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

```
public final class Utils {

    private Utils(){} //can't be instantiated

    /**Returns the extension of a file.
     *
     * @param file the file to check
     * @return the extension
     * @return <code>null</code> if the file's a folder
     */
    public static String getExtension(File file){
        String extension = null;
        String name = file.getName(); //like foo.txt
        int dot = name.lastIndexOf('.');

        if (dot > 0 &&  dot < name.length() - 1) {
            //there is an extension and it's not at the end
            extension = name.substring(dot+1).toLowerCase();
        }
        return extension;
    }


    /**Tests to see if the given string ends with any of the options. Used often
with files/extensions.
     *
     * @param test the string to test
     * @param ends the strings that you want to see at the end
     * @return true if the string ends with one of the specified string, false
otherwise
     */
    public static boolean endsWith(String test, String... ends){
        for(String end : ends){
            if(test.endsWith(end)) return true;
        }
        return false;
    }

    /** Cleans up a URL, because Java doesn't like URLs with spaces in them
     *
     * @param url the raw URL with spaces
     * @return the URL with %20's instead of spaces
     */
    public static String sanitizeURL(String url){
        return url.replaceAll(" ","%20");
    }

    /**
     * Decides if the random chance goes through or not (a probability trial.)
     * @param chance the decimal chance that it will happen (0.1 = 10%)
     * @return true if the random chance goes through, false otherwise
     */
    public static boolean pushLuck(double chance) {
        return Math.random() < chance;
    }

    /** Finds the sum of an array of ints.
     *
     * @param nums the array
     * @return their sum
     */
    public static int arraySum(int[] nums){
        int sum = 0;
        for(int i : nums)
            sum += i;
        return sum;
    }

    /* Finds the average of the numbers.
     * @param nums as many integers as you want
     * @return the average, shortened to an int
     */
    public static int average(int... nums){
        int count = 0;
        int total = 0;
        for(int num : nums){
            total += num;
            count++;
        }
        return total / count;
    }

    /**Converts the given selection into a percent.
     *
     * @param chosen the numerator, or the number of chosen items
     * @param total the denominator, or the total number of items
     * @return the percent, formatted as xx% or just x%
     */
    public static String toPercent(int chosen, int total){
        //for example, let's say we had 2/3 passed
        int percent = percent(chosen,total);
        String percentString = percent + "%"; //67 + %
        return percentString;
    }

    /**Returns the percentage, like 5.
     *
     * @param chosen the numerator
     * @param total the denominator
     * @return the percent
     */
    public static int percent(int chosen, int total){
        if(total == 0){
            //dividing by 0 is bad; just return 0%
            return 0;
        }
        //let's say we had 2,3 passed
        double decimal = chosen / (total + 0.0); //2/3.0 is 0.6666
        int percent = (int)(Math.round(decimal * 100)); //0.6666 * 100 ~= 0.67
(we're rounding)
        return percent;
    }

    /**
     * Converts the given number of days to milliseconds.
     * @param days  a certain number of days
     * @return  a number of milliseconds equal to that. It's a long.
     */
    public static long daysToMillis(int days){
        return days * 24 * 60 * 60 * 1000; //24 hours, 60 mins, 60 sec, 1000 ms
    }

    /**
     * Turns the given int[] array into an Integer[] array.
     * @param a an array of ints
     * @return an array of Integers with the same contents
     */
    public static Integer[] toIntegerArray(int[] a){
        Integer[] Ints = new Integer[a.length];
        for(int i=0; i<a.length; i++){
            Ints[i] = new Integer(a[i]);
        }

        return Ints;
    }

    /**
     * Returns the given string with an "s" at the end if the quantity is present.
     * stringWithPlural("dog", 5) -> "dogs"
     * Warning: doesn't take into account unusual endings (curse you, English.)
"child" would become "childs" :(
     * @param string some sort of noun that can be counted
     * @param quantity the number of things there are of the string
     * @return  the string + "s" if there's anything but 1, the original string only
if there's only 1
     */
    public static String stringWithPlural(String string, int quantity){
        if(quantity != 1)
            return string + "s";
        else
            return string;
    }

    /**
     * Returns the number of digits in x.
     * @param x a whole number.
     * @return 1 if x is 0, otherwise the number of digits.
     */
    public static int numDigits(int x){
        if(x == 0) return 1;
        return (int)(Math.log10(x)) + 1; //250 -> log250 = 2.something -> 2+1 -> 3
    }
```

```

    /**
     * Given a number, pads it with leading zeros so the string representation has the given number of digits.
     * If there are more digits in x than the parameter digits, then the representation of x is returned.
     * @param x a number >= 0.
     * @param digits a number >= 0.
     * @return e.g. "00034" if x=34 and digits=5. "532" if x=532 and digits=2 (digits ignored since digits in x > [digits])
     */
    public static String padWithLeadingZeroes(int x, int digits){
        String string = x + ""; //default string representation
        int zeroesInFront = digits - numDigits(x); //total digits - digits in number is number to add on front

        //add specified # of 0s on front of string
        for(int i=0; i<zeroesInFront; i++){
            string = "0" + string;
        }

        return string;
    }

    /**
     * Given a string array, converts it into an array representation.
     * @param array an array of strings. Elements should not have semicolons.
     * @return the stringified array. Use arrayFromString() to decode it.
     */
    public static String stringFromArray(String[] array){
        String string = "[";
        //tack on each string, add a semicolon to separate them
        for(String arrayString : array){
            string += arrayString + ";";
        }
        //chop off the last bit
        string = string.substring(0, string.length() - 1);
        string += "]";

        return string;
    }

    /**
     * Given a string representing an encoded list of strings (from stringFromArray), changes it into a list of strings.
     * @param stringified represents a string array.
     * @return an array of the strings inside stringified.
     */
    public static String[] arrayFromString(String stringified){
        //chop off starting and ending brackets
        stringified = stringified.substring(1, stringified.length() - 1);
        //now each item is delimited by a semicolon; split them
        String[] strings = stringified.split(";");
        return strings;
    }


    /**
     * Guts the given folder. All files inside it are deleted. The given folder is not deleted.
     * @param file a folder
     */
    public static void purgeFolder(File folder){
        //delete everything inside this
        for(File file : folder.listFiles()){
```

```
            obliterate(file);
        }
    }

    /**
     * Completely obliterates the given file, leaving no trace of it.
     * @param file
     */
    public static void obliterate(File file){
        if(file.isFile()){
            file.delete();
            file.deleteOnExit();
        }
        else if(file.isDirectory()){
            //delete everything inside, then delete this on exit
            for(File child : file.listFiles()){
                obliterate(child);
            }
            file.delete();
            file.deleteOnExit();
        }
    }

    //GUI UTILITIES

    /** Creates a JPanel that shows some advice to the user.
     *
     * @param text the advice (can be HTML)
     * @return the created JPanel
     */
    public static JPanel createAdvicePanel(String text){
```

```
                JPanel panel = new JPanel(false){
                @Override
                        public void paintComponent(Graphics g){
                            super.paintComponent(g);
                            Utils.drawEmblem(this, g);

                        }
                };
                JLabel filler = new JLabel(text);
                filler.setHorizontalAlignment(JLabel.CENTER);
                //filler.setVerticalAlignment(JLabel.CENTER);
                panel.setLayout(new java.awt.GridLayout(1, 1));
                panel.add(filler);
                return panel;
        }

        /** Launches the user's browser to the specified URL.
         *
         * @param url the url to open to
         */
        public static void browse(String url){
            url = sanitizeURL(url);
            java.net.URI uri = null;
            java.awt.Desktop desktop = java.awt.Desktop.getDesktop();
            try{
                uri = new java.net.URI(url);
                desktop.browse(uri);
            }
            catch(java.io.IOException io){
                //io error
                System.err.println("Error launching browser! Details: " + io);
            }
            catch(java.net.URISyntaxException ex){
                //error with syntax of URI
                System.err.println("Bad URI!: " + uri + ", details: " + ex);
            }
        }

        /** Creates a JEditorPane (for viewing the web) and puts it in a panel and
    returns it
         *
         * @param pageURL the URL of the page to open in this text pane
         * @param width, height
         */
        public static JPanel createEditorPane(String pageURL, int width, int height){
            //build editor pane
            JEditorPane editorPane = new JEditorPane();
            editorPane.setEditable(false);
            editorPane.setFont(FontManager.PREFERRED_FONT);

            //set the page
            java.net.URL url = null;
            try{
                url = new java.net.URL(pageURL);
            }
            catch(java.net.MalformedURLException m){
                //bad URL
                System.err.println("Bad url! " + m);
            }
            if(url != null){
                try{
                    editorPane.setPage(url);
                }
                catch(java.io.IOException io){
                    //error with setting page
                    //System.err.println("Error loading page " + url);
                }
            }

            //enable hyperlinks
            editorPane.addHyperlinkListener(new HyperlinkListener() {
                public void hyperlinkUpdate(HyperlinkEvent evt) {
                    if(evt.getEventType() == HyperlinkEvent.EventType.ACTIVATED){
                        Utils.browse(evt.getURL().toString());
                    }
                }
            });

            //put it in a scroll pane
            JScrollPane scrollPane = new JScrollPane(editorPane);
    scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
    //mostly for the license, which has preformatted text

            //put it in a JPanel
            JPanel panel = new JPanel(new java.awt.BorderLayout());
            panel.add(scrollPane);
            panel.setPreferredSize(new java.awt.Dimension(width,height));
            return panel;
        }

        /**
         * Uses a JEditorPane to open the given URL in a dialog
         * @param url the url to load
         * @param title the title of the dialog
         * @param frame the parent frame
         * @param visible true if you want the dialog to be visible, false if you want it to be invisible to user
         */
        public static void openURLinDialog(String url, String title, JFrame frame, boolean visible){
            openURLinDialog(url, title, "goat16.png", frame, visible);
        }

        /**
         * Uses a JEditorPane to open the given URL in a dialog
         * @param url the url to load
         * @param title the title of the dialog
         * @param frame the parent frame
         * @param visible true if you want the dialog to be visible, false if you want it to be invisible to user
         * @param iconPath the path to the image icon for the dialog.
         */
        public static void openURLinDialog(String url, String title,  String iconPath, JFrame frame, boolean visible){
            //create panel with page in it
            final int WIDTH = 400;
            final int HEIGHT = 300;
            JPanel webView = createEditorPane(url, WIDTH, HEIGHT);

            //put it in a dialog
            JDialog dialog = new JDialog(frame);
            dialog.setTitle(title);
            dialog.setIconImage(GUI.createImageIcon(iconPath).getImage());
            dialog.setContentPane(webView);
            dialog.pack();
            dialog.setResizable(false);
```

```
            dialog.setModal(true);
            Utils.centerComponent(dialog,frame);

            //open it
            dialog.setVisible(visible);
        }


        /** Changes the frame's location to a more central screen location
         * @param frame the frame to be moved
         * @param X how far right to move the frame
         * @param Y how far down to move the frame
         */
        public static void changeFrameLocation(Component frame, int X, int Y){
            Point location = frame.getLocation(); //the window's current location
            //move the window over and down a certain amount of pixels
            location.translate(X, Y);
            //set the location
            frame.setLocation(location);
        }

        /** Centers the given component in relation to its owner.
         *
         * @param component the component to center
         * @param owner the parent frame
         */
        public static void centerComponent(Component component,Component owner){
            //find the difference in width to see the offsets
            int widthDifference = owner.getWidth() - component.getWidth();
            int heightDifference = owner.getHeight() - component.getHeight();

            //we can divide the differences by 2 and add that to the owner's top left
            //and then make that the top left of the component
            //to center the frame
            int leftOffset = widthDifference / 2;
            int topOffset = heightDifference / 2;

            //these are the new locations
            int left = owner.getX() + leftOffset;
            int top = owner.getY() + topOffset;

            Utils.changeFrameLocation(component, left, top);
        }

          /**
           * Centers the given component on the user's screen.
           * @param component a component (usually a frame.)
           */
          public static void centerOnScreen(Component component){
                Toolkit toolkit = java.awt.Toolkit.getDefaultToolkit();
                Dimension screenSize = toolkit.getScreenSize();
                int screenWidth = (int)screenSize.getWidth();
                int screenHeight = (int)screenSize.getHeight();

                int componentWidth = component.getWidth();
                int componentHeight = component.getHeight();

                int top = (screenHeight - componentHeight) / 2;
                int left = (screenWidth - componentWidth) / 2;

                Utils.changeFrameLocation(component, left, top);
          }

```

```
    /** Draws an emblem (based on current theme) in the bottom left of the given
component.
     * Call this in paintComponent() of the component.
     *
     * @param component the component to draw on
     * @param g the Graphics object from paintComponent()
     */
    public static void drawEmblem(JComponent component, Graphics g){
            Themes currentTheme = Themes.getCurrentTheme();
            ImageIcon image = GUI.createImageIcon("translucent/" +
currentTheme.getImageIconPath());

            /*if(Themes.getCurrentTheme() == Themes.SNOW){
                image = GUI.createImageIcon("translucent/snow.png");
            }*/

            int imageWidth = image.getIconWidth();
            int imageHeight = image.getIconHeight();
            //top left corner of where to start drawing
            int topLeftX = component.getWidth() - imageWidth; //x (horizontal)
coordinate
            int topLeftY = component.getHeight() - imageHeight; //y (vertical)
coordinate

            //draw in bottom right corner
            g.drawImage(image.getImage(), topLeftX, topLeftY,
(java.awt.image.ImageObserver)null);
            //g.drawImage(image.getImage(), 0,0,
(java.awt.image.ImageObserver)null);
    }

    /**
     * Utility method to put the given panel in a JDialog. You'll have to show it on
your own.
```

```
     * @param panel the panel to put in a dialog
     * @param owner the JFrame that owns the dialog. You can pass null.
     * @param dialogTitle the title for the dialog
     * @param iconPath the path to the dialog's icon, such as foo.png
     * @param width the dialog's width. Pass -1 to pack().
     * @param height the dialog's height. Pass -1 to pack().
     * @return the JDialog the panel is in
     */
    public static JDialog putPanelInDialog(JPanel panel, JFrame owner, String
dialogTitle,

            String iconPath, int width, int height){
        JDialog dialog = new JDialog(owner, dialogTitle, true); //boolean means
modality

        dialog.add(panel);
        dialog.setIconImage(ImageManager.createImageIcon(iconPath).getImage());
        if(width < 0 && height < 0)
            dialog.pack();
        else
            dialog.setSize(width, height);
        dialog.setResizable(false);
        if(owner != null)
            Utils.centerComponent(dialog, owner);
        else
            dialog.setLocationRelativeTo(null);

        return dialog;
    }

    /**
     * Creates and shows a dialog.
     * @param frame the parent frame. Can be null.
     * @param whatToSay the body of the dialog.
     * @param title the title of the dialog.
     * @param iconPath the icon's name (x.png)
     */
    public static void showDialog(JFrame frame,String whatToSay,String title){
        showDialog(frame, whatToSay, title, "goat64.png");
    }

    /**
     * Creates and shows an alert dialog.
     * @param frame the parent frame. Can be null.
     * @param whatToSay the body of the dialog.
     * @param title the title of the dialog.
     * @param iconPath the icon's name (x.png)
     */
    public static void showDialog(JFrame frame,String whatToSay,String title, String
iconPath){
        JOptionPane.showMessageDialog(
                                            frame, //parent
                                            whatToSay, //text
                                            title, //title
                                            JOptionPane.INFORMATION_MESSAGE, //mesage type
                                            GUI.createImageIcon(iconPath) //icon
                                       );
    }

    /**
     * Convenience overload for debug(Exception, String.) Title is default.
     * @param e the exception to show
     */
    /*public static void debug(Exception e){
        debug(e, "Cabra error");
    }*/

    /**
     * Shows an alert dialog for when an exception occurs.
     * @param e the exception to show
     * @param title the title of the dialog
     */
    public static void debug(Exception e, String title){
        e.printStackTrace();
        String text = "<i>" + e.toString() + "</i><br>";
        //add stack trace
        for(StackTraceElement ste : e.getStackTrace()){
            text += ste.toString() + "<br>";
        }
        text += "<br><b>Please take a screenshot and email it to
<u>neel@hathix.com</u>.</b>";

        showDialog(null, "<html>Sorry! Cabra has encountered an error. Details:<br>
<br>" + text,title);
    }
}
```

Status.java

```
public enum Status {

    //number is default reps
    A('A',0,"FF0000","These cards are new, so you'll study them the most."),
        //red
    B('B',2,"FF7F00","You know these cards just a little, so you'll study them a
lot."),
        //orange
    C('C',4,"FFC800","You're in the process of learning these cards, so you'll
study these occasionally."),
        //gold
    D('D',8,"0094FF","You know these cards very well, so you won't study them often."),
        //blue
    E('E',12,"00E500","You know these cards cold, so you'll study them rarely.")
        //green
    ;

    /* constants */
    /** Original status of all cards.
     *
     */
    public static final Status DEFAULT_STATUS = Status.A;

    /* class stuff */

    /** Name of rank (A,B,C)
     *
     */
    private char rank;

    /* The number of sessions left until this card is studied. If it's 0, the card will be studied immediately.
     * Otherwise it'll be reduced by one until it hits 0.
     *
     */
    private int sessionsLeft;

    /** The color of this status's bar graph and other stuff
     *
     */
    private Color color;

    /**
     * Text shown in a tool tip when the bar graph for this is moused over.
     */
    private String toolTipText;

    public int getReps(){
        return sessionsLeft;
    }

    public Color getColor(){
        return color;
    }

    public String getToolTipText(){
        return toolTipText;
    }

    public ImageIcon getImageIcon(){
        return GUI.createImageIcon(name() + ".png");
    }

    @Override
    public String toString(){
        String string = new String(new char[]{ rank });
        return string;
    }

    /** Returns the rank after this one. If rank is A, it returns B.
     *
     * @return the rank after this one.
     */
    public Status nextRank(){
        //determine it based on this rank
        switch(rank){
            case 'A':
                return Status.B;
            case 'B':
                return Status.C;
            case 'C':
                return Status.D;
            case 'D':
                return Status.E;
            case 'E':
                return Status.E; //can't go any higher
            default:
                return Status.DEFAULT_STATUS; //shouldn't happen, it's just here
to please compiler
        }
    }

   /** Returns the rank before this one. If this rank is C, this method returns
B.
     *
     * @return the rank before this one.
     */
    public Status previousRank(){
        switch(rank){
            case 'A':
                return Status.A; //can't go any lower
            case 'B':
                return Status.A;
            case 'C':
                    return Status.B;
                case 'D':
                    return Status.C;
                case 'E':
                    return Status.D;
                default:
                    return Status.DEFAULT_STATUS; //shouldn't happen, it's just here to please compiler
            }
    }

    /** A convenient overload that lets you pass the hex code of the color and not the color itself
     *
     * @param rank the letter to display for the rank (A-E)
     * @param defaultReps how many study sessions elapse between studying.
     * @param hexCode the hex code of the color, like FF0000
     * @param toolTipText the text displayed on the bar graph's tooltip
     */
    Status(char rank, int defaultReps, String hexCode, String toolTipText){
        this(rank,defaultReps,ColorManager.createColor(hexCode),toolTipText);
    }

    Status(char rank, int defaultReps, Color color, String toolTipText){
        this.rank = rank;
        this.sessionsLeft = defaultReps;
        this.color = color;
        this.toolTipText = toolTipText;
    }

    /** Tries to find the status with the given name.
     *
     * @param statusName A0,E3,B1... first rank, then number of reps left
     * @return the status, or the default if nothing is found
     */
    public static Status getStatus(String statusName){
        try{
            Status status = Status.valueOf(statusName);

            return status;
        }
        catch(IllegalArgumentException e){
            //not a valid status
            //importing statuses from 0.4.x; learned is B, not learned/not
studied is A
            if(statusName.equals("learned"))
                return Status.B;
            else
                return Status.A;
        }
    }

    /** Takes a status string (new or from 0.4.x) and changes it to the new
version.
     *
     * @param past the status string from the text file.
     * @return the new status string, like "A0" or "C2"
     */
    public static String importFromPast(String past){
        if(past.equals("learned")){
            //c status
            return "C" + Status.C.sessionsLeft;
        }
        else if(past.equals("not_learned")){
            //a status
            return "A" + Status.A.sessionsLeft;
        }
        else if(past.equals("not_studied")){
            //a status
            return "A" + Status.A.sessionsLeft;
        }
        else{
            //it's new, so no need to adapt it
            return past;
        }
    }
}
```

Session.java

```
public class Session extends Object{

    private Project project; //the project we're studying for

    private int currentIndex = 0; //how many cards we have studied
    private ArrayList<Card> cards; //cards we'll study
    private Card currentCard; //current card we're studying
    private int numLearned = 0;
    private int numNotLearned = 0;
    private int numSortOf = 0;
    private int numSkipped = 0;

    public Session(Project project){
        //new Exception().printStackTrace();
        this.project = project;

        project.setSession(this);

        cards = new ArrayList<Card>();
        setupSession(project.getCards());
    }

    /** Sets up the session; creates the list of cards that will be studied.
     *
     * @param allCards
     */
    private void setupSession(ArrayList<Card> allCards){
        //determine the maximum number of cards the user wants to study in a session
        //and use that as a limit
        int maxCards = UserData.getIntPref("MaxSession");
        int added = 0;
        for(int i=0; i<allCards.size() && added < maxCards; i++){
            Card card = allCards.get(i);
            if(card.isDueForStudying()){
                cards.add(card);
                added++;
            }
            else{
                //we won't study it
                card.skip();
            }
        }
    }

    /**Update the numbers to match the current project's new numbers (so it's called when a card is added to the active project)
     *
     * @return true if the session has ended, false otherwise
     */
    public boolean update(){
        //has the session ended?
        return this.getNumCards() == 0;
    }

     /*   //since totalCards was hard-coded, change it to reflect any new cards
or something
        int oldCards = totalCards;
        totalCards = getAcceptedCards().size();

        if(oldCards != totalCards){
            //a card was just added, but that doesn't really matter
            //as long as totalCards in incremented we're happy (the card is added
on the end of deck)
        }
        return totalCards == 0; //session ends if no cards are left
    }*/

    /**Signals this session that it's been ended, normally or prematurely.
     *
     */
    public void end(){
        //if the session was ended prematurely, then the number of total cards
won't equal the number of learned/notlearned/notstudied

        //if the user quit early, don't let the cards you didn't study show up as
skipped
        //numCards = numLearned + numNotLearned + numSkipped;

        //totalCards = numLearned + numNotLearned + numSkipped;
        project.setSession(null);
    }

    /**Gets a card based on the filters and project.
     *
     * @return the chosen card
     * @return null if the session has ended
     */
    public Card getCard(){
        if(currentIndex >= numCards()){
            //this session should be over
            return null;
        }
        //just grab a card that's been chosen
        currentCard = cards.get(currentIndex);
        currentIndex++; //this could go over, that's ok since the session should
always be checked after each card

        return currentCard;
    }

    /**Doesn't do anything, this guy just wants to know when something happens.
     *
     * @param card the card that is about to be removed
     */
    /*public void removeCard(Card card){
        if(currentIndex == totalCards){
            //because this is the last card of the deck (3 of 3), roll everything
back 1 (2 of 2.)
            //since nextCard() increases currentIndex by 1, reduce currentIndex
by 2.

            totalCards--;
            currentIndex -= 2;

        }
        else{
            totalCards--;
            currentIndex--;
        }
    }*/

    /**Adds data about how well the user did to this session.
     *
     * @param status Status.LEARNED if they got it, Status.NOT_LEARNED if they
didn't
     */
    public void putResult(KnowPanel.Choices choice){
        switch(choice){
            case YES:
                numLearned++;
                break;
            case NO:
                numNotLearned++;
                break;
            case SORT_OF:
                numSortOf++;
                break;
            case SKIPPED:
                //handled in cardSkipped()
                break;
        }
    }

    /**Finds this session's current card and sets it as active. Handy if you just
switched from another project.
     *
```

```
     */
    public Card reloadCard(){
        return currentCard;
    }

    /**As usual, this guy only wants to be informed. Call it when Skip is pressed
     *
     */
    public void cardSkipped(){
        numSkipped++;
        //card doesn't need to be informed
    }

    public int getCurrentIndex(){
        return currentIndex;
    }

    /*public void decreaseCurrentIndex(){
        currentIndex--;
    }*/

    /* GETTERS */

    public int getNumCards(){
        //before you give it back make sure there aren't any cards you missed
from quitting early
        return cards.size();
    }

    /**Alias for getNumCards().
     *
     * @return the number of cards in this session
```

```
         */
        public int numCards(){
            return getNumCards();
        }

        /**Returns the stats of this session.
         *
         * @return an int[] with 4 values: learned, not learned, sort of, skipped.
         */
        public int[] getCardStats(){
            return new int[] { numLearned, numNotLearned, numSortOf, numSkipped };
        }

        /**Determines if this session is done.
         *
         * @return true if it is done, false otherwise.
         */
        public boolean isFinished(){
            //current index is 0 based, so anything too large returns true
            //System.out.println(currentIndex + " out of " + numCards());
            return currentIndex > numCards();
        }

        /** Finds out if there are no cards to be studied.
         *
         * @return true if there are cards to be studied, false otherwise

         */
        public boolean isEmpty(){
            return numCards() == 0;
        }
    }
```

Sanitizer.java

```
public abstract class Sanitizer {

    private Sanitizer(){}

    /**Marks which characters cannot be allowed in file names
     *
     */
    public static final String DISALLOWED_CHARS = "\\/:?*.<>|\"";

    /** Determines if a certain String has any disallowed chars in it
     *
     * @param string the String to test
     * @return true if it has a disallowed char, false if not
     */

    public static boolean hasDisallowedChar(String string){
        //go through each char and see if it's disallowed
        for(char letter : string.toCharArray()){
            if(DISALLOWED_CHARS.indexOf(letter + "") != -1){
                //if the disallowed chars string contains this char, it's bad
                return true;
            }
        }

        return false;
    }

    /**Finds any disallowed chars in the string and removes them.
     *
     * @param string the string to test

     * @return the sanitized string
     */

    public static String sanitize(String string){
        if(hasDisallowedChar(string) == false)
            return string; //nothing to change
        StringBuilder builder = new StringBuilder(string);
        for(int i=0;i<builder.length();i++){
            char letter = builder.charAt(i);
            if(DISALLOWED_CHARS.indexOf(letter + "") != -1){
                //this char is disallowed
                builder = builder.deleteCharAt(i);
                i--;
            }
        }

        return builder.toString();
    }


    /**Replaces any spaces in the string with underscores, i.e. "Forty two"
becomes "Forty_two"
     *
     * @param string the string to remove the spaces from
     * @return the new string
     */

    public static String removeSpaces(String string){
        return string.replaceAll(" ","_");
    }
    /** Replaces underscores in the string with spaces, i.e. "Forty_two" becomes
"Forty two"
     *
     * @param string the string to remove the underscores from
     * @return the new string
     */

    public static String removeUnderscores(String string){
        return string.replaceAll("_"," ");
    }
}
```

Project.java

```
public class Project implements Comparable<Project>{
    private Session session = null; //if a study session is going on, it's here
    private Deck deck; //all the cards in this project are here
    private String name; //the name of this project, like "History Test"
    //private Card currentCard; //just for tracking, the current card you're on
    private ArrayList<Note> notes; //each project has its own notes

    public Project(String name){
        //name is something like "History Test"
        this.name = name;
        deck = new Deck();
        notes = new ArrayList<Note>();
    }

    /* SESSION STUFF */
    public void setSession(Session session){

        this.session = session;
    }


    /** Creates a new session for this project.
     *
     */
    public void newSession(){
        do{
            setSession(new Session(this));
        }
        while(getSession().isEmpty());
    }

    public Session getSession(){
        return this.session;
    }

    /* NOTE STUFF */

    /**Add an existing note to this project
     *
     * @param note the note to add
     */
    public void addNote(Note note){
        notes.add(note);
    }

    /** Takes the given note out of the notes list.
     *
     * @param note the note to remove
     */
    public void removeNote(Note note){
        notes.remove(note);

        //delete the file

        File noteFile = new File(SaveLoad.getProjectFolder() + "/" + name + "/" +
note.getName() + "." + Note.EXTENSION);
            //get the note's file name

        //deletes the file
        noteFile.delete();
    }

    /***
     * Returns this project's notes.
     * @return the notes, in ArrayList form
     */
    public ArrayList<Note> getNotes(){
        return notes;
    }

    /**
     * Returns how many notes there are.
     * @return the number of notes
     */
    public int numNotes(){
        return notes.size();
    }

    /* CARD STUFF */

    public void addCard(Card card,Status status){
        card.setStatus(status);
        deck.add(card);

        //if the card has a picture, move the picture over here
        if(card.hasPicture()){
            /*if(!new File(this.getPathTo(card.getPictureName())).exists()){
                //the picture file is corrupted or doesn't exist... remove it
                card.removePicture();

                return;
            }*/

            File copiedFile =
copyPictureFile(card.getPictureFile());

            //resize the image so it's the same size as the studying picture
panel; reduces file size
            /** DISABLED so we can see image in full size some time **/
            /*ImageManager.saveImage(ImageManager.scaleImage(

GUI.createImageIconFromFullPath(copiedFile.getAbsolutePath()),
                    PicturePanel.PICTURE_WIDTH,
                    PicturePanel.PICTURE_HEIGHT),
                    copiedFile);*/

            card.setPictureName(copiedFile.getAbsolutePath());
            //and now trim the card's picture file... we won't need the full path
any more
            card.trimPictureFile();
        }

        saveCards();


        //since there's a new card, notify the session

        if(session != null){
            session.update();
        }
    }

    public void addCard(Card card){
        //we don't know if it's important or not
        //however, the card knows if it's important or not... let's ask
        addCard(card,card.getStatus()); //ask the card if it's importnat
    }

    public void addCards(ArrayList<Card> givenCards){
        //called during initialization to create cards
        //significantly reduces overhead by only saving at end
        for(Card card : givenCards){ //go through each card
            deck.add(card);
        }
        //now that the cards have been added, shuffle and save
        shuffle();

        //session = new Session(this);
    }

    /** Removes the given card from the card array... that's it. Well, it also
saves.
     *
     * @param cardToRemove the card to get rid of
     */


    public void removeCard(Card cardToRemove){
        deck.remove(cardToRemove);
        //if the card being removed was active (it probably was), set active card
to null
        if(cardToRemove.equals(deck.getCurrentCard())){
            deck.makeCurrentCardNull();
        }
        //delete the card's picture, if it has one
        if(cardToRemove.hasPicture()){
            String path = getPathTo(cardToRemove.getPictureName()); //finds the
full path to the image
            File fileToRemove = new File(path);
            //delete the file
            fileToRemove.delete();
        }
        //save
        saveCards();
    }

    /** Copies the given picture file to this guy's folder.
     *
     * @param pictureFile the picture file to be copied
     * @return the new location of the file
     */

    public File copyPictureFile(File pictureFile){
        String fileName = pictureFile.getName();
        File newFile = new File(SaveLoad.getProjectFolder() + "/" + name + "/" +
fileName);
        ImageManager.copyImage(pictureFile,newFile);
        return newFile;
    }

    /** Retrieves an imageicon that is stored in this project's folder
```

```
     *
     * @param imageName the name of the icon (foo.png)
     * @return the created imageicon, or null if the image cannot be found
     */

    public ImageIcon getImageIcon(String imageName){
        return GUI.createImageIconFromFullPath(getPathTo(imageName));
    }

    /** Finds the absolute location of a card/note based on its name
     *
     * @param thing the name of the card/note/picture's file, like foo.png
     * @return the full path to foo.png
     */

    public String getPathTo(String thing){
        String folderPath = SaveLoad.getProjectFolder().getAbsolutePath() + "/" +
this.getName(); //to the folder of the image
        String absolutePath = folderPath + "/" + thing; //the absolute path to
the image
        return absolutePath;
    }

    public void save(){
        //called when this project needs to be saved

        //the methods are split up for convenience
        saveCards();

        saveNotes();
    }

    public void saveCards(){
        //save all the cards
        //new Thread(new Runnable(){
        //    public synchronized void run(){
            try{
                BufferedWriter writer = new BufferedWriter(new FileWriter(new
File(SaveLoad.getProjectFolder().getAbsolutePath() + "/" + name +
"/cards.txt"))); //write to my card file
                for(Card card : deck.getCards()){
                    //write down each card
                    writer.write(card.toString()); //card's toString() does
that question/answer thing
                    writer.newLine();
                }
                writer.close();
            }
            catch(IOException io){
                System.out.println("Couldn't save cards!");
            }
        //    }
        //}).start();
    }

    public void saveNotes(){
        //save notes
        final Project proj = this;
        //save in background

        new Thread(new Runnable(){
            public void run(){
                SaveLoad.saveNotes(proj);
            }
        }).start();
```

```
            SaveLoad.saveNotes(this);
    }

    /** Tells this project to load notes from the saved files. Call this when
switching to a new active project
     *
     */

    public void loadNotes(){
        notes = SaveLoad.getNotesFromProject(this);
    }

    public void shuffle(){
        deck.shuffle();
        saveCards();
    }

    public String getName(){
        return name;
    }

    public void setName(String newName){
        File folder = getFolder(); //folder of this project with old name

        this.name = newName;

        //rename project's folder
        folder.renameTo(new File(SaveLoad.getProjectFolder() + "/" + newName));
    }

    /***
     * Prints out this project's cards. The user earns some points by doing this.
     * @param controller the controller. Used to gain points.
     */
    public void print(Controller controller){
        Printer.print(this, getCards());

       //earn the points
       controller.gainPoints(PointEnums.Activity.PRINT_CARDS);
    }

    /** Resets all cards in this deck to not studied
     *
     */
    public void resetAllCards(){
        for(Card card : deck.getCards()){
            card.setStatus(Status.DEFAULT_STATUS);
        }
        saveCards();
    }

    /** The entire session was skipped.
     *
     */
    public void skipAll(){
        for(Card card : getCards()){
            card.skip();
        }
    }

    public ArrayList<Card> getCards(){
        return deck.getCards();
    }

    /***
     * Returns true if and only if there are 0 cards in the project.
     * @return true if there are 0 cards, false otherwise
     */
    public boolean isEmpty(){
        return numCards() == 0;
    }

    public int numCards(){
        //returns the number of cards in the card list
        return deck.numCards();
    }

    /** Returns, for example, how many not studied cards there are.
     *
     * @param status the status to check for (learned, not learned, not studied)
     * @return the number of matching cards
     */
    public int numMatchingCards(Status status){
        return deck.numMatchingCards(status);
    }

    /** Returns the statuses of the cards: [A,B,C,D,E]
     *
     * @return [cards with status A, cards with B, C, D, E]
     */
    public int[] cardStatuses(){
        return new int[]{
            numMatchingCards(Status.A),
            numMatchingCards(Status.B),

            numMatchingCards(Status.C),
            numMatchingCards(Status.D),
            numMatchingCards(Status.E)
        };
    }

    public Card nextCard(){
        return deck.getCard();
    }

    public File getFolder(){
        return new File(SaveLoad.getProjectFolder().getAbsolutePath() + "/" +
name);
    }

    public Card getCurrentCard(){
        return deck.getCurrentCard();
    }

    public int getCurrentIndex(){
        return deck.getCurrentIndex();
    }

    @Override
            public String toString(){
                //like "History Test"
                return name; //this guy's toString is just his name
            }
    @Override
            public boolean equals(Object aProject){
                if(aProject == null) return false;
```

```
                    if(aProject instanceof Project == false)return false;
                    try{
                        Project project = (Project)aProject;
                        //if(project == null) return false;
                        return project.name.equals(this.name);//compare by name, i.e.
"History Test"
                    }
                    catch(Exception e){
                        return false;
                    }
                }

    /**
     * Compares the two projects based on name, case insensitive. "ABC" > "XYZ".
     * @param other
     * @return +ve if this project is bigger than other, -ve if it's smaller, 0
if they are equal (names are the same)
     */
    public int compareTo(Project other){
        if(this.equals(other)) return 0;
        String thisname = this.name.toLowerCase();
        String othername = other.name.toLowerCase();
        return thisname.compareTo(othername);
    }

    @Override
    public int hashCode() {
        int hash = 3;
        hash = 19 * hash + (this.name != null ? this.name.hashCode() : 0);
        return hash;
    }


    public static void createSampleProject(Controller controller){
        Project project = controller.addProject("Sample", true);
        ArrayList<Card> cards = new ArrayList<Card>();
        cards.add(new Card(
                "What is the ultimate answer to life, the universe, and everything?",
                "42"));
        project.addCards(cards);

        //return project;
        controller.refresh();
    }
}
```

Deck.java

```
public class Deck extends Object{

    private ArrayList<Card> cards; //flash cards of owner project
    private Card currentCard = null;
    private int currentIndex; //the index of the current card being viewed. Between 0 and length of cards

    public Deck(){
        cards = new ArrayList<Card>();
        currentIndex = 0;
    }

    public ArrayList<Card> getCards(){
        return cards;
    }

    public Card getCurrentCard(){
        return currentCard;
    }

    public void makeCurrentCardNull(){
        currentCard = null;
    }

    public int getCurrentIndex(){
        return currentIndex;
    }

    public int numCards(){
        return cards.size();
    }

    /**Returns the number of cards with the given status.
     *
     * @param status the status you want to look for
     * @return the number of cards with that status
     */
    public int numMatchingCards(Status status){
        int numSelected = 0;

        for(Card card : cards){
            if(card.getStatus() == status){
                numSelected++;
            }
        }

        return numSelected;
    }

    public void add(Card card){
        cards.add(card);
    }

    public void remove(Card card){
        cards.remove(card);
    }

    //actual meat of the class here

    /**
     * Shuffles the deck by randomizing the list of cards
     */
    public void shuffle(){
        ArrayList<Card> newCards = new ArrayList<Card>(); //cards will be moved
to here

        while(cards.isEmpty() == false){
            //keep going until there are no more cards
            int randomIndex = (int)(Math.random() * cards.size());
            Card randomCard = cards.get(randomIndex);
            //move it from old deck to new one
            cards.remove(randomCard);
            newCards.add(randomCard);
        }

        cards = newCards;
        currentIndex = 0; //now we'll draw from the top of the deck
    }


    /** Finds the next card in the deck and returns it.
     *
     * @return the next card
     */
    private Card nextCard(){
        Card card = null;
        try{
            card = cards.get(currentIndex);
        }
        catch(IndexOutOfBoundsException e){
            //tried to access a bad location, so shuffle and try again
            shuffle();
            return nextCard(); //return a new card
        }
        currentIndex++;
        if(currentIndex >= cards.size()){
            //we've run out of cards
            shuffle(); //for next time
        }
        return card;
    }

    public Card getCard(){

        if(numCards() == 0)
            return null; //no cards here

            currentCard = nextCard();

            return currentCard;
        //  }
      }

}
```

Controller.java

```
public final class Controller extends Object{
    //communicates with the GUI and object classes to get stuff done

    private GUI gui; //the GUI that is used here
    private PointManager pointManager;
    private ArrayList<Project> projects;
    private Project activeProject; //the project that you create cards for, study
from, etc.

    public static final double CHANCE_TO_GET_LUCKY = 0.05;

    public Controller(){
        /* the plan:
         * IF no existing projects:
         *      GET new project and make a project with it
         * ELSE: (existing projects)
         *      LOAD projects:
         *          LOAD cards and give them to project
         *          LOAD notes and give them to project
         * FINALLY:
         *      INITIALIZE GUI
         *      BUILD the GUI using the projects
         *      IF userData exists:
         *          LOAD it
         *      ELSE:
         *          CREATE a new one with defaults
         *      TELL the GUI to adapt to these changes
         *
         *
         */
        try{
        /** Is it the first time the program's being booted up?
         *
         */
        boolean firstRun = false;

        //if there isn't a cabraprojects file, create it
        if(SaveLoad.getProjectFolder().exists() == false){
            SaveLoad.getProjectFolder().mkdir();
            //unless the user deleted their data directory, this means this is
the first run
            firstRun = true;
        }

        UserData.load();
        //USER DATA IS NOW LOADED; do any init of prefs or such here

        //create point manager
        try{
            pointManager = new PointManager();
        }
        catch(NumberFormatException nfe){
            //if there's an exception like this, the wrong data was loaded into
the User Data
            //probably done by 0.6.0

            //alert user
            Utils.showDialog(null,
                    "Sorry! Your user data seems to have been corrupted and has been reset.",
                    "User data corrupted");

            //clear all data since something's corrupted
            UserData.makeAllDefault();

            //reload points
            pointManager = new PointManager();
        }

        //user data is set, so update font
        updatePreferredFont(
                UserData.getPref("FontName"),
                UserData.getInt("Prefs.FontSize")
                );

        //load projects
        ArrayList<Project> loadedProjects = loadProjectsFromFile(); //these projects are all stocked with cards/notes
        this.projects = loadedProjects;
        this.gui = new GUI(this,loadedProjects);

        //lack of projects matters now
        if(projects.isEmpty()){
            //no active project
            setNoActiveProject();
        }
        else{
            //there is an active project
            String projectName = UserData.getString("Project"); //the raw name of the project
            setActiveProject(projectName,false);
```

```
    }

    Themes theme = Themes.getThemeByName(UserData.getString("Theme"));
    setTheme(theme);

    gui.makeFrameVisible();

    //give first run info
    if(firstRun){
      //show advice
Utils.openURLinDialog("http://www.cabra.hathix.com/cabra/welcome.php",
          "Welcome to Cabra!",
          gui.getFrame(), true);

      //set user data's latest version to this
      UserData.setString("Version", About.VERSION);

      //add a default project
      //this.addProject(new Project("My First Project"), true);
    }

    //show changelog if this is a new version
    boolean upgrade = UserData.getString("Version").equals(About.VERSION) ==
false;
    if(!firstRun && upgrade){
      //not first run (then new version doesn't matter)   &&   old version
!= new version

      //store new version
      String version = About.VERSION;
      UserData.setString("Version", About.VERSION);

```

```
            //show changelog
            if(About.NIGHTLY){
                //don't bother with showing nightly changelog; docs are rarely
written for nightlies
              /*  Utils.showDialog(gui.getFrame(),
                        "<html><center>Thanks for testing Cabra " + version + "!
<br>As thanks, here's <b>100</b> points!",
                        "Thanks for upgrading to Cabra " + version + "!"
                        );  */
            }
            else{
               /* Utils.openURLinDialog("http://cabra.hathix.com/changelog/" +
Utils.sanitizeURL(version) + ".php",
                        "Thanks for upgrading to Cabra " + version + "!",
                        gui.getFrame(), true);  */
            }

            //earn points for upgrading
            if(About.PRERELEASE)
                gainPoints(Activity.USE_BETA);
            else
                gainPoints(Activity.USE_NEW_VERSION);
        }

        //if you're lucky, you earn some free points; also don't do it on first
run and overwhelm them w/dialogs
        if(!firstRun && !upgrade && Utils.pushLuck(CHANCE_TO_GET_LUCKY)){
            //earn points!
            int points = Activity.GET_LUCKY.getPoints();
            Utils.showDialog(gui.getFrame(),
                    "<html><center>"
                    + "I'm feeling generous, so here's <b>" +  points + "</b>
free points! Enjoy!",
                    "You got lucky!",
                    "goatgift.png");
            gainPoints(Activity.GET_LUCKY);
        }

        //is it time for an upgrade? see how long it's been since the last check
        long lastCheck = Long.parseLong(UserData.getString("LastUpdateCheck"));
        long rightNow = Calendar.getInstance().getTimeInMillis();
        long updateInterval =
Utils.daysToMillis(UserData.getIntPref("UpdateInterval"));
        if(rightNow - lastCheck >= updateInterval){
            //time to check for updates, it's been more than the chosen interval
since the last one
            Updates.checkForUpdates(gui);
        }

        gui.update();
        gui.refresh();
        }
        catch(Exception e){
            //some sort of exception threw off the whole thing
            Utils.debug(e, "Fatal error");
        }
    }

    public GUI getGUI(){
        return gui;
    }

    /**
     * Returns how many points the user has.
     * @return the amount of points the user has
     */
    public int getPoints(){
        return pointManager.getPoints();
    }

    /**
     * The user gains points by doing an activity.
     * @param activity the activity that the user did to gain these points.
     * @param refresh if the GUI should refresh.
     */
    public void gainPoints(PointEnums.Activity activity, boolean refresh){
        pointManager.gainPoints(activity);

        //show how many points were earned
        gui.showPointsBadge(activity.getPoints());

        if(refresh)
            gui.refresh();
    }

    /**
     * The user gains points. The GUI will refresh.
     * @param activity the activity that the user did to gain those points.
     */
    public void gainPoints(PointEnums.Activity activity){
        gainPoints(activity, true);
    }

    /**
     * Returns the vault manager used to control buying and display of prizes.
     * @param pointLabel the label that will be used to display the points the
```

```
user has. Should be pre-made and added to view.
     */
    public VaultManager createVaultManager(JLabel pointLabel){
        return new VaultManager(pointManager, gui, pointLabel);
    }

    /**
     * A wrapper around FontManager.updatePreferredFont() that works better.
     * Updates the PREFERRED_FONT to the given parameters. You should only pass
one. NOTE: you have to validate the frame after this
     * @param fontName the new font name/family. pass null if you don't want to
change it.
     * @param fontSize the new size of the font. pass 0 if you don't want to
change it.
     */
    public void updatePreferredFont(String fontName, int fontSize){
        FontManager.updatePreferredFont(fontName, fontSize);

        //validate frame so the changes take effect
        if(gui != null)
            gui.update();
    }

    /** Finds all the projects that the user has and returns them
     *
     * @return the user's projects
     */
    private ArrayList<Project> loadProjectsFromFile(){
        //looks for existing project files and, if they're there, creates the
projects
        File mainProjectFolder = SaveLoad.getProjectFolder();
        if(!mainProjectFolder.exists()){
            //there is no projects folder, since you're a first-time user
```

```
                mainProjectFolder.mkdir();
                //we know there's nothing in the folder so let's leave
                return new ArrayList<Project>();
            }

            ArrayList<Project> loadedProjects = new ArrayList<Project>();

            for(File projectFolder : mainProjectFolder.listFiles()){
                //projectFolder is a folder that contains a project

                if(projectFolder.isFile())
                    continue; //that means it's probably UserData... but regardless,
don't mess with it

                Project project = new Project(projectFolder.getName());
                loadedProjects.add(project);

                //give it some cards
                SaveLoad.loadCardsFromProject(project);

                //do this regardless of the user's having cards in the project
                //and now give notes to the project
                SaveLoad.loadNotesFromProject(project);
            } //end foreach

            return loadedProjects;
        }


        /** Adds a note to the active project, and while doing that creates the note
panel
         *
```

```
     * @param note the note to add to the active project
     * @param tabPane the note tab pane that invokes this method
     * @return the created note panel
     */

    public NotePanel addNoteToActiveProject(NoteTabPane tabPane,Note note){
        activeProject.addNote(note);

        //save while we're at it
        activeProject.saveNotes(); //no need to save cards too

        return new NotePanel(tabPane,gui,this,note);
    }


    public void setTheme(Themes theme){
        Themes.setTheme(theme); //that'll do it all for us

        //change and save user data
        UserData.setString("Theme",theme.getName());


        //update the look
        refresh();
        gui.repaint();
    }

    public Project getActiveProject(){

        return activeProject;
    }

    public ArrayList<Project> getAllProjects(){
        return projects;
    }

    public int getNumberOfProjects(){
        return projects.size();
    }

    /** Same as refresh() except it happens in this thread.
     *
     */
    public void refreshNow(){
        gui.refresh();
    }

    public void refresh(){
        //called when the active project is changed or has its cards manipulated
        //helps disable/enable buttons
        //Runnable r = new Runnable(){
        //    public synchronized void run(){
                refreshNow();
        //    }
        //};
        //javax.swing.SwingUtilities.invokeLater(r);
        //new Thread(r).start();
        //gui.refresh();
    }


    public void refreshHomePage(){
        Runnable r = new Runnable(){
            public void run(){
                gui.refreshHomePage();
            }
        };
        //javax.swing.SwingUtilities.invokeLater(r);
        new Thread(r).start();
    }

    /** Differs from setActive project in that that just changes active project,
this handles
     *  user interaction
     * @param projectName the name of the project you wish to be made active
     * @param shouldSave whether or not user data should be saved
     */
    public void setActiveProject(Project project, boolean shouldSave){
        //if nothing has matched, there's a problem
        if(project == null){
            //set the first project as active, then call this method again
            setActiveProject(projects.get(0),false);
            return;
        }

        if(shouldSave){
            //quick! Save the old notes if they weren't saved yet
            if(activeProject != null){
                try{
                    gui.saveAllNotes(); //that should do it
                }
                catch(NullPointerException n){
                    //error with saving project
                    System.out.println("Error saving project notes!");
                }
            }
        }

        activeProject = project;
        //alert all
        gui.newActiveProject(project);

        //set user data
        UserData.setString("Project",project.getName());

        //load the notes for the project
       // activeProject.loadNotes();

        //fix the home panel, which shows nothing unless this is done
        refresh();
    }

    /** Differs from setActive project in that that just changes active project,
this handles
     *  user interaction
     * @param projectName the name of the project you wish to be made active
     * @param shouldSave whether or not user data should be saved
     */

    public void setActiveProject(String projectName, boolean shouldSave){
        //find the matching project
        Project project = null;
        for(Project proj : projects){
            //System.out.println(proj.getName());
            if(proj.getName().equals(projectName))
                project = proj;
        }

        setActiveProject(project, shouldSave);
    }

    /**Sets nothing as the active project.
     *
     */
    public void setNoActiveProject(){
        UserData.makeDefault("Project");
        activeProject = null;
        gui.setFrameTitleByProject(null);
        //only refresh the tab pane (that removes all the panels and shows a new
one)
        refresh();
    }

    /** Creates a new project and adds it
     *
     * @param projectName the name of the project you want made
     * @param shouldSave true if the userData should be saved, false otherwise
     * @return the created project
     */

    public Project addProject(String projectName, boolean shouldSave){
        Project project = new Project(projectName);

        File projectFolder = new
File(SaveLoad.getProjectFolder().getAbsolutePath() + "/" + project.getName());
//puts the new folder in the projects folder
            projectFolder.mkdir();

        //really all this method does is make a project and tell addProject to do
its stuff using the project
```

```
        return addProject(project,shouldSave);
    }

    private Project addProject(Project project, boolean shouldSave){
        projects.add(project);
        gui.addProject(project);

        //add project data
            //create the card file
            //File cardFile = new File(projectFolder.getPath() + "/cards.txt");

        project.saveCards(); //forces the creation of cards.txt

        //make this project active
        setActiveProject(project, shouldSave);

        //sort project list
        Collections.sort(projects);

        return project;
    }

    /**
     * Renames the given project so it has the given name.
     * @param project the project
     * @param newName the project's new name.
     */
    public void renameProject(Project project, String newName){
        project.setName(newName);
        //re-sort projects; the name change may have put project out of order
        Collections.sort(projects);
    }


    /** Creates a project assuming you have all the files (i.e. you've just imported it.)
     *
     * @param projectName the name of the project
     * @param projectFolder the path to the project's folder (inside the Project Folder)
     */
    public void createProjectFromExistingFile(String projectName,File projectFolder){
        Project project = new Project(projectName);

        //add cards
        SaveLoad.loadCardsFromProject(project);

        //add notes
        SaveLoad.loadNotesFromProject(project);

        addProject(project,true);
    }

    public void removeProject(Project project){
        //removes the project at the given index
        //by the time we get here, we know something will be deleted

        //first delete the project file

        //Project project = projects.get(projectIndexInList); //get the project slated for deletion
        int projectIndexInList = projects.indexOf(project); //location of the project in the list
        File projectFile = new File(SaveLoad.getProjectFolder() + "/"+project.getName());
        //before we delete the directory we need to delete files inside
        for(File file : projectFile.listFiles()){
            file.delete();
        }
        //now delete the directory
        projectFile.delete();

        //what if the project to be removed was the active one?
        if(project.equals(activeProject)){
            //set the previous active project
            if(projects.size() == 1){
                //the last project was deleted, so nothing's left
                setNoActiveProject(); //takes care of making activeproject = null
            }
            else{
                //there's still a project left
                if(projectIndexInList == 0){
                    setActiveProject(projects.get(1),true);
                }
                else{
                    setActiveProject(projects.get(projectIndexInList-
1),true);
                }
                refresh();
            }
        }

        //now remove the project from list
        projects.remove(projectIndexInList);

        //update project list panel
    }

    public void addCardToActiveProject(Card card){
        //add the card to the project... it'll save itself

        if(activeProject != null){
            activeProject.addCard(card);

            //gain points
            gainPoints(Activity.CREATE_CARD);
            if(card.hasPicture()){
                gainPoints(Activity.ADD_IMAGE);
            }

            refresh();
        }
        //if there's no active project, take no action
    }


}
```

Card.java

```
public class Card extends Object{
    //a simple quiz card with a question and answer

    private String questionText;
    private String answerText;
    private String pictureName; //this card might have a picture
    private Status status; //rank
    private int sessionsLeft; //sessions until next study

    public static final String NO_PICTURE_STRING = " "; //not really empty, just
```

```
represents no picture

    public static final String DELIMITER = "//"; //separates fields
    public static final String NEWLINE_REPLACER = "-nl-"; //\n's are replaced
with this string during saving and loading
    public static final String NEWLINE = "\n"; //signifies a new line in card
text



    //the ultimate one
    //the rest cascade under this, overloading to the one above it
    public Card(Status status, int sessionsLeft, String question, String answer,
String pictureName){
        setStatus(status);
        this.sessionsLeft = sessionsLeft;

        this.questionText = question;
        this.answerText = answer;
        this.pictureName = pictureName;
    }

    public Card(Status status,String question,String answer,String pictureName){
        this(status,status.getReps(),question,answer,pictureName);
    }

    public Card(String question,String answer,String pictureName){
        this(Status.DEFAULT_STATUS,question,answer,pictureName);
    }

    public Card(String question,String answer){
        this(question,answer,NO_PICTURE_STRING);
    }


    public void trimPictureFile(){
        //right now we have to full path to the picture... get just the name
"foo.png"
        File picture = new File(pictureName);
        this.pictureName = picture.getName();
    }

    /** Tells if this card has a picture
     *
     * @return true if it has a picture, false otherwise
     */
    public boolean hasPicture(){
        return !pictureName.equals(NO_PICTURE_STRING);
    }


    /** Status stuff */
    public void setStatus(Status status){
        //System.out.println("Setting status: " + status.toString());
        this.status = status;
        this.sessionsLeft = status.getReps();
        //System.out.println(status.toString() + sessionsLeft);
    }

    public Status getStatus(){
        return this.status;
    }

    public int sessionsLeft(){
        return sessionsLeft;
    }


    public boolean isDueForStudying(){
        return sessionsLeft <= 0;
    }

    /** This card is studied.
     *
     * @param result Choices.YES if it was known, Choices.NO if it wasn't
     */
    public void study(KnowPanel.Choices result){
        switch(result){
            case YES:
                //send rank up
                setStatus(status.nextRank());
                break;
            case NO:
                //send it back to bottom
                setStatus(Status.A);
                break;
            case SORT_OF:
                //send rank down 1
                setStatus(status.previousRank());
                break;
            case SKIPPED:
                break;
        }
    }

    /** This card isn't studied this round
     *
     */
    public void skip(){
        if(sessionsLeft > 0)
            sessionsLeft--;
    }

    public String getQuestion(){
        //replace newlines
        String text = bringBackNewlines(questionText);
        //return question
        return text;
    }
    public String getAnswer(){
        return bringBackNewlines(answerText); //replace the newline replacer with
the actual \n character
    }

    public void setQuestion(String text){
        if(text != null && text.equals("")==false)
            this.questionText = text;
    }

    public void setAnswer(String text){
        if(text != null && text.equals("")==false)
            this.answerText = text;
    }

    public String getPictureName(){
        return pictureName;
    }

    public File getPictureFile(){
        return new File(pictureName);
    }

    /** Changes the name (path) of the picture.
     *
     * @param name the path (absolute or relative) of the picture file
     */
    public void setPictureName(String name){
       this.pictureName = name;
    }

    /** Removes the picture from this card.
     *
     */
    public void removePicture(){
        setPictureName(Card.NO_PICTURE_STRING);
    }

    /**Replaces the newline replacers in the given text with real newlines
     *
     * @param string the string to be fixed
     * @return the fixed string
     */

    public static String bringBackNewlines(String string){
        //escape all characters
        //String literal = Matcher.quoteReplacement(string);
        String fixed = string.replaceAll(Card.NEWLINE_REPLACER, Card.NEWLINE);
        return fixed;
    }

    /** Replaces the newlines in a string of text with the newline replacer.
     *
     * @param string the string to be messed with
     * @return the new string
     */

    public static String replaceNewlines(String string){
            //get a literal interpretation of the string
            //String literal = Matcher.quoteReplacement(string);
            //now replace stuff
            string = string.replaceAll(Card.NEWLINE, Card.NEWLINE_REPLACER);
            return string;
    }

    /** Creates a card based on the raw data string passed.
     *
     * @param text the raw text
     * @return the created card
     */
    public static Card createCardBasedOnText(String text){
        try{
            //create a card for each line here
            String[] stuff = text.split(Card.DELIMITER);
            //first string is status, next string is question, then answer, then
image

            //cards made in older versions need to be slightly adapted
            String fixedFirst = Status.importFromPast(stuff[0]);

            Card card = new Card(
            Status.getStatus(fixedFirst.substring(0,1)),
            Integer.parseInt(fixedFirst.substring(1,fixedFirst.length())), //grab
however many digits
            stuff[1],
            stuff[2],
            stuff[3]);

            return card;
        }
        catch(Exception e){
            //a malformed line, maybe?
            System.out.println("Malformed card! Details:" + e);
            return null;
        }
    }

    @Override
    public String toString(){
        //used during saving of this card
        String text = status.toString() + sessionsLeft + Card.DELIMITER
                    + questionText + Card.DELIMITER
                    + answerText + Card.DELIMITER
                    + pictureName;
            //replace newlines with the newline replacer
            text = replaceNewlines(text);
            //now that it's cleaned up return it
            return text;
    }

    @Override
            public boolean equals(Object aCard){
                if(aCard == null) return false;
                if(aCard instanceof Card == false)return false; //if it's not a
```

```
card, stop it
                    Card card = (Card)aCard;
                    if(card == null) return false;
                    return card.answerText.equals(this.answerText) &&
card.questionText.equals(this.questionText)
                            && card.pictureName.equals(this.pictureName)
                            && card.status == this.status;
                    //everything must match

            }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 43 * hash + (this.questionText != null ?
this.questionText.hashCode() : 0);
        hash = 43 * hash + (this.answerText != null ? this.answerText.hashCode()
: 0);
        hash = 43 * hash + (this.pictureName != null ?
this.pictureName.hashCode() : 0);
        hash = 43 * hash + (this.status != null ? this.status.hashCode() : 0);
        return hash;
    }
}
```

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