		0 / 0	---	
public void removePicture()	1 / 1	100 %		0 / 0	---		0 / 0	---	
public static String bringBackNewlines(String string)	2 / 2	100 %		0 / 0	---		0 / 0	---	
public static String replaceNewlines(String string)	2 / 2	100 %		0 / 0	---		0 / 0	---	
public static Card createCardBasedOnText(String text)	0 / 6	0 %		0 / 2	0 %		0 / 0	---	
@Override public String toString()	3 / 3	100 %		0 / 0	---		0 / 0	---	
@Override public boolean equals(Object aCard)	3 / 5	60 %		4 / 6	66 %		4 / 6	66 %	
@Override public int hashCode()	0 / 6	0 %		0 / 0	---		0 / 0	---	

Utils.java

```

19 public final class Utils {
20
21     private Utils(){} //can't be instantiated
22

```

```
23  /**Returns the extension of a file.
24  *
25  * @param file the file to check
26  * @return the extension
27  * @return <code>null</code> if the file's a folder
28  */
29  public static String getExtension(File file){
30      String extension = null;
31      String name = file.getName(); //like foo.txt
32      int dot = name.lastIndexOf('.');
33
34      if (dot > 0 && dot < name.length() - 1) {
35          //there is an extension and it's not at the end
36          extension = name.substring(dot+1).toLowerCase();
37      }
38      return extension;
39  }
40
41
42  /**Tests to see if the given string ends with any of the options. Used often
    with files/extensions.
43  *
44  * @param test the string to test
45  * @param ends the strings that you want to see at the end
46  * @return true if the string ends with one of the specified string, false
    otherwise
47  */
48  public static boolean endsWith(String test, String... ends){
49      for(String end : ends){
50          if(test.endsWith(end)) return true;
51      }
52      return false;
```

```
53     }
54
55     /** Cleans up a URL, because Java doesn't like URLs with spaces in them
56     *
57     * @param url the raw URL with spaces
58     * @return the URL with %20's instead of spaces
59     */
60     public static String sanitizeURL(String url){
61         return url.replaceAll(" ", "%20");
62     }
63
64     /**
65     * Decides if the random chance goes through or not (a probability trial.)
66     * @param chance the decimal chance that it will happen (0.1 = 10%)
67     * @return true if the random chance goes through, false otherwise
68     */
69     public static boolean pushLuck(double chance) {
70         return Math.random() < chance;
71     }
72
73     /** Finds the sum of an array of ints.
74     *
75     * @param nums the array
76     * @return their sum
77     */
78     public static int arraySum(int[] nums){
79         int sum = 0;
80         for(int i : nums)
81             sum += i;
82         return sum;
83     }
```

```
84
85  /* Finds the average of the numbers.
86
87   * @param nums as many integers as you want
88   * @return the average, shortened to an int
89   */
89  public static int average(int... nums){
90      int count = 0;
91      int total = 0;
92      for(int num : nums){
93          total += num;
94          count++;
95      }
96      return total / count;
97  }
98
99  /**Converts the given selection into a percent.
100
101   *
102   * @param chosen the numerator, or the number of chosen items
103   * @param total the denominator, or the total number of items
104   * @return the percent, formatted as xx% or just x%
105   */
105  public static String toPercent(int chosen, int total){
106      //for example, let's say we had 2/3 passed
107      int percent = percent(chosen,total);
108      String percentString = percent + "%"; //67 + %
109      return percentString;
110  }
111
112  /**Returns the percentage, like 5.
113
114   *
114   * @param chosen the numerator
```

```
115     * @param total the denominator
116     * @return the percent
117     */
118     public static int percent(int chosen, int total){
119         if(total == 0){
120             //dividing by 0 is bad; just return 0%
121             return 0;
122         }
123         //let's say we had 2,3 passed
124         double decimal = chosen / (total + 0.0); //2/3.0 is 0.6666
125         int percent = (int)(Math.round(decimal * 100)); //0.6666 * 100 ~= 0.67
126         return percent;
127     }
128
129     /**
130     * Converts the given number of days to milliseconds.
131     * @param days a certain number of days
132     * @return a number of milliseconds equal to that. It's a long.
133     */
134     public static long daysToMillis(int days){
135         return days * 24 * 60 * 60 * 1000; //24 hours, 60 mins, 60 sec, 1000 ms
136     }
137
138     /**
139     * Turns the given int[] array into an Integer[] array.
140     * @param a an array of ints
141     * @return an array of Integers with the same contents
142     */
143     public static Integer[] toIntegerArray(int[] a){
144         Integer[] Ints = new Integer[a.length];
145         for(int i=0; i<a.length; i++){
```

```
146         Ints[i] = new Integer(a[i]);
147     }
148
149     return Ints;
150 }
151
152 /**
153  * Returns the given string with an "s" at the end if the quantity is present.
154  * stringWithPlural("dog", 5) -> "dogs"
155  * Warning: doesn't take into account unusual endings (curse you, English.)
156  * "child" would become "childs" :(
157  * @param string some sort of noun that can be counted
158  * @param quantity the number of things there are of the string
159  * @return the string + "s" if there's anything but 1, the original string only
160  * if there's only 1
161  */
162 public static String stringWithPlural(String string, int quantity){
163     if(quantity != 1)
164         return string + "s";
165     else
166         return string;
167 }
168
169 /**
170  * Returns the number of digits in x.
171  * @param x a whole number.
172  * @return 1 if x is 0, otherwise the number of digits.
173  */
174 public static int numDigits(int x){
175     if(x == 0) return 1;
176     return (int)(Math.log10(x)) + 1; //250 -> log250 = 2.something -> 2+1 -> 3
177 }
```

```
176
177     /**
178      * Given a number, pads it with leading zeros so the string representation has
the given number of digits.
179      * If there are more digits in x than the parameter digits, then the
representation of x is returned.
180      * @param x a number >= 0.
181      * @param digits a number >= 0.
182      * @return e.g. "00034" if x=34 and digits=5. "532" if x=532 and digits=2
(digits ignored since digits in x > [digits])
183      */
184     public static String padWithLeadingZeroes(int x, int digits){
185         String string = x + ""; //default string representation
186         int zeroesInFront = digits - numDigits(x); //total digits - digits in number
is number to add on front
187
188         //add specified # of 0s on front of string
189         for(int i=0; i<zeroesInFront; i++){
190             string = "0" + string;
191         }
192
193         return string;
194     }
195
196     /**
197      * Given a string array, converts it into an array representation.
198      * @param array an array of strings. Elements should not have semicolons.
199      * @return the stringified array. Use arrayFromString() to decode it.
200      */
201     public static String stringFromArray(String[] array){
202         String string = "[";
203         //tack on each string, add a semicolon to separate them
204         for(String arrayString : array){
```

```
205         string += arrayString + ";";
206     }
207     //chop off the last bit
208     string = string.substring(0, string.length() - 1);
209     string += "];";
210
211     return string;
212 }
213
214 /**
215  * Given a string representing an encoded list of strings (from
216  * stringFromArray), changes it into a list of strings.
217  * @param stringified represents a string array.
218  * @return an array of the strings inside stringified.
219  */
219 public static String[] arrayFromString(String stringified){
220     //chop off starting and ending brackets
221     stringified = stringified.substring(1, stringified.length() - 1);
222     //now each item is delimited by a semicolon; split them
223     String[] strings = stringified.split(";");
224     return strings;
225 }
226
227
228 /**
229  * Guts the given folder. All files inside it are deleted. The given folder is
230  * not deleted.
231  * @param file a folder
232  */
232 public static void purgeFolder(File folder){
233     //delete everything inside this
234     for(File file : folder.listFiles()){
```



```
235         obliterate(file);
236     }
237 }
238
239 /**
240  * Completely obliterates the given file, leaving no trace of it.
241  * @param file
242  */
243 public static void obliterate(File file){
244     if(file.isFile()){
245         file.delete();
246         file.deleteOnExit();
247     }
248     else if(file.isDirectory()){
249         //delete everything inside, then delete this on exit
250         for(File child : file.listFiles()){
251             obliterate(child);
252         }
253         file.delete();
254         file.deleteOnExit();
255     }
256 }
257
258 //GUI UTILITIES
259
260 /** Creates a JPanel that shows some advice to the user.
261  *
262  * @param text the advice (can be HTML)
263  * @return the created JPanel
264  */
265 public static JPanel createAdvicePanel(String text){
```

```
266     JPanel panel = new JPanel(false){
267         @Override
268             public void paintComponent(Graphics g){
269                 super.paintComponent(g);
270                 Utils.drawEmblem(this, g);
271             }
272     };
273     JLabel filler = new JLabel(text);
274     filler.setHorizontalAlignment(JLabel.CENTER);
275     //filler.setVerticalAlignment(JLabel.CENTER);
276     panel.setLayout(new java.awt.GridLayout(1, 1));
277     panel.add(filler);
278     return panel;
279 }

280
281 /** Launches the user's browser to the specified URL.
282  *
283  * @param url the url to open to
284  */
285 public static void browse(String url){
286     url = sanitizeURL(url);
287     java.net.URI uri = null;
288     java.awt.Desktop desktop = java.awt.Desktop.getDesktop();
289     try{
290         uri = new java.net.URI(url);
291         desktop.browse(uri);
292     }
293     catch(java.io.IOException io){
294         //io error
295         System.err.println("Error launching browser! Details: " + io);
296     }
}
```

```
297     catch(java.net.URISyntaxException ex){
298         //error with syntax of URI
299         System.err.println("Bad URI!: " + uri + ", details: " + ex);
300     }
301 }
302
303 /** Creates a JEditorPane (for viewing the web) and puts it in a panel and
returns it
304 *
305 * @param pageURL the URL of the page to open in this text pane
306 * @param width, height
307 */
308 public static JPanel createEditorPane(String pageURL, int width, int height){
309     //build editor pane
310     JEditorPane editorPane = new JEditorPane();
311     editorPane.setEditable(false);
312     editorPane.setFont(FontManager.PREFERRED_FONT);
313
314     //set the page
315     java.net.URL url = null;
316     try{
317         url = new java.net.URL(pageURL);
318     }
319     catch(java.net.MalformedURLException m){
320         //bad URL
321         System.err.println("Bad url! " + m);
322     }
323     if(url != null){
324         try{
325             editorPane.setPage(url);
326         }
327         catch(java.io.IOException io){
```

```
328         //error with setting page
329         //System.err.println("Error loading page " + url);
330     }
331 }
332
333 //enable hyperlinks
334 editorPane.addHyperlinkListener(new HyperlinkListener() {
335     public void hyperlinkUpdate(HyperlinkEvent evt) {
336         if (evt.getEventType() == HyperlinkEvent.EventType.ACTIVATED) {
337             Utils.browse(evt.getURL().toString());
338         }
339     }
340 });
341
342 //put it in a scroll pane
343 JScrollPane scrollPane = new JScrollPane(editorPane);
344
345 scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
346
347 scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
348 //mostly for the license, which has preformatted text
349
350 //put it in a JPanel
351 JPanel panel = new JPanel(new java.awt.BorderLayout());
352 panel.add(scrollPane);
353 panel.setPreferredSize(new java.awt.Dimension(width,height));
354 return panel;
355 }
356
357 /**
358  * Uses a JEditorPane to open the given URL in a dialog
359  * @param url the url to load
```

```
357     * @param title the title of the dialog
358     * @param frame the parent frame
359     * @param visible true if you want the dialog to be visible, false if you want
it to be invisible to user
360     */
361     public static void openURLinDialog(String url, String title, JFrame frame,
boolean visible){
362         openURLinDialog(url, title, "goat16.png", frame, visible);
363     }
364
365     /**
366     * Uses a JEditorPane to open the given URL in a dialog
367     * @param url the url to load
368     * @param title the title of the dialog
369     * @param frame the parent frame
370     * @param visible true if you want the dialog to be visible, false if you want
it to be invisible to user
371     * @param iconPath the path to the image icon for the dialog.
372     */
373     public static void openURLinDialog(String url, String title, String iconPath,
JFrame frame, boolean visible){
374         //create panel with page in it
375         final int WIDTH = 400;
376         final int HEIGHT = 300;
377         JPanel webView = createEditorPane(url, WIDTH, HEIGHT);
378
379         //put it in a dialog
380         JDialog dialog = new JDialog(frame);
381         dialog.setTitle(title);
382         dialog.setIconImage(GUI.createImageIcon(iconPath).getImage());
383         dialog.setContentPane(webView);
384         dialog.pack();
385         dialog.setResizable(false);
```

```
386         dialog.setModal(true);
387         Utils.centerComponent(dialog, frame);
388
389         //open it
390         dialog.setVisible(visible);
391     }
392
393     /** Changes the frame's location to a more central screen location
394     * @param frame the frame to be moved
395     * @param X how far right to move the frame
396     * @param Y how far down to move the frame
397     */
398     public static void changeFrameLocation(Component frame, int X, int Y){
399         Point location = frame.getLocation(); //the window's current location
400         //move the window over and down a certain amount of pixels
401         location.translate(X, Y);
402         //set the location
403         frame.setLocation(location);
404     }
405
406     /** Centers the given component in relation to its owner.
407     *
408     * @param component the component to center
409     * @param owner the parent frame
410     */
411     public static void centerComponent(Component component, Component owner){
412         //find the difference in width to see the offsets
413         int widthDifference = owner.getWidth() - component.getWidth();
414         int heightDifference = owner.getHeight() - component.getHeight();
415
416         //we can divide the differences by 2 and add that to the owner's top left
```

```
417 //and then make that the top left of the component
418 //to center the frame
419 int leftOffset = widthDifference / 2;
420 int topOffset = heightDifference / 2;
421
422 //these are the new locations
423 int left = owner.getX() + leftOffset;
424 int top = owner.getY() + topOffset;
425
426 Utils.changeFrameLocation(component, left, top);
427 }
428
429 /**
430  * Centers the given component on the user's screen.
431  * @param component a component (usually a frame.)
432  */
433 public static void centerOnScreen(Component component){
434     Toolkit toolkit = java.awt.Toolkit.getDefaultToolkit();
435     Dimension screenSize = toolkit.getScreenSize();
436     int screenWidth = (int)screenSize.getWidth();
437     int screenHeight = (int)screenSize.getHeight();
438
439     int componentWidth = component.getWidth();
440     int componentHeight = component.getHeight();
441
442     int top = (screenHeight - componentHeight) / 2;
443     int left = (screenWidth - componentWidth) / 2;
444
445     Utils.changeFrameLocation(component, left, top);
446 }
447
```

```
448     /** Draws an emblem (based on current theme) in the bottom left of the given
component.
449     * Call this in paintComponent() of the component.
450     *
451     * @param component the component to draw on
452     * @param g the Graphics object from paintComponent()
453     */
454     public static void drawEmblem(JComponent component, Graphics g){
455         Themes currentTheme = Themes.getCurrentTheme();
456         ImageIcon image = GUI.createImageIcon("translucent/" +
currentTheme.getImageIconPath());
457
458         /*if(Themes.getCurrentTheme() == Themes.SNOW){
459             image = GUI.createImageIcon("translucent/snow.png");
460         }*/
461
462         int imageWidth = image.getIconWidth();
463         int imageHeight = image.getIconHeight();
464         //top left corner of where to start drawing
465         int topLeftX = component.getWidth() - imageWidth; //x (horizontal)
coordinate
466         int topLeftY = component.getHeight() - imageHeight; //y (vertical)
coordinate
467
468         //draw in bottom right corner
469         g.drawImage(image.getImage(), topLeftX, topLeftY,
(java.awt.image.ImageObserver)null);
470         //g.drawImage(image.getImage(), 0,0,
(java.awt.image.ImageObserver)null);
471     }
472
473     /**
474     * Utility method to put the given panel in a JDialog. You'll have to show it on
your own.
```



```
475     * @param panel the panel to put in a dialog
476     * @param owner the JFrame that owns the dialog. You can pass null.
477     * @param dialogTitle the title for the dialog
478     * @param iconPath the path to the dialog's icon, such as foo.png
479     * @param width the dialog's width. Pass -1 to pack().
480     * @param height the dialog's height. Pass -1 to pack().
481     * @return the JDialog the panel is in
482     */
483     public static JDialog putPanelInDialog(JPanel panel, JFrame owner, String
dialogTitle,
484         String iconPath, int width, int height){
485         JDialog dialog = new JDialog(owner, dialogTitle, true); //boolean means
modality
486
487         dialog.add(panel);
488         dialog.setIconImage(ImageManager.createImageIcon(iconPath).getImage());
489         if(width < 0 && height < 0)
490             dialog.pack();
491         else
492             dialog.setSize(width, height);
493         dialog.setResizable(false);
494         if(owner != null)
495             Utils.centerComponent(dialog, owner);
496         else
497             dialog.setLocationRelativeTo(null);
498
499         return dialog;
500     }
501
502     /**
503     * Creates and shows a dialog.
504     * @param frame the parent frame. Can be null.
```

```
505     * @param whatToSay the body of the dialog.
506     * @param title the title of the dialog.
507     * @param iconPath the icon's name (x.png)
508     */
509     public static void showDialog(JFrame frame,String whatToSay,String title){
510         showDialog(frame, whatToSay, title, "goat64.png");
511     }
512
513     /**
514     * Creates and shows an alert dialog.
515     * @param frame the parent frame. Can be null.
516     * @param whatToSay the body of the dialog.
517     * @param title the title of the dialog.
518     * @param iconPath the icon's name (x.png)
519     */
520     public static void showDialog(JFrame frame,String whatToSay,String title, String
iconPath){
521         JOptionPane.showMessageDialog(
522             frame, //parent
523             whatToSay, //text
524             title, //title
525             JOptionPane.INFORMATION_MESSAGE, //message type
526             GUI.createImageIcon(iconPath) //icon
527         );
528     }
529
530     /**
531     * Convenience overload for debug(Exception, String.) Title is default.
532     * @param e the exception to show
533     */
534     /*public static void debug(Exception e){
535         debug(e, "Cabra error");
```

```
536     */
537
538     /**
539      * Shows an alert dialog for when an exception occurs.
540      * @param e the exception to show
541      * @param title the title of the dialog
542      */
543     public static void debug(Exception e, String title){
544         e.printStackTrace();
545         String text = "<i>" + e.toString() + "</i><br>";
546         //add stack trace
547         for(StackTraceElement ste : e.getStackTrace()){
548             text += ste.toString() + "<br>";
549         }
550         text += "<br><b>Please take a screenshot and email it to
551         <u>neel@hathix.com</u>.</b>";
552         showDialog(null, "<html>Sorry! Cabra has encountered an error. Details:<br>
553         <br>" + text,title);
554     }
```

Status.java

```
14     public enum Status {
15
16         //number is default reps
17         A('A',0,"FF0000","These cards are new, so you'll study them the most."),
18         //red
19         B('B',2,"FF7F00","You know these cards just a little, so you'll study them a
20         lot."),
21         //orange
22         C('C',4,"FFC800","You're in the process of learning these cards, so you'll
23         study these occasionally."),
```

```
22         //gold
23         D('D',8,"0094FF","You know these cards very well, so you won't study them
often."),
24         //blue
25         E('E',12,"00E500","You know these cards cold, so you'll study them rarely.")
26         //green
27     ;
28
29     /* constants */
30     /** Original status of all cards.
31     *
32     */
33     public static final Status DEFAULT_STATUS = Status.A;
34
35     /* class stuff */
36
37     /** Name of rank (A,B,C)
38     *
39     */
40     private char rank;
41
42     /** The number of sessions left until this card is studied. If it's 0, the
card will be studied immediately.
43     * Otherwise it'll be reduced by one until it hits 0.
44     *
45     */
46     private int sessionsLeft;
47
48     /** The color of this status's bar graph and other stuff
49     *
50     */
51     private Color color;
```

```
52
53  /**
54   * Text shown in a tool tip when the bar graph for this is moused over.
55   */
56  private String toolTipText;
57
58  public int getReps(){
59      return sessionsLeft;
60  }
61
62  public Color getColor(){
63      return color;
64  }
65
66  public String getToolTipText(){
67      return toolTipText;
68  }
69
70  public ImageIcon getImageIcon(){
71      return GUI.createImageIcon(name() + ".png");
72  }
73
74  @Override
75  public String toString(){
76      String string = new String(new char[]{ rank });
77      return string;
78  }
79
80  /** Returns the rank after this one. If rank is A, it returns B.
81   *
82   * @return the rank after this one.
```

```
83     */
84     public Status nextRank(){
85         //determine it based on this rank
86         switch(rank){
87             case 'A':
88                 return Status.B;
89             case 'B':
90                 return Status.C;
91             case 'C':
92                 return Status.D;
93             case 'D':
94                 return Status.E;
95             case 'E':
96                 return Status.E; //can't go any higher
97             default:
98                 return Status.DEFAULT_STATUS; //shouldn't happen, it's just here
99         }
100     }
101
102     /** Returns the rank before this one. If this rank is C, this method returns
103     B.
104     *
105     * @return the rank before this one.
106     */
107     public Status previousRank(){
108         switch(rank){
109             case 'A':
110                 return Status.A; //can't go any lower
111             case 'B':
112                 return Status.A;
113             case 'C':
```

```
113         return Status.B;
114     case 'D':
115         return Status.C;
116     case 'E':
117         return Status.D;
118     default:
119         return Status.DEFAULT_STATUS; //shouldn't happen, it's just here
to please compiler
120     }
121 }
122
123 /** A convenient overload that lets you pass the hex code of the color and
not the color itself
124 *
125 * @param rank the letter to display for the rank (A-E)
126 * @param defaultReps how many study sessions elapse between studying.
127 * @param hexCode the hex code of the color, like FF0000
128 * @param tooltipText the text displayed on the bar graph's tooltip
129 */
130 Status(char rank, int defaultReps, String hexCode, String tooltipText){
131     this(rank,defaultReps,ColorManager.createColor(hexCode),tooltipText);
132 }
133
134 Status(char rank, int defaultReps, Color color, String tooltipText){
135     this.rank = rank;
136     this.sessionsLeft = defaultReps;
137     this.color = color;
138     this.tooltipText = tooltipText;
139 }
140
141 /** Tries to find the status with the given name.
142 *
```

```
143 * @param statusName A0,E3,B1... first rank, then number of reps left
144 * @return the status, or the default if nothing is found
145 */
146 public static Status getStatus(String statusName){
147     try{
148         Status status = Status.valueOf(statusName);
149
150         return status;
151     }
152     catch(IllegalArgumentException e){
153         //not a valid status
154         //importing statuses from 0.4.x; learned is B, not learned/not
studied is A
155         if(statusName.equals("learned"))
156             return Status.B;
157         else
158             return Status.A;
159     }
160 }
161
162 /** Takes a status string (new or from 0.4.x) and changes it to the new
version.
163 *
164 * @param past the status string from the text file.
165 * @return the new status string, like "A0" or "C2"
166 */
167 public static String importFromPast(String past){
168     if(past.equals("learned")){
169         //c status
170         return "C" + Status.C.sessionsLeft;
171     }
172     else if(past.equals("not_learned")){
```



```
173         //a status
174         return "A" + Status.A.sessionsLeft;
175     }
176     else if(past.equals("not_studied")){
177         //a status
178         return "A" + Status.A.sessionsLeft;
179     }
180     else{
181         //it's new, so no need to adapt it
182         return past;
183     }
184 }
185 }
```

Session.java

```
14 public class Session extends Object{
15
16     private Project project; //the project we're studying for
17
18     private int currentIndex = 0; //how many cards we have studied
19     private ArrayList<Card> cards; //cards we'll study
20     private Card currentCard; //current card we're studying
21     private int numLearned = 0;
22     private int numNotLearned = 0;
23     private int numSortOf = 0;
24     private int numSkipped = 0;
25
26     public Session(Project project){
27         //new Exception().printStackTrace();
28         this.project = project;
29
30         project.setSession(this);
```

```
31
32     cards = new ArrayList<Card>();
33     setupSession(project.getCards());
34 }
35
36 /** Sets up the session; creates the list of cards that will be studied.
37  *
38  * @param allCards
39  */
40 private void setupSession(ArrayList<Card> allCards){
41     //determine the maximum number of cards the user wants to study in a
session
42     //and use that as a limit
43     int maxCards = UserData.getIntPref("MaxSession");
44     int added = 0;
45     for(int i=0; i<allCards.size() && added < maxCards; i++){
46         Card card = allCards.get(i);
47         if(card.isDueForStudying()){
48             cards.add(card);
49             added++;
50         }
51         else{
52             //we won't study it
53             card.skip();
54         }
55     }
56 }
57
58 /**Update the numbers to match the current project's new numbers (so it's
called when a card is added to the active project)
59  *
60  * @return true if the session has ended, false otherwise
```

```
61     */
62     public boolean update(){
63         //has the session ended?
64         return this.getNumCards() == 0;
65     }
66
67     /*    //since totalCards was hard-coded, change it to reflect any new cards
or something
68         int oldCards = totalCards;
69         totalCards = getAcceptedCards().size();
70
71         if(oldCards != totalCards){
72             //a card was just added, but that doesn't really matter
73             //as long as totalCards is incremented we're happy (the card is added
on the end of deck)
74         }
75         return totalCards == 0; //session ends if no cards are left
76     }*/
77
78     /**Signals this session that it's been ended, normally or prematurely.
79     *
80     */
81     public void end(){
82         //if the session was ended prematurely, then the number of total cards
won't equal the number of learned/notlearned/notstudied
83
84         //if the user quit early, don't let the cards you didn't study show up as
skipped
85         //numCards = numLearned + numNotLearned + numSkipped;
86
87         //totalCards = numLearned + numNotLearned + numSkipped;
88         project.setSession(null);
89     }
```

```
90
91  /**Gets a card based on the filters and project.
92  *
93  * @return the chosen card
94  * @return null if the session has ended
95  */
96  public Card getCard(){
97      if(currentIndex >= numCards()){
98          //this session should be over
99          return null;
100      }
101      //just grab a card that's been chosen
102      currentCard = cards.get(currentIndex);
103      currentIndex++; //this could go over, that's ok since the session should
always be checked after each card
104
105      return currentCard;
106  }
107
108  /**Doesn't do anything, this guy just wants to know when something happens.
109  *
110  * @param card the card that is about to be removed
111  */
112  /*public void removeCard(Card card){
113      if(currentIndex == totalCards){
114          //because this is the last card of the deck (3 of 3), roll everything
back 1 (2 of 2.)
115          //since nextCard() increases currentIndex by 1, reduce currentIndex
by 2.
116
117          totalCards--;
118          currentIndex -= 2;
```

```
119         }
120         else{
121             totalCards--;
122             currentIndex--;
123         }
124     }*/
125
126     /**Adds data about how well the user did to this session.
127     *
128     * @param status Status.LEARNED if they got it, Status.NOT_LEARNED if they
129     * didn't
130     */
131     public void putResult(KnowPanel.Choices choice){
132         switch(choice){
133             case YES:
134                 numLearned++;
135                 break;
136             case NO:
137                 numNotLearned++;
138                 break;
139             case SORT_OF:
140                 numSortOf++;
141                 break;
142             case SKIPPED:
143                 //handled in cardSkipped()
144                 break;
145         }
146     }
147
148     /**Finds this session's current card and sets it as active. Handy if you just
149     switched from another project.
150     *
```

```
149     */
150     public Card reloadCard(){
151         return currentCard;
152     }
153
154     /**As usual, this guy only wants to be informed. Call it when Skip is pressed
155     *
156     */
157     public void cardSkipped(){
158         numSkipped++;
159         //card doesn't need to be informed
160     }
161
162     public int getCurrentIndex(){
163         return currentIndex;
164     }
165
166     /*public void decreaseCurrentIndex(){
167         currentIndex--;
168     }*/
169
170     /* GETTERS */
171
172     public int getNumCards(){
173         //before you give it back make sure there aren't any cards you missed
from quitting early
174         return cards.size();
175     }
176
177     /**Alias for getNumCards().
178     *
179     * @return the number of cards in this session
```

```
180     */
181     public int numCards(){
182         return getNumCards();
183     }
184
185     /**Returns the stats of this session.
186     *
187     * @return an int[] with 4 values: learned, not learned, sort of, skipped.
188     */
189     public int[] getCardStats(){
190         return new int[] { numLearned, numNotLearned, numSortOf, numSkipped };
191     }
192
193     /**Determines if this session is done.
194     *
195     * @return true if it is done, false otherwise.
196     */
197     public boolean isFinished(){
198         //current index is 0 based, so anything too large returns true
199         //System.out.println(currentIndex + " out of " + numCards());
200         return currentIndex > numCards();
201     }
202
203     /** Finds out if there are no cards to be studied.
204     *
205     * @return true if there are cards to be studied, false otherwise
206     */
207     public boolean isEmpty(){
208         return numCards() == 0;
209     }
210 }
```

Sanitizer.java

```
12 public abstract class Sanitizer {
13
14     private Sanitizer(){}
15
16     /**Marks which characters cannot be allowed in file names
17     *
18     */
19     public static final String DISALLOWED_CHARS = "\\/:?*.<>|\"";
20
21     /** Determines if a certain String has any disallowed chars in it
22     *
23     * @param string the String to test
24     * @return true if it has a disallowed char, false if not
25     */
26
27     public static boolean hasDisallowedChar(String string){
28         //go through each char and see if it's disallowed
29         for(char letter : string.toCharArray()){
30             if(DISALLOWED_CHARS.indexOf(letter + "") != -1){
31                 //if the disallowed chars string contains this char, it's bad
32                 return true;
33             }
34         }
35
36         return false;
37     }
38
39     /**Finds any disallowed chars in the string and removes them.
40     *
41     * @param string the string to test
```



```
41
42     * @return the sanitized string
43     */
44
45     public static String sanitize(String string){
46         if(hasDisallowedChar(string) == false)
47             return string; //nothing to change
48         StringBuilder builder = new StringBuilder(string);
49         for(int i=0; i<builder.length(); i++){
50             char letter = builder.charAt(i);
51             if(DISALLOWED_CHARS.indexOf(letter + "") != -1){
52                 //this char is disallowed
53                 builder = builder.deleteCharAt(i);
54                 i--;
55             }
56         }
57
58         return builder.toString();
59     }
60
61
62     /**Replaces any spaces in the string with underscores, i.e. "Forty two"
63     becomes "Forty_two"
64     *
65     * @param string the string to remove the spaces from
66     * @return the new string
67     */
68
69     public static String removeSpaces(String string){
70         return string.replaceAll(" ", "_");
71     }
```

```
71
72     /** Replaces underscores in the string with spaces, i.e. "Forty_two" becomes
73     "Forty two"
74     *
75     * @param string the string to remove the underscores from
76     * @return the new string
77     */
78     public static String removeUnderscores(String string){
79         return string.replaceAll("_", " ");
80     }
81 }
```

Project.java

```
17 public class Project implements Comparable<Project>{
18     private Session session = null; //if a study session is going on, it's here
19     private Deck deck; //all the cards in this project are here
20     private String name; //the name of this project, like "History Test"
21     //private Card currentCard; //just for tracking, the current card you're on
22     private ArrayList<Note> notes; //each project has its own notes
23
24     public Project(String name){
25         //name is something like "History Test"
26         this.name = name;
27         deck = new Deck();
28         notes = new ArrayList<Note>();
29     }
30
31     /* SESSION STUFF */
32     public void setSession(Session session){
33         this.session = session;
34     }
```

```
34
35
36  /** Creates a new session for this project.
37  *
38  */
39  public void newSession(){
40      do{
41          setSession(new Session(this));
42      }
43      while(getSession().isEmpty());
44  }
45
46  public Session getSession(){
47      return this.session;
48  }
49
50  /* NOTE STUFF */
51
52  /**Add an existing note to this project
53  *
54  * @param note the note to add
55  */
56  public void addNote(Note note){
57      notes.add(note);
58  }
59
60  /** Takes the given note out of the notes list.
61  *
62  * @param note the note to remove
63  */
64  public void removeNote(Note note){
```

```
65         notes.remove(note);
66
67         //delete the file
68
69         File noteFile = new File(SaveLoad.getProjectFolder() + "/" + name + "/" +
note.getName() + "." + Note.EXTENSION);
70         //get the note's file name
71
72         //deletes the file
73         noteFile.delete();
74     }
75
76     /**
77      * Returns this project's notes.
78      * @return the notes, in ArrayList form
79      */
80     public ArrayList<Note> getNotes(){
81         return notes;
82     }
83
84     /**
85      * Returns how many notes there are.
86      * @return the number of notes
87      */
88     public int numNotes(){
89         return notes.size();
90     }
91
92     /* CARD STUFF */
93
94     public void addCard(Card card, Status status){
95         card.setStatus(status);
```

```
96     deck.add(card);
97
98     //if the card has a picture, move the picture over here
99     if(card.hasPicture()){
100         /*if(!new File(this.getPathTo(card.getPictureName())).exists()){
101             //the picture file is corrupted or doesn't exist... remove it
102             card.removePicture();
103
104             return;
105         }*/
106
107         File copiedFile =
108         copyPictureFile(card.getPictureFile());
109
110         //resize the image so it's the same size as the studying picture
111         panel; reduces file size
112         /** DISABLED so we can see image in full size some time **/
113         /*ImageManager.saveImage(ImageManager.scaleImage(
114             GUI.createImageIconFromFullPath(copiedFile.getAbsolutePath()),
115             PicturePanel.PICTURE_WIDTH,
116             PicturePanel.PICTURE_HEIGHT),
117             copiedFile);*/
118
119         card.setPictureName(copiedFile.getAbsolutePath());
120         //and now trim the card's picture file... we won't need the full path
121         any more
122         card.trimPictureFile();
123     }
124
125     saveCards();
126
127     //since there's a new card, notify the session
```

```
124
125     if(session != null){
126         session.update();
127     }
128 }
129
130 public void addCard(Card card){
131     //we don't know if it's important or not
132     //however, the card knows if it's important or not... let's ask
133     addCard(card,card.getStatus()); //ask the card if it's importnat
134 }
135
136 public void addCards(ArrayList<Card> givenCards){
137     //called during initialization to create cards
138     //significantly reduces overhead by only saving at end
139     for(Card card : givenCards){ //go through each card
140         deck.add(card);
141     }
142     //now that the cards have been added, shuffle and save
143     shuffle();
144
145     //session = new Session(this);
146 }
147
148 /** Removes the given card from the card array... that's it. Well, it also
149     saves.
150     *
151     * @param cardToRemove the card to get rid of
152     */
153
153 public void removeCard(Card cardToRemove){
```

```
154         deck.remove(cardToRemove);
155         //if the card being removed was active (it probably was), set active card
to null
156         if(cardToRemove.equals(deck.getCurrentCard())){
157             deck.makeCurrentCardNull();
158         }
159         //delete the card's picture, if it has one
160         if(cardToRemove.hasPicture()){
161             String path = getPathTo(cardToRemove.getPictureName()); //finds the
full path to the image
162             File fileToRemove = new File(path);
163             //delete the file
164             fileToRemove.delete();
165         }
166         //save
167         saveCards();
168     }
169
170     /** Copies the given picture file to this guy's folder.
171     *
172     * @param pictureFile the picture file to be copied
173     * @return the new location of the file
174     */
175
176     public File copyPictureFile(File pictureFile){
177         String fileName = pictureFile.getName();
178         File newFile = new File(SaveLoad.getProjectFolder() + "/" + name + "/" +
fileName);
179         ImageManager.copyImage(pictureFile,newFile);
180         return newFile;
181     }
182
183     /** Retrieves an imageicon that is stored in this project's folder
```

```
184      *
185      * @param imageName the name of the icon (foo.png)
186      * @return the created imageicon, or null if the image cannot be found
187      */
188
189      public ImageIcon getImageIcon(String imageName){
190          return GUI.createImageIconFromFullPath(getPathTo(imageName));
191      }
192
193      /** Finds the absolute location of a card/note based on its name
194      *
195      * @param thing the name of the card/note/picture's file, like foo.png
196      * @return the full path to foo.png
197      */
198
199      public String getPathTo(String thing){
200          String folderPath = SaveLoad.getProjectFolder().getAbsolutePath() + "/" +
this.getName(); //to the folder of the image
201          String absolutePath = folderPath + "/" + thing; //the absolute path to
the image
202          return absolutePath;
203      }
204
205      public void save(){
206          //called when this project needs to be saved
207
208          //the methods are split up for convenience
209          saveCards();
210
211          saveNotes();
212      }
213
```



```
214     public void saveCards(){
215         //save all the cards
216         //new Thread(new Runnable(){
217             //    public synchronized void run(){
218                 try{
219                     BufferedWriter writer = new BufferedWriter(new FileWriter(new
220                     File(SaveLoad.getProjectFolder().getAbsolutePath() + "/" + name +
221                     "/cards.txt"))); //write to my card file
222                     for(Card card : deck.getCards()){
223                         //write down each card
224                         writer.write(card.toString()); //card's toString() does
225                         that question/answer thing
226                         writer.newLine();
227                     }
228                     writer.close();
229                 }
230             }
231         }).start();
232     }
233
234     public void saveNotes(){
235         //save notes
236         final Project proj = this;
237         //save in background
238         new Thread(new Runnable(){
239             public void run(){
240                 SaveLoad.saveNotes(proj);
241             }
242         }).start();
```

```
243         SaveLoad.saveNotes(this);
244     }
245
246     /** Tells this project to load notes from the saved files. Call this when
switching to a new active project
247     *
248     */
249
250     public void loadNotes(){
251         notes = SaveLoad.getNotesFromProject(this);
252     }
253
254     public void shuffle(){
255         deck.shuffle();
256         saveCards();
257     }
258
259     public String getName(){
260         return name;
261     }
262
263     public void setName(String newName){
264         File folder = getFolder(); //folder of this project with old name
265
266         this.name = newName;
267
268         //rename project's folder
269         folder.renameTo(new File(SaveLoad.getProjectFolder() + "/" + newName));
270     }
271
272     /**
273     * Prints out this project's cards. The user earns some points by doing this.
```

```
274     * @param controller the controller. Used to gain points.
275     */
276 public void print(Controller controller){
277     Printer.print(this, getCards());
278
279     //earn the points
280     controller.gainPoints(PointEnums.Activity.PRINT_CARDS);
281 }
282
283 /** Resets all cards in this deck to not studied
284     *
285     */
286 public void resetAllCards(){
287     for(Card card : deck.getCards()){
288         card.setStatus(Status.DEFAULT_STATUS);
289     }
290     saveCards();
291 }
292
293 /** The entire session was skipped.
294     *
295     */
296 public void skipAll(){
297     for(Card card : getCards()){
298         card.skip();
299     }
300 }
301
302 public ArrayList<Card> getCards(){
303     return deck.getCards();
304 }
```

```
305
306  /**
307   * Returns true if and only if there are 0 cards in the project.
308   * @return true if there are 0 cards, false otherwise
309   */
310  public boolean isEmpty(){
311      return numCards() == 0;
312  }
313
314  public int numCards(){
315      //returns the number of cards in the card list
316      return deck.numCards();
317  }
318
319  /** Returns, for example, how many not studied cards there are.
320   *
321   * @param status the status to check for (learned, not learned, not studied)
322   * @return the number of matching cards
323   */
324  public int numMatchingCards(Status status){
325      return deck.numMatchingCards(status);
326  }
327
328  /** Returns the statuses of the cards: [A,B,C,D,E]
329   *
330   * @return [cards with status A, cards with B, C, D, E]
331   */
332  public int[] cardStatuses(){
333      return new int[]{
334          numMatchingCards(Status.A),
335          numMatchingCards(Status.B),
```

```
335
336         numMatchingCards(Status.C),
337         numMatchingCards(Status.D),
338         numMatchingCards(Status.E)
339     };
340 }
341
342 public Card nextCard(){
343     return deck.getCard();
344 }
345
346 public File getFolder(){
347     return new File(SaveLoad.getProjectFolder().getAbsolutePath() + "/" +
348         name);
349 }
350
351 public Card getCurrentCard(){
352     return deck.getCurrentCard();
353 }
354
355 public int getCurrentIndex(){
356     return deck.getCurrentIndex();
357 }
358
359 @Override
360     public String toString(){
361         //like "History Test"
362         return name; //this guy's toString is just his name
363     }
364
365 @Override
366     public boolean equals(Object aProject){
367         if(aProject == null) return false;
```

```
366         if(aProject instanceof Project == false) return false;
367         try{
368             Project project = (Project)aProject;
369             //if(project == null) return false;
370             return project.name.equals(this.name); //compare by name, i.e.
"History Test"
371         }
372         catch(Exception e){
373             return false;
374         }
375     }
376
377     /**
378     * Compares the two projects based on name, case insensitive. "ABC" > "XYZ".
379     * @param other
380     * @return +ve if this project is bigger than other, -ve if it's smaller, 0
    if they are equal (names are the same)
381     */
382     public int compareTo(Project other){
383         if(this.equals(other)) return 0;
384         String thisname = this.name.toLowerCase();
385         String othername = other.name.toLowerCase();
386         return thisname.compareTo(othername);
387     }
388
389     @Override
390     public int hashCode() {
391         int hash = 3;
392         hash = 19 * hash + (this.name != null ? this.name.hashCode() : 0);
393         return hash;
394     }
395
```

```
396
397     public static void createSampleProject(Controller controller){
398         Project project = controller.addProject("Sample", true);
399         ArrayList<Card> cards = new ArrayList<Card>();
400         cards.add(new Card(
401             "What is the ultimate answer to life, the universe, and
everything?",
402             "42"));
403         project.addCards(cards);
404
405         //return project;
406         controller.refresh();
407     }
408 }
```

Deck.java

```
14 public class Deck extends Object{
15
16     private ArrayList<Card> cards; //flash cards of owner project
17     private Card currentCard = null;
18     private int currentIndex; //the index of the current card being viewed.
Between 0 and length of cards
19
20     public Deck(){
21         cards = new ArrayList<Card>();
22         currentIndex = 0;
23     }
24
25     public ArrayList<Card> getCards(){
26         return cards;
27     }
28
```

```
29 public Card getCurrentCard(){
30     return currentCard;
31 }
32
33 public void makeCurrentCardNull(){
34     currentCard = null;
35 }
36
37 public int getCurrentIndex(){
38     return currentIndex;
39 }
40
41 public int numCards(){
42     return cards.size();
43 }
44
45 /**Returns the number of cards with the given status.
46  *
47  * @param status the status you want to look for
48  * @return the number of cards with that status
49  */
50 public int numMatchingCards(Status status){
51     int numSelected = 0;
52
53     for(Card card : cards){
54         if(card.getStatus() == status){
55             numSelected++;
56         }
57     }
58
59     return numSelected;
```



```
60     }
61
62     public void add(Card card){
63         cards.add(card);
64     }
65
66     public void remove(Card card){
67         cards.remove(card);
68     }
69
70     //actual meat of the class here
71
72     /**
73      * Shuffles the deck by randomizing the list of cards
74      */
75     public void shuffle(){
76         ArrayList<Card> newCards = new ArrayList<Card>(); //cards will be moved
to here
77
78         while(cards.isEmpty() == false){
79             //keep going until there are no more cards
80             int randomIndex = (int)(Math.random() * cards.size());
81             Card randomCard = cards.get(randomIndex);
82             //move it from old deck to new one
83             cards.remove(randomCard);
84             newCards.add(randomCard);
85         }
86
87         cards = newCards;
88         currentIndex = 0; //now we'll draw from the top of the deck
89     }
90
```

```
91
92  /** Finds the next card in the deck and returns it.
93  *
94  * @return the next card
95  */
96  private Card nextCard(){
97      Card card = null;
98      try{
99          card = cards.get(currentIndex);
100      }
101      catch(IndexOutOfBoundsException e){
102          //tried to access a bad location, so shuffle and try again
103          shuffle();
104          return nextCard(); //return a new card
105      }
106      currentIndex++;
107      if(currentIndex >= cards.size()){
108          //we've run out of cards
109          shuffle(); //for next time
110      }
111      return card;
112  }
113
114  public Card getCard(){
115
116      if(numCards() == 0)
117          return null; //no cards here
118
119      currentCard = nextCard();
120
121      return currentCard;
```

```
122         // }
123     }
124
125 }
```

Controller.java

```
22 public final class Controller extends Object{
23     //communicates with the GUI and object classes to get stuff done
24
25     private GUI gui; //the GUI that is used here
26     private PointManager pointManager;
27     private ArrayList<Project> projects;
28     private Project activeProject; //the project that you create cards for, study
    from, etc.
29
30     public static final double CHANCE_TO_GET_LUCKY = 0.05;
31
32     public Controller(){
33         /* the plan:
34          * IF no existing projects:
35          *     GET new project and make a project with it
36          * ELSE: (existing projects)
37          *     LOAD projects:
38          *         LOAD cards and give them to project
39          *         LOAD notes and give them to project
40          * FINALLY:
41          *     INITIALIZE GUI
42          *     BUILD the GUI using the projects
43          *     IF userData exists:
44          *         LOAD it
45          *     ELSE:
46          *         CREATE a new one with defaults
```

```
47      *      TELL the GUI to adapt to these changes
48      *
49      *
50      */
51  try{
52      /** Is it the first time the program's being booted up?
53      *
54      */
55      boolean firstRun = false;
56
57      //if there isn't a cabraprojects file, create it
58      if(SaveLoad.getProjectFolder().exists() == false){
59          SaveLoad.getProjectFolder().mkdir();
60          //unless the user deleted their data directory, this means this is
the first run
61          firstRun = true;
62      }
63
64      UserData.load();
65      //USER DATA IS NOW LOADED; do any init of prefs or such here
66
67      //create point manager
68      try{
69          pointManager = new PointManager();
70      }
71      catch(NumberFormatException nfe){
72          //if there's an exception like this, the wrong data was loaded into
the User Data
73          //probably done by 0.6.0
74
75          //alert user
76          Utils.showDialog(null,
```

```
77      "Sorry! Your user data seems to have been corrupted and has
been reset.",
78      "User data corrupted");
79
80      //clear all data since something's corrupted
81      UserData.makeAllDefault();
82
83      //reload points
84      pointManager = new PointManager();
85  }
86
87      //user data is set, so update font
88      updatePreferredFont(
89          UserData.getPref("FontName"),
90          UserData.getInt("Prefs.FontSize")
91      );
92
93      //load projects
94      ArrayList<Project> loadedProjects = loadProjectsFromFile(); //these
projects are all stocked with cards/notes
95      this.projects = loadedProjects;
96      this.gui = new GUI(this,loadedProjects);
97
98      //lack of projects matters now
99      if(projects.isEmpty()){
100          //no active project
101          setNoActiveProject();
102      }
103      else{
104          //there is an active project
105          String projectName = UserData.getString("Project"); //the raw name of
the project
106          setActiveProject(projectName,false);
```

```
107     }
108
109     Themes theme = Themes.getThemeByName(UserData.getString("Theme"));
110     setTheme(theme);
111
112     gui.makeFrameVisible();
113
114     //give first run info
115     if(firstRun){
116         //show advice
117         Utils.openURLinDialog("http://www.cabra.hathix.com/cabra/welcome.php",
118                             "Welcome to Cabra!",
119                             gui.getFrame(), true);
120
121         //set user data's latest version to this
122         UserData.setString("Version", About.VERSION);
123
124         //add a default project
125         //this.addProject(new Project("My First Project"), true);
126     }
127
128     //show changelog if this is a new version
129     boolean upgrade = UserData.getString("Version").equals(About.VERSION) ==
false;
130     if(!firstRun && upgrade){
131         //not first run (then new version doesn't matter)    &&    old version
!= new version
132
133         //store new version
134         String version = About.VERSION;
135         UserData.setString("Version", About.VERSION);
136
```

```
137         //show changelog
138         if(About.NIGHTLY){
139             //don't bother with showing nightly changelog; docs are rarely
written for nightlies
140             /*  Utils.showDialog(gui.getFrame(),
141                 "<html><center>Thanks for testing Cabra " + version + "!
<br>As thanks, here's <b>100</b> points!",
142                 "Thanks for upgrading to Cabra " + version + "!",
143                 );  */
144         }
145         else{
146             /* Utils.openURLinDialog("http://cabra.hathix.com/changelog/" +
Utils.sanitizeURL(version) + ".php",
147                 "Thanks for upgrading to Cabra " + version + "!",
148                 gui.getFrame(), true);  */
149         }
150
151         //earn points for upgrading
152         if(About.PRERELEASE)
153             gainPoints(Activity.USE_BETA);
154         else
155             gainPoints(Activity.USE_NEW_VERSION);
156     }
157
158     //if you're lucky, you earn some free points; also don't do it on first
run and overwhelm them w/dialogs
159     if(!firstRun && !upgrade && Utils.pushLuck(CHANCE_TO_GET_LUCKY)){
160         //earn points!
161         int points = Activity.GET_LUCKY.getPoints();
162         Utils.showDialog(gui.getFrame(),
163             "<html><center>"
164             + "I'm feeling generous, so here's <b>" + points + "</b>
free points! Enjoy!",
```

```
165         "You got lucky!",
166         "goatgift.png");
167         gainPoints(Activity.GET_LUCKY);
168     }
169
170     //is it time for an upgrade? see how long it's been since the last check
171     long lastCheck = Long.parseLong(UserData.getString("LastUpdateCheck"));
172     long rightNow = Calendar.getInstance().getTimeInMillis();
173     long updateInterval =
174     Utils.daysToMillis(UserData.getIntPref("UpdateInterval"));
175     if(rightNow - lastCheck >= updateInterval){
176         //time to check for updates, it's been more than the chosen interval
177         since the last one
178         Updates.checkForUpdates(gui);
179     }
180     gui.update();
181     gui.refresh();
182 }
183 catch(Exception e){
184     //some sort of exception threw off the whole thing
185     Utils.debug(e, "Fatal error");
186 }
187
188 public GUI getGUI(){
189     return gui;
190 }
191
192 /**
193  * Returns how many points the user has.
194  * @return the amount of points the user has
```



```
195     */
196     public int getPoints(){
197         return pointManager.getPoints();
198     }
199
200     /**
201     * The user gains points by doing an activity.
202     * @param activity the activity that the user did to gain these points.
203     * @param refresh if the GUI should refresh.
204     */
205     public void gainPoints(PointEnums.Activity activity, boolean refresh){
206         pointManager.gainPoints(activity);
207
208         //show how many points were earned
209         gui.showPointsBadge(activity.getPoints());
210
211         if(refresh)
212             gui.refresh();
213     }
214
215     /**
216     * The user gains points. The GUI will refresh.
217     * @param activity the activity that the user did to gain those points.
218     */
219     public void gainPoints(PointEnums.Activity activity){
220         gainPoints(activity, true);
221     }
222
223     /**
224     * Returns the vault manager used to control buying and display of prizes.
225     * @param pointLabel the label that will be used to display the points the
```

```
user has. Should be pre-made and added to view.
226     */
227     public VaultManager createVaultManager(JLabel pointLabel){
228         return new VaultManager(pointManager, gui, pointLabel);
229     }
230
231     /**
232      * A wrapper around FontManager.updatePreferredFont() that works better.
233      * Updates the PREFERRED_FONT to the given parameters. You should only pass
234      * one. NOTE: you have to validate the frame after this
235      * @param fontName the new font name/family. pass null if you don't want to
236      * change it.
237      * @param fontSize the new size of the font. pass 0 if you don't want to
238      * change it.
239      */
240     public void updatePreferredFont(String fontName, int fontSize){
241         FontManager.updatePreferredFont(fontName, fontSize);
242
243         //validate frame so the changes take effect
244         if(gui != null)
245             gui.update();
246     }
247
248     /** Finds all the projects that the user has and returns them
249      *
250      * @return the user's projects
251      */
252     private ArrayList<Project> loadProjectsFromFile(){
253         //looks for existing project files and, if they're there, creates the
254         projects
255         File mainProjectFolder = SaveLoad.getProjectFolder();
256         if(!mainProjectFolder.exists()){
257             //there is no projects folder, since you're a first-time user
```

```
254         mainProjectFolder.mkdir();
255         //we know there's nothing in the folder so let's leave
256         return new ArrayList<Project>();
257     }
258
259     ArrayList<Project> loadedProjects = new ArrayList<Project>();
260
261     for(File projectFolder : mainProjectFolder.listFiles()){
262         //projectFolder is a folder that contains a project
263
264         if(projectFolder.isFile())
265             continue; //that means it's probably UserData... but regardless,
don't mess with it
266
267         Project project = new Project(projectFolder.getName());
268         loadedProjects.add(project);
269
270         //give it some cards
271         SaveLoad.loadCardsFromProject(project);
272
273         //do this regardless of the user's having cards in the project
274         //and now give notes to the project
275         SaveLoad.loadNotesFromProject(project);
276     } //end foreach
277
278     return loadedProjects;
279 }
280
281
282 /** Adds a note to the active project, and while doing that creates the note
panel
283     *
```

```
284 * @param note the note to add to the active project
285 * @param tabPane the note tab pane that invokes this method
286 * @return the created note panel
287 */
288
289 public NotePanel addNoteToActiveProject(NoteTabPane tabPane, Note note){
290     activeProject.addNote(note);
291
292     //save while we're at it
293     activeProject.saveNotes(); //no need to save cards too
294
295     return new NotePanel(tabPane,gui,this,note);
296 }
297
298
299 public void setTheme(Themes theme){
300     Themes.setTheme(theme); //that'll do it all for us
301
302     //change and save user data
303     UserData.setString("Theme",theme.getName());
304
305
306     //update the look
307     refresh();
308     gui.repaint();
309 }
310
311 public Project getActiveProject(){
312     return activeProject;
313 }
314
```

```
315 public ArrayList<Project> getAllProjects(){
316     return projects;
317 }
318
319 public int getNumberOfProjects(){
320     return projects.size();
321 }
322
323 /** Same as refresh() except it happens in this thread.
324  *
325  */
326 public void refreshNow(){
327     gui.refresh();
328 }
329
330 public void refresh(){
331     //called when the active project is changed or has its cards manipulated
332     //helps disable/enable buttons
333     //Runnable r = new Runnable(){
334     //    public synchronized void run(){
335         refreshNow();
336     //    }
337     //};
338     //javax.swing.SwingUtilities.invokeLater(r);
339     //new Thread(r).start();
340     //gui.refresh();
341 }
342
343 public void refreshHomePage(){
344     Runnable r = new Runnable(){
345         public void run(){
```

```
346         gui.refreshHomePage();
347     }
348 };
```

```
349 //javax.swing.SwingUtilities.invokeLater(r);
350 new Thread(r).start();
351 }
352
353 /** Differs from setActive project in that that just changes active project,
this handles
354 * user interaction
355 * @param projectName the name of the project you wish to be made active
356 * @param shouldSave whether or not user data should be saved
357 */
358 public void setActiveProject(Project project, boolean shouldSave){
359     //if nothing has matched, there's a problem
360     if(project == null){
361         //set the first project as active, then call this method again
362         setActiveProject(projects.get(0), false);
363         return;
364     }
365
366     if(shouldSave){
367         //quick! Save the old notes if they weren't saved yet
368         if(activeProject != null){
369             try{
370                 gui.saveAllNotes(); //that should do it
371             }
372             catch(NullPointerException n){
373                 //error with saving project
374                 System.out.println("Error saving project notes!");
375             }

```

```
376         }
377     }
378
379     activeProject = project;
380     //alert all
381     gui.newActiveProject(project);
382
383     //set user data
384     UserData.setString("Project",project.getName());
385
386     //load the notes for the project
387     // activeProject.loadNotes();
388
389     //fix the home panel, which shows nothing unless this is done
390     refresh();
391 }
392
393 /** Differs from setActive project in that that just changes active project,
394     this handles
395     * user interaction
396     * @param projectName the name of the project you wish to be made active
397     * @param shouldSave whether or not user data should be saved
398     */
399 public void setActiveProject(String projectName, boolean shouldSave){
400     //find the matching project
401     Project project = null;
402     for(Project proj : projects){
403         //System.out.println(proj.getName());
404         if(proj.getName().equals(projectName))
405             project = proj;
406     }
```

```
407
408     setActiveProject(project, shouldSave);
409 }
410
411 /**Sets nothing as the active project.
412  *
413  */
414 public void setNoActiveProject(){
415     UserData.makeDefault("Project");
416     activeProject = null;
417     gui.setFrameTitleByProject(null);
418     //only refresh the tab pane (that removes all the panels and shows a new
one)
419     refresh();
420 }
421
422 /** Creates a new project and adds it
423  *
424  * @param projectName the name of the project you want made
425  * @param shouldSave true if the userData should be saved, false otherwise
426  * @return the created project
427  */
428
429 public Project addProject(String projectName, boolean shouldSave){
430     Project project = new Project(projectName);
431
432     File projectFolder = new
File(SaveLoad.getProjectFolder().getAbsolutePath() + "/" + project.getName());
    //puts the new folder in the projects folder
433     projectFolder.mkdir();
434
435     //really all this method does is make a project and tell addProject to do
its stuff using the project
```



```
436         return addProject(project,shouldSave);
437     }
438
439     private Project addProject(Project project, boolean shouldSave){
440         projects.add(project);
441         gui.addProject(project);
442
443         //add project data
444         //create the card file
445         //File cardFile = new File(projectFolder.getPath() + "/cards.txt");
446
447         project.saveCards(); //forces the creation of cards.txt
448
449         //make this project active
450         setActiveProject(project, shouldSave);
451
452         //sort project list
453         Collections.sort(projects);
454
455         return project;
456     }
457
458     /**
459     * Renames the given project so it has the given name.
460     * @param project the project
461     * @param newName the project's new name.
462     */
463     public void renameProject(Project project, String newName){
464         project.setName(newName);
465         //re-sort projects; the name change may have put project out of order
466         Collections.sort(projects);
```

```
467     }
468
469
470     /** Creates a project assuming you have all the files (i.e. you've just
imported it.)
471     *
472     * @param projectName the name of the project
473     * @param projectFolder the path to the project's folder (inside the Project
Folder)
474     */
475     public void createProjectFromExistingFile(String projectName, File
projectFolder){
476         Project project = new Project(projectName);
477
478         //add cards
479         SaveLoad.loadCardsFromProject(project);
480
481         //add notes
482         SaveLoad.loadNotesFromProject(project);
483
484         addProject(project, true);
485     }
486
487     public void removeProject(Project project){
488         //removes the project at the given index
489         //by the time we get here, we know something will be deleted
490
491         //first delete the project file
492
493         //Project project = projects.get(projectIndexInList); //get the project
slated for deletion
493         int projectIndexInList = projects.indexOf(project); //location of the
project in the list
494         File projectFile = new File(SaveLoad.getProjectFolder() +
"/"+project.getName());
```

```
495 //before we delete the directory we need to delete files inside
496 for(File file : projectFile.listFiles()){
497     file.delete();
498 }
499 //now delete the directory
500 projectFile.delete();
501
502 //what if the project to be removed was the active one?
503 if(project.equals(activeProject)){
504     //set the previous active project
505     if(projects.size() == 1){
506         //the last project was deleted, so nothing's left
507         setNoActiveProject(); //takes care of making activeproject = null
508     }
509     else{
510         //there's still a project left
511         if(projectIndexInList == 0){
512             setActiveProject(projects.get(1),true);
513         }
514         else{
515             setActiveProject(projects.get(projectIndexInList-
516 1),true);
517         }
518         refresh();
519     }
520 }
521 //now remove the project from list
522 projects.remove(projectIndexInList);
523
524 //update project list panel
525 }
```

```
526
527 public void addCardToActiveProject(Card card){
528     //add the card to the project... it'll save itself
529
530     if(activeProject != null){
531         activeProject.addCard(card);
532
533         //gain points
534         gainPoints(Activity.CREATE_CARD);
535         if(card.hasPicture()){
536             gainPoints(Activity.ADD_IMAGE);
537         }
538
539         refresh();
540     }
541     //if there's no active project, take no action
542 }
543
544
545 }
```

Card.java

```
15 public class Card extends Object{
16     //a simple quiz card with a question and answer
17
18     private String questionText;
19     private String answerText;
20     private String pictureName; //this card might have a picture
21     private Status status; //rank
22     private int sessionsLeft; //sessions until next study
23
24     public static final String NO_PICTURE_STRING = " "; //not really empty, just
```

```
24 represents no picture
25 public static final String DELIMITER = "//"; //separates fields
26 public static final String NEWLINE REPLACER = "-nl-"; //\n's are replaced
with this string during saving and loading
27 public static final String NEWLINE = "\n"; //signifies a new line in card
text
28
29
30 //the ultimate one
31 //the rest cascade under this, overloading to the one above it
32 public Card(Status status, int sessionsLeft, String question, String answer,
String pictureName){
33     setStatus(status);
34     this.sessionsLeft = sessionsLeft;
35
36     this.questionText = question;
37     this.answerText = answer;
38     this.pictureName = pictureName;
39 }
40
41 public Card(Status status,String question,String answer,String pictureName){
42     this(status,status.getReps(),question,answer,pictureName);
43 }
44
45 public Card(String question,String answer,String pictureName){
46     this(Status.DEFAULT_STATUS,question,answer,pictureName);
47 }
48
49 public Card(String question,String answer){
50     this(question,answer,NO_PICTURE_STRING);
51 }
52
```

```
53
54     public void trimPictureFile(){
55         //right now we have to full path to the picture... get just the name
56         "foo.png"
57         File picture = new File(pictureName);
58         this.pictureName = picture.getName();
59     }
60
61     /** Tells if this card has a picture
62     *
63     * @return true if it has a picture, false otherwise
64     */
65     public boolean hasPicture(){
66         return !pictureName.equals(NO_PICTURE_STRING);
67     }
68
69     /** Status stuff */
70     public void setStatus(Status status){
71         //System.out.println("Setting status: " + status.toString());
72         this.status = status;
73         this.sessionsLeft = status.getReps();
74         //System.out.println(status.toString() + sessionsLeft);
75     }
76
77     public Status getStatus(){
78         return this.status;
79     }
80
81     public int sessionsLeft(){
82         return sessionsLeft;
83     }
```

```
83
84
85 public boolean isDueForStudying(){
86     return sessionsLeft <= 0;
87 }
88
89 /** This card is studied.
90  *
91  * @param result Choices.YES if it was known, Choices.NO if it wasn't
92  */
93 public void study(KnowPanel.Choices result){
94     switch(result){
95         case YES:
96             //send rank up
97             setStatus(status.nextRank());
98             break;
99         case NO:
100             //send it back to bottom
101             setStatus(Status.A);
102             break;
103         case SORT_OF:
104             //send rank down 1
105             setStatus(status.previousRank());
106             break;
107         case SKIPPED:
108             break;
109     }
110 }
111
112 /** This card isn't studied this round
113  *
```

```
114     */
115     public void skip(){
116         if(sessionsLeft > 0)
117             sessionsLeft--;
118     }
119
120     public String getQuestion(){
121         //replace newlines
122         String text = bringBackNewlines(questionText);
123         //return question
124         return text;
125     }
126     public String getAnswer(){
127         return bringBackNewlines(answerText); //replace the newline replacer with
the actual \n character
128     }
129
130     public void setQuestion(String text){
131         if(text != null && text.equals("")==false)
132             this.questionText = text;
133     }
134
135     public void setAnswer(String text){
136         if(text != null && text.equals("")==false)
137             this.answerText = text;
138     }
139
140     public String getPictureName(){
141         return pictureName;
142     }
143
```



```
144 public File getPictureFile(){
145     return new File(pictureName);
146 }
147
148 /** Changes the name (path) of the picture.
149  *
150  * @param name the path (absolute or relative) of the picture file
151  */
152 public void setPictureName(String name){
153     this.pictureName = name;
154 }
155
156 /** Removes the picture from this card.
157  *
158  */
159 public void removePicture(){
160     setPictureName(Card.NO_PICTURE_STRING);
161 }
162
163 /**Replaces the newline replacers in the given text with real newlines
164  *
165  * @param string the string to be fixed
166  * @return the fixed string
167  */
168
169 public static String bringBackNewlines(String string){
170     //escape all characters
171     //String literal = Matcher.quoteReplacement(string);
172     String fixed = string.replaceAll(Card.NEWLINE_REPLACER, Card.NEWLINE);
173     return fixed;
174 }
```

```
175
176 /** Replaces the newlines in a string of text with the newline replacer.
177  *
178  * @param string the string to be messed with
179  * @return the new string
180  */
181
182 public static String replaceNewlines(String string){
183     //get a literal interpretation of the string
184     //String literal = Matcher.quoteReplacement(string);
185     //now replace stuff
186     string = string.replaceAll(Card.NEWLINE, Card.NEWLINE_REPLACER);
187     return string;
188 }
189
190 /** Creates a card based on the raw data string passed.
191  *
192  * @param text the raw text
193  * @return the created card
194  */
195 public static Card createCardBasedOnText(String text){
196     try{
197         //create a card for each line here
198         String[] stuff = text.split(Card.DELIMITER);
199         //first string is status, next string is question, then answer, then
200         image
201         //cards made in older versions need to be slightly adapted
202         String fixedFirst = Status.importFromPast(stuff[0]);
203
204         Card card = new Card(
205             Status.getStatus(fixedFirst.substring(0,1)),
```

```
206         Integer.parseInt(fixedFirst.substring(1,fixedFirst.length())), //grab
however many digits
207         stuff[1],
208         stuff[2],
209         stuff[3]);
210
211         return card;
212     }
213     catch(Exception e){
214         //a malformed line, maybe?
215         System.out.println("Malformed card! Details:" + e);
216         return null;
217     }
218 }
219
220 @Override
221 public String toString(){
222     //used during saving of this card
223     String text = status.toString() + sessionsLeft + Card.DELIMITER
224                 + questionText + Card.DELIMITER
225                 + answerText + Card.DELIMITER
226                 + pictureName;
227     //replace newlines with the newline replacer
228     text = replaceNewlines(text);
229     //now that it's cleaned up return it
230     return text;
231 }
232
233 @Override
234 public boolean equals(Object aCard){
235     if(aCard == null) return false;
236     if(aCard instanceof Card == false) return false; //if it's not a
```

```
card, stop it
237         Card card = (Card)aCard;
238         if(card == null) return false;
239         return card.answerText.equals(this.answerText) &&
card.questionText.equals(this.questionText)
240         && card.pictureName.equals(this.pictureName)
241         && card.status == this.status;
242         //everything must match
243
244     }
245
246     @Override
247     public int hashCode() {
248         int hash = 7;
249         hash = 43 * hash + (this.questionText != null ?
this.questionText.hashCode() : 0);
250         hash = 43 * hash + (this.answerText != null ? this.answerText.hashCode()
: 0);
251         hash = 43 * hash + (this.pictureName != null ?
this.pictureName.hashCode() : 0);
252         hash = 43 * hash + (this.status != null ? this.status.hashCode() : 0);
253         return hash;
254     }
255 }
```

created on March 8, 2014 1:15:45 PM MST with CodeCover