

```
1 package edu.seaaddicts.brockbutler.coursemanager;
2
3 import java.util.ArrayList;
4
5 import android.app.Activity;
6 import android.os.Bundle;
7 import android.util.Log;
8 import android.util.SparseBooleanArray;
9 import android.view.View;
10 import android.view.View.OnClickListener;
11 import android.widget.AdapterView;
12 import android.widget.AdapterView.OnItemClickListener;
13 import android.widget.ArrayAdapter;
14 import android.widget.Button;
15 import android.widget.LinearLayout;
16 import android.widget.ListView;
17 import android.widget.Spinner;
18 import edu.seaaddicts.brockbutler.R;
19
20 public class AddCourseActivity extends Activity {
21
22     private static final String TAG = "AddCourseActivity";
23
24     //private static final int DATE_DIALOG_ID = 0;
25     private static final int VISIBLE = 0;
26     //private static final int INVISIBLE = 4;
27     private static final int GONE = 8;
28
29     ArrayList<String> mLecs;
30     ArrayList<String> mLabs;
31     ArrayList<String> mTuts;
32     ArrayList<String> mSems;
33
34     ArrayList<Offering> mLecsOfferings;
35     ArrayList<Offering> mLabsOfferings;
36     ArrayList<Offering> mTutsOfferings;
37     ArrayList<Offering> mSemsOfferings;
38
39     private String mSubject;
40     private String mCode;
41
42     private Course mCourse;
43
44     private Button mSaveButton;
45     private Button mCancelButton;
46
47     private CourseHandler mCourseHandle;
48     ArrayList<Offering> mOfferings;
49
50     private Spinner mSubjectSpinner;
51     private Spinner mCodesSpinner;
52
53     private ListView mLecsListView;
54     private ListView mSemsListView;
55     private ListView mTutsListView;
56     private ListView mLabsListView;
57
58     @Override
59     protected void onCreate(Bundle savedInstanceState) {
60         super.onCreate(savedInstanceState);
61         setContentView(R.layout.activity_add_course);
62         mCourseHandle = new CourseHandler(this.getApplicationContext());
63         init();
64     }
65
66     /*
67     * Initialize all views and sets Button OnClickListener.
68     */
69     private void init() {
70
71         mSaveButton = (Button) findViewById(R.id.add_course_save_button);
```

```

72     mCancelButton = (Button) findViewById(R.id.add_course_cancel_button);
73
74     mSubjectSpinner = (Spinner) findViewById(R.id.add_course_subjects_spinner);
75     try {
76         mSubjectSpinner.setAdapter(new ArrayAdapter<String>(this,
77             android.R.layout.simple_spinner_dropdown_item,
78             mCourseHandle.getSubjects()));
79     } catch (Exception e) {
80         e.printStackTrace();
81     }
82
83     mCodesSpinner = (Spinner) findViewById(R.id.add_course_codes_spinner);
84     mSubjectSpinner.setOnItemSelectedListener(new OnItemSelectedListener() {
85         public void onItemSelected(AdapterView<?> arg0, View arg1,
86             int arg2, long arg3) {
87             try {
88                 mSubject = arg0.getItemAtPosition(arg2).toString();
89                 mCodesSpinner.setAdapter(new ArrayAdapter<String>(
90                     AddCourseActivity.this,
91                     android.R.layout.simple_spinner_dropdown_item,
92                     mCourseHandle.getCodes(mSubject)));
93             } catch (Exception e) {
94                 e.printStackTrace();
95             }
96         }
97
98         public void onNothingSelected(AdapterView<?> arg0) {
99             // Do nothing.
100         }
101     });
102     mCodesSpinner.setOnItemSelectedListener(new OnItemSelectedListener() {
103
104         public void onItemSelected(AdapterView<?> arg0, View arg1,
105             int arg2, long arg3) {
106             mCode = arg0.getItemAtPosition(arg2).toString();
107             mCourse = mCourseHandle.getCourseOfferings(mSubject, mCode);
108             mOfferings = mCourse.mOfferings;
109
110             String s;
111             mLecs = new ArrayList<String>();
112             mLabs = new ArrayList<String>();
113             mTuts = new ArrayList<String>();
114             mSems = new ArrayList<String>();
115
116             mLecsOfferings = new ArrayList<Offering>();
117             mLabsOfferings = new ArrayList<Offering>();
118             mTutsOfferings = new ArrayList<Offering>();
119             mSemsOfferings = new ArrayList<Offering>();
120
121             for (int i = 0; i < mOfferings.size(); i++) {
122                 s = mOfferings.get(i).mType;
123
124                 // Some offerings don't have any of what we are looking for,
125                 // so check length to make sure.
126                 if (s.length() > 2) {
127                     String ss = s.substring(0, 3).trim();
128                     if (ss.equalsIgnoreCase("lec")) {
129                         mLecs.add(s + ", SEC " + mOfferings.get(i).mSection);
130                         mLecsOfferings.add(mOfferings.get(i));
131                     } else if (ss.equalsIgnoreCase("lab")) {
132                         mLabs.add(s + ", SEC " + mOfferings.get(i).mSection);
133                         mLabsOfferings.add(mOfferings.get(i));
134                     } else if (ss.equalsIgnoreCase("tut")) {
135                         mTuts.add(s + ", SEC " + mOfferings.get(i).mSection);
136                         mTutsOfferings.add(mOfferings.get(i));
137                     } else if (ss.equalsIgnoreCase("sem")) {
138                         mSems.add(s + ", SEC " + mOfferings.get(i).mSection);
139                         mSemsOfferings.add(mOfferings.get(i));
140                     }
141                 }
142             }

```

```

143
144     // Check if offerings available, and add ListView to display if
145     // so.
146     LinearLayout lec_layout = (LinearLayout) findViewById(R.id.layout_add_lecs);
147     LinearLayout lab_layout = (LinearLayout) findViewById(R.id.layout_add_labs);
148     LinearLayout tut_layout = (LinearLayout) findViewById(R.id.layout_add_tuts);
149     LinearLayout sem_layout = (LinearLayout) findViewById(R.id.layout_add_sems);
150
151     if (mLecs.size() > 0) {
152         lec_layout.setVisibility(VISIBLE);
153         mLecsListView = (ListView) findViewById(R.id.add_course_add_lecs);
154         mLecsListView.setAdapter(new ArrayAdapter<String>(
155             getApplicationContext(),
156             android.R.layout.simple_list_item_multiple_choice,
157             mLecs));
158     } else {
159         lec_layout.setVisibility(GONE);
160     }
161     if (mLabs.size() > 0) {
162         lab_layout.setVisibility(VISIBLE);
163         mLabsListView = (ListView) findViewById(R.id.add_course_add_labs);
164         mLabsListView.setAdapter(new ArrayAdapter<String>(
165             getApplicationContext(),
166             android.R.layout.simple_list_item_multiple_choice,
167             mLabs));
168     } else {
169         lab_layout.setVisibility(GONE);
170     }
171     if (mTuts.size() > 0) {
172         tut_layout.setVisibility(VISIBLE);
173         mTutsListView = (ListView) findViewById(R.id.add_course_add_tuts);
174         mTutsListView.setAdapter(new ArrayAdapter<String>(
175             getApplicationContext(),
176             android.R.layout.simple_list_item_multiple_choice,
177             mTuts));
178     } else {
179         tut_layout.setVisibility(GONE);
180     }
181     if (mSemsOfferings.size() > 0) {
182         sem_layout.setVisibility(VISIBLE);
183         mSemsListView = (ListView) findViewById(R.id.add_course_add_sems);
184         mSemsListView.setAdapter(new ArrayAdapter<String>(
185             getApplicationContext(),
186             android.R.layout.simple_list_item_multiple_choice,
187             mSems));
188     } else {
189         sem_layout.setVisibility(GONE);
190     }
191 }
192
193 public void onNothingSelected(AdapterView<?> arg0) {
194 }
195 });
196
197 mSaveButton.setOnClickListener(new OnClickListener() {
198     public void onClick(View v) {
199         Course c = new Course();
200         c.mSubject = mSubject;
201         c.mCode = mCode;
202         c.mInstructor = mCourseHandle.getCourseOfferings(mSubject,
203             mCode).mInstructor;
204         try {
205             c.mInstructor_email = ""+c.mInstructor.charAt(c.mInstructor.length()-1);
206             c.mInstructor_email += c.mInstructor.substring(0,c.mInstructor.length()-2);
207             c.mInstructor_email += "@brocku.ca";
208         } catch (Exception e){}
209
210         SparseBooleanArray sbal, sba2, sba3, sba4;
211         if (mLecsListView != null) {
212             sbal = mLecsListView.getCheckedItemPositions();
213             for (int i = 0; i < mLecsListView.getCount(); i++) {

```

```

214         if (sbal.get(i) == true) {
215             c.mOfferings.add(mLecsOfferings.get(i));
216             Log.d(TAG, "Added: " + mLecsOfferings.get(i).mSubj
217                 + " " + mOfferings.get(i).mCode + ", Type "
218                 + mOfferings.get(i).mType + ", Section "
219                 + mOfferings.get(i).mSection
220                 + " to Offerings");
221         }
222     }
223 }
224
225 if (mLabsListView != null) {
226     sba2 = mLabsListView.getCheckedItemPositions();
227
228     for (int i = 0; i < mLabsListView.getCount(); i++) {
229         if (sba2.get(i) == true) {
230             c.mOfferings.add(mLabsOfferings.get(i));
231             Log.d(TAG, "Added: " + mLabsOfferings.get(i).mSubj
232                 + " " + mOfferings.get(i).mCode + ", Type "
233                 + mOfferings.get(i).mType + ", Section "
234                 + mOfferings.get(i).mSection
235                 + " to Offerings");
236         }
237     }
238 }
239
240 if (mTutsListView != null) {
241     sba3 = mTutsListView.getCheckedItemPositions();
242
243     for (int i = 0; i < mTutsListView.getCount(); i++) {
244         if (sba3.get(i) == true) {
245             c.mOfferings.add(mTutsOfferings.get(i));
246             Log.d(TAG, "Added: " + mTutsOfferings.get(i).mSubj
247                 + " " + mOfferings.get(i).mCode + ", Type "
248                 + mOfferings.get(i).mType + ", Section "
249                 + mOfferings.get(i).mSection
250                 + " to Offerings");
251         }
252     }
253 }
254
255 if (mSemsListView != null) {
256     sba4 = mSemsListView.getCheckedItemPositions();
257
258     for (int i = 0; i < mSemsListView.getCount(); i++) {
259         if (sba4.get(i) == true) {
260             c.mOfferings.add(mSemsOfferings.get(i));
261             Log.d(TAG, "Added: " + mSemsOfferings.get(i).mSubj
262                 + " " + mOfferings.get(i).mCode + ", Type "
263                 + mOfferings.get(i).mType + ", Section "
264                 + mOfferings.get(i).mSection
265                 + " to Offerings");
266         }
267     }
268 }
269
270 if (c.mSubject != null) {
271     try {
272         mCourseHandle.addCourse(c);
273         Log.d("# ADDED OFFERINGS:", "" + c.mOfferings.size());
274         for (int i = 0; i < c.mOfferings.size(); i++) {
275             Log.d(TAG, "Added: " + c.mOfferings.get(i).mSubj
276                 + " " + mOfferings.get(i).mCode + ", Type "
277                 + mOfferings.get(i).mType + ", Section "
278                 + mOfferings.get(i).mSection
279                 + " to Offerings");
280         }
281         onBackPressed();
282     } catch (Exception e) {
283         e.printStackTrace();
284     }

```

```
285         }
286     }
287 }));
288
289 mCancelButton.setOnClickListener(new OnClickListener() {
290     public void onClick(View v) {
291         onBackPressed();
292     }
293 });
294 }
295 }
296
```

```

1  /**
2   * Brocku.java
3   * Brock Butler
4   * Connects to the Brock University registrars' website to obtain course
5   * information from the current timetable
6   * Created by James Grisdale on 2013-02-24
7   * * Copyright (c) 2013 Sea Addicts. All rights reserved.
8   */
9
10 package edu.seaaddicts.brockbutler.coursemanager;
11
12 import java.io.BufferedReader;
13 import java.io.InputStreamReader;
14 import java.net.URI;
15 import java.util.ArrayList;
16 import org.apache.http.client.HttpClient;
17 import org.apache.http.client.methods.HttpGet;
18 import org.apache.http.impl.client.*;
19 import org.apache.http.HttpResponse;
20 import android.os.AsyncTask;
21 //Using AsyncTask to do operations off the main thread
22 public class Brocku extends AsyncTask<Void, Void, ArrayList<MasterCourse>> {
23
24     /* doInBackground - connects to Brock University's registrar's office website to
25     * information on courses being offered, then returns an arraylist of MasterCourse
26     * objects which hold the data for all offerings at Brock.
27     */
28     protected ArrayList<MasterCourse> doInBackground(Void... Void) {
29         String codes[] = new String[74];
30         BufferedReader in = null;
31         String info = new String();
32         String substring = new String();
33         MasterCourse course;
34         boolean done;
35         ArrayList<MasterCourse> courseList = new ArrayList<MasterCourse>();
36         courseList.ensureCapacity(8000);
37         //test = "working";
38         try{
39             HttpClient client = new DefaultHttpClient();
40             HttpGet request = new HttpGet();
41             URI BTimeTable = new URI(
42                 "http://www.brocku.ca/registrar/guides/returning/timetable/a_get_subj.php?subj=C
43                 OSC");
44             request.setURI(BTimeTable);
45             HttpResponse response = client.execute(request);
46             in = new BufferedReader(new InputStreamReader(response.getEntity().getContent
47                 ()));
48             for (int i=0; i<3; i++) in.readLine();
49             for (int i=0; i<74; i++){
50                 info = in.readLine();
51                 codes[i] = info.substring(19,23);
52             }//retrieving all subjects
53             in.close();
54             for (int h=0; h<codes.length ; h++){
55                 done = false;
56                 BTimeTable = new URI(
57                     "http://www.brocku.ca/registrar/guides/returning/timetable/a_get_subj.php?subj
58                     =" + codes[h]);
59                 request.setURI(BTimeTable);
60                 response = client.execute(request);
61                 in = new BufferedReader(new InputStreamReader(response.getEntity().getContent
62                     ()));
63                 for (int i=0; i<98; i++) {in.readLine();}
64                 info = in.readLine();
65                 //for each subjects get all course offering information
66                 if (info.length()<50){
67                     while(!done){
68                         course = new MasterCourse();
69                         substring = info.substring(24, info.length() - 5);
70                         course.id = substring;

```

```

65         for (int i=0; i<2; i++) {in.readLine();}
66         info = in.readLine();
67         substring = info.substring(123, 127);
68         course.subj = substring;
69         substring = info.substring(128, 132);
70         course.code = substring;
71         in.readLine();
72         info = in.readLine();
73         substring = info.substring(26,info.length() - 7);
74         course.desc = substring;
75         for (int i=0; i<4; i++) {in.readLine();}
76         info = in.readLine();
77         substring = info.substring(24,26);
78         course.dur = substring;
79         info = in.readLine();
80         substring = info.substring(24,info.length() - 5);
81         course.type = substring;
82         info = in.readLine();
83         substring = info.substring(24,info.length() - 5);
84         course.sec = substring;
85         info = in.readLine();
86         substring = info.substring(24,30);
87         course.days = substring;
88         info = in.readLine();
89         substring = info.substring(24,info.length() - 5);
90         course.time = substring;
91         info = in.readLine();
92         substring = info.substring(24,info.length() - 5);
93         if (info.substring(24,26).equals("<a"))
94             substring = info.substring(94, info.length()-9);
95         course.location = substring;
96         info = in.readLine();
97         if (info.length()>16)
98             substring = info.substring(9,info.length() - 5);
99         else substring = " ";
100        course.instructor = substring;
101        in.readLine();
102        in.readLine();
103        info = in.readLine();
104        courseList.add(course);
105        if (info.length() <20){
106            info = in.readLine();
107            done = true;
108        }
109    }
110 }
111 in.close();
112 }
113 }
114 //if there's an error return the error information in a course object
115 catch (Exception e){
116     info = e.toString();
117     course = new MasterCourse();
118     course.id = info;
119     courseList.add(course);
120 }
121 return courseList;
122 }
123
124 //not used
125 protected void onPostExecute(MasterCourse course){
126     posttest(course);
127 }
128
129 //not used
130 public MasterCourse posttest(MasterCourse course){
131     return course;
132 }
133 }
134

```

```
1  /**
2   * Course.java
3   * Brock Butler
4   * A wrapper class for Course information
5   * Created by James Grisdale on 2013-02-24
6   * Copyright (c) 2013 Sea Addicts. All rights reserved.
7   **/
8
9  package edu.seaaddicts.brockbutler.coursemanager;
10
11  import java.util.ArrayList;
12
13  import edu.seaaddicts.brockbutler.contacts.Contact;
14  import edu.seaaddicts.brockbutler.scheduler.Task;
15
16  public class Course {
17      public int mId; //course ID
18      public String mSubject; //subject name
19      public String mCode; //course code
20      public String mDesc; //course description
21      public String mInstructor; //instructor name
22      public String mInstructor_email; //instructor's email
23      public ArrayList<Offering> mOfferings; //list of offerings for this course
24      public ArrayList<Task> mTasks; //list of tasks associated with this course
25      public ArrayList<Contact> mContacts; //contacts for this course
26
27      //Constructor - initializes the arraylists for offerings, tasks and contacts
28      public Course() {
29          mOfferings = new ArrayList<Offering>();
30          mTasks = new ArrayList<Task>();
31          mContacts = new ArrayList<Contact>();
32      }
33  }
```



```
1  /**
2   * CourseHandler.java
3   * Brock Butler
4   * A class to allow easy access to database functions
5   * Created by James Grisdale on 2013-02-24
6   * Copyright (c) 2013 Sea Addicts. All rights reserved.
7   */
8
9  package edu.seaaddicts.brockbutler.coursemanager;
10 import java.util.ArrayList;
11
12 import android.content.Context;
13 import android.database.Cursor;
14 import edu.seaaddicts.brockbutler.contacts.Contact;
15 import edu.seaaddicts.brockbutler.scheduler.Task;
16
17 public class CourseHandler {
18     // Context context;
19     CurrentCoursesHandler CH;
20     CourseListHandler courseList;
21
22     /* Constructor - opens and closes database to ensure the database exists
23      * and if not, it copies over the installed database of course offerings
24      * @param context - application context
25      */
26     public CourseHandler(Context context) {
27         // this.context = context;
28         CH = new CurrentCoursesHandler(context);
29         courseList = new CourseListHandler(context);
30         try{
31             courseList.createDataBase();
32             courseList.openDataBase();
33             courseList.close();
34         }
35         catch(Exception e){};
36         //SQLiteDatabase db = courseList.getWritableDatabase();
37         //db.close();
38     }
39
40     /* updateAll - updates all course information from the Brock University registrar's
41      * office website
42      */
43     public void updateAll(){
44         courseList.addCourse();
45     }
46
47     /* Depreciated - getAllCourses - grabs course data from the registrar's timetable
48     and
49     * inserts data into the masterlist table.
50     * Depreciated due to information no longer being available on website
51     */
52     public void getAllCourses() {
53         courseList.addCourse();
54     }
55
56     /* getCourse - gets all information for a given course subject and code
57      * @param subj - subject name to get
58      * @param code - course code to get
59      */
60     public Course getCourse(final String subj, final String code) {
61         return CH.getCourse(subj, code);
62     }
63
64     /* updateCourse - updates all the information for a given course
65      * @param course - course information to update
66      */
67     public void updateCourse(Course course) {
68         CH.addCourse(course);
69     }
70
71     /* getSubjects - gets a list of subjects available from the master list
```

```

71     * returns an arraylist of subject offerings
72     */
73     public ArrayList<String> getSubjects() throws Exception {
74         return courseList.getSubjects();
75     }
76
77     /* getCodes - gets a list of codes for a given subject from the master list
78     * returns an arraylist of subject codes
79     * @param subj - return codes for this subject
80     */
81     public ArrayList<String> getCodes(String subj) {
82         return courseList.getCodes(subj);
83     }
84
85     /* getCourseOfferings - returns all offerings offered for a given course
86     * Converts the offerings from MasterCourse format to Course format
87     * @param subj - subject name
88     * @param code - course code
89     */
90     public Course getCourseOfferings(String subj, String code) {
91         //get list of offerings as a list of MasterCourse objects
92         ArrayList<MasterCourse> list = courseList.getCourses(subj, code);
93         Course course = new Course(); //create a new course object
94         ArrayList<OfferingTime> offeringtimes;
95         course.mSubject = list.get(0).subj;
96         course.mCode = list.get(0).code;
97         course.mInstructor = list.get(0).instructor;
98         course.mDesc = list.get(0).desc;
99         Offering offering;
100        int tindex = 0;
101        OfferingTime otime;
102        //add all offerings for a particular course
103        ArrayList<Offering> offerings = new ArrayList<Offering>();
104        for (int i = 0; i < list.size(); i++) {
105            offering = new Offering();
106            offering.mSubj = list.get(i).subj;
107            offering.mCode = list.get(i).code;
108            offering.mType = list.get(i).type;
109            offering.mSection = Integer.parseInt(list.get(i).sec);
110            //add all the offeringtimes associated with all the offerings
111            offeringtimes = new ArrayList<OfferingTime>();
112            for (int j = 0; j < 5; j++) {
113                if (list.get(i).days.charAt(j) != ' ') {
114                    otime = new OfferingTime();
115                    otime.mDay = list.get(i).days.substring(j, j+1);
116                    otime.mLocation = list.get(i).location;
117                    for (int h = 0; h < list.get(i).time.length(); h++) {
118                        if (list.get(i).time.charAt(h) == '-') {
119                            tindex = h;
120                            break;
121                        }
122                    }
123                    //get the times for each offering
124                    otime.mStartTime = list.get(i).time.substring(0, tindex);
125                    otime.mEndTime = list.get(i).time.substring(tindex + 1,
126                        list.get(i).time.length());
127                    offeringtimes.add(otime);
128                }
129            }
130            offering.mOfferingTimes = offeringtimes;
131            offerings.add(offering);
132        }
133        course.mOfferings = offerings;
134
135        return course; //return the course object
136    }
137
138    /* addCourse - adds information for a course into the database
139    * returns a 0 if sucessful, 1 if the add failed
140    * @param course - the course object to be added

```

```
142     */
143     public int addCourse(Course course) throws Exception {
144         try {
145             CH.addCourse(course);
146             return 0;
147         } catch (Exception e) {
148             return 1;
149         }
150     }
151
152     /* removeCourse - deletes all information from the database for a course
153     * returns a 0 on success, returns 1 if failure
154     * @param course - the course information to be deleted
155     */
156     public int removeCourse(Course course) {
157         try {
158             CH.deleteCourse(course);
159             return 0;
160         } catch (Exception e) {
161             return 1;
162         }
163     }
164
165     /*
166     * getRegisteredCourses - returns all information for all courses in the
167     * current courses database
168     */
169     public ArrayList<Course> getRegisteredCourses() {
170         return CH.getRegCourses();
171     }
172
173     /* getOfferings - get all offerings for a certain course
174     * @param subj - course name
175     * @param code - course code
176     */
177     public ArrayList<Offering> getOfferings(String subj, String code) {
178         return CH.getOfferings(subj, code);
179     }
180
181     /* getTasks - gets all tasks from the database
182     */
183     public ArrayList<Task> getTasks() {
184         return CH.getTasks();
185     }
186
187     /* addTask - adds a given task to the task table in the database
188     * returns 0 if sucessful, returns 1 if it fails
189     * @param task - the task information to be added to the database
190     */
191     public int addTask(Task task) {
192         try {
193             CH.addTasks(task);
194             return 0;
195         } catch (Exception e) {
196             return 1;
197         }
198     }
199
200     // addTask - adds the tasks for a given course to the task table in the
201     // database
202     public int addTask(Course course) {
203         try {
204             CH.addTasks(course);
205             return 0;
206         } catch (Exception e) {
207             return 1;
208         }
209     }
210
211     /* removeTask - deletes task information from the database for a given task
212     * returns 0 if sucessful, 1 if failure
```

```
213     * @param task - the task information to be removed from the database
214     */
215     public int removeTask(Task task) {
216         try {
217             CH.removeTask(task);
218             return 0;
219         } catch (Exception e) {
220             return 1;
221         }
222     }
223
224     /* getBase - returns the total base mark for a particular course
225     * given base information from the course
226     * @param course - the course to calculate the total base for
227     */
228     public float getBase(Course course) {
229         float base = 0;
230         for (int i = 0; i < course.mTasks.size(); i++)
231             base += course.mTasks.get(i).mWeight;
232         return base;
233     }
234
235     /* getMark - returns the calculated progress mark for a course
236     * given mark information from the course, the mark is calculated and returned
237     * as a float
238     * @param course - the course to calculate the marks for
239     */
240     public float getMark(Course course) {
241         float mark = 0;
242         for (int i = 0; i < course.mTasks.size(); i++){
243             if(course.mTasks.get(i).mBase !=0){
244                 mark += (course.mTasks.get(i).mMark / course.mTasks.get(i).mBase)
245                     * course.mTasks.get(i).mWeight;
246             } else mark+=0;
247         }
248
249         return mark;
250     }
251
252     /* getSize - returns the number of courses added to the course database
253     */
254     public int getSize() {
255         return courseList.size();
256     }
257
258     /* removeContact - removes contact information from the database
259     * @param contact - the contact information to be removed
260     */
261     public void removeContact(Contact contact){
262         CH.removeContact(contact);
263     }
264
265     /* Query - returns a cursor with results for a custom query
266     * @param query - a string with a sqlite query
267     */
268     public Cursor Query(String query) {
269         return CH.Query(query);
270     }
271 }
272
```

```

1  /**
2   * CourseListHandler.java
3   * Brock Butler
4   * Creates a database table for a full list of offerings on the registrar's
5   * timetable and allows the table to have inserts or be read
6   * Created by James Grisdale on 2013-02-24
7   * Copyright (c) 2013 Sea Addicts. All rights reserved.
8   */
9
10 package edu.seaaddicts.brockbutler.coursemanager;
11
12 import java.io.FileOutputStream;
13 import java.io.IOException;
14 import java.io.InputStream;
15 import java.io.OutputStream;
16 import java.util.ArrayList;
17
18 import android.content.ContentValues;
19 import android.content.Context;
20 import android.database.Cursor;
21 import android.database.SQLException;
22 import android.database.sqlite.SQLiteDatabase;
23 import android.database.sqlite.SQLiteException;
24 import android.database.sqlite.SQLiteOpenHelper;
25 import android.os.Looper;
26
27 public class CourseListHandler extends SQLiteOpenHelper {
28
29     // All Static variables
30     // Database Version
31     private static final int DATABASE_VERSION = 1;
32     private static final String DATABASE_NAME = "Database";
33     private static String DB_PATH =
34         "/data/data/edu.seaaddicts.brockbutler.cousemanager/databases";
35     private SQLiteDatabase myDataBase;
36     // Database Name
37
38     // Full course list table name
39     private static final String TABLE_MCOURSES = "MasterList";
40     //current courses table names
41     //private static final String TABLE_COURSES = "courses";
42     //private static final String TABLE_TASKS = "tasks";
43     //private static final String TABLE_OFFERINGS = "offerings";
44     //private static final String TABLE_OFFERING_TIMES = "offering_times";
45     //private static final String TABLE_CONTACTS = "contacts";
46     // All field names used in the database
47     private static final String KEY_SUBJ = "subj";
48     private static final String KEY_CODE = "code";
49     private static final String KEY_DESC = "desc";
50     private static final String KEY_INSTRUCTOR = "instructor";
51     private static final String KEY_ID = "id";
52     private static final String KEY_TYPE = "type";
53     private static final String KEY_SEC = "sec";
54     //private static final String KEY_DAY = "day";
55     //private static final String KEY_TIMES = "time_start";
56     //private static final String KEY_TIMEE = "time_end";
57     private static final String KEY_LOCATION = "location";
58     private static final String KEY_DUR = "dur";
59     //private static final String KEY_ASSIGN = "assign";
60     //private static final String KEY_NAME = "name";
61     //private static final String KEY_MARK = "mark";
62     //private static final String KEY_BASE = "base";
63     //private static final String KEY_WEIGHT = "weight";
64     //private static final String KEY_DUE = "due";
65     //private static final String KEY_CREATE_DATE = "create_date";
66     //private static final String KEY_CID = "cid";
67     //private static final String KEY_FNAME = "fname";
68     //private static final String KEY_LNAME = "lname";
69     //private static final String KEY_EMAIL = "email";
70     //private static final String KEY_PRIORITY = "priority";

```

```

71 //private static final String KEY_INSTREMAIL = "instructor_email";
72 private static final String KEY_DAYS = "days";
73 private static final String KEY_TIME = "time";
74 Context context;//holds the application context
75
76 /* CourseListHandler - constructor. Takes the application context and initializes
the
77 * database, creates the database if it does not exist and creates the tables
78 * @param context - the application context
79 */
80 public CourseListHandler(Context context) {
81     super(context, DATABASE_NAME, null, DATABASE_VERSION);//initialize database
82     this.context = context;//get context
83     DB_PATH = this.context.getDatabasePath(DATABASE_NAME).getAbsolutePath();//get
database path
84 }
85
86 /* onCreate - creates the tables for the database if they do not exist in
87 * the database. This method is deprecated since the database is being added
88 * from a prebuilt database in the assets folder
89 * @param db - reference to the database
90 */
91 @Override
92 public void onCreate(SQLiteDatabase db) {
93     /* All tables are not being built by the app since BrockU is no longer
94     * being used. The tables in the database are now preloaded from the
95     * database file in the assests folder
96     *
97     * Saved here when new courses are available for the new year of school
98     */
99     String CREATE_COURSES_TABLE = "CREATE TABLE " + TABLE_MCOURSES + "("
100         + KEY_ID + " TEXT," + KEY_SUBJ + " TEXT," + KEY_CODE + " TEXT,"
101         + KEY_DESC + " TEXT," + KEY_TYPE + " TEXT," + KEY_SEC
102         + " TEXT," + KEY_DUR + " TEXT," + KEY_DAYS + " TEXT,"
103         + KEY_TIME + " TEXT," + KEY_LOCATION + " TEXT,"
104         + KEY_INSTRUCTOR + " TEXT" + ")";
105     db.execSQL(CREATE_COURSES_TABLE);
106
107     String CREATE_COURSES = "CREATE TABLE " + TABLE_COURSES + "("
108         + KEY_SUBJ + " TEXT," + KEY_CODE + " TEXT," + KEY_DESC
109         + " TEXT," + KEY_INSTRUCTOR + " TEXT," + KEY_INSTREMAIL
110         + " TEXT," + "PRIMARY KEY(" + KEY_SUBJ + "," + KEY_CODE + ")"
111         + ")";
112
113     String CREATE_TASKS = "CREATE TABLE " + TABLE_TASKS + "(" + KEY_SUBJ
114         + " TEXT," + KEY_CODE + " TEXT," + KEY_ASSIGN + " INTEGER,"
115         + KEY_NAME + " TEXT," + KEY_MARK + " INTEGER," + KEY_BASE
116         + " INTEGER," + KEY_WEIGHT + " REAL," + KEY_DUE + " TEXT,"
117         + KEY_CREATE_DATE + " TEXT," + KEY_PRIORITY + " INTEGER,"
118         + "PRIMARY KEY(" + KEY_SUBJ + "," + KEY_CODE + "," + KEY_ASSIGN
119         + ")" + ")"; //+ "FOREIGN KEY(" + KEY_SUBJ + "," + KEY_CODE
120         //+ ") REFERENCES " + TABLE_COURSES + "(" + KEY_SUBJ + ","
121         //+ KEY_CODE + ")" + ")";
122
123     String CREATE_OFFERINGS = "CREATE TABLE " + TABLE_OFFERINGS + "("
124         + KEY_ID + " INTEGER," + KEY_SUBJ + " TEXT ," + KEY_CODE
125         + " TEXT ," + KEY_TYPE + " TEXT," + KEY_SEC + " INTEGER,"
126         + "PRIMARY KEY(" + KEY_ID + ")" + ")"; //+ "FOREIGN KEY(" + KEY_SUBJ
127         //+ "," + KEY_CODE + ") REFERENCES " + TABLE_COURSES + "("
128         //+ KEY_SUBJ + "," + KEY_CODE + ")" + ")";
129
130     String CREATE_OFFERING_TIMES = "CREATE TABLE " + TABLE_OFFERING_TIMES
131         + "(" + KEY_ID + " INTEGER," + KEY_DAY + " TEXT," + KEY_TIMES
132         + " TEXT ," + KEY_TIMEE + " TEXT," + KEY_LOCATION + " TEXT,"
133         + "PRIMARY KEY(" + KEY_ID + "," + KEY_DAY + ")" + ")";
134         //+ "FOREIGN KEY(" + KEY_ID + ") REFERENCES " + TABLE_OFFERINGS
135         //+ "(" + KEY_ID + ")" + ")";
136
137     String CREATE_CONTACTS = "CREATE TABLE " + TABLE_CONTACTS + "("
138         + KEY_SUBJ + " TEXT," + KEY_CODE + " TEXT," + KEY_CID
139         + " INTEGER," + KEY_FNAME + " TEXT," + KEY_LNAME + " TEXT,"

```

```

140         + KEY_EMAIL + " TEXT," + "PRIMARY KEY(" + KEY_CID + ")"+ ")";
141         //+ "FOREIGN KEY(" + KEY_SUBJ + "," + KEY_CODE + ") REFERENCES "
142         //+ TABLE_COURSES + "(" + KEY_SUBJ + "," + KEY_CODE + ")" + ")";
143
144         db.execSQL(CREATE_COURSES);
145         db.execSQL(CREATE_TASKS);
146         db.execSQL(CREATE_OFFERINGS);
147         db.execSQL(CREATE_OFFERING_TIMES);
148         db.execSQL(CREATE_CONTACTS);
149     */
150 }
151
152 /* createDataBase - if the database does not currently exist then the database
153 * we read data from the included database to copy to a newly created one
154 */
155 public void createDataBase() throws IOException{
156
157     boolean dbExist = checkDataBase();
158
159     if(dbExist){
160         //do nothing - database already exist
161     }else{
162         //By calling this method an empty database will be created into the default
163         //system path
164         //of the application so that it can be overwritten by the included database.
165         this.getReadableDatabase();
166         try {
167             copyDataBase();
168         } catch (IOException e) {
169             throw new Error("Error copying database");
170         }
171     }
172
173 /* checkDataBase - checks if the database for the app currently exists
174 */
175 private boolean checkDataBase(){
176
177     SQLiteDatabase checkDB = null;
178
179     try{
180         String myPath = DB_PATH;// + DATABASE_NAME;
181         checkDB = SQLiteDatabase.openDatabase(myPath, null, SQLiteDatabase.
182             OPEN_READONLY);
183
184     }catch(SQLiteException e){
185         //database doesn't exist yet.
186     }
187     if(checkDB != null){
188         checkDB.close();
189     }
190     return checkDB != null ? true : false;
191 }
192
193 /* copyDataBase - copies all the data from the included database in the assets
194 * folder and copies that information to the newly created application
195 * database
196 */
197 private void copyDataBase() throws IOException{
198     //Open the asset db as the input stream
199     InputStream myInput = this.context.getAssets().open(DATABASE_NAME);
200     // Path to the just created empty db
201     String outFileName = DB_PATH;
202     //Open the empty db as the output stream
203     OutputStream myOutput = new FileOutputStream(outFileName);
204     //transfer bytes from the inputfile to the outputfile
205     byte[] buffer = new byte[1024];
206     int length;
207     while ((length = myInput.read(buffer))>0){
208         myOutput.write(buffer, 0, length);

```

```

209     }
210     //Close the streams
211     myOutput.flush();
212     myOutput.close();
213     myInput.close();
214 }
215
216 /* openDataBase - open the database from the set database path
217 */
218 public void openDataBase() throws SQLException{
219     //Open the database
220     String myPath = DB_PATH;// + DATABASE_NAME;
221     myDataBase = SQLiteDatabase.openDatabase(myPath, null, SQLiteDatabase.
222     OPEN_READONLY);
223 }
224
225 /* close - closes the streams for the database. checks if the database is open */
226 @Override
227 public synchronized void close() {
228     if(myDataBase != null)
229         myDataBase.close();
230     super.close();
231 }
232
233 /* onUpgrade - upgrading the database will drop the table and recreate */
234 @Override
235 public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
236     // Drop older table if existed
237     db.execSQL("DROP TABLE IF EXISTS " + TABLE_MCOURSES);
238     // Create tables again
239     onCreate(db);
240 }
241
242 /* addCourse - initializes Brocku which get all course information from the
243 * Brock Univeristy registrar's webiste. Gets a list of all offerings and
244 * stores the information into the MasterList table in the database
245 */
246 public void addCourse() {
247     Looper myLooper;
248     Brocku list = new Brocku();
249     myLooper = Looper.myLooper();
250     Looper.loop();
251     myLooper.quit();
252     ArrayList<MasterCourse> course = new ArrayList<MasterCourse>();
253     try {
254         course = list.execute().get();
255         SQLiteDatabase db = this.getWritableDatabase();
256         //start a bulk transaction to the database
257         db.beginTransaction();
258         for (int i = 0; i < course.size(); i++) {
259             ContentValues values = new ContentValues();
260             values.put(KEY_ID, course.get(i).id); // Course id
261             values.put(KEY_SUBJ, course.get(i).subj); // subject code
262             values.put(KEY_CODE, course.get(i).code);
263             values.put(KEY_DESC, course.get(i).desc);
264             values.put(KEY_TYPE, course.get(i).type);
265             values.put(KEY_SEC, course.get(i).sec);
266             values.put(KEY_DUR, course.get(i).dur);
267             values.put(KEY_DAYS, course.get(i).days);
268             values.put(KEY_TIME, course.get(i).time);
269             values.put(KEY_LOCATION, course.get(i).location);
270             values.put(KEY_INSTRUCTOR, course.get(i).instructor);
271             // Inserting Row to the table
272             db.insert(TABLE_MCOURSES, null, values);
273         }
274         //bulk transaction is successful
275         db.setTransactionSuccessful();
276         db.endTransaction();
277         //bulk transaction is complete
278         db.close(); // Closing database connection
279     } catch (Exception e) {}

```



```

279     }
280
281     /* getCourses - returns a list of offerings for a particular subject and
282     * code, returns an arraylist of courses
283     * @param subj - subject name
284     * @param code - subject code
285     */
286     public ArrayList<MasterCourse> getCourses(String subj, String code) {
287         SQLiteDatabase db = this.getReadableDatabase();
288         ArrayList<MasterCourse> courseList = new ArrayList<MasterCourse>();
289         courseList.ensureCapacity(50);
290         MasterCourse course;
291         //search the db for all items with subj and code
292         Cursor c = db.rawQuery("SELECT * FROM " + TABLE_MCOURSES + " where "
293             + KEY_SUBJ + " = '" + subj + "' and " + KEY_CODE + " = '" + code
294             + "'", null);
295         if (c != null) {
296             //start at the first element
297             if (c.moveToFirst()) {
298                 do {
299                     //enter the data from the query into a MasterCourse object
300                     course = new MasterCourse();
301                     course.id = c.getString(c.getColumnIndex(KEY_ID));
302                     course.subj = c.getString(c.getColumnIndex(KEY_SUBJ));
303                     course.code = c.getString(c.getColumnIndex(KEY_CODE));
304                     course.desc = c.getString(c.getColumnIndex(KEY_DESC));
305                     course.type = c.getString(c.getColumnIndex(KEY_TYPE));
306                     course.sec = c.getString(c.getColumnIndex(KEY_SEC));
307                     course.dur = c.getString(c.getColumnIndex(KEY_DUR));
308                     course.days = c.getString(c.getColumnIndex(KEY_DAYS));
309                     course.time = c.getString(c.getColumnIndex(KEY_TIME));
310                     course.location = c.getString(c
311                         .getColumnIndex(KEY_LOCATION));
312                     course.instructor = c.getString(c
313                         .getColumnIndex(KEY_INSTRUCTOR));
314                     courseList.add(course); //add this offering to the list
315                 } while (c.moveToNext());
316             }
317         }
318         c.close();
319         db.close();
320         return courseList; //return the list of offerings
321     }
322
323     /* getSubjects - returns a list of all subjects from the database */
324     public ArrayList<String> getSubjects() {
325         // String subjects;
326         ArrayList<String> subj = new ArrayList<String>();
327         try {
328             SQLiteDatabase db = this.getReadableDatabase();
329             //query the database for distinct subjects
330             Cursor c = db.rawQuery("SELECT DISTINCT " + KEY_SUBJ + " FROM "
331                 + TABLE_MCOURSES + " ORDER BY " + KEY_SUBJ + " ASC", null);
332             if (c != null) {
333                 //start at the first entry
334                 if (c.moveToFirst()) {
335                     do { //add the subjects to an arraylist
336                         subj.add(c.getString(c.getColumnIndex(KEY_SUBJ)));
337                     } while (c.moveToNext());
338                 }
339             }
340             db.close(); //close the db
341             c.close(); //close the cursor
342         } catch (Exception e) {
343             subj.add(e.toString()); //add an error to the list
344         }
345         return subj; //return the subject list
346     }
347
348     /* getCodes - returns a list of codes for a subject from the database
349     * @param subj - the subject to get all the codes for

```

```

350     */
351     public ArrayList<String> getCodes(String subj) {
352         ArrayList<String> codes = new ArrayList<String>();
353         try {
354             SQLiteDatabase db = this.getReadableDatabase();
355             //query for all distinct subject codes given the subject
356             Cursor c = db.rawQuery("SELECT DISTINCT " + KEY_CODE + " FROM "
357                 + TABLE_MCOURSES + " WHERE " + KEY_SUBJ + "='" + subj
358                 + "' ORDER BY " + KEY_SUBJ + " ASC", null);
359             if (c != null) {
360                 //start at the first element
361                 if (c.moveToFirst()) {
362                     do {
363                         //add the code to the code list
364                         codes.add(c.getString(c.getColumnIndex(KEY_CODE)));
365                     } while (c.moveToNext());
366                 }
367             }
368             db.close();//close the db
369             c.close();//close the cursor
370         } catch (Exception e) {
371             codes.add(e.toString());//add error to list if the query fails
372         }
373         return codes; //return the list of codes
374     }
375
376     /* size - returns the total number of entries in the masterList table */
377     public int size() {
378         int i = 0;
379         try {
380             SQLiteDatabase db = this.getReadableDatabase();
381             Cursor c = db.rawQuery("SELECT COUNT(*) FROM " + TABLE_MCOURSES,
382                 null);
383             if (c != null) {
384                 //move to first entry which will be the count we want
385                 c.moveToFirst();
386                 i = c.getInt(0);
387             }
388             db.close();//close the db
389             c.close();//close the cursor
390         } catch (Exception e) {
391             i = 0;//return 0 if there are no entries in the table
392         }
393         return i; //return the number of entries in the table
394     }
395 }

```

```

1  package edu.seaaddicts.brockbutler.coursemanager;
2
3  import java.util.ArrayList;
4
5  import android.app.Activity;
6  import android.app.ProgressDialog;
7  import android.content.Context;
8  import android.content.Intent;
9  import android.net.Uri;
10 import android.os.Bundle;
11 import android.os.Handler;
12 import android.util.Log;
13 import android.view.ContextMenu;
14 import android.view.ContextMenu.ContextMenuInfo;
15 import android.view.View.OnClickListener;
16 import android.view.Gravity;
17 import android.view.Menu;
18 import android.view.MenuItem;
19 import android.view.View;
20 import android.view.ViewGroup;
21 import android.widget.AdapterView;
22 import android.widget.AdapterView.Adapter;
23 import android.widget.Button;
24 import android.widget.LinearLayout;
25 import android.widget.ListView;
26 import android.widget.TextView;
27 import android.widget.Toast;
28 import edu.seaaddicts.brockbutler.R;
29 import edu.seaaddicts.brockbutler.animation.ExpandAnimation;
30 import edu.seaaddicts.brockbutler.help.HelpActivity;
31
32 public class CourseManagerActivity extends Activity {
33     public static final int CODE_COURSE_MODIFIED = 0;
34     public static final int CODE_COURSE_UNMODIFIED = 1;
35     public static final int CODE_ADD_COURSE = 2;
36
37     public static final String CODE_COURSE_SUBJECT = "csubj";
38     public static final String CODE_COURSE_CODE = "ccode";
39     public static final String CODE_COURSE_DESC = "cdesc";
40     public static final String CODE_COURSE_INSTRUCTOR = "cinstruct";
41     public static final String CODE_COURSE_INSTRUCTOR_EMAIL = "cinstructemail";
42     public static final String CODE_COURSE_OFFERINGS = "coffs";
43
44     private static final String TAG = "CourseManagerActivity";
45
46     private static final int VISIBLE = 0;
47     private static final int GONE = 8;
48
49     private ArrayList<Course> mRegisteredCoursesList;
50     private CourseHandler mCourseHandle = null;
51     private ListView mRegisteredCoursesListView = null;
52
53     @Override
54     protected void onCreate(Bundle savedInstanceState) {
55         super.onCreate(savedInstanceState);
56         setContentView(R.layout.activity_coursemanager);
57         mCourseHandle = new CourseHandler(this.getApplicationContext());
58
59         if (mCourseHandle.getSize() < 1) {
60             updateCourseDatabaseFromRegistrar();
61         }
62     }
63
64     @Override
65     protected void onResume() {
66         super.onResume();
67         populateCoursesLayout();
68     }
69
70     /**
71      * Populates the ListView with registered classes and brief details, i.e.

```

```

72     * instructor name and class times.
73     */
74     private void populateCoursesLayout() {
75         TextView tvNoCourses = (TextView) findViewById(R.id.tv_no_courses);
76         mRegisteredCoursesList = mCourseHandle.getRegisteredCourses();
77         mRegisteredCoursesListView = (ListView) findViewById(R.id.course_manager_list);
78         if (mRegisteredCoursesList.size() == 0) {
79             // There are no registered courses so set message.
80             mRegisteredCoursesListView.setVisibility(GONE);
81             tvNoCourses.setVisibility(VISIBLE);
82         } else {
83             // We have registered courses so populate ListView.
84             // Creating the list adapter and populating the list
85             ArrayAdapter<String> listAdapter = new CustomListAdapter(this,
86                 R.layout.course_list_item);
87
88             for (int i = 0; i < mRegisteredCoursesList.size(); i++) {
89                 listAdapter.add(mRegisteredCoursesList.get(i).mSubject + " "
90                     + mRegisteredCoursesList.get(i).mCode);
91
92                 for (int j = 0; j < mRegisteredCoursesList.get(i).mOfferings
93                     .size(); j++)
94                     Log.d(TAG, "Offerings: "
95                         + mRegisteredCoursesList.get(i).mOfferings.get(j));
96
97                 Log.d(TAG, "# Offerings: "
98                     + mRegisteredCoursesList.get(i).mOfferings.size());
99             }
100             mRegisteredCoursesListView.setAdapter(listAdapter);
101             tvNoCourses.setVisibility(GONE);
102             mRegisteredCoursesListView.setVisibility(VISIBLE);
103             mRegisteredCoursesListView
104                 .setOnClickListener(new AdapterView.OnItemClickListener() {
105                 public void onItemClick(AdapterView<?> parent,
106                     final View view, int position, long id) {
107                     showHideToolbar(view, position);
108                 }
109             });
110             registerContextMenu(mRegisteredCoursesListView);
111         }
112     }
113
114     public void showHelp(MenuItem item) {
115         Intent intent = new Intent(CourseManagerActivity.this,
116             HelpActivity.class);
117         Bundle bundle = new Bundle();
118         bundle.putString("activity", "coursemanager");
119         intent.putExtras(bundle);
120         startActivity(intent);
121     }
122
123     @Override
124     public void onCreateContextMenu(ContextMenu menu, View v,
125         ContextMenuInfo menuInfo) {
126         if (v.getId() == R.id.course_manager_list) {
127             //AdapterView.AdapterContextMenuInfo info =
128             (AdapterView.AdapterContextMenuInfo) menuInfo;
129             String[] menuItems = getResources().getStringArray(
130                 R.array.course_manager_context_menu);
131             for (int i = 0; i < menuItems.length; i++) {
132                 menu.add(Menu.NONE, i, i, menuItems[i]);
133             }
134         }
135     }
136
137     /**
138     * Determines which MenuItem was selected and acts appropriately depending
139     * on choice.
140     */
141     @Override
142     public boolean onContextItemSelected(MenuItem item) {

```

```

142     AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo)
        item
143         .getMenuInfo();
144     int menuItemIndex = item.getItemId();
145     Log.d(TAG, "" + menuItemIndex);
146     Course thisCourse = mRegisteredCoursesList.get(info.position);
147     switch (menuItemIndex) {
148     case 0:
149
150         // Start Intent with Course as Extra
151         Intent i = new Intent(CourseManagerActivity.this,
152             ModifyCourseActivity.class);
153         i.putExtra(CODE_COURSE_SUBJECT, thisCourse.mSubject);
154         i.putExtra(CODE_COURSE_CODE, thisCourse.mCode);
155         startActivity(i);
156         break;
157     case 1:
158         Course c = mRegisteredCoursesList.get(info.position);
159         mCourseHandle.removeCourse(c);
160     }
161     populateCoursesLayout();
162     return true;
163 }
164
165 @Override
166 protected void onActivityResult(int requestCode, int resultCode, Intent data) {
167     if (resultCode == RESULT_OK) {
168         switch (requestCode) {
169         case (CODE_ADD_COURSE):
170             // Course c = (Course)
171             // data.getSerializableExtra(CODE_COURSE_OBJECT);
172             // Toast.makeText(getApplicationContext(),
173             // c.mSubject + " " + c.mCode + " added.",
174             // Toast.LENGTH_LONG).show();
175             break;
176         default:
177             break;
178         }
179     }
180     if (resultCode == RESULT_OK && requestCode == CODE_COURSE_MODIFIED) {
181         if (data.hasExtra("returnKey1")) {
182             Toast.makeText(this, data.getExtras().getString("returnKey1"),
183                 Toast.LENGTH_SHORT).show();
184         }
185     }
186 }
187
188 /**
189  * Shows/hides verbose description of course.
190  *
191  * @param view
192  *      The selected view to hide/show.
193  */
194 private void showHideToolbar(View view, int position) {
195     View toolbar = view.findViewById(R.id.course_manager_toolbar);
196
197     // Get current course info.
198     String subj = mRegisteredCoursesList.get(position).mSubject;
199     String code = mRegisteredCoursesList.get(position).mCode;
200     String offering = "";
201     ArrayList<Offering> offs = mRegisteredCoursesList.get(position).mOfferings;
202     ArrayList<OfferingTime> offTimes;
203
204     // Creating the expand animation for the item
205     ExpandAnimation expandAni = new ExpandAnimation(toolbar,
206         ExpandAnimation.ANIMATE_SHORT);
207
208     // Start the animation on the toolbar
209     toolbar.startAnimation(expandAni);
210
211     ((TextView) view.findViewById(R.id.tv_prof_name))

```

```

212         .setText(mRegisteredCoursesList.get(position).mInstructor);
213     final String e= mRegisteredCoursesList.get(position).mInstructor_email;
214     ((Button) view.findViewById(R.id.prof_email_button))
215     .setOnClickListener(new OnClickListener() {
216         public void onClick(View v) {
217             sendEmail(e);
218         }
219     });
220
221     Log.d(TAG, "Number of Offerings for " + subj + " " + code + ": " + offs.size());
222
223     // Add Offerings registered for.
224     for (int i = 0; i < offs.size(); i++) {
225         String what = offs.get(i).mType.substring(0, 3).trim();
226
227         offTimes = offs.get(i).mOfferingTimes;
228
229         Log.d(TAG, "Offering Type: " + what + ", # " + offTimes.size());
230
231         offering = "";
232
233         // Loop through OfferingTimes for each Offering to populate
234         for (int j = 0; j < offTimes.size(); j++) {
235             offering += offTimes.get(j).mDay + " "
236                 + offTimes.get(j).mStartTime + " - "
237                 + offTimes.get(j).mEndTime + " @ "
238                 + offTimes.get(j).mLocation + "\n";
239
240             // Check for type of Offering and add as appropriate
241             if (what.equalsIgnoreCase("lec")) {
242                 TextView tv = ((TextView) view
243                     .findViewById(R.id.tv_lecture));
244                 tv.setText(offering);
245             }
246
247             else if (what.equalsIgnoreCase("lab"))
248                 ((TextView) view.findViewById(R.id.tv_lab))
249                     .setText(offering);
250
251             else if (what.equalsIgnoreCase("tut"))
252                 ((TextView) view.findViewById(R.id.tv_tutorial))
253                     .setText(offering);
254
255             else if (what.equalsIgnoreCase("sem"))
256                 ((TextView) view.findViewById(R.id.tv_seminar))
257                     .setText(offering);
258         }
259     }
260
261     float mark = mCourseHandle.getMark(mRegisteredCoursesList.get(position));
262     float base = mCourseHandle.getBase(mRegisteredCoursesList.get(position));
263     float total=0;
264     if(base != 0){
265         total= (mark/base)*100;
266     }
267
268     String grade= ""+(int)mark+"/"+(int)base+" = "+ (int)total+"%";
269     ((TextView) view.findViewById(R.id.course_grade_grade))
270     .setText(grade);
271
272     /*
273     * Hide class type if none available
274     */
275     if (((TextView) view.findViewById(R.id.tv_lecture)).getText()
276         .toString().equalsIgnoreCase("none"))
277         view.findViewById(R.id.row_lec).setVisibility(GONE);
278     if (((TextView) view.findViewById(R.id.tv_lab)).getText()
279         .toString().equalsIgnoreCase("none"))
280         view.findViewById(R.id.row_lab).setVisibility(GONE);
281     if (((TextView) view.findViewById(R.id.tv_tutorial)).getText()
282         .toString().equalsIgnoreCase("none"))

```

```

283     view.findViewById(R.id.row_tut).setVisibility(GONE);
284     if (((TextView) view.findViewById(R.id.tv_seminar)).getText()
285         .toString().equalsIgnoreCase("none"))
286         view.findViewById(R.id.row_sem).setVisibility(GONE);
287 }
288
289 @Override
290 public boolean onCreateOptionsMenu(Menu menu) {
291     // Inflate the menu; this adds items to the action bar if it is present.
292     getMenuInflater().inflate(R.menu.activity_coursemanager, menu);
293     return true;
294 }
295
296 /**
297  * A simple implementation of list adapter to populate ListView with
298  * courses.
299  */
300 class CustomListAdapter extends ArrayAdapter<String> {
301
302     public CustomListAdapter(Context context, int textViewResourceId) {
303         super(context, textViewResourceId);
304     }
305
306     @Override
307     public View getView(int position, View convertView, ViewGroup parent) {
308
309         if (convertView == null) {
310             convertView = getLayoutInflater().inflate(
311                 R.layout.course_list_item, null);
312         }
313
314         ((TextView) convertView.findViewById(R.id.course_list_item_title))
315             .setText(getItem(position));
316
317         // Resets the toolbar to be closed
318         View toolbar = convertView
319             .findViewById(R.id.course_manager_toolbar);
320         ((LinearLayout.LayoutParams) toolbar.getLayoutParams()).bottomMargin = -50;
321         toolbar.setVisibility(View.GONE);
322         return convertView;
323     }
324 }
325
326 /**
327  * Launches AddCourseActivity as intent.
328  *
329  * @param menu
330  *         MenuItem selected.
331  */
332 public void addCourse(MenuItem menu) {
333     Intent i = new Intent(CourseManagerActivity.this,
334         AddCourseActivity.class);
335     startActivity(i);
336 }
337
338 /**
339  * Allows user to manually fetch course calendar offerings from Registrar
340  *
341  * @param item
342  */
343 public void updateMaster(MenuItem item) {
344     updateCourseDatabaseFromRegistrar();
345 }
346
347 private void sendEmail(String instr_email) {
348     Intent i = new Intent(Intent.ACTION_SENDTO);
349     i.setType("message/rfc822");
350     i.setData(Uri.parse("mailto:" + instr_email));
351     try {
352         startActivity(Intent.createChooser(i, "Send mail..."));
353     } catch (android.content.ActivityNotFoundException ex) {

```

```

354         Toast.makeText(CourseManagerActivity.this,
355             "There are no email clients installed.", Toast.LENGTH_SHORT)
356         .show();
357     }
358 }
359
360 /**
361  * Updates the course calendar offerings master table. Is called at first
362  * run (if table does not exist) and manually when user wishes to check for
363  * updates. Progress bar to prevent hanging on main thread.
364  */
365 private void updateCourseDatabaseFromRegistrar() {
366     final Handler handler = new Handler();
367     final ProgressDialog progressDialog;
368
369     TextView title = new TextView(CourseManagerActivity.this);
370     title.setText(R.string.loading_courses_registrar);
371     title.setGravity(Gravity.FILL);
372
373     TextView msg = new TextView(CourseManagerActivity.this);
374     msg.setText(R.string.loading_courses_registrar_msg);
375     msg.setGravity(Gravity.FILL);
376
377     progressDialog = ProgressDialog.show(this, "Course Timetable Update",
378         "Updating course timetable from registrar. Please be patient.");
379
380     Thread thread = new Thread() {
381         public void run() {
382             mCourseHandle.updateAll();
383             // this will handle the post task.
384             // it will run when the time consuming task get finished
385             handler.post(new Runnable() {
386                 public void run() {
387
388                     // Update your UI or
389                     // do any Post job after the time consuming task
390                     // remember to dismiss the progress dialog here.
391                     // updateUI();
392                     progressDialog.dismiss();
393                     populateCoursesLayout();
394                 }
395             });
396         }
397     };
398     thread.start();
399 }
400 }
401

```



```

1  /**
2   * CurrentCoursesHandler.java
3   * Brock Butler
4   * Handles database table creation and queries for course information
5   * Created by James Grisdale on 2013-02-24
6   * Copyright (c) 2013 Sea Addicts. All rights reserved.
7   */
8
9  package edu.seaaddicts.brockbutler.coursemanager;
10
11  import java.util.ArrayList;
12
13  import android.content.ContentValues;
14  import android.content.Context;
15  import android.database.Cursor;
16  import android.database.DatabaseUtils;
17  import android.database.sqlite.SQLiteDatabase;
18  import android.database.sqlite.SQLiteOpenHelper;
19  import edu.seaaddicts.brockbutler.contacts.Contact;
20  import edu.seaaddicts.brockbutler.scheduler.Task;
21
22  public class CurrentCoursesHandler extends SQLiteOpenHelper {
23      private static final int DATABASE_VERSION = 1;
24      // Database Name
25      private static final String DATABASE_NAME = "Database";
26      // table names
27      private static final String TABLE_COURSES = "courses";
28      private static final String TABLE_TASKS = "tasks";
29      private static final String TABLE_OFFERINGS = "offerings";
30      private static final String TABLE_OFFERING_TIMES = "offering_times";
31      private static final String TABLE_CONTACTS = "contacts";
32      // field names
33      private static final String KEY_SUBJ = "subj";
34      private static final String KEY_CODE = "code";
35      private static final String KEY_DESC = "desc";
36      private static final String KEY_INSTRUCTOR = "instructor";
37      private static final String KEY_ID = "id";
38      private static final String KEY_TYPE = "type";
39      private static final String KEY_SEC = "sec";
40      private static final String KEY_DAY = "day";
41      private static final String KEY_TIMES = "time_start";
42      private static final String KEY_TIMEEE = "time_end";
43      private static final String KEY_LOCATION = "location";
44      private static final String KEY_ASSIGN = "assign";
45      private static final String KEY_NAME = "name";
46      private static final String KEY_MARK = "mark";
47      private static final String KEY_BASE = "base";
48      private static final String KEY_WEIGHT = "weight";
49      private static final String KEY_DUE = "due";
50      private static final String KEY_CREATE_DATE = "create_date";
51      private static final String KEY_IS_DONE = "is_done";
52      private static final String KEY_CID = "cid";
53      private static final String KEY_FNAME = "fname";
54      private static final String KEY_LNAME = "lname";
55      private static final String KEY_EMAIL = "email";
56      private static final String KEY_PRIORITY = "priority";
57      private static final String KEY_INSTREMAIL = "instructor_email";
58
59      /* Constructor for the database helper */
60      public CurrentCoursesHandler(Context context) {
61          super(context, DATABASE_NAME, null, DATABASE_VERSION);
62      }
63
64      /* Create tables for courses, tasks, offerings, offering times, and contacts
65       * in the database if they do not exist when the database helper is first
66       * called
67       */
68      @Override
69      public void onCreate(SQLiteDatabase db) {
70      }
71

```

```

72  /* on an upgrade drop tables and recreate */
73  @Override
74  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
75      // Drop older table if existed
76      db.execSQL("DROP TABLE IF EXISTS " + TABLE_COURSES);
77      db.execSQL("DROP TABLE IF EXISTS " + TABLE_TASKS);
78      db.execSQL("DROP TABLE IF EXISTS " + TABLE_OFFERINGS);
79      db.execSQL("DROP TABLE IF EXISTS " + TABLE_OFFERING_TIMES);
80      db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS);
81      // Create tables again
82      onCreate(db);
83  }
84
85  /* addCourse - adds all information for a course to the database adding
86   * course, offerings, tasks and contacts information, if information exists
87   * for the course then an update is done, otherwise and insert is done
88   * @param course - the course information to add to the course table
89   */
90  public void addCourse(Course course) {
91      SQLiteDatabase db = this.getWritableDatabase();
92      ContentValues values = new ContentValues();
93      long num = 0;
94      boolean update = false;
95      //check if the course already exists in the table
96      num = DatabaseUtils.queryNumEntries(db, TABLE_COURSES, KEY_SUBJ + " = '"
97          + course.mSubject + "' AND " + KEY_CODE + " = '" + course.mCode
98          + "'");
99      if (num > 0) //if it exists then do an update
100         update = true;
101      // values to be added to the table
102      values.put(KEY_SUBJ, course.mSubject); // subject code
103      values.put(KEY_CODE, course.mCode);
104      values.put(KEY_DESC, course.mDesc);
105      values.put(KEY_INSTRUCTOR, course.mInstructor);
106      values.put(KEY_INSTREMAIL, course.mInstructor_email);
107      // Inserting or updating Row
108      if (update)
109         db.update(TABLE_COURSES, values, KEY_SUBJ + " = '" + course.mSubject
110             + "' AND " + KEY_CODE + " = '" + course.mCode + "'", null);
111      else
112         db.insert(TABLE_COURSES, null, values);
113      db.close(); // Closing database connection
114      values.clear();
115      addOfferings(course); //add the offerings for the course
116      addTasks(course); //add the tasks for the course
117      addContacts(course.mContacts); //add the contacts for the course
118      db.close(); //close the database
119  }
120
121  /* deleteCourse - removes all information for the given course from the
122   * database
123   * @param course - the course data to be removed from the course table
124   */
125  public void deleteCourse(Course course) {
126      SQLiteDatabase db = this.getWritableDatabase();
127      //delete the row for the selected course
128      db.delete(TABLE_COURSES, KEY_SUBJ + " = '" + course.mSubject + "' AND "
129          + KEY_CODE + " = '" + course.mCode + "'", null);
130      db.close(); //close the db
131      //delete all the offerings
132      for (int i=0; i<course.mOfferings.size(); i++){
133         deleteOffering(course.mOfferings.get(i));
134      }
135      //delete all the tasks
136      for (int i=0; i<course.mTasks.size(); i++){
137         removeTask(course.mTasks.get(i));
138      }
139  }
140
141  /* getCourse - retrieves all information for the given course
142   * @param subj - the course name

```

```

143     * @param code - the course code
144     */
145     public Course getCourse(String subj, String code) {
146         SQLiteDatabase db = this.getReadableDatabase();
147         Course course = new Course();
148         //query to retrieve all information for the given subj and code
149         Cursor c = db.rawQuery("SELECT * FROM " + TABLE_COURSES + " where "
150             + KEY_SUBJ + " = '" + subj + "'" and " + KEY_CODE + " = '" + code
151             + "'", null);
152         if (c != null) {
153             //start at the first record
154             if (c.moveToFirst()) {
155                 do {
156                     //set values in the course object from the table
157                     course.mSubject = c.getString(c.getColumnIndex(KEY_SUBJ));
158                     course.mCode = c.getString(c.getColumnIndex(KEY_CODE));
159                     course.mDesc = c.getString(c.getColumnIndex(KEY_DESC));
160                     course.mInstructor = c.getString(c
161                         .getColumnIndex(KEY_INSTRUCTOR));
162                     course.mInstructor_email = c.getString(c
163                         .getColumnIndex(KEY_INSTREMAIL));
164                     course.mOfferings = getOfferings(course.mSubject,
165                         course.mCode);
166                     course.mTasks = getTasks(course); //get the tasks for the course
167                     course.mContacts = getContacts(course); //get the contacts for the course
168                 } while (c.moveToNext());
169             }
170         }
171         c.close(); //close the cursor
172         db.close(); //close the database
173         return course; //return the course object
174     }
175
176     /* addOfferings - adds all offerings offered by a particular course as well
177     * as their offering times
178     * @param course - the course to add the offerings from
179     */
180     public void addOfferings(Course course) {
181         Offering offering;
182         OfferingTime offeringtime;
183         ContentValues values = new ContentValues();
184         SQLiteDatabase db = this.getWritableDatabase();
185         long num = 0;
186         boolean update = false;
187         for (int i = 0; i < course.mOfferings.size(); i++) {
188             offering = course.mOfferings.get(i);
189             num = 0;
190             update = false;
191             //find if the offering already exists
192             num = DatabaseUtils.queryNumEntries(db, TABLE_OFFERINGS, KEY_SUBJ
193                 + " = '" + offering.mSubj + "'" AND " + KEY_CODE + " = '"
194                 + offering.mCode + "'" AND " + KEY_TYPE + " = '"
195                 + offering.mType + "'" AND " + KEY_SEC + " = '"
196                 + offering.mSection);
197             if (num > 0) //if the offering exists then do an update
198                 update = true;
199             //set the feilds and values
200             values.put(KEY_SUBJ, course.mSubject);
201             values.put(KEY_CODE, course.mCode);
202             values.put(KEY_TYPE, offering.mType);
203             values.put(KEY_SEC, offering.mSection);
204             if (update) //update the offering information
205                 db.update(TABLE_OFFERINGS, values, KEY_SUBJ + " = '"
206                     + offering.mSubj + "'" AND " + KEY_CODE + " = '"
207                     + offering.mCode + "'" AND " + KEY_TYPE + " = '"
208                     + offering.mType + "'" AND " + KEY_SEC + " = '"
209                     + offering.mSection, null);
210             else //insert the offering information
211                 db.insert(TABLE_OFFERINGS, null, values);
212             values.clear();
213         }

```

```

214 SQLiteDatabase rdb = this.getReadableDatabase();
215 for (int i=0; i<course.mOfferings.size(); i++){
216     offering = course.mOfferings.get(i);
217     //now adding the offering times for each offering
218     for (int j = 0; j < offering.mOfferingTimes.size(); j++) {
219         offeringtime = offering.mOfferingTimes.get(j);
220         num = 0;
221         update = false;
222         //see if the offering time exists
223         num = DatabaseUtils.queryNumEntries(db, TABLE_OFFERING_TIMES,
224             KEY_ID + " =" + offering.mId+ " AND "+KEY_DAY+"='"+offeringtime.mDay+"'");
225         if (num > 1)//if exists then update
226             update = true;
227         //query for offering id for each offering time
228         Cursor c = rdb.rawQuery("SELECT " + KEY_ID + " FROM "
229             + TABLE_OFFERINGS + " WHERE " + KEY_SUBJ + "='"
230             + offering.mSubj + "' AND " + KEY_CODE + "='"
231             + offering.mCode + "' AND " + KEY_TYPE + "='"
232             + offering.mType + "' AND " + KEY_SEC + "='"
233             + offering.mSection, null);
234         c.moveToFirst();
235         //set fields and values to be added
236         offering.mId = c.getInt(c.getColumnIndex(KEY_ID));
237         values.put(KEY_ID, offering.mId);
238         values.put(KEY_DAY, offeringtime.mDay);
239         values.put(KEY_TIMES, offeringtime.mStartTime);
240         values.put(KEY_TIMEE, offeringtime.mEndTime);
241         values.put(KEY_LOCATION, offeringtime.mLocation);
242         if (update)//update the record
243             db.update(TABLE_OFFERING_TIMES, values, KEY_ID + " ="
244                 + offering.mId, null);
245         else//insert the record
246             db.insert(TABLE_OFFERING_TIMES, null, values);
247         values.clear();
248         c.close();//close the cursor
249     }
250 }
251 rdb.close();//close database connection
252 db.close();//close database connection
253 }
254
255
256 /* deleteOffering - removes all information from the databse for the given
257 * offering
258 * @param offering - the offering to be removed from the offerings table
259 */
260 public void deleteOffering(Offering offering) {
261     int id;
262     SQLiteDatabase rdb = this.getReadableDatabase();
263     //query to get the id of the offering to be deleted
264     Cursor c = rdb.rawQuery("SELECT " + KEY_ID + " FROM "
265         + TABLE_OFFERINGS + " WHERE " + KEY_SUBJ + "='"
266         + offering.mSubj + "' AND " + KEY_CODE + "='"
267         + offering.mCode + "' AND " + KEY_TYPE + "='"
268         + offering.mType + "' AND " + KEY_SEC + "='"
269         + offering.mSection, null);
270     c.moveToFirst();
271     id = c.getInt(c.getColumnIndex(KEY_ID));
272     c.close();//close cursor
273     SQLiteDatabase db = this.getWritableDatabase();
274     //delete the offering times associated to the offering
275     db.delete(TABLE_OFFERING_TIMES, KEY_ID +"="+id, null);
276     //delete the offering from the offerings table
277     db.delete(TABLE_OFFERINGS, KEY_SUBJ
278         + " =" + offering.mSubj + "' AND " + KEY_CODE + "='"
279         + offering.mCode + "' AND " + KEY_TYPE + "='"
280         + offering.mType + "' AND " + KEY_SEC + "='"
281         + offering.mSection, null);
282     db.close();//close the database
283 }
284

```

```

285  /* addTasks - adds all tasks associated with a given course
286  * @param course - the course object with the tasks to be added
287  */
288  public void addTasks(Course course) {
289      Task task;
290      ContentValues values = new ContentValues();
291      SQLiteDatabase db = this.getWritableDatabase();
292      long num = 0;
293      boolean update = false;
294      for (int i = 0; i < course.mTasks.size(); i++) {
295          task = course.mTasks.get(i);
296          num = 0;
297          update = false;
298          //see if the task exists already
299          num = DatabaseUtils.queryNumEntries(db, TABLE_TASKS, KEY_ASSIGN
300              + " ='" + task.mAssign + "' AND " + KEY_SUBJ + " ='"
301              + task.mSubj + "' AND " + KEY_CODE + " ='" + task.mCode
302              + "'");
303          if (num > 0) //if exists then update
304              update = true;
305          values.put(KEY_SUBJ, task.mSubj);
306          values.put(KEY_CODE, task.mCode);
307          //if the task number is not 0 then use that value
308          try {
309              if (task.mAssign != 0)
310                  values.put(KEY_ASSIGN, task.mAssign);
311          } catch (NullPointerException e) {}
312          //set all values to be added
313          values.put(KEY_NAME, task.mName);
314          values.put(KEY_MARK, task.mMark);
315          values.put(KEY_BASE, task.mBase);
316          values.put(KEY_WEIGHT, task.mWeight);
317          values.put(KEY_DUE, task.mDueDate);
318          values.put(KEY_CREATE_DATE, task.mCreationDate);
319          values.put(KEY_PRIORITY, task.mPriority);
320          values.put(KEY_IS_DONE, task.mIsDone);
321          if (update) //update the row
322              db.update(TABLE_TASKS, values, KEY_ASSIGN + " ='" + task.mAssign
323                  + "' AND " + KEY_SUBJ + " ='" + task.mSubj + "' AND "
324                  + KEY_CODE + " ='" + task.mCode + "'", null);
325          else //insert the row
326              db.insert(TABLE_TASKS, null, values);
327          values.clear();
328      }
329      db.close(); //close the database
330  }
331
332  /* addContacts - add contacts to the contacts table in the database for the
333  * given list of contacts
334  * @param contacts - the list of contacts to be added to the contacts table
335  */
336  public void addContacts(ArrayList<Contact> contacts) {
337      Contact contact;
338      ContentValues values = new ContentValues();
339      SQLiteDatabase db = this.getWritableDatabase();
340      long num = 0;
341      boolean update = false;
342      for (int j = 0; j < contacts.size(); j++) {
343          contact = contacts.get(j);
344          num = 0;
345          update = false;
346          //check if the contact exists
347          num = DatabaseUtils.queryNumEntries(db, TABLE_CONTACTS, KEY_SUBJ
348              + " ='" + contact.mSubj + "' AND " + KEY_CODE + " ='"
349              + contact.mCode + "' AND " + KEY_FNAME + " ='" + contact.mFirstName
350              + "' AND " + KEY_LNAME + " ='" + contact.mLastName + "'");
351          if (num > 0) //if exists then update
352              update = true;
353          //set the fields and the values
354          values.put(KEY_SUBJ, contact.mSubj);
355          values.put(KEY_CODE, contact.mCode);

```

```

356     values.put(KEY_FNAME, contact.mFirstName);
357     values.put(KEY_LNAME, contact.mLastName);
358     values.put(KEY_EMAIL, contact.mEmail);
359     if (update)//update the record
360         db.update(TABLE_CONTACTS, values, KEY_SUBJ + " = '"
361             + contact.mSubj + "' AND " + KEY_CODE + " = '"
362             + contact.mCode + "' AND " + KEY_FNAME + " = '"
363             + contact.mFirstName + "' AND " + KEY_LNAME + " = '"
364             + contact.mLastName + "' AND " + KEY_EMAIL + " = '"
365             + contact.mEmail + "'", null);
366     else//insert the record
367         db.insert(TABLE_CONTACTS, null, values);
368     values.clear();
369 }
370 db.close();//close the database
371 }
372
373 /* addTasks - adds a task for a certain course using the addTasks(course) method
374  * @param task - the task to be added
375  */
376 public void addTasks(Task task) {
377     Course course = new Course();
378     course.mTasks.add(task);
379     addTasks(course);//add the tasks for the course object
380 }
381
382 /* getOfferings - gets all offerings for a given subject and code
383  * @param subj - the course subject
384  * @param code - the course code
385  */
386 public ArrayList<Offering> getOfferings(String subj, String code) {
387     ArrayList<Offering> offerings = new ArrayList<Offering>();
388     ArrayList<OfferingTime> offtimes;
389     Offering offering;
390     OfferingTime otime;
391     SQLiteDatabase db = this.getReadableDatabase();
392     //get all offerings for the subj and code
393     Cursor c = db.rawQuery("SELECT * FROM " + TABLE_OFFERINGS + " WHERE "
394         + KEY_SUBJ + " = '" + subj + "' and " + KEY_CODE + " = '" + code
395         + "'", null);
396     try {
397         if (c != null) {
398             if (c.moveToFirst()) { //start at the first record
399                 do {
400                     offering = new Offering();
401                     //add the data into a new offering object
402                     offering.mId = c.getInt(c.getColumnIndex(KEY_ID));
403                     offering.mSubj = c
404                         .getString(c.getColumnIndex(KEY_SUBJ));
405                     offering.mCode = c
406                         .getString(c.getColumnIndex(KEY_CODE));
407                     offering.mType = c
408                         .getString(c.getColumnIndex(KEY_TYPE));
409                     offering.mSection = c.getInt(c.getColumnIndex(KEY_SEC));
410                     offerings.add(offering);
411                 }while (c.moveToNext()); //get next record
412                 c.close();//close cursor
413                 //get all the offering times for each offering
414                 for (int i=0; i<offerings.size(); i++){
415                     offtimes = new ArrayList<OfferingTime>();
416                     //get the id for the offering
417                     Cursor o = db.rawQuery("SELECT * FROM "
418                         + TABLE_OFFERING_TIMES + " WHERE " + KEY_ID
419                         + " = " + offerings.get(i).mId, null);
420                     if (o != null) {
421                         if (o.moveToFirst()) { //move to first offering time
422                             do {
423                                 otime = new OfferingTime();
424                                 //insert data from table to OfferingTime object
425                                 otime.mOid = o.getInt(o
426                                     .getColumnIndex(KEY_ID));

```

```

427         otime.mDay = o.getString(o
428             .getColumnIndex(KEY_DAY));
429         otime.mStartTime = o.getString(o
430             .getColumnIndex(KEY_TIMES));
431         otime.mEndTime = o.getString(o
432             .getColumnIndex(KEY_TIMEE));
433         otime.mLocation = o.getString(o
434             .getColumnIndex(KEY_LOCATION));
435         offtimes.add(otime);
436     } while (o.moveToNext()); //get next time
437 }
438 }
439 offerings.get(i).mOfferingTimes = offtimes;
440 o.close(); //close cursor
441 }
442 }
443 }
444 db.close(); //close database
445 } catch (Exception e) {}
446 return offerings; //return the offerings
447 }
448
449 /* getTasks - gets all tasks a person may have from the database */
450 public ArrayList<Task> getTasks() {
451     ArrayList<Task> tasks = new ArrayList<Task>();
452     SQLiteDatabase db = this.getReadableDatabase();
453     Task task;
454     //get all tasks from the tasks table
455     Cursor c = db.rawQuery("SELECT * FROM " + TABLE_TASKS, null);
456     if (c != null) {
457         if (c.moveToFirst()) { //start at the first record
458             do {
459                 task = new Task();
460                 //insert data from the table into a new task object
461                 task.mSubj = c.getString(c.getColumnIndex(KEY_SUBJ));
462                 task.mCode = c.getString(c.getColumnIndex(KEY_CODE));
463                 task.mAssign = c.getInt(c.getColumnIndex(KEY_ASSIGN));
464                 task.mName = c.getString(c.getColumnIndex(KEY_NAME));
465                 task.mMark = c.getInt(c.getColumnIndex(KEY_MARK));
466                 task.mBase = c.getInt(c.getColumnIndex(KEY_BASE));
467                 task.mWeight = c.getFloat(c.getColumnIndex(KEY_WEIGHT));
468                 task.mDueDate = c.getString(c.getColumnIndex(KEY_DUE));
469                 task.mIsDone = c.getInt(c.getColumnIndex(KEY_IS_DONE));
470                 task.mCreationDate = c.getString(c.getColumnIndex(KEY_CREATE_DATE));
471                 task.mPriority = c.getInt(c.getColumnIndex(KEY_PRIORITY));
472                 tasks.add(task); //add the task to the list
473             } while (c.moveToNext());
474         }
475     }
476     c.close(); //close cursor
477     db.close(); //close database
478     return tasks; //return the list of tasks
479 }
480
481 /* getTasks - gets all tasks for a particular course
482 * @param course - the course to get the tasks for
483 */
484 private ArrayList<Task> getTasks(Course course) {
485     ArrayList<Task> tasks = new ArrayList<Task>();
486     SQLiteDatabase db = this.getReadableDatabase();
487     Task task;
488     // get all task information for the choosen course
489     Cursor c = db.rawQuery("SELECT * FROM " + TABLE_TASKS + " WHERE "
490         + KEY_SUBJ + "='" + course.mSubject + "' AND " + KEY_CODE
491         + "='" + course.mCode + "'", null);
492     if (c != null) {
493         if (c.moveToFirst()) { //start at the first record
494             do {
495                 task = new Task();
496                 //insert data from the table to a new task object
497                 task.mSubj = c.getString(c.getColumnIndex(KEY_SUBJ));

```

```

498         task.mCode = c.getString(c.getColumnIndex(KEY_CODE));
499         task.mAssign = c.getInt(c.getColumnIndex(KEY_ASSIGN));
500         task.mName = c.getString(c.getColumnIndex(KEY_NAME));
501         task.mMark = c.getInt(c.getColumnIndex(KEY_MARK));
502         task.mBase = c.getInt(c.getColumnIndex(KEY_BASE));
503         task.mWeight = c.getFloat(c.getColumnIndex(KEY_WEIGHT));
504         task.mDueDate = c.getString(c.getColumnIndex(KEY_DUE));
505         task.mIsDone = c.getInt(c.getColumnIndex(KEY_IS_DONE));
506         task.mCreationDate = c.getString(c.getColumnIndex(KEY_CREATE_DATE));
507         task.mPriority = c.getInt(c.getColumnIndex(KEY_PRIORITY));
508         task.mContacts = getContacts(course);
509         tasks.add(task); //add task to the list of tasks
510     } while (c.moveToNext()); //get next record
511 }
512 }
513 c.close(); //close cursor
514 db.close(); //close database
515 return tasks; //return the list of tasks
516 }
517
518 /* getContacts - get all contacts for a specified course
519  * @param course - the course object to get contacts for
520  */
521 private ArrayList<Contact> getContacts(Course course) {
522     ArrayList<Contact> contacts = new ArrayList<Contact>();
523     SQLiteDatabase db = this.getReadableDatabase();
524     Contact contact;
525     //get all contacts from the contacts table for the specified course
526     Cursor c = db.rawQuery("SELECT * FROM " + TABLE_CONTACTS + " WHERE "
527         + KEY_SUBJ + "='" + course.mSubject + "' AND " + KEY_CODE
528         + "='" + course.mCode + "'", null);
529     if (c != null) {
530         if (c.moveToFirst()) { //get first record
531             do {
532                 contact = new Contact();
533                 //insert data from the contacts table to a new contact object
534                 contact.mSubj = c.getString(c.getColumnIndex(KEY_SUBJ));
535                 contact.mCode = c.getString(c.getColumnIndex(KEY_CODE));
536                 contact.mId = c.getInt(c.getColumnIndex(KEY_CID));
537                 contact.mFirstName = c.getString(c
538                     .getColumnIndex(KEY_FNAME));
539                 contact.mLastName = c
540                     .getString(c.getColumnIndex(KEY_LNAME));
541                 contact.mEmail = c.getString(c.getColumnIndex(KEY_EMAIL));
542                 contacts.add(contact); //add contact to list of contacts
543             } while (c.moveToNext()); //get next record
544         }
545     }
546     c.close(); //close cursor
547     db.close(); //close database
548     return contacts; //return list of contacts
549 }
550
551 /* removeTask - deletes a given task from the tasks table of the database
552  * @param task - the task to be removed from the tasks table
553  */
554 public void removeTask(Task task) {
555     SQLiteDatabase db = this.getWritableDatabase();
556     //delete the record for the given task
557     db.delete(TABLE_TASKS, KEY_SUBJ
558         + "='" + task.mSubj + "' AND " + KEY_CODE + "='"
559         + task.mCode + "' AND " + KEY_ASSIGN + "="
560         + task.mAssign, null);
561     db.close(); //close database
562 }
563
564 /* removeContact - deletes a contact from the contacts table from the database
565  * @param contact - the contact information to be deleted from the table
566  */
567 public void removeContact(Contact contact){
568     SQLiteDatabase db = this.getWritableDatabase();

```



```

569         //delete the record associated with the contact id
570         db.delete(TABLE_CONTACTS, KEY_CID + "=" + contact.mId, null );
571     }
572
573     /* getRegCourses - gets all courses added to the courses table of the
574     * database and all of it's components
575     */
576     public ArrayList<Course> getRegCourses() {
577         ArrayList<Course> courses = new ArrayList<Course>();
578         SQLiteDatabase db = this.getReadableDatabase();
579         try {
580             //get all courses from the course table
581             Cursor c = db.rawQuery("SELECT * FROM " + TABLE_COURSES, null);
582             if (c != null) {
583                 if (c.moveToFirst()) { //start at the first record
584                     do {
585                         //get course information for each course found in courses table
586                         courses.add(getCourse(
587                             c.getString(c.getColumnIndex(KEY_SUBJ)),
588                             c.getString(c.getColumnIndex(KEY_CODE))));
589                     } while (c.moveToNext()); //get next record
590                 }
591             }
592             c.close(); //close cursor
593         } catch (Exception e) {}
594         db.close(); //close database
595         return courses; //return list of current courses
596     }
597
598     /* Query - a general method to allow a query to be done that has not been
599     * specified. it returns a cursor object to allow the data to be read
600     * @param s - a query sent as a string to perform the query on the database
601     */
602     public Cursor Query(String s) {
603         SQLiteDatabase db = this.getReadableDatabase();
604         Cursor c = db.rawQuery(s, null); //perform query
605         db.close();
606         return c; //return cursor object
607     }
608 }
609

```

```
1  /**
2   * MasterCourse.java
3   * Brock Butler
4   * A wrapper class for course timetable information
5   * Created by James Grisdale on 2013-02-24
6   * Copyright (c) 2013 Sea Addicts. All rights reserved.
7   **/
8
9  package edu.seaaddicts.brockbutler.coursemanager;
10
11  public class MasterCourse {
12      public String id; //course id
13      public String code; //course code
14      public String subj; //faculty name
15      public String desc; //course description
16      public String type; //course type
17      public String sec; //section
18      public String dur; //duration
19      public String days; //days offered
20      public String time; //time offered
21      public String location; //location
22      public String instructor; //instructor
23  }
24
```

```

1  package edu.seaaddicts.brockbutler.coursemanager;
2
3  import java.util.ArrayList;
4
5  import android.app.Activity;
6  import android.os.Bundle;
7  import android.util.Log;
8  import android.util.SparseBooleanArray;
9  import android.view.View;
10 import android.view.View.OnClickListener;
11 import android.widget.AdapterView;
12 import android.widget.Button;
13 import android.widget.LinearLayout;
14 import android.widget.ListView;
15 import android.widget.TextView;
16 import edu.seaaddicts.brockbutler.R;
17 import edu.seaaddicts.brockbutler.contacts.Contact;
18 import edu.seaaddicts.brockbutler.scheduler.Task;
19
20 public class ModifyCourseActivity extends Activity {
21
22     private static final String TAG = "ModifyCourseActivity";
23
24     private static final int VISIBLE = 0;
25     private static final int GONE = 8;
26
27     ArrayList<String> mLecs;
28     ArrayList<String> mLabs;
29     ArrayList<String> mTuts;
30     ArrayList<String> mSems;
31
32     ArrayList<Offering> mLecsOfferings;
33     ArrayList<Offering> mLabsOfferings;
34     ArrayList<Offering> mTutsOfferings;
35     ArrayList<Offering> mSemsOfferings;
36
37     public String mSubject;
38     public String mCode;
39     public String mDesc;
40     public String mInstructor;
41     public String mInstructor_email;
42     public ArrayList<Offering> mOfferings;
43     public ArrayList<Task> mTasks;
44     public ArrayList<Contact> mContacts;
45     public Course course;
46     public Course brock;
47
48     private Button mSaveButton;
49     private Button mCancelButton;
50
51     private ListView mLecsListView;
52     private ListView mSemsListView;
53     private ListView mTutsListView;
54     private ListView mLabsListView;
55
56     private CurrentCoursesHandler mCourseHandle;
57
58     private TextView mSubjectTextView;
59
60     @Override
61     protected void onCreate(Bundle savedInstanceState) {
62         super.onCreate(savedInstanceState);
63         setContentView(R.layout.activity_modify_course);
64         Bundle bundle = this.getIntent().getExtras();
65
66         brock = new CourseHandler(getApplicationContext()).getCourseOfferings(
67             bundle.getString(CourseManagerActivity.CODE_COURSE_SUBJECT),
68             bundle.getString(CourseManagerActivity.CODE_COURSE_CODE));
69         mCourseHandle = new CurrentCoursesHandler(this(getApplicationContext());
70         course = mCourseHandle.getCourse(bundle.getString(CourseManagerActivity.
71             CODE_COURSE_SUBJECT),

```

```

71         bundle.getString(CourseManagerActivity.CODE_COURSE_CODE));
72
73     init();
74 }
75
76 /*
77  * Initialize all views and sets Button OnClickListener.
78  */
79 private void init() {
80     // Set Buttons
81     mSaveButton = (Button) findViewById(R.id.add_course_save_button);
82     mCancelButton = (Button) findViewById(R.id.add_course_cancel_button);
83
84     // Instantiate TextView
85     mSubjectTextView = (TextView) findViewById(R.id.modify_course_subject_textview);
86
87     // Set TextView
88     mSubjectTextView.setText(course.mSubject+" "+course.mCode);
89
90     mCode = course.mCode;
91     mOfferings = brock.mOfferings;
92
93     String s;
94     mLecs = new ArrayList<String>();
95     mLabs = new ArrayList<String>();
96     mTuts = new ArrayList<String>();
97     mSems = new ArrayList<String>();
98
99     mLecsOfferings = new ArrayList<Offering>();
100    mLabsOfferings = new ArrayList<Offering>();
101    mTutsOfferings = new ArrayList<Offering>();
102    mSemsOfferings = new ArrayList<Offering>();
103
104    for (int i = 0; i < mOfferings.size(); i++) {
105        s = mOfferings.get(i).mType;
106
107        // Some offerings don't have any of what we are looking for,
108        // so check length to make sure.
109        if (s.length() > 2) {
110            String ss = s.substring(0, 3).trim();
111            if (ss.equalsIgnoreCase("lec")) {
112                mLecs.add(s + ", SEC " + mOfferings.get(i).mSection);
113                mLecsOfferings.add(mOfferings.get(i));
114            } else if (ss.equalsIgnoreCase("lab")) {
115                mLabs.add(s + ", SEC " + mOfferings.get(i).mSection);
116                mLabsOfferings.add(mOfferings.get(i));
117            } else if (ss.equalsIgnoreCase("tut")) {
118                mTuts.add(s + ", SEC " + mOfferings.get(i).mSection);
119                mTutsOfferings.add(mOfferings.get(i));
120            } else if (ss.equalsIgnoreCase("sem")) {
121                mSems.add(s + ", SEC " + mOfferings.get(i).mSection);
122                mSemsOfferings.add(mOfferings.get(i));
123            }
124        }
125    }
126
127    // Check if offerings available, and add ListView to display if so.
128    LinearLayout lec_lay = (LinearLayout) findViewById(R.id.layout_add_lecs);
129    LinearLayout lab_lay = (LinearLayout) findViewById(R.id.layout_add_labs);
130    LinearLayout tut_lay = (LinearLayout) findViewById(R.id.layout_add_tuts);
131    LinearLayout sem_lay = (LinearLayout) findViewById(R.id.layout_add_sems);
132
133    if (mLecs.size() > 0) {
134        lec_lay.setVisibility(VISIBLE);
135        mLecsListView = (ListView) findViewById(R.id.add_course_add_lecs);
136        mLecsListView.setAdapter(new ArrayAdapter<String>(
137            getApplicationContext(),
138            android.R.layout.simple_list_item_multiple_choice,
139            mLecs));
140    } else {
141        lec_lay.setVisibility(GONE);

```

```

142     }
143     if (mLabs.size() > 0) {
144         lab_layout.setVisibility(VISIBLE);
145         mLabsListView = (ListView) findViewById(R.id.add_course_add_labs);
146         mLabsListView.setAdapter(new ArrayAdapter<String>(
147             getApplicationContext(),
148             android.R.layout.simple_list_item_multiple_choice,
149             mLabs));
150     } else {
151         lab_layout.setVisibility(GONE);
152     }
153     if (mTuts.size() > 0) {
154         tut_layout.setVisibility(VISIBLE);
155         mTutsListView = (ListView) findViewById(R.id.add_course_add_tuts);
156         mTutsListView.setAdapter(new ArrayAdapter<String>(
157             getApplicationContext(),
158             android.R.layout.simple_list_item_multiple_choice,
159             mTuts));
160     } else {
161         tut_layout.setVisibility(GONE);
162     }
163     if (mSemsOfferings.size() > 0) {
164         sem_layout.setVisibility(VISIBLE);
165         mSemsListView = (ListView) findViewById(R.id.add_course_add_sems);
166         mSemsListView.setAdapter(new ArrayAdapter<String>(
167             getApplicationContext(),
168             android.R.layout.simple_list_item_multiple_choice,
169             mSems));
170     } else {
171         sem_layout.setVisibility(GONE);
172     }
173
174     // Set OnClickListener
175     mSaveButton.setOnClickListener(new OnClickListener() {
176         public void onClick(View v) {
177             Course c = new Course();
178             c.mSubject = course.mSubject;
179             c.mCode = course.mCode;
180             c.mDesc = course.mDesc;
181             c.mInstructor = course.mInstructor;
182             c.mInstructor_email = course.mInstructor_email;
183             for (Task t : course.mTasks)
184                 c.mTasks.add(t);
185
186             for (Offering o : course.mOfferings)
187                 mCourseHandle.deleteOffering(o);
188
189             SparseBooleanArray sbal, sba2, sba3, sba4;
190             if (mLecsListView != null) {
191                 sbal = mLecsListView.getCheckedItemPositions();
192                 for (int i = 0; i < mLecsListView.getCount(); i++) {
193                     if (sbal.get(i) == true) {
194                         c.mOfferings.add(mLecsOfferings.get(i));
195                         Log.d(TAG, "Added: " + mLecsOfferings.get(i).mSubj
196                             + " " + mOfferings.get(i).mCode + ", Type "
197                             + mOfferings.get(i).mType + ", Section "
198                             + mOfferings.get(i).mSection
199                             + " to Offerings");
200                     }
201                 }
202             }
203
204             if (mLabsListView != null) {
205                 sba2 = mLabsListView.getCheckedItemPositions();
206
207                 for (int i = 0; i < mLabsListView.getCount(); i++) {
208                     if (sba2.get(i) == true) {
209                         c.mOfferings.add(mLabsOfferings.get(i));
210                         Log.d(TAG, "Added: " + mLabsOfferings.get(i).mSubj
211                             + " " + mOfferings.get(i).mCode + ", Type "
212                             + mOfferings.get(i).mType + ", Section "

```

```

213         + mOfferings.get(i).mSection
214         + " to Offerings");
215     }
216 }
217 }
218
219 if (mTutsListView != null) {
220     sba3 = mTutsListView.getCheckedItemPositions();
221
222     for (int i = 0; i < mTutsListView.getCount(); i++) {
223         if (sba3.get(i) == true) {
224             c.mOfferings.add(mTutsOfferings.get(i));
225             Log.d(TAG, "Added: " + mTutsOfferings.get(i).mSubj
226                 + " " + mOfferings.get(i).mCode + ", Type "
227                 + mOfferings.get(i).mType + ", Section "
228                 + mOfferings.get(i).mSection
229                 + " to Offerings");
230         }
231     }
232 }
233
234 if (mSemsListView != null) {
235     sba4 = mSemsListView.getCheckedItemPositions();
236
237     for (int i = 0; i < mSemsListView.getCount(); i++) {
238         if (sba4.get(i) == true) {
239             c.mOfferings.add(mSemsOfferings.get(i));
240             Log.d(TAG, "Added: " + mSemsOfferings.get(i).mSubj
241                 + " " + mOfferings.get(i).mCode + ", Type "
242                 + mOfferings.get(i).mType + ", Section "
243                 + mOfferings.get(i).mSection
244                 + " to Offerings");
245         }
246     }
247 }
248 if (c.mSubject != null) {
249     try {
250         mCourseHandle.addCourse(c);
251         onBackPressed();
252     } catch (Exception e) {
253         // TODO Auto-generated catch block
254         e.printStackTrace();
255     }
256 }
257 }
258 });
259
260 mCancelButton.setOnClickListener(new OnClickListener() {
261     public void onClick(View v) {
262         // Do nothing.
263         onBackPressed();
264     }
265 });
266 }
267 }
268

```

```
1  /**
2   * Offering.java
3   * Brock Butler
4   * A wrapper class for Offering information
5   * Created by James Grisdale on 2013-02-24
6   * Copyright (c) 2013 Sea Addicts. All rights reserved.
7   **/
8
9  package edu.seaaddicts.brockbutler.coursemanager;
10
11  import java.util.ArrayList;
12
13  public class Offering {
14      public int mId;        //offering id
15      public String mSubj;   //faculty name
16      public String mCode;   //course code
17      public int mSection;   //section
18      public String mType;   //type
19      public ArrayList<OfferingTime> mOfferingTimes; //list of times offered
20      public int mOid;       //extra id if needed
21
22      //constructor - initializes arraylist of offering times
23      public Offering(){
24          mOfferingTimes = new ArrayList<OfferingTime>();
25      }
26  }
```

```
1  /**
2   * OfferingTime.java
3   * Brock Butler
4   * A wrapper class for OfferingTime information
5   * Created by James Grisdale on 2013-02-24
6   * Copyright (c) 2013 Sea Addicts. All rights reserved.
7  **/
8
9  package edu.seaaddicts.brockbutler.coursemanager;
10
11  public class OfferingTime {
12      public int mOid; //id associated with offering
13      public String mStartTime; //start time of offering
14      public String mEndTime; //offering end time
15      public String mDay; //day available
16      public String mLocation; //location
17  }
18
```



```
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2   xmlns:tools="http://schemas.android.com/tools"
3   android:layout_width="match_parent"
4   android:layout_height="match_parent"
5   android:layout_margin="10sp"
6   android:orientation="vertical"
7   android:scrollbars="vertical"
8   tools:context=".AddTaskActivity" >
9
10  <LinearLayout
11    android:layout_width="fill_parent"
12    android:layout_height="wrap_content"
13    android:orientation="horizontal" >
14
15    <TextView
16      android:layout_width="85sp"
17      android:layout_height="wrap_content"
18      android:text="@string/add_course_subject" />
19
20    <Spinner
21      android:id="@+id/add_course_subjects_spinner"
22      android:layout_width="180sp"
23      android:layout_height="40sp"
24      android:layout_marginLeft="25sp" />
25  </LinearLayout>
26
27  <LinearLayout
28    android:layout_width="fill_parent"
29    android:layout_height="wrap_content"
30    android:orientation="horizontal" >
31
32    <TextView
33      android:layout_width="85sp"
34      android:layout_height="wrap_content"
35      android:text="@string/add_course_code" />
36
37    <Spinner
38      android:id="@+id/add_course_codes_spinner"
39      android:layout_width="180sp"
40      android:layout_height="40sp"
41      android:layout_marginLeft="25sp" />
42  </LinearLayout>
43
44  <LinearLayout
45    android:id="@+id/layout_add_lecs"
46    android:layout_width="fill_parent"
47    android:layout_height="wrap_content"
48    android:layout_marginTop="20sp"
49    android:orientation="horizontal"
50    android:visibility="gone" >
51
52    <TextView
53      android:layout_width="85sp"
54      android:layout_height="wrap_content"
55      android:text="@string/add_course_add_lecs" />
56
57    <ListView
58      android:id="@+id/add_course_add_lecs"
59      android:layout_width="fill_parent"
60      android:layout_height="100sp"
61      android:layout_marginLeft="25sp"
62      android:choiceMode="singleChoice" />
63  </LinearLayout>
64
65  <LinearLayout
66    android:id="@+id/layout_add_labs"
67    android:layout_width="fill_parent"
68    android:layout_height="wrap_content"
69    android:layout_marginTop="20sp"
70    android:orientation="horizontal"
71    android:visibility="gone" >
```

```
72
73     <TextView
74         android:layout_width="85sp"
75         android:layout_height="wrap_content"
76         android:text="@string/add_course_add_labs" />
77
78     <ListView
79         android:id="@+id/add_course_add_labs"
80         android:layout_width="fill_parent"
81         android:layout_height="100sp"
82         android:layout_marginLeft="25sp"
83         android:choiceMode="singleChoice"
84         android:text="@string/add_course_add_labs" />
85 </LinearLayout>
86
87 <LinearLayout
88     android:id="@+id/layout_add_tuts"
89     android:layout_width="fill_parent"
90     android:layout_height="wrap_content"
91     android:layout_marginTop="20sp"
92     android:orientation="horizontal"
93     android:visibility="gone" >
94
95     <TextView
96         android:layout_width="85sp"
97         android:layout_height="wrap_content"
98         android:text="@string/add_course_add_tuts" />
99
100     <ListView
101         android:id="@+id/add_course_add_tuts"
102         android:layout_width="fill_parent"
103         android:layout_height="100sp"
104         android:layout_marginLeft="25sp"
105         android:choiceMode="singleChoice"
106         android:text="@string/add_course_add_tuts" />
107 </LinearLayout>
108
109 <LinearLayout
110     android:id="@+id/layout_add_sems"
111     android:layout_width="fill_parent"
112     android:layout_height="wrap_content"
113     android:layout_marginTop="20sp"
114     android:orientation="horizontal"
115     android:visibility="gone" >
116
117     <TextView
118         android:layout_width="85sp"
119         android:layout_height="wrap_content"
120         android:text="@string/add_course_add_sems" />
121
122     <ListView
123         android:id="@+id/add_course_add_sems"
124         android:layout_width="fill_parent"
125         android:layout_height="100sp"
126         android:layout_marginLeft="25sp"
127         android:choiceMode="singleChoice"
128         android:text="@string/add_course_add_sems" />
129 </LinearLayout>
130
131 <LinearLayout
132     android:layout_width="fill_parent"
133     android:layout_height="wrap_content"
134     android:gravity="center_horizontal"
135     android:orientation="horizontal"
136     android:layout_marginTop="10sp" >
137
138     <Button
139         android:id="@+id/add_course_save_button"
140         android:layout_width="100sp"
141         android:layout_height="wrap_content"
142         android:text="@string/button_save" />
```

```
143
144         <Button
145             android:id="@+id/add_course_cancel_button"
146             android:layout_width="100sp"
147             android:layout_height="wrap_content"
148             android:layout_marginLeft="20sp"
149             android:text="@string/button_cancel" />
150     </LinearLayout>
151
152 </LinearLayout>
```

```
1  <menu xmlns:android="http://schemas.android.com/apk/res/android" >
2
3      <item
4          android:id="@+id/add_course"
5          android:orderInCategory="100"
6          android:showAsAction="never"
7          android:onClick="addCourse"
8          android:title="@string/add_course_add"/>
9
10     <item
11         android:id="@+id/update_master_list"
12         android:orderInCategory="100"
13         android:showAsAction="never"
14         android:onClick="updateMaster"
15         android:title="@string/add_course_update_master_list"/>
16     <item
17         android:id="@+id/menu_show_coursemanager_help"
18         android:orderInCategory="100"
19         android:showAsAction="never"
20         android:onClick="showHelp"
21         android:title="@string/menu_show_coursemanager_help"/>
22
23 </menu>
```

```
1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:id="@+id/course_manager_layout"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent" >
6
7      <ListView
8          android:id="@+id/course_manager_list"
9          android:layout_width="fill_parent"
10         android:layout_height="fill_parent"
11         android:visibility="gone"
12         android:divider="@drawable/grad_course_title" >
13  </ListView>
14
15  <TextView
16      android:id="@+id/tv_no_courses"
17      android:layout_width="fill_parent"
18      android:layout_height="fill_parent"
19      android:text="@string/no_courses"
20      android:textColor="#ddd"
21      android:textStyle="italic"
22      android:textSize="20sp"
23      android:gravity="center"
24      android:visibility="gone" />
25  </LinearLayout>
```

```
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2   xmlns:tools="http://schemas.android.com/tools"
3   android:layout_width="match_parent"
4   android:layout_height="match_parent"
5   android:layout_margin="10sp"
6   android:orientation="vertical"
7   android:scrollbars="vertical"
8   tools:context=".ModifyTaskActivity" >
9
10  <LinearLayout
11    android:layout_width="fill_parent"
12    android:layout_height="wrap_content"
13    android:orientation="horizontal" >
14
15    <TextView
16      android:id="@+id/modify_course_subject_textview"
17      android:layout_width="85sp"
18      android:layout_height="wrap_content"
19      android:text="@string/add_course_subject" />
20
21  </LinearLayout>
22
23  <LinearLayout
24    android:id="@+id/layout_add_lecs"
25    android:layout_width="fill_parent"
26    android:layout_height="wrap_content"
27    android:layout_marginTop="20sp"
28    android:orientation="horizontal"
29    android:visibility="gone" >
30
31    <TextView
32      android:layout_width="85sp"
33      android:layout_height="wrap_content"
34      android:text="@string/add_course_add_lecs" />
35
36    <ListView
37      android:id="@+id/add_course_add_lecs"
38      android:layout_width="fill_parent"
39      android:layout_height="100sp"
40      android:layout_marginLeft="25sp"
41      android:choiceMode="singleChoice" />
42  </LinearLayout>
43
44  <LinearLayout
45    android:id="@+id/layout_add_labs"
46    android:layout_width="fill_parent"
47    android:layout_height="wrap_content"
48    android:layout_marginTop="20sp"
49    android:orientation="horizontal"
50    android:visibility="gone" >
51
52    <TextView
53      android:layout_width="85sp"
54      android:layout_height="wrap_content"
55      android:text="@string/add_course_add_labs" />
56
57    <ListView
58      android:id="@+id/add_course_add_labs"
59      android:layout_width="fill_parent"
60      android:layout_height="100sp"
61      android:layout_marginLeft="25sp"
62      android:choiceMode="singleChoice"
63      android:text="@string/add_course_add_labs" />
64  </LinearLayout>
65
66  <LinearLayout
67    android:id="@+id/layout_add_tuts"
68    android:layout_width="fill_parent"
69    android:layout_height="wrap_content"
70    android:layout_marginTop="20sp"
71    android:orientation="horizontal"
```

```
72         android:visibility="gone" >
73
74         <TextView
75             android:layout_width="85sp"
76             android:layout_height="wrap_content"
77             android:text="@string/add_course_add_tuts" />
78
79         <ListView
80             android:id="@+id/add_course_add_tuts"
81             android:layout_width="fill_parent"
82             android:layout_height="100sp"
83             android:layout_marginLeft="25sp"
84             android:choiceMode="singleChoice"
85             android:text="@string/add_course_add_tuts" />
86     </LinearLayout>
87
88     <LinearLayout
89         android:id="@+id/layout_add_sems"
90         android:layout_width="fill_parent"
91         android:layout_height="wrap_content"
92         android:layout_marginTop="20sp"
93         android:orientation="horizontal"
94         android:visibility="gone" >
95
96         <TextView
97             android:layout_width="85sp"
98             android:layout_height="wrap_content"
99             android:text="@string/add_course_add_sems" />
100
101         <ListView
102             android:id="@+id/add_course_add_sems"
103             android:layout_width="fill_parent"
104             android:layout_height="100sp"
105             android:layout_marginLeft="25sp"
106             android:choiceMode="singleChoice"
107             android:text="@string/add_course_add_sems" />
108     </LinearLayout>
109
110     <LinearLayout
111         android:layout_width="fill_parent"
112         android:layout_height="wrap_content"
113         android:gravity="center_horizontal"
114         android:orientation="horizontal"
115         android:layout_marginTop="10sp" >
116
117         <Button
118             android:id="@+id/add_course_save_button"
119             android:layout_width="100sp"
120             android:layout_height="wrap_content"
121             android:text="@string/button_save" />
122
123         <Button
124             android:id="@+id/add_course_cancel_button"
125             android:layout_width="100sp"
126             android:layout_height="wrap_content"
127             android:layout_marginLeft="20sp"
128             android:text="@string/button_cancel" />
129     </LinearLayout>
130
131 </LinearLayout>
```