```java
package edu.seaaddicts.brockbutler.coursemanager;

import java.util.ArrayList;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.util.SparseBooleanArray;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.ListView;
import android.widget.Spinner;
import edu.seaaddicts.brockbutler.R;

public class AddCourseActivity extends Activity {

  private static final String TAG = "AddCourseActivity";

  //private static final int DATE_DIALOG_ID = 0;
  private static final int VISIBLE = 0;
  //private static final int INVISIBLE = 4;
  private static final int GONE = 8;

  ArrayList<String> mLecs;
  ArrayList<String> mLabs;
  ArrayList<String> mTuts;
  ArrayList<String> mSems;

  ArrayList<Offering> mLecsOfferings;
  ArrayList<Offering> mLabsOfferings;
  ArrayList<Offering> mTutsOfferings;
  ArrayList<Offering> mSemsOfferings;

  private String mSubject;
  private String mCode;

  private Course mCourse;

  private Button mSaveButton;
  private Button mCancelButton;

  private CourseHandler mCourseHandle;
  ArrayList<Offering> mOfferings;

  private Spinner mSubjectSpinner;
  private Spinner mCodesSpinner;

  private ListView mLecsListView;
  private ListView mSemsListView;
  private ListView mTutsListView;
  private ListView mLabsListView;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_add_course);
    mCourseHandle = new CourseHandler(this.getApplicationContext());
    init();
  }

  /*
   * Initialize all views and sets Button OnClickListeners.
   */
  private void init() {

    mSaveButton = (Button) findViewById(R.id.add_course_save_button);
```

```
        mCancelButton = (Button) findViewById(R.id.add_course_cancel_button);

        mSubjectSpinner = (Spinner) findViewById(R.id.add_course_subjects_spinner);
        try {
          mSubjectSpinner.setAdapter(new ArrayAdapter<String>(this,
              android.R.layout.simple_spinner_dropdown_item,
              mCourseHandle.getSubjects()));
        } catch (Exception e) {
          e.printStackTrace();
        }

        mCodesSpinner = (Spinner) findViewById(R.id.add_course_codes_spinner);
        mSubjectSpinner.setOnItemSelectedListener(new OnItemSelectedListener() {
          public void onItemSelected(AdapterView<?> arg0, View arg1,
              int arg2, long arg3) {
            try {
              mSubject = arg0.getItemAtPosition(arg2).toString();
              mCodesSpinner.setAdapter(new ArrayAdapter<String>(
                  AddCourseActivity.this,
                  android.R.layout.simple_spinner_dropdown_item,
                  mCourseHandle.getCodes(mSubject)));
            } catch (Exception e) {
              e.printStackTrace();
            }
          }

          public void onNothingSelected(AdapterView<?> arg0) {
            // Do nothing.
          }
        });
        mCodesSpinner.setOnItemSelectedListener(new OnItemSelectedListener() {

          public void onItemSelected(AdapterView<?> arg0, View arg1,
              int arg2, long arg3) {
            mCode = arg0.getItemAtPosition(arg2).toString();
            mCourse = mCourseHandle.getCourseOfferings(mSubject, mCode);
            mOfferings = mCourse.mOfferings;

            String s;
            mLecs = new ArrayList<String>();
            mLabs = new ArrayList<String>();
            mTuts = new ArrayList<String>();
            mSems = new ArrayList<String>();

            mLecsOfferings = new ArrayList<Offering>();
            mLabsOfferings = new ArrayList<Offering>();
            mTutsOfferings = new ArrayList<Offering>();
            mSemsOfferings = new ArrayList<Offering>();

            for (int i = 0; i < mOfferings.size(); i++) {
              s = mOfferings.get(i).mType;

              // Some offerings don't have any of what we are looking for,
              // so check length to make sure.
              if (s.length() > 2) {
                String ss = s.substring(0, 3).trim();
                if (ss.equalsIgnoreCase("lec")) {
                  mLecs.add(s + ", SEC " + mOfferings.get(i).mSection);
                  mLecsOfferings.add(mOfferings.get(i));
                } else if (ss.equalsIgnoreCase("lab")) {
                  mLabs.add(s + ", SEC " + mOfferings.get(i).mSection);
                  mLabsOfferings.add(mOfferings.get(i));
                } else if (ss.equalsIgnoreCase("tut")) {
                  mTuts.add(s + ", SEC " + mOfferings.get(i).mSection);
                  mTutsOfferings.add(mOfferings.get(i));
                } else if (ss.equalsIgnoreCase("sem")) {
                  mSems.add(s + ", SEC " + mOfferings.get(i).mSection);
                  mSemsOfferings.add(mOfferings.get(i));
                }
              }
            }
```

```java

          // Check if offerings available, and add ListView to display if
          // so.
          LinearLayout lec_lay = (LinearLayout) findViewById(R.id.layout_add_lecs);
          LinearLayout lab_lay = (LinearLayout) findViewById(R.id.layout_add_labs);
          LinearLayout tut_lay = (LinearLayout) findViewById(R.id.layout_add_tuts);
          LinearLayout sem_lay = (LinearLayout) findViewById(R.id.layout_add_sems);

          if (mLecs.size() > 0) {
            lec_lay.setVisibility(VISIBLE);
            mLecsListView = (ListView) findViewById(R.id.add_course_add_lecs);
            mLecsListView.setAdapter(new ArrayAdapter<String>(
                getApplicationContext(),
                android.R.layout.simple_list_item_multiple_choice,
                mLecs));
          } else {
            lec_lay.setVisibility(GONE);
          }
          if (mLabs.size() > 0) {
            lab_lay.setVisibility(VISIBLE);
            mLabsListView = (ListView) findViewById(R.id.add_course_add_labs);
            mLabsListView.setAdapter(new ArrayAdapter<String>(
                getApplicationContext(),
                android.R.layout.simple_list_item_multiple_choice,
                mLabs));
          } else {
            lab_lay.setVisibility(GONE);
          }
          if (mTuts.size() > 0) {
            tut_lay.setVisibility(VISIBLE);
            mTutsListView = (ListView) findViewById(R.id.add_course_add_tuts);
            mTutsListView.setAdapter(new ArrayAdapter<String>(
                getApplicationContext(),
                android.R.layout.simple_list_item_multiple_choice,
                mTuts));
          } else {
            tut_lay.setVisibility(GONE);
          }
          if (mSemsOfferings.size() > 0) {
            sem_lay.setVisibility(VISIBLE);
            mSemsListView = (ListView) findViewById(R.id.add_course_add_sems);
            mSemsListView.setAdapter(new ArrayAdapter<String>(
                getApplicationContext(),
                android.R.layout.simple_list_item_multiple_choice,
                mSems));
          } else {
            sem_lay.setVisibility(GONE);
          }
        }

        public void onNothingSelected(AdapterView<?> arg0) {
        }
      });

      mSaveButton.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
          Course c = new Course();
          c.mSubject = mSubject;
          c.mCode = mCode;
          c.mInstructor = mCourseHandle.getCourseOfferings(mSubject,
              mCode).mInstructor;
          try {
            c.mInstructor_email = ""+c.mInstructor.charAt(c.mInstructor.length()-1);
            c.mInstructor_email += c.mInstructor.substring(0,c.mInstructor.length()-2);
            c.mInstructor_email += "@brocku.ca";
          } catch (Exception e){}

          SparseBooleanArray sba1, sba2, sba3, sba4;
          if (mLecsListView != null) {
            sba1 = mLecsListView.getCheckedItemPositions();
            for (int i = 0; i < mLecsListView.getCount(); i++) {
```

```
          if (sba1.get(i) == true) {
            c.mOfferings.add(mLecsOfferings.get(i));
            Log.d(TAG, "Added: " + mLecsOfferings.get(i).mSubj
                + " " + mOfferings.get(i).mCode + ", Type "
                + mOfferings.get(i).mType + ", Section "
                + mOfferings.get(i).mSection
                + " to Offerings");
          }
        }
      }

      if (mLabsListView != null) {
        sba2 = mLabsListView.getCheckedItemPositions();

        for (int i = 0; i < mLabsListView.getCount(); i++) {
          if (sba2.get(i) == true) {
            c.mOfferings.add(mLabsOfferings.get(i));
            Log.d(TAG, "Added: " + mLabsOfferings.get(i).mSubj
                + " " + mOfferings.get(i).mCode + ", Type "
                + mOfferings.get(i).mType + ", Section "
                + mOfferings.get(i).mSection
                + " to Offerings");
          }
        }
      }

      if (mTutsListView != null) {
        sba3 = mTutsListView.getCheckedItemPositions();

        for (int i = 0; i < mTutsListView.getCount(); i++) {
          if (sba3.get(i) == true) {
            c.mOfferings.add(mTutsOfferings.get(i));
            Log.d(TAG, "Added: " + mTutsOfferings.get(i).mSubj
                + " " + mOfferings.get(i).mCode + ", Type "
                + mOfferings.get(i).mType + ", Section "
                + mOfferings.get(i).mSection
                + " to Offerings");
          }
        }
      }

      if (mSemsListView != null) {
        sba4 = mSemsListView.getCheckedItemPositions();

        for (int i = 0; i < mSemsListView.getCount(); i++) {
          if (sba4.get(i) == true) {
            c.mOfferings.add(mSemsOfferings.get(i));
            Log.d(TAG, "Added: " + mSemsOfferings.get(i).mSubj
                + " " + mOfferings.get(i).mCode + ", Type "
                + mOfferings.get(i).mType + ", Section "
                + mOfferings.get(i).mSection
                + " to Offerings");
          }
        }
      }

      if (c.mSubject != null) {
        try {
          mCourseHandle.addCourse(c);
          Log.d("# ADDED OFFERINGS:", "" + c.mOfferings.size());
          for (int i = 0; i < c.mOfferings.size(); i++) {
            Log.d(TAG, "Added: " + c.mOfferings.get(i).mSubj
                + " " + mOfferings.get(i).mCode + ", Type "
                + mOfferings.get(i).mType + ", Section "
                + mOfferings.get(i).mSection
                + " to Offerings");
          }
          onBackPressed();
        } catch (Exception e) {
          e.printStackTrace();
        }
```

```
          }
        }
      });

      mCancelButton.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
          onBackPressed();
        }
      });
    }
  }

```

```java
/**
 * Brocku.java
 * Brock Butler
 * Connects to the Brock University registrars' website to obtain course
 * information from the current timetable
 * Created by James Grisdale on 2013-02-24
 * * Copyright (c) 2013 Sea Addicts. All rights reserved.
**/

package edu.seaaddicts.brockbutler.coursemanager;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URI;
import java.util.ArrayList;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.*;
import org.apache.http.HttpResponse;
import android.os.AsyncTask;
//Using AsyncTask to do operations off the main thread
public class Brocku extends AsyncTask<Void, Void, ArrayList<MasterCourse>> {

  /* doInBackground - connects to Brock University's registrar's office website to
  gather
   * information on courses being offered, then returns an arraylist of MasterCourse
   * objects which hold the data for all offerings at Brock.
   */
  protected ArrayList<MasterCourse> doInBackground(Void... Void) {
    String codes[] = new String[74];
    BufferedReader in = null;
    String info = new String();
    String substring = new String();
    MasterCourse course;
    boolean done;
    ArrayList<MasterCourse> courseList = new ArrayList<MasterCourse>();
    courseList.ensureCapacity(8000);
    //test = "working";
    try{
      HttpClient client = new DefaultHttpClient();
      HttpGet request = new HttpGet();
      URI BTimeTable = new URI(
      "http://www.brocku.ca/registrar/guides/returning/timetable/a_get_subj.php?subj=C
      OSC");
      request.setURI(BTimeTable);
      HttpResponse response = client.execute(request);
      in = new BufferedReader(new InputStreamReader(response.getEntity().getContent
      ()));
      for (int i=0; i<3; i++) in.readLine();
      for (int i=0; i<74; i++){
        info = in.readLine();
        codes[i] = info.substring(19,23);
      }//retrieving all subjects
      in.close();
      for (int h=0; h<codes.length ; h++){
        done = false;
        BTimeTable = new URI(
        "http://www.brocku.ca/registrar/guides/returning/timetable/a_get_subj.php?subj
        ="+codes[h]);
        request.setURI(BTimeTable);
        response = client.execute(request);
        in = new BufferedReader(new InputStreamReader(response.getEntity().getContent
        ()));
        for (int i=0; i<98; i++) {in.readLine();}
        info = in.readLine();
        //for each subjects get all course offering information
        if (info.length()<50){
          while(!done){
            course = new MasterCourse();
            substring = info.substring(24, info.length() - 5);
            course.id = substring;
```

```
              for (int i=0; i<2; i++) {in.readLine();}
              info = in.readLine();
              substring = info.substring(123, 127);
              course.subj = substring;
              substring = info.substring(128, 132);
              course.code = substring;
              in.readLine();
              info = in.readLine();
              substring = info.substring(26,info.length() - 7);
              course.desc = substring;
              for (int i=0; i<4; i++) {in.readLine();}
              info = in.readLine();
              substring = info.substring(24,26);
              course.dur = substring;
              info = in.readLine();
              substring = info.substring(24,info.length() - 5);
              course.type = substring;
              info = in.readLine();
              substring = info.substring(24,info.length() - 5);
              course.sec = substring;
              info = in.readLine();
              substring = info.substring(24,30);
              course.days = substring;
              info = in.readLine();
              substring = info.substring(24,info.length() - 5);
              course.time = substring;
              info = in.readLine();
              substring = info.substring(24,info.length() - 5);
              if (info.substring(24,26).equals("<a"))
                substring = info.substring(94, info.length()-9);
              course.location = substring;
              info = in.readLine();
              if (info.length()>16)
                substring = info.substring(9,info.length() - 5);
              else substring = " ";
              course.instructor = substring;
              in.readLine();
              in.readLine();
              info = in.readLine();
              courseList.add(course);
              if (info.length() <20){
                info = in.readLine();
                done = true;
              }
            }
          }
          in.close();
        }
      }
      //if there's an error return the error information in a course object
      catch (Exception e){
      info = e.toString();
      course = new MasterCourse();
      course.id = info;
      courseList.add(course);
      }
      return courseList;
    }

    //not used
    protected void onPostExecute(MasterCourse course){
      posttest(course);
    }

    //not used
    public MasterCourse posttest(MasterCourse course){
      return course;
    }
  }

```

```java
/**
 * Course.java
 * Brock Butler
 * A wrapper class for Course information
 * Created by James Grisdale on 2013-02-24
 * Copyright (c) 2013 Sea Addicts. All rights reserved.
 **/

package edu.seaaddicts.brockbutler.coursemanager;

import java.util.ArrayList;

import edu.seaaddicts.brockbutler.contacts.Contact;
import edu.seaaddicts.brockbutler.scheduler.Task;

public class Course {
  public int mId; //course ID
  public String mSubject;  //subject name
  public String mCode;  //course code
  public String mDesc;  //course description
  public String mInstructor;  //instructor name
  public String mInstructor_email;  //instructor's email
  public ArrayList<Offering> mOfferings;  //list of offerings for this course
  public ArrayList<Task> mTasks; //list of tasks associated with this course
  public ArrayList<Contact> mContacts;  //contacts for this course

  //Constructor - initializes the arraylists for offerings, tasks and contacts
  public Course() {
    mOfferings = new ArrayList<Offering>();
    mTasks = new ArrayList<Task>();
    mContacts = new ArrayList<Contact>();
  }
}
```

```
/**
 * CourseHandler.java
 * Brock Butler
 * A class to allow easy access to database functions
 * Created by James Grisdale on 2013-02-24
 * Copyright (c) 2013 Sea Addicts. All rights reserved.
 **/

package edu.seaaddicts.brockbutler.coursemanager;
import java.util.ArrayList;

import android.content.Context;
import android.database.Cursor;
import edu.seaaddicts.brockbutler.contacts.Contact;
import edu.seaaddicts.brockbutler.scheduler.Task;

public class CourseHandler {
  // Context context;
  CurrentCoursesHandler CH;
  CourseListHandler courseList;

  /* Constructor - opens and closes database to ensure the database exists
   *    and if not, it copies over the installed database of course offerings
   * @param context - application context
   */
  public CourseHandler(Context context) {
    // this.context = context;
    CH = new CurrentCoursesHandler(context);
    courseList = new CourseListHandler(context);
    try{
    courseList.createDataBase();
    courseList.openDataBase();
    courseList.close();
    }
    catch(Exception e){};
    //SQLiteDatabase db = courseList.getWritableDatabase();
    //db.close();
  }

  /* updateAll - updates all course information from the Brock University registrar's
   * office website
   */
  public void updateAll(){
    courseList.addCourse();
  }

  /* Depreciated - getAllCourses - grabs course data from the registrar's timetable
  and
   * inserts data into the masterlist table.
   * Depreciated due to information no longer being available on website
   */
  public void getAllCourses() {
    courseList.addCourse();
  }

  /* getCourse - gets all information for a given course subject and code
   * @param subj - subject name to get
   * @param code - course code to get
  */
  public Course getCourse(final String subj, final String code) {
    return CH.getCourse(subj, code);
  }

  /* updateCourse - updates all the information for a given course
   * @param course - course information to update
   */
  public void updateCourse(Course course) {
    CH.addCourse(course);
  }

  /* getSubjects - gets a list of subjects available from the master list
```

```java
     * returns an arraylist of subject offerings
     */
    public ArrayList<String> getSubjects() throws Exception {
      return courseList.getSubjects();
    }

    /* getCodes - gets a list of codes for a given subject from the master list
     * returns an arraylist of subject codes
     * @param subj - return codes for this subject
     */
    public ArrayList<String> getCodes(String subj) {
      return courseList.getCodes(subj);
    }

    /* getCourseOfferings - returns all offerings offered for a given course
     * Converts the offerings from MasterCourse format to Course format
     * @param subj - subject name
     * @param code - course code
     */
    public Course getCourseOfferings(String subj, String code) {
      //get list of offerings as a list of MasterCourse objects
      ArrayList<MasterCourse> list = courseList.getCourses(subj, code);
      Course course = new Course(); //create a new course object
      ArrayList<OfferingTime> offeringtimes;
      course.mSubject = list.get(0).subj;
      course.mCode = list.get(0).code;
      course.mInstructor = list.get(0).instructor;
      course.mDesc = list.get(0).desc;
      Offering offering;
      int tindex = 0;
      OfferingTime otime;
      //add all offerings for a particular course
      ArrayList<Offering> offerings = new ArrayList<Offering>();
      for (int i = 0; i < list.size(); i++) {
        offering = new Offering();
        offering.mSubj = list.get(i).subj;
        offering.mCode = list.get(i).code;
        offering.mType = list.get(i).type;
        offering.mSection = Integer.parseInt(list.get(i).sec);
        //add all the offeringtimes associated with all the offerings
        offeringtimes = new ArrayList<OfferingTime>();
        for (int j = 0; j < 5; j++) {
          if (list.get(i).days.charAt(j) != ' ') {
            otime = new OfferingTime();
            otime.mDay = list.get(i).days.substring(j, j+1);
            otime.mLocation = list.get(i).location;
            for (int h = 0; h < list.get(i).time.length(); h++) {
              if (list.get(i).time.charAt(h) == '-') {
                tindex = h;
                break;
              }
            }
            //get the times for each offering
            otime.mStartTime = list.get(i).time.substring(0, tindex);
            otime.mEndTime = list.get(i).time.substring(tindex + 1,
                list.get(i).time.length());
            offeringtimes.add(otime);
          }
        }
        offering.mOfferingTimes = offeringtimes;
        offerings.add(offering);

      }
      course.mOfferings = offerings;

      return course;//return the course object
    }

    /* addCourse - adds information for a course into the database
     * returns a 0 if sucessful, 1 if the add failed
     * @param course - the course object to be added
     */
    public int addCourse(Course course) throws Exception {
      try {
        CH.addCourse(course);
        return 0;
      } catch (Exception e) {
        return 1;
      }
    }

    /* removeCourse - deletes all information from the database for a course
     * returns a 0 on success, returns 1 if failure
     * @param course - the course information to be deleted
     */
    public int removeCourse(Course course) {
      try {
        CH.deleteCourse(course);
        return 0;
      } catch (Exception e) {
        return 1;
      }
    }

    /*
     * getRegisteredCourses - returns all information for all courses in the
     * current courses database
     */
    public ArrayList<Course> getRegisteredCourses() {
      return CH.getRegCourses();
    }

    /* getOfferings - get all offerings for a certain course
     * @param subj - course name
     * @param code - course code
     */
    public ArrayList<Offering> getOfferings(String subj, String code) {
      return CH.getOfferings(subj, code);
    }

    /* getTasks - gets all tasks from the database
     */
    public ArrayList<Task> getTasks() {
      return CH.getTasks();
    }

    /* addTask - adds a given task to the task table in the database
     * returns 0 if sucessful, returns 1 if it fails
     * @param task - the task information to be added to the database
     */
    public int addTask(Task task) {
      try {
        CH.addTasks(task);
        return 0;
      } catch (Exception e) {
        return 1;
      }
    }

    // addTask - adds the tasks for a given course to the task table in the
      // database
    public int addTask(Course course) {
      try {
        CH.addTasks(course);
        return 0;
      } catch (Exception e) {
        return 1;
      }
    }

    /* removeTask - deletes task information from the database for a given task
     * returns 0 if sucessful, 1 if failure
     * @param task -  the task information to be removed from the database
     */
    public int removeTask(Task task) {
      try {
        CH.removeTask(task);
        return 0;
      } catch (Exception e) {
        return 1;
      }
    }

    /* getBase - returns the total base mark for a particular course
     * given base information from the course
     * @param course - the course to calculate the total base for
     */
    public float getBase(Course course) {
      float base = 0;
      for (int i = 0; i < course.mTasks.size(); i++)
        base +=course.mTasks.get(i).mWeight;
       return base;
    }

    /* getMark - returns the calculated progress mark for a course
     * given mark information from the course, the mark is calculated and returned
     * as a float
     * @param course - the course to calculate the marks for
     */
    public float getMark(Course course) {
      float mark = 0;
      for (int i = 0; i < course.mTasks.size(); i++){
        if(course.mTasks.get(i).mBase !=0){
        mark += (course.mTasks.get(i).mMark / course.mTasks.get(i).mBase)
            * course.mTasks.get(i).mWeight;}
        else mark+=0;
      }

       return mark;
    }

    /* getSize - returns the number of courses added to the course database
     */
    public int getSize() {
      return courseList.size();
    }

    /* removeContact - removes contact information from the database
     * @param contact - the contact information to be removed
     */
    public void removeContact(Contact contact){
      CH.removeContact(contact);
    }

    /* Query - returns a cursor with results for a custom query
     * @param query - a string with a sqlite query
     */
    public Cursor Query(String query) {
      return CH.Query(query);
    }
  }

```

```java
/**
 * CourseListHandler.java
 * Brock Butler
 * Creates a database table for a full list of offerings on the registrar's
 * timetable and allows the table to have inserts or be read
 * Created by James Grisdale on 2013-02-24
 * Copyright (c) 2013 Sea Addicts. All rights reserved.
 **/

package edu.seaaddicts.brockbutler.coursemanager;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Looper;

public class CourseListHandler extends SQLiteOpenHelper {

  // All Static variables
  // Database Version
  private static final int DATABASE_VERSION = 1;
  private static final String DATABASE_NAME = "Database";
  private static String DB_PATH =
  "/data/data/edu.seaaddicts.brockbutler.cousemanager/databases";
    private SQLiteDatabase myDataBase;
  // Database Name


  // Full course list table name
  private static final String TABLE_MCOURSES = "MasterList";
  //current courses table names
  //private static final String TABLE_COURSES = "courses";
  //private static final String TABLE_TASKS = "tasks";
  //private static final String TABLE_OFFERINGS = "offerings";
  //private static final String TABLE_OFFERING_TIMES = "offering_times";
  //private static final String TABLE_CONTACTS = "contacts";
  // All field names used in the database
  private static final String KEY_SUBJ = "subj";
  private static final String KEY_CODE = "code";
  private static final String KEY_DESC = "desc";
  private static final String KEY_INSTRUCTOR = "instructor";
  private static final String KEY_ID = "id";
  private static final String KEY_TYPE = "type";
  private static final String KEY_SEC = "sec";
  //private static final String KEY_DAY = "day";
  //private static final String KEY_TIMES = "time_start";
  //private static final String KEY_TIMEE = "time_end";
  private static final String KEY_LOCATION = "location";
  private static final String KEY_DUR = "dur";
  //private static final String KEY_ASSIGN = "assign";
  //private static final String KEY_NAME = "name";
  //private static final String KEY_MARK = "mark";
  //private static final String KEY_BASE = "base";
  //private static final String KEY_WEIGHT = "weight";
  //private static final String KEY_DUE = "due";
  //private static final String KEY_CREATE_DATE = "create_date";
  //private static final String KEY_CID = "cid";
  //private static final String KEY_FNAME = "fname";
  //private static final String KEY_LNAME = "lname";
  //private static final String KEY_EMAIL = "email";
  //private static final String KEY_PRIORITY = "priority";
```

```
    //private static final String KEY_INSTREMAIL = "instructor_email";
    private static final String KEY_DAYS = "days";
    private static final String KEY_TIME = "time";
    Context context;//holds the application context

    /* CourseListHandler - constructor.  Takes the application context and initializes the
     * database, creates the database if it does not exist and creates the tables
     * @param context - the application context
     */
    public CourseListHandler(Context context) {
      super(context, DATABASE_NAME, null, DATABASE_VERSION);//initialize database
      this.context = context;//get context
      DB_PATH = this.context.getDatabasePath(DATABASE_NAME).getAbsolutePath();//get database path
    }

    /* onCreate - creates the tables for the database if they do not exist in
     * the database. This method is depreciated since the database is being added
     * from a prebuilt database in the assets folder
     * @param db - reference to the database
     */
    @Override
    public void onCreate(SQLiteDatabase db) {
      /* All tables are not being built by the app since BrockU is no longer
       * being used.  The tables in the database are now preloaded from the
       * database file in the assests folder
       *
       * Saved here when new courses are available for the new year of school
       *
      String CREATE_COURSES_TABLE = "CREATE TABLE " + TABLE_MCOURSES + "("
          + KEY_ID + " TEXT," + KEY_SUBJ + " TEXT," + KEY_CODE + " TEXT,"
          + KEY_DESC + " TEXT," + KEY_TYPE + " TEXT," + KEY_SEC
          + " TEXT," + KEY_DUR + " TEXT," + KEY_DAYS + " TEXT,"
          + KEY_TIME + " TEXT," + KEY_LOCATION + " TEXT,"
          + KEY_INSTRUCTOR + " TEXT" + ")";
      db.execSQL(CREATE_COURSES_TABLE);

      String CREATE_COURSES = "CREATE TABLE " + TABLE_COURSES + "("
          + KEY_SUBJ + " TEXT," + KEY_CODE + " TEXT," + KEY_DESC
          + " TEXT," + KEY_INSTRUCTOR + " TEXT," + KEY_INSTREMAIL
          + " TEXT," + "PRIMARY KEY(" + KEY_SUBJ + "," + KEY_CODE + ")"
          + ")";

      String CREATE_TASKS = "CREATE TABLE " + TABLE_TASKS + "(" + KEY_SUBJ
          + " TEXT," + KEY_CODE + " TEXT," + KEY_ASSIGN + " INTEGER,"
          + KEY_NAME + " TEXT," + KEY_MARK + " INTEGER," + KEY_BASE
          + " INTEGER," + KEY_WEIGHT + " REAL," + KEY_DUE + " TEXT,"
          + KEY_CREATE_DATE + " TEXT," + KEY_PRIORITY + " INTEGER,"
          + "PRIMARY KEY(" + KEY_SUBJ + "," + KEY_CODE + "," + KEY_ASSIGN
          + ")"+ ")"; //+ "FOREIGN KEY(" + KEY_SUBJ + "," + KEY_CODE
          //+ ") REFERENCES " + TABLE_COURSES + "(" + KEY_SUBJ + ","
          //+ KEY_CODE + ")" + ")";

      String CREATE_OFFERINGS = "CREATE TABLE " + TABLE_OFFERINGS + "("
          + KEY_ID + " INTEGER," + KEY_SUBJ + " TEXT ," + KEY_CODE
          + " TEXT ," + KEY_TYPE + " TEXT," + KEY_SEC + " INTEGER,"
          + "PRIMARY KEY(" + KEY_ID + ")"+ ")";// + "FOREIGN KEY(" + KEY_SUBJ
          //+ "," + KEY_CODE + ") REFERENCES " + TABLE_COURSES + "("
          //+ KEY_SUBJ + "," + KEY_CODE + ")" + ")";

      String CREATE_OFFERING_TIMES = "CREATE TABLE " + TABLE_OFFERING_TIMES
          + "(" + KEY_ID + " INTEGER," + KEY_DAY + " TEXT," + KEY_TIMES
          + " TEXT ," + KEY_TIMEE + " TEXT," + KEY_LOCATION + " TEXT,"
          + "PRIMARY KEY(" + KEY_ID + "," + KEY_DAY + ")"+ ")";
          //+ "FOREIGN KEY(" + KEY_ID + ") REFERENCES " + TABLE_OFFERINGS
          //+ "(" + KEY_ID + ")" + ")";

      String CREATE_CONTACTS = "CREATE TABLE " + TABLE_CONTACTS + "("
          + KEY_SUBJ + " TEXT," + KEY_CODE + " TEXT," + KEY_CID
          + " INTEGER," + KEY_FNAME + " TEXT," + KEY_LNAME + " TEXT,"
```

```
            + KEY_EMAIL + " TEXT," + "PRIMARY KEY(" + KEY_CID + ")"+ ")";
            //+ "FOREIGN KEY(" + KEY_SUBJ + "," + KEY_CODE + ") REFERENCES "
            //+ TABLE_COURSES + "(" + KEY_SUBJ + "," + KEY_CODE + ")" + ")";

        db.execSQL(CREATE_COURSES);
        db.execSQL(CREATE_TASKS);
        db.execSQL(CREATE_OFFERINGS);
        db.execSQL(CREATE_OFFERING_TIMES);
        db.execSQL(CREATE_CONTACTS);
        */
    }

    /* createDataBase - if the database does not currently exist then the database
     * we read data from the included database to copy to a newly created one
     */
    public void createDataBase() throws IOException{

        boolean dbExist = checkDataBase();

        if(dbExist){
          //do nothing - database already exist
        }else{
        //By calling this method an empty database will be created into the default
        system path
        //of the application so that it can be overwritten by the included database.
          this.getReadableDatabase();
          try {
            copyDataBase();
          } catch (IOException e) {
            throw new Error("Error copying database");
          }
        }
    }

    /* checkDataBase - checks if the database for the app currently exists
     */
    private boolean checkDataBase(){

        SQLiteDatabase checkDB = null;

        try{
          String myPath = DB_PATH;// + DATABASE_NAME;
          checkDB = SQLiteDatabase.openDatabase(myPath, null, SQLiteDatabase.
          OPEN_READONLY);

        }catch(SQLiteException e){
          //database does't exist yet.
        }
        if(checkDB != null){
          checkDB.close();
        }
         return checkDB != null ? true : false;
    }


     /* copyDataBase - copies all the data from the included database in the assests
      * folder and copies that information to the newly created application
      * database
      */
    private void copyDataBase() throws IOException{
      //Open the asset db as the input stream
      InputStream myInput = this.context.getAssets().open(DATABASE_NAME);
      // Path to the just created empty db
      String outFileName = DB_PATH;
      //Open the empty db as the output stream
      OutputStream myOutput = new FileOutputStream(outFileName);
      //transfer bytes from the inputfile to the outputfile
      byte[] buffer = new byte[1024];
      int length;
      while ((length = myInput.read(buffer))>0){
        myOutput.write(buffer, 0, length);
      }
      //Close the streams
      myOutput.flush();
      myOutput.close();
      myInput.close();
    }

    /* openDataBase - open the database from the set database path
     */
    public void openDataBase() throws SQLException{
      //Open the database
      String myPath = DB_PATH;// + DATABASE_NAME;
      myDataBase = SQLiteDatabase.openDatabase(myPath, null, SQLiteDatabase.
      OPEN_READONLY);
    }

    /* close - closes the streams for the database. checks if the database is open */
      @Override
    public synchronized void close() {
      if(myDataBase != null)
      myDataBase.close();
      super.close();
    }

    /* onUpgrade - upgrading the database will drop the table and recreate */
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
      // Drop older table if existed
      db.execSQL("DROP TABLE IF EXISTS " + TABLE_MCOURSES);
      // Create tables again
      onCreate(db);
    }

    /* addCourse - initializes Brocku which get all course information from the
     * Brock Univeristy registrar's webiste.  Gets a list of all offerings and
     * stores the information into the MasterList table in the database
     */
    public void addCourse() {
      Looper myLooper;
      Brocku list = new Brocku();
      myLooper = Looper.myLooper();
          Looper.loop();
          myLooper.quit();
      ArrayList<MasterCourse> course = new ArrayList<MasterCourse>();
      try {
        course = list.execute().get();
        SQLiteDatabase db = this.getWritableDatabase();
        //start a bulk transaction to the database
        db.beginTransaction();
        for (int i = 0; i < course.size(); i++) {
          ContentValues values = new ContentValues();
          values.put(KEY_ID, course.get(i).id); // Course id
          values.put(KEY_SUBJ, course.get(i).subj); // subject code
          values.put(KEY_CODE, course.get(i).code);
          values.put(KEY_DESC, course.get(i).desc);
          values.put(KEY_TYPE, course.get(i).type);
          values.put(KEY_SEC, course.get(i).sec);
          values.put(KEY_DUR, course.get(i).dur);
          values.put(KEY_DAYS, course.get(i).days);
          values.put(KEY_TIME, course.get(i).time);
          values.put(KEY_LOCATION, course.get(i).location);
          values.put(KEY_INSTRUCTOR, course.get(i).instructor);
          // Inserting Row to the table
          db.insert(TABLE_MCOURSES, null, values);
        }
        //bulk transaction is successful
        db.setTransactionSuccessful();
        db.endTransaction();
        //bulk transaction is complete
        db.close(); // Closing database connection
      } catch (Exception e) {}
```

```java
    }

    /* getCourses - returns a list of offerings for a particular subject and
     * code, returns an arraylist of courses
     * @param subj - subject name
     * @param code - subject code
    */
    public ArrayList<MasterCourse> getCourses(String subj, String code) {
      SQLiteDatabase db = this.getReadableDatabase();
      ArrayList<MasterCourse> courseList = new ArrayList<MasterCourse>();
      courseList.ensureCapacity(50);
      MasterCourse course;
      //search the db for all items with subj and code
      Cursor c = db.rawQuery("SELECT * FROM " + TABLE_MCOURSES + " where "
          + KEY_SUBJ + "= '" + subj + "' and " + KEY_CODE + " = '" + code
          + "'", null);
      if (c != null) {
        //start at the first element
        if (c.moveToFirst()) {
          do {
            //enter the data from the query into a MasterCourse object
            course = new MasterCourse();
            course.id = c.getString(c.getColumnIndex(KEY_ID));
            course.subj = c.getString(c.getColumnIndex(KEY_SUBJ));
            course.code = c.getString(c.getColumnIndex(KEY_CODE));
            course.desc = c.getString(c.getColumnIndex(KEY_DESC));
            course.type = c.getString(c.getColumnIndex(KEY_TYPE));
            course.sec = c.getString(c.getColumnIndex(KEY_SEC));
            course.dur = c.getString(c.getColumnIndex(KEY_DUR));
            course.days = c.getString(c.getColumnIndex(KEY_DAYS));
            course.time = c.getString(c.getColumnIndex(KEY_TIME));
            course.location = c.getString(c
                .getColumnIndex(KEY_LOCATION));
            course.instructor = c.getString(c
                .getColumnIndex(KEY_INSTRUCTOR));
            courseList.add(course);//add this offering to the list
          } while (c.moveToNext());
        }
      }
      c.close();
      db.close();
      return courseList;//return the list of offerings
    }

    /* getSubjects - returns a list of all subjects from the database */
    public ArrayList<String> getSubjects() {
      // String subjects;
      ArrayList<String> subj = new ArrayList<String>();
      try {
        SQLiteDatabase db = this.getReadableDatabase();
        //query the database for distinct subjects
        Cursor c = db.rawQuery("SELECT DISTINCT " + KEY_SUBJ + " FROM "
            + TABLE_MCOURSES + " ORDER BY " + KEY_SUBJ + " ASC", null);
        if (c != null) {
          //start at the first entry
          if (c.moveToFirst()) {
            do {//add the subjects to an arraylist
              subj.add(c.getString(c.getColumnIndex(KEY_SUBJ)));
            } while (c.moveToNext());
          }
        }
        db.close();//close the db
        c.close();//close the cursor
      } catch (Exception e) {
        subj.add(e.toString());//add an error to the list
      }
      return subj; //return the subject list
    }

    /* getCodes - returns a list of codes for a subject from the database
     * @param subj - the subject to get all the codes for
     */
    public ArrayList<String> getCodes(String subj) {
      ArrayList<String> codes = new ArrayList<String>();
      try {
        SQLiteDatabase db = this.getReadableDatabase();
        //query for all distict subject codes given the subject
        Cursor c = db.rawQuery("SELECT DISTINCT " + KEY_CODE + " FROM "
            + TABLE_MCOURSES + " WHERE " + KEY_SUBJ + "='" + subj
            + "' ORDER BY " + KEY_SUBJ + " ASC", null);
        if (c != null) {
          //start at the first element
          if (c.moveToFirst()) {
            do {
              //add the code to the code list
              codes.add(c.getString(c.getColumnIndex(KEY_CODE)));
            } while (c.moveToNext());
          }
        }
        db.close();//close the db
        c.close();//close the cursor
      } catch (Exception e) {
        codes.add(e.toString());//add error to list if the query fails
      }
      return codes; //return the list of codes
    }

    /* size - returns the total number of entries in the masterList table */
    public int size() {
      int i = 0;
      try {
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor c = db.rawQuery("SELECT COUNT(*) FROM " + TABLE_MCOURSES,
            null);
        if (c != null) {
          //move to first entry which will be the count we want
          c.moveToFirst();
          i = c.getInt(0);
        }
        db.close();//close the db
        c.close();//close the cursor
      } catch (Exception e) {
        i = 0;//return 0 if there are no entries in the table
      }
      return i; //return the number of entries in the table
    }
  }
```

```java
package edu.seaaddicts.brockbutler.coursemanager;

import java.util.ArrayList;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.View.OnClickListener;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import edu.seaaddicts.brockbutler.R;
import edu.seaaddicts.brockbutler.animation.ExpandAnimation;
import edu.seaaddicts.brockbutler.help.HelpActivity;

public class CourseManagerActivity extends Activity {
  public static final int CODE_COURSE_MODIFIED = 0;
  public static final int CODE_COURSE_UNMODIFIED = 1;
  public static final int CODE_ADD_COURSE = 2;

  public static final String CODE_COURSE_SUBJECT = "csubj";
  public static final String CODE_COURSE_CODE = "ccode";
  public static final String CODE_COURSE_DESC = "cdesc";
  public static final String CODE_COURSE_INSTRUCTOR = "cinstruct";
  public static final String CODE_COURSE_INSTRUCTOR_EMAIL = "cinstructemail";
  public static final String CODE_COURSE_OFFERINGS = "coffs";

  private static final String TAG = "CourseManagerActivity";

  private static final int VISIBLE = 0;
  private static final int GONE = 8;

  private ArrayList<Course> mRegisteredCoursesList;
  private CourseHandler mCourseHandle = null;
  private ListView mRegisteredCoursesListView = null;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_coursemanager);
    mCourseHandle = new CourseHandler(this.getApplicationContext());

    if (mCourseHandle.getSize() < 1) {
      updateCourseDatabaseFromRegistrar();
    }
  }

  @Override
  protected void onResume() {
    super.onResume();
    populateCoursesLayout();
  }

  /**
   * Populates the ListView with registered classes and brief details, i.e.
```

```java
     * instructor name and class times.
     */
    private void populateCoursesLayout() {
      TextView tvNoCourses = (TextView) findViewById(R.id.tv_no_courses);
      mRegisteredCoursesList = mCourseHandle.getRegisteredCourses();
      mRegisteredCoursesListView = (ListView) findViewById(R.id.course_manager_list);
      if (mRegisteredCoursesList.size() == 0) {
        // There are no registered courses so set message.
        mRegisteredCoursesListView.setVisibility(GONE);
        tvNoCourses.setVisibility(VISIBLE);
      } else {
        // We have registered courses so populate ListView.
        // Creating the list adapter and populating the list
        ArrayAdapter<String> listAdapter = new CustomListAdapter(this,
            R.layout.course_list_item);

        for (int i = 0; i < mRegisteredCoursesList.size(); i++) {
          listAdapter.add(mRegisteredCoursesList.get(i).mSubject + " "
              + mRegisteredCoursesList.get(i).mCode);

          for (int j = 0; j < mRegisteredCoursesList.get(i).mOfferings
              .size(); j++)
            Log.d(TAG, "Offerings: "
                + mRegisteredCoursesList.get(i).mOfferings.get(j));

          Log.d(TAG, "# Offerings: "
              + mRegisteredCoursesList.get(i).mOfferings.size());
        }
        mRegisteredCoursesListView.setAdapter(listAdapter);
        tvNoCourses.setVisibility(GONE);
        mRegisteredCoursesListView.setVisibility(VISIBLE);
        mRegisteredCoursesListView
            .setOnItemClickListener(new AdapterView.OnItemClickListener() {
              public void onItemClick(AdapterView<?> parent,
                  final View view, int position, long id) {
                showHideToolbar(view, position);
              }
            });
        registerForContextMenu(mRegisteredCoursesListView);
      }
    }

    public void showHelp(MenuItem item) {
      Intent intent = new Intent(CourseManagerActivity.this,
          HelpActivity.class);
      Bundle bundle = new Bundle();
      bundle.putString("activity", "coursemanager");
      intent.putExtras(bundle);
      startActivity(intent);
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenuInfo menuInfo) {
      if (v.getId() == R.id.course_manager_list) {
        //AdapterView.AdapterContextMenuInfo info =
        (AdapterView.AdapterContextMenuInfo) menuInfo;
        String[] menuItems = getResources().getStringArray(
            R.array.course_manager_context_menu);
        for (int i = 0; i < menuItems.length; i++) {
          menu.add(Menu.NONE, i, i, menuItems[i]);
        }
      }
    }

    /**
     * Determines which MenuItem was selected and acts appropriately depending
     * on choice.
     */
    @Override
    public boolean onContextItemSelected(MenuItem item) {
```

```java
		AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo) 
		item
				.getMenuInfo();
		int menuItemIndex = item.getItemId();
		Log.d(TAG, "" + menuItemIndex);
		Course thisCourse = mRegisteredCoursesList.get(info.position);
		switch (menuItemIndex) {
		case 0:

			// Start Intent with Course as Extra
			Intent i = new Intent(CourseManagerActivity.this,
					ModifyCourseActivity.class);
			i.putExtra(CODE_COURSE_SUBJECT, thisCourse.mSubject);
			i.putExtra(CODE_COURSE_CODE, thisCourse.mCode);
			startActivity(i);
			break;
		case 1:
				Course c = mRegisteredCoursesList.get(info.position);
				mCourseHandle.removeCourse(c);
		}
		populateCoursesLayout();
		return true;
	}

	@Override
	protected void onActivityResult(int requestCode, int resultCode, Intent data) {
		if (resultCode == RESULT_OK) {
			switch (requestCode) {
			case (CODE_ADD_COURSE):
				// Course c = (Course)
				// data.getSerializableExtra(CODE_COURSE_OBJECT);
				// Toast.makeText(getApplicationContext(),
				// c.mSubject + " " + c.mCode + " added.",
				// Toast.LENGTH_LONG).show();
				break;
			default:
				break;
			}
		}
		if (resultCode == RESULT_OK && requestCode == CODE_COURSE_MODIFIED) {
			if (data.hasExtra("returnKey1")) {
				Toast.makeText(this, data.getExtras().getString("returnKey1"),
						Toast.LENGTH_SHORT).show();
			}
		}
	}

	/**
	 * Shows/hides verbose description of course.
	 * 
	 * @param view
	 *            The selected view to hide/show.
	 */
	private void showHideToolbar(View view, int position) {
		View toolbar = view.findViewById(R.id.course_manager_toolbar);

		// Get current course info.
		String subj = mRegisteredCoursesList.get(position).mSubject;
		String code = mRegisteredCoursesList.get(position).mCode;
		String offering = "";
		ArrayList<Offering> offs = mRegisteredCoursesList.get(position).mOfferings;
		ArrayList<OfferingTime> offTimes;

		// Creating the expand animation for the item
		ExpandAnimation expandAni = new ExpandAnimation(toolbar,
				ExpandAnimation.ANIMATE_SHORT);

		// Start the animation on the toolbar
		toolbar.startAnimation(expandAni);

		((TextView) view.findViewById(R.id.tv_prof_name))
```

```
          .setText(mRegisteredCoursesList.get(position).mInstructor);
      final String e= mRegisteredCoursesList.get(position).mInstructor_email;
      ((Button) view.findViewById(R.id.prof_email_button))
      .setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
          sendEmail(e);
        }
      });

      Log.d(TAG,"Number of Offerings for " + subj + " " + code + ": " + offs.size());

      // Add Offerings registered for.
      for (int i = 0; i < offs.size(); i++) {
        String what = offs.get(i).mType.substring(0, 3).trim();

        offTimes = offs.get(i).mOfferingTimes;

        Log.d(TAG, "Offering Type: " + what + ", # " + offTimes.size());

        offering = "";

        // Loop through OfferingTimes for each Offering to populate
        for (int j = 0; j < offTimes.size(); j++) {
          offering += offTimes.get(j).mDay + " "
              + offTimes.get(j).mStartTime + " - "
              + offTimes.get(j).mEndTime + " @ "
              + offTimes.get(j).mLocation + "\n";

          // Check for type of Offering and add as appropriate
          if (what.equalsIgnoreCase("lec")) {
            TextView tv = ((TextView) view
                .findViewById(R.id.tv_lecture));
            tv.setText(offering);
          }

          else if (what.equalsIgnoreCase("lab"))
            ((TextView) view.findViewById(R.id.tv_lab))
                .setText(offering);

          else if (what.equalsIgnoreCase("tut"))
            ((TextView) view.findViewById(R.id.tv_tutorial))
                .setText(offering);

          else if (what.equalsIgnoreCase("sem"))
            ((TextView) view.findViewById(R.id.tv_seminar))
                .setText(offering);
        }
      }

      float mark  = mCourseHandle.getMark(mRegisteredCoursesList.get(position));
      float base  = mCourseHandle.getBase(mRegisteredCoursesList.get(position));
      float total=0;
      if(base != 0){
        total= (mark/base)*100;
      }

      String grade= ""+(int)mark+"/"+(int)base+" = "+ (int)total+"%";
      ((TextView) view.findViewById(R.id.course_grade_grade))
      .setText(grade);

      /*
       * Hide class type if none available
       */
      if (((TextView) view.findViewById(R.id.tv_lecture)).getText()
          .toString().equalsIgnoreCase("none"))
        view.findViewById(R.id.row_lec).setVisibility(GONE);
      if (((TextView) view.findViewById(R.id.tv_lab)).getText()
          .toString().equalsIgnoreCase("none"))
        view.findViewById(R.id.row_lab).setVisibility(GONE);
      if (((TextView) view.findViewById(R.id.tv_tutorial)).getText()
          .toString().equalsIgnoreCase("none"))
        view.findViewById(R.id.row_tut).setVisibility(GONE);
      if (((TextView) view.findViewById(R.id.tv_seminar)).getText()
          .toString().equalsIgnoreCase("none"))
        view.findViewById(R.id.row_sem).setVisibility(GONE);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
      // Inflate the menu; this adds items to the action bar if it is present.
      getMenuInflater().inflate(R.menu.activity_coursemanager, menu);
      return true;
    }

    /**
     * A simple implementation of list adapter to populate ListView with
     * courses.
     */
    class CustomListAdapter extends ArrayAdapter<String> {

      public CustomListAdapter(Context context, int textViewResourceId) {
        super(context, textViewResourceId);
      }

      @Override
      public View getView(int position, View convertView, ViewGroup parent) {

        if (convertView == null) {
          convertView = getLayoutInflater().inflate(
              R.layout.course_list_item, null);
        }

        ((TextView) convertView.findViewById(R.id.course_list_item_title))
            .setText(getItem(position));

        // Resets the toolbar to be closed
        View toolbar = convertView
            .findViewById(R.id.course_manager_toolbar);
        ((LinearLayout.LayoutParams) toolbar.getLayoutParams()).bottomMargin = -50;
        toolbar.setVisibility(View.GONE);
        return convertView;
      }
    }

    /**
     * Launches AddCourseActivity as intent.
     *
     * @param menu
     *            MenuItem selected.
     */
    public void addCourse(MenuItem menu) {
      Intent i = new Intent(CourseManagerActivity.this,
          AddCourseActivity.class);
      startActivity(i);
    }

    /**
     * Allows user to manually fetch course calendar offerings from Registrar
     *
     * @param item
     */
    public void updateMaster(MenuItem item) {
      updateCourseDatabaseFromRegistrar();
    }

    private void sendEmail(String instr_email) {
      Intent i = new Intent(Intent.ACTION_SENDTO);
      i.setType("message/rfc822");
      i.setData(Uri.parse("mailto:"+instr_email));
      try {
        startActivity(Intent.createChooser(i, "Send mail..."));
      } catch (android.content.ActivityNotFoundException ex) {
        Toast.makeText(CourseManagerActivity.this,
            "There are no email clients installed.", Toast.LENGTH_SHORT)
            .show();
      }
    }

    /**
     * Updates the course calendar offerings master table. Is called at first
     * run (if table does not exist) and manually when user wishes to check for
     * updates. Progress bar to prevent hanging on main thread.
     */
    private void updateCourseDatabaseFromRegistrar() {
      final Handler handler = new Handler();
      final ProgressDialog progressDialog;

      TextView title = new TextView(CourseManagerActivity.this);
      title.setText(R.string.loading_courses_registrar);
      title.setGravity(Gravity.FILL);

      TextView msg = new TextView(CourseManagerActivity.this);
      msg.setText(R.string.loading_courses_registrar_msg);
      msg.setGravity(Gravity.FILL);

      progressDialog = ProgressDialog.show(this, "Course Timetable Update",
          "Updating course timetable from registrar. Please be patient.");

      Thread thread = new Thread() {
        public void run() {
          mCourseHandle.updateAll();
          // this will handle the post task.
          // it will run when the time consuming task get finished
          handler.post(new Runnable() {
            public void run() {

              // Update your UI or
              // do any Post job after the time consuming task
              // remember to dismiss the progress dialog here.
              // updateUI();
              progressDialog.dismiss();
              populateCoursesLayout();
            }
          });
        }
      };
      thread.start();
    }
  }

```

```java
/**
 * CurrentCoursesHandler.java
 * Brock Butler
 * Handles database table creation and queries for course information
 * Created by James Grisdale on 2013-02-24
 * Copyright (c) 2013 Sea Addicts. All rights reserved.
 **/

package edu.seaaddicts.brockbutler.coursemanager;

import java.util.ArrayList;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.DatabaseUtils;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import edu.seaaddicts.brockbutler.contacts.Contact;
import edu.seaaddicts.brockbutler.scheduler.Task;

public class CurrentCoursesHandler extends SQLiteOpenHelper {
  private static final int DATABASE_VERSION = 1;
  // Database Name
  private static final String DATABASE_NAME = "Database";
  // table names
  private static final String TABLE_COURSES = "courses";
  private static final String TABLE_TASKS = "tasks";
  private static final String TABLE_OFFERINGS = "offerings";
  private static final String TABLE_OFFERING_TIMES = "offering_times";
  private static final String TABLE_CONTACTS = "contacts";
  // field names
  private static final String KEY_SUBJ = "subj";
  private static final String KEY_CODE = "code";
  private static final String KEY_DESC = "desc";
  private static final String KEY_INSTRUCTOR = "instructor";
  private static final String KEY_ID = "id";
  private static final String KEY_TYPE = "type";
  private static final String KEY_SEC = "sec";
  private static final String KEY_DAY = "day";
  private static final String KEY_TIMES = "time_start";
  private static final String KEY_TIMEE = "time_end";
  private static final String KEY_LOCATION = "location";
  private static final String KEY_ASSIGN = "assign";
  private static final String KEY_NAME = "name";
  private static final String KEY_MARK = "mark";
  private static final String KEY_BASE = "base";
  private static final String KEY_WEIGHT = "weight";
  private static final String KEY_DUE = "due";
  private static final String KEY_CREATE_DATE = "create_date";
  private static final String KEY_IS_DONE = "is_done";
  private static final String KEY_CID = "cid";
  private static final String KEY_FNAME = "fname";
  private static final String KEY_LNAME = "lname";
  private static final String KEY_EMAIL = "email";
  private static final String KEY_PRIORITY = "priority";
  private static final String KEY_INSTREMAIL = "instructor_email";

  /* Constructor for the database helper */
  public CurrentCoursesHandler(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
  }

  /* Create tables for courses, tasks, offerings, offering times, and contacts
   * in the database if they do not exist when the database helper is first
   * called
   */
  @Override
  public void onCreate(SQLiteDatabase db) {
  }

```

```
    /* on an upgrade drop tables and recreate */
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
      // Drop older table if existed
      db.execSQL("DROP TABLE IF EXISTS " + TABLE_COURSES);
      db.execSQL("DROP TABLE IF EXISTS " + TABLE_TASKS);
      db.execSQL("DROP TABLE IF EXISTS " + TABLE_OFFERINGS);
      db.execSQL("DROP TABLE IF EXISTS " + TABLE_OFFERING_TIMES);
      db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS);
      // Create tables again
      onCreate(db);
    }

    /* addCourse - adds all information for a course to the database adding
     * course, offerings, tasks and contacts information, if information exists
     * for the course then an update is done, otherwise and insert is done
     * @param course - the course information to add to the course table
     */
    public void addCourse(Course course) {
      SQLiteDatabase db = this.getWritableDatabase();
      ContentValues values = new ContentValues();
      long num = 0;
      boolean update = false;
      //check if the course already exists in the table
      num = DatabaseUtils.queryNumEntries(db, TABLE_COURSES, KEY_SUBJ + " ='"
          + course.mSubject + "' AND " + KEY_CODE + "='" + course.mCode
          + "'");
      if (num > 0)//if it exists then do an update
        update = true;
      // values to be added to the table
      values.put(KEY_SUBJ, course.mSubject); // subject code
      values.put(KEY_CODE, course.mCode);
      values.put(KEY_DESC, course.mDesc);
      values.put(KEY_INSTRUCTOR, course.mInstructor);
      values.put(KEY_INSTREMAIL, course.mInstructor_email);
      // Inserting or updating Row
      if (update)
        db.update(TABLE_COURSES, values, KEY_SUBJ + " ='" + course.mSubject
            + "' AND " + KEY_CODE + "='" + course.mCode + "'", null);
      else
        db.insert(TABLE_COURSES, null, values);
      db.close(); // Closing database connection
      values.clear();
      addOfferings(course);//add the offerings for the course
      addTasks(course);//add the tasks for the course
      addContacts(course.mContacts);//add the contacts for the course
      db.close();//close the database
    }

    /* deleteCourse - removes all information for the given course from the
     * database
     * @param course - the course data to be removed from the course table
     */
    public void deleteCourse(Course course) {
      SQLiteDatabase db = this.getWritableDatabase();
      //delete the row for the selected course
      db.delete(TABLE_COURSES, KEY_SUBJ + "='" + course.mSubject + "' AND "
          + KEY_CODE + "='" + course.mCode + "'", null);
      db.close(); //close the db
      //delete all the offerings
      for (int i=0; i<course.mOfferings.size(); i++){
        deleteOffering(course.mOfferings.get(i));
      }
      //delete all the tasks
      for (int i=0; i<course.mTasks.size(); i++){
        removeTask(course.mTasks.get(i));
      }
    }

    /* getCourse - retrieves all information for the given course
     * @param subj - the course name
     * @param code - the course code
     */
    public Course getCourse(String subj, String code) {
      SQLiteDatabase db = this.getReadableDatabase();
      Course course = new Course();
      //query to retrieve all information for the given subj and code
      Cursor c = db.rawQuery("SELECT * FROM " + TABLE_COURSES + " where "
          + KEY_SUBJ + "= '" + subj + "' and " + KEY_CODE + " = '" + code
          + "'", null);
      if (c != null) {
        //start at the first record
        if (c.moveToFirst()) {
          do {
            //set values in the course object from the table
            course.mSubject = c.getString(c.getColumnIndex(KEY_SUBJ));
            course.mCode = c.getString(c.getColumnIndex(KEY_CODE));
            course.mDesc = c.getString(c.getColumnIndex(KEY_DESC));
            course.mInstructor = c.getString(c
                .getColumnIndex(KEY_INSTRUCTOR));
            course.mInstructor_email = c.getString(c
                .getColumnIndex(KEY_INSTREMAIL));
            course.mOfferings = getOfferings(course.mSubject,
                course.mCode);
            course.mTasks = getTasks(course);//get the tasks for the course
            course.mContacts = getContacts(course);//get the contacts for the course
          } while (c.moveToNext());
        }
      }
      c.close();//close the cursor
      db.close();//close the database
      return course;//return the course object
    }

     /* addOfferings - adds all offerings offered by a particular course as well
     * as their offering times
     * @param course - the course to add the offerings from
     */
    public void addOfferings(Course course) {
      Offering offering;
      OfferingTime offeringtime;
      ContentValues values = new ContentValues();
      SQLiteDatabase db = this.getWritableDatabase();
      long num = 0;
      boolean update = false;
      for (int i = 0; i < course.mOfferings.size(); i++) {
        offering = course.mOfferings.get(i);
        num = 0;
        update = false;
        //find if the offering already exists
        num = DatabaseUtils.queryNumEntries(db, TABLE_OFFERINGS, KEY_SUBJ
            + " ='" + offering.mSubj + "' AND " + KEY_CODE + "='"
            + offering.mCode + "' AND " + KEY_TYPE + "='"
            + offering.mType + "' AND " + KEY_SEC + "="
            + offering.mSection);
        if (num > 0)//if the offering exists then do an update
          update = true;
        //set the feilds and values
        values.put(KEY_SUBJ, course.mSubject);
        values.put(KEY_CODE, course.mCode);
        values.put(KEY_TYPE, offering.mType);
        values.put(KEY_SEC, offering.mSection);
        if (update)//update the offering information
          db.update(TABLE_OFFERINGS, values, KEY_SUBJ + " ='"
              + offering.mSubj + "' AND " + KEY_CODE + "='"
              + offering.mCode + "' AND " + KEY_TYPE + "='"
              + offering.mType + "' AND " + KEY_SEC + "="
              + offering.mSection, null);
        else//insert the offering information
          db.insert(TABLE_OFFERINGS, null, values);
        values.clear();
      }
```

```
        SQLiteDatabase rdb = this.getReadableDatabase();
        for (int i=0; i<course.mOfferings.size(); i++){
          offering = course.mOfferings.get(i);
          //now adding the offering times for each offering
          for (int j = 0; j < offering.mOfferingTimes.size(); j++) {
            offeringtime = offering.mOfferingTimes.get(j);
            num = 0;
            update = false;
            //see if the offering time exists
            num = DatabaseUtils.queryNumEntries(db, TABLE_OFFERING_TIMES,
                KEY_ID + " =" + offering.mId+ " AND "+KEY_DAY+"='"+offeringtime.mDay+"'");
            if (num > 1)//if exists then update
              update = true;
            //query for offering id for each offering time
            Cursor c = rdb.rawQuery("SELECT " + KEY_ID + " FROM "
                + TABLE_OFFERINGS + " WHERE " + KEY_SUBJ + "='"
                + offering.mSubj + "' AND " + KEY_CODE + "='"
                + offering.mCode + "' AND " + KEY_TYPE + "='"
                + offering.mType + "' AND " + KEY_SEC + "="
                + offering.mSection, null);
            c.moveToFirst();
            //set fields and values to be added
            offering.mId = c.getInt(c.getColumnIndex(KEY_ID));
            values.put(KEY_ID, offering.mId);
            values.put(KEY_DAY, offeringtime.mDay);
            values.put(KEY_TIMES, offeringtime.mStartTime);
            values.put(KEY_TIMEE, offeringtime.mEndTime);
            values.put(KEY_LOCATION, offeringtime.mLocation);
            if (update)//update the record
              db.update(TABLE_OFFERING_TIMES, values, KEY_ID + " ="
                  + offering.mId, null);
            else//insert the record
              db.insert(TABLE_OFFERING_TIMES, null, values);
            values.clear();
            c.close();//close the cursor
          }
        }
        rdb.close();//close database connection
        db.close();//close database connection
      }


      /* deleteOffering - removes all information from the databse for the given
       * offering
       * @param offering - the offering to be removed from the offerings table
       */
      public void deleteOffering(Offering offering) {
        int id;
        SQLiteDatabase rdb = this.getReadableDatabase();
        //query to get the id of the offering to be deleted
        Cursor c = rdb.rawQuery("SELECT " + KEY_ID + " FROM "
            + TABLE_OFFERINGS + " WHERE " + KEY_SUBJ + "='"
            + offering.mSubj + "' AND " + KEY_CODE + "='"
            + offering.mCode + "' AND " + KEY_TYPE + "='"
            + offering.mType + "' AND " + KEY_SEC + "="
            + offering.mSection, null);
        c.moveToFirst();
        id = c.getInt(c.getColumnIndex(KEY_ID));
        c.close();//close cursor
        SQLiteDatabase db = this.getWritableDatabase();
        //delete the offering times associated to the offering
        db.delete(TABLE_OFFERING_TIMES, KEY_ID +"="+id, null);
        //delete the offering from the offerings table
        db.delete(TABLE_OFFERINGS, KEY_SUBJ
            + " ='" + offering.mSubj + "' AND " + KEY_CODE + "='"
            + offering.mCode + "' AND " + KEY_TYPE + "='"
            + offering.mType + "' AND " + KEY_SEC + "="
            + offering.mSection, null);
        db.close();//close the database
      }

```

```
    /* addTasks - adds all tasks associated with a given course
     * @param course - the course object with the tasks to be added
     */
    public void addTasks(Course course) {
      Task task;
      ContentValues values = new ContentValues();
      SQLiteDatabase db = this.getWritableDatabase();
      long num = 0;
      boolean update = false;
      for (int i = 0; i < course.mTasks.size(); i++) {
        task = course.mTasks.get(i);
        num = 0;
        update = false;
        //see if the task exists already
        num = DatabaseUtils.queryNumEntries(db, TABLE_TASKS, KEY_ASSIGN
            + " ='" + task.mAssign + "' AND " + KEY_SUBJ + " ='"
            + task.mSubj + "' AND " + KEY_CODE + "='" + task.mCode
            + "'");
        if (num > 0)//if exists then update
          update = true;
        values.put(KEY_SUBJ, task.mSubj);
        values.put(KEY_CODE, task.mCode);
        //if the task number is not 0 then use that value
        try {
          if (task.mAssign != 0)
            values.put(KEY_ASSIGN, task.mAssign);
        } catch (NullPointerException e){}
        //set all values to be added
        values.put(KEY_NAME, task.mName);
        values.put(KEY_MARK, task.mMark);
        values.put(KEY_BASE, task.mBase);
        values.put(KEY_WEIGHT, task.mWeight);
        values.put(KEY_DUE, task.mDueDate);
        values.put(KEY_CREATE_DATE, task.mCreationDate);
        values.put(KEY_PRIORITY, task.mPriority);
        values.put(KEY_IS_DONE, task.mIsDone);
        if (update)//update the row
          db.update(TABLE_TASKS, values, KEY_ASSIGN + " ='" + task.mAssign
              + "' AND " + KEY_SUBJ + " ='" + task.mSubj + "' AND "
              + KEY_CODE + "='" + task.mCode + "'", null);
        else//insert the row
          db.insert(TABLE_TASKS, null, values);
        values.clear();
      }
      db.close();//close the database
    }

    /* addContacts - add contacts to the contacts table in the database for the
     * given list of contacts
     * @param contacts - the list of contacts to be added to the contacts table
     */
    public void addContacts(ArrayList<Contact> contacts) {
      Contact contact;
      ContentValues values = new ContentValues();
      SQLiteDatabase db = this.getWritableDatabase();
      long num = 0;
      boolean update = false;
      for (int j = 0; j < contacts.size(); j++) {
        contact = contacts.get(j);
        num = 0;
        update = false;
        //check if the contact exists
        num = DatabaseUtils.queryNumEntries(db, TABLE_CONTACTS, KEY_SUBJ
            + " ='" + contact.mSubj + "' AND " + KEY_CODE + "='"
            + contact.mCode + "' AND " + KEY_FNAME + "='" + contact.mFirstName
            + "' AND " + KEY_LNAME + "='"+ contact.mLastName+"'");
        if (num > 0)//if exits then update
          update = true;
        //set the fields and the values
        values.put(KEY_SUBJ, contact.mSubj);
        values.put(KEY_CODE, contact.mCode);
        values.put(KEY_FNAME, contact.mFirstName);
        values.put(KEY_LNAME, contact.mLastName);
        values.put(KEY_EMAIL, contact.mEmail);
        if (update)//update the record
          db.update(TABLE_CONTACTS, values, KEY_SUBJ + " ='"
              + contact.mSubj + "' AND " + KEY_CODE + "='"
              + contact.mCode + "' AND " + KEY_FNAME + "='"
              + contact.mFirstName + "' AND " + KEY_LNAME + "='"
              + contact.mLastName + "' AND " + KEY_EMAIL + "='"
              + contact.mEmail + "'", null);
        else//insert the record
          db.insert(TABLE_CONTACTS, null, values);
        values.clear();
      }
      db.close();//close the database
    }

    /* addTasks - adds a task for a certain course using the addTasks(course) method
     * @param task - the task to be added
     */
    public void addTasks(Task task) {
      Course course = new Course();
      course.mTasks.add(task);
      addTasks(course);//add the tasks for the course object
    }

    /* getOfferings - gets all offerings for a given subject and code
     * @param subj - the course subject
     * @param code - the course code
     */
    public ArrayList<Offering> getOfferings(String subj, String code) {
      ArrayList<Offering> offerings = new ArrayList<Offering>();
      ArrayList<OfferingTime> offtimes;
      Offering offering;
      OfferingTime otime;
      SQLiteDatabase db = this.getReadableDatabase();
      //get all offerings for the subj and code
      Cursor c = db.rawQuery("SELECT *  FROM " + TABLE_OFFERINGS + " WHERE "
          + KEY_SUBJ + "='" + subj + "' and " + KEY_CODE + "='" + code
          + "'", null);
      try {
        if (c != null) {
          if (c.moveToFirst()) {//start at the first record
            do {
              offering = new Offering();
              //add the data into a new offering object
              offering.mId = c.getInt(c.getColumnIndex(KEY_ID));
              offering.mSubj = c
                  .getString(c.getColumnIndex(KEY_SUBJ));
              offering.mCode = c
                  .getString(c.getColumnIndex(KEY_CODE));
              offering.mType = c
                  .getString(c.getColumnIndex(KEY_TYPE));
              offering.mSection = c.getInt(c.getColumnIndex(KEY_SEC));
              offerings.add(offering);
            }while (c.moveToNext());//get next record
            c.close();//close cursor
            //get all the offering times for each offering
            for (int i=0; i<offerings.size(); i++){
              offtimes = new ArrayList<OfferingTime>();
              //get the id for the offering
              Cursor o = db.rawQuery("SELECT *  FROM "
                  + TABLE_OFFERING_TIMES + " WHERE " + KEY_ID
                  + "=" + offerings.get(i).mId, null);
              if (o != null) {
                if (o.moveToFirst()) {//move to first offering time
                  do {
                    otime = new OfferingTime();
                    //insert data from table to OfferingTime oject
                    otime.mOid = o.getInt(o
                        .getColumnIndex(KEY_ID));
                    otime.mDay = o.getString(o
                        .getColumnIndex(KEY_DAY));
                    otime.mStartTime = o.getString(o
                        .getColumnIndex(KEY_TIMES));
                    otime.mEndTime = o.getString(o
                        .getColumnIndex(KEY_TIMEE));
                    otime.mLocation = o.getString(o
                        .getColumnIndex(KEY_LOCATION));
                    offtimes.add(otime);
                  } while (o.moveToNext());//get next time
                }
              }
              offerings.get(i).mOfferingTimes = offtimes;
              o.close();//close cursor
            }
          }
        }
        db.close();//close database
      } catch (Exception e) {}
      return offerings;//return the offerings
    }

    /* getTasks - gets all tasks a person may have from the database */
    public ArrayList<Task> getTasks() {
      ArrayList<Task> tasks = new ArrayList<Task>();
      SQLiteDatabase db = this.getReadableDatabase();
      Task task;
      //get all tasks from the tasks table
      Cursor c = db.rawQuery("SELECT * FROM " + TABLE_TASKS, null);
      if (c != null) {
        if (c.moveToFirst()) {//start at the first record
          do {
            task = new Task();
            //insert data from the table into a new task object
            task.mSubj = c.getString(c.getColumnIndex(KEY_SUBJ));
            task.mCode = c.getString(c.getColumnIndex(KEY_CODE));
            task.mAssign = c.getInt(c.getColumnIndex(KEY_ASSIGN));
            task.mName = c.getString(c.getColumnIndex(KEY_NAME));
            task.mMark = c.getInt(c.getColumnIndex(KEY_MARK));
            task.mBase = c.getInt(c.getColumnIndex(KEY_BASE));
            task.mWeight = c.getFloat(c.getColumnIndex(KEY_WEIGHT));
            task.mDueDate = c.getString(c.getColumnIndex(KEY_DUE));
            task.mIsDone = c.getInt(c.getColumnIndex(KEY_IS_DONE));
            task.mCreationDate = c.getString(c.getColumnIndex(KEY_CREATE_DATE));
            task.mPriority = c.getInt(c.getColumnIndex(KEY_PRIORITY));
            tasks.add(task);//add the task to the list
          } while (c.moveToNext());
        }
      }
      c.close();//close cursor
      db.close();//close database
      return tasks;//return the list of tasks
    }

    /* getTasks - gets all tasks for a particular course
     * @param course - the course to get the tasks for
     */
    private ArrayList<Task> getTasks(Course course) {
      ArrayList<Task> tasks = new ArrayList<Task>();
      SQLiteDatabase db = this.getReadableDatabase();
      Task task;
      // get all task information for the choosen course
      Cursor c = db.rawQuery("SELECT * FROM " + TABLE_TASKS + " WHERE "
          + KEY_SUBJ + "='" + course.mSubject + "' AND " + KEY_CODE
          + "='" + course.mCode + "'", null);
      if (c != null) {
        if (c.moveToFirst()) {//start at the first record
          do {
            task = new Task();
            //insert data from the table to a new task object
            task.mSubj = c.getString(c.getColumnIndex(KEY_SUBJ));
            task.mCode = c.getString(c.getColumnIndex(KEY_CODE));
            task.mAssign = c.getInt(c.getColumnIndex(KEY_ASSIGN));
            task.mName = c.getString(c.getColumnIndex(KEY_NAME));
            task.mMark = c.getInt(c.getColumnIndex(KEY_MARK));
            task.mBase = c.getInt(c.getColumnIndex(KEY_BASE));
            task.mWeight = c.getFloat(c.getColumnIndex(KEY_WEIGHT));
            task.mDueDate = c.getString(c.getColumnIndex(KEY_DUE));
            task.mIsDone = c.getInt(c.getColumnIndex(KEY_IS_DONE));
            task.mCreationDate = c.getString(c.getColumnIndex(KEY_CREATE_DATE));
            task.mPriority = c.getInt(c.getColumnIndex(KEY_PRIORITY));
            task.mContacts = getContacts(course);
            tasks.add(task);//add task to the list of tasks
          } while (c.moveToNext());//get next record
        }
      }
      c.close();//close cursor
      db.close();//close database
      return tasks;//return the list of tasks
    }

    /* getContacts - get all contacts for a specified course
     * @param course - the course object to get contacts for
     */
    private ArrayList<Contact> getContacts(Course course) {
      ArrayList<Contact> contacts = new ArrayList<Contact>();
      SQLiteDatabase db = this.getReadableDatabase();
      Contact contact;
      //get all contacts from the contacts table for the specified course
      Cursor c = db.rawQuery("SELECT * FROM " + TABLE_CONTACTS + " WHERE "
          + KEY_SUBJ + "='" + course.mSubject + "' AND " + KEY_CODE
          + "='" + course.mCode + "'", null);
      if (c != null) {
        if (c.moveToFirst()) {//get first record
          do {
            contact = new Contact();
            //insert data from the contacts table to a new contact object
            contact.mSubj = c.getString(c.getColumnIndex(KEY_SUBJ));
            contact.mCode = c.getString(c.getColumnIndex(KEY_CODE));
            contact.mId = c.getInt(c.getColumnIndex(KEY_CID));
            contact.mFirstName = c.getString(c
                .getColumnIndex(KEY_FNAME));
            contact.mLastName = c
                .getString(c.getColumnIndex(KEY_LNAME));
            contact.mEmail = c.getString(c.getColumnIndex(KEY_EMAIL));
            contacts.add(contact);//add contact to list of contacts
          } while (c.moveToNext());//get next record
        }
      }
      c.close(); //close cursor
      db.close(); //close database
      return contacts; //return list of contacts
    }

    /* removeTask - deletes a given task from the tasks table of the database
     * @param task - the task to be removed from the tasks table
     */
    public void removeTask(Task task) {
      SQLiteDatabase db = this.getWritableDatabase();
      //delete the record for the given task
      db.delete(TABLE_TASKS, KEY_SUBJ
          + " ='" + task.mSubj + "' AND " + KEY_CODE + "='"
          + task.mCode + "' AND " + KEY_ASSIGN + "="
          + task.mAssign, null);
      db.close();//close database
    }

    /* removeContact - deletes a contact from the contacts table from the database
     * @param contact - the contact information to be deleted from the table
     */
    public void removeContact(Contact contact){
      SQLiteDatabase db = this.getWritableDatabase();
      //delete the record associated with the contact id
      db.delete(TABLE_CONTACTS, KEY_CID +"="+contact.mId, null );
    }

    /* getRegCourses - gets all courses added to the courses table of the
     * database and all of it's components
     */
    public ArrayList<Course> getRegCourses() {
      ArrayList<Course> courses = new ArrayList<Course>();
      SQLiteDatabase db = this.getReadableDatabase();
      try {
        //get all courses from the course table
        Cursor c = db.rawQuery("SELECT * FROM " + TABLE_COURSES, null);
        if (c != null) {
          if (c.moveToFirst()) {//start at the first record
            do {
              //get course information for each course found in courses table
              courses.add(getCourse(
                  c.getString(c.getColumnIndex(KEY_SUBJ)),
                  c.getString(c.getColumnIndex(KEY_CODE))));
            } while (c.moveToNext());//get next record
          }
        }
        c.close();//close cursor
      } catch (Exception e) {}
      db.close();//close database
      return courses; //return list of current courses
    }

    /* Query - a general method to allow a query to be done that has not been
     * specified. it returns a cursor object to allow the data to be read
     * @param s - a query sent as a string to perform the query on the database
     */
    public Cursor Query(String s) {
      SQLiteDatabase db = this.getReadableDatabase();
      Cursor c = db.rawQuery(s, null);//perform query
      db.close();
      return c;//return cursor object
    }
  }

```

```java
/**
 * MasterCourse.java
 * Brock Butler
 * A wrapper class for course timetable information
 * Created by James Grisdale on 2013-02-24
 * Copyright (c) 2013 Sea Addicts. All rights reserved.
**/

package edu.seaaddicts.brockbutler.coursemanager;

public class MasterCourse {
  public String id;  //course id
  public String code;  //course code
  public String subj;  //faculty name
  public String desc;  //course description
  public String type;  //course type
  public String sec;  //section
  public String dur;  //duration
  public String days;  //days offered
  public String time;  //time offered
  public String location;  //location
  public String instructor;  //instructor
}

```

```java
package edu.seaaddicts.brockbutler.coursemanager;

import java.util.ArrayList;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.util.SparseBooleanArray;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.ListView;
import android.widget.TextView;
import edu.seaaddicts.brockbutler.R;
import edu.seaaddicts.brockbutler.contacts.Contact;
import edu.seaaddicts.brockbutler.scheduler.Task;

public class ModifyCourseActivity extends Activity {

  private static final String TAG = "ModifyCourseActivity";

  private static final int VISIBLE = 0;
  private static final int GONE = 8;

  ArrayList<String> mLecs;
  ArrayList<String> mLabs;
  ArrayList<String> mTuts;
  ArrayList<String> mSems;

  ArrayList<Offering> mLecsOfferings;
  ArrayList<Offering> mLabsOfferings;
  ArrayList<Offering> mTutsOfferings;
  ArrayList<Offering> mSemsOfferings;

  public String mSubject;
  public String mCode;
  public String mDesc;
  public String mInstructor;
  public String mInstructor_email;
  public ArrayList<Offering> mOfferings;
  public ArrayList<Task> mTasks;
  public ArrayList<Contact> mContacts;
  public Course course;
  public Course brock;

  private Button mSaveButton;
  private Button mCancelButton;

  private ListView mLecsListView;
  private ListView mSemsListView;
  private ListView mTutsListView;
  private ListView mLabsListView;

  private CurrentCoursesHandler mCourseHandle;

  private TextView mSubjectTextView;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_modify_course);
    Bundle bundle = this.getIntent().getExtras();

    brock = new CourseHandler(getApplicationContext()).getCourseOfferings(
        bundle.getString(CourseManagerActivity.CODE_COURSE_SUBJECT),
        bundle.getString(CourseManagerActivity.CODE_COURSE_CODE));
    mCourseHandle = new CurrentCoursesHandler(this.getApplicationContext());
    course = mCourseHandle.getCourse(bundle.getString(CourseManagerActivity.
    CODE_COURSE_SUBJECT),
        bundle.getString(CourseManagerActivity.CODE_COURSE_CODE));

    init();
  }

  /*
   * Initialize all views and sets Button OnClickListeners.
   */
  private void init() {
    // Set Buttons
    mSaveButton = (Button) findViewById(R.id.add_course_save_button);
    mCancelButton = (Button) findViewById(R.id.add_course_cancel_button);

    // Instantiate TextView
    mSubjectTextView = (TextView) findViewById(R.id.modify_course_subject_textview);

    // Set TextView
    mSubjectTextView.setText(course.mSubject+" "+course.mCode);

    mCode = course.mCode;
    mOfferings = brock.mOfferings;

    String s;
    mLecs = new ArrayList<String>();
    mLabs = new ArrayList<String>();
    mTuts = new ArrayList<String>();
    mSems = new ArrayList<String>();

    mLecsOfferings = new ArrayList<Offering>();
    mLabsOfferings = new ArrayList<Offering>();
    mTutsOfferings = new ArrayList<Offering>();
    mSemsOfferings = new ArrayList<Offering>();

    for (int i = 0; i < mOfferings.size(); i++) {
      s = mOfferings.get(i).mType;

      // Some offerings don't have any of what we are looking for,
      // so check length to make sure.
      if (s.length() > 2) {
        String ss = s.substring(0, 3).trim();
        if (ss.equalsIgnoreCase("lec")) {
          mLecs.add(s + ", SEC " + mOfferings.get(i).mSection);
          mLecsOfferings.add(mOfferings.get(i));
        } else if (ss.equalsIgnoreCase("lab")) {
          mLabs.add(s + ", SEC " + mOfferings.get(i).mSection);
          mLabsOfferings.add(mOfferings.get(i));
        } else if (ss.equalsIgnoreCase("tut")) {
          mTuts.add(s + ", SEC " + mOfferings.get(i).mSection);
          mTutsOfferings.add(mOfferings.get(i));
        } else if (ss.equalsIgnoreCase("sem")) {
          mSems.add(s + ", SEC " + mOfferings.get(i).mSection);
          mSemsOfferings.add(mOfferings.get(i));
        }
      }
    }

    // Check if offerings available, and add ListView to display if so.
    LinearLayout lec_lay = (LinearLayout) findViewById(R.id.layout_add_lecs);
    LinearLayout lab_lay = (LinearLayout) findViewById(R.id.layout_add_labs);
    LinearLayout tut_lay = (LinearLayout) findViewById(R.id.layout_add_tuts);
    LinearLayout sem_lay = (LinearLayout) findViewById(R.id.layout_add_sems);

    if (mLecs.size() > 0) {
      lec_lay.setVisibility(VISIBLE);
      mLecsListView = (ListView) findViewById(R.id.add_course_add_lecs);
      mLecsListView.setAdapter(new ArrayAdapter<String>(
          getApplicationContext(),
          android.R.layout.simple_list_item_multiple_choice,
          mLecs));
    } else {
      lec_lay.setVisibility(GONE);
```

```java
        }
        if (mLabs.size() > 0) {
          lab_lay.setVisibility(VISIBLE);
          mLabsListView = (ListView) findViewById(R.id.add_course_add_labs);
          mLabsListView.setAdapter(new ArrayAdapter<String>(
              getApplicationContext(),
              android.R.layout.simple_list_item_multiple_choice,
              mLabs));
        } else {
          lab_lay.setVisibility(GONE);
        }
        if (mTuts.size() > 0) {
          tut_lay.setVisibility(VISIBLE);
          mTutsListView = (ListView) findViewById(R.id.add_course_add_tuts);
          mTutsListView.setAdapter(new ArrayAdapter<String>(
              getApplicationContext(),
              android.R.layout.simple_list_item_multiple_choice,
              mTuts));
        } else {
          tut_lay.setVisibility(GONE);
        }
        if (mSemsOfferings.size() > 0) {
          sem_lay.setVisibility(VISIBLE);
          mSemsListView = (ListView) findViewById(R.id.add_course_add_sems);
          mSemsListView.setAdapter(new ArrayAdapter<String>(
              getApplicationContext(),
              android.R.layout.simple_list_item_multiple_choice,
              mSems));
        } else {
          sem_lay.setVisibility(GONE);
        }

        // Set OnClickListener
        mSaveButton.setOnClickListener(new OnClickListener() {
          public void onClick(View v) {
            Course c = new Course();
            c.mSubject          = course.mSubject;
            c.mCode             = course.mCode;
            c.mDesc             = course.mDesc;
            c.mInstructor       = course.mInstructor;
            c.mInstructor_email = course.mInstructor_email;
            for (Task t : course.mTasks)
              c.mTasks.add(t);

            for (Offering o : course.mOfferings)
              mCourseHandle.deleteOffering(o);

            SparseBooleanArray sba1, sba2, sba3, sba4;
            if (mLecsListView != null) {
              sba1 = mLecsListView.getCheckedItemPositions();
              for (int i = 0; i < mLecsListView.getCount(); i++) {
                if (sba1.get(i) == true) {
                  c.mOfferings.add(mLecsOfferings.get(i));
                  Log.d(TAG, "Added: " + mLecsOfferings.get(i).mSubj
                      + " " + mOfferings.get(i).mCode + ", Type "
                      + mOfferings.get(i).mType + ", Section "
                      + mOfferings.get(i).mSection
                      + " to Offerings");
                }
              }
            }

            if (mLabsListView != null) {
              sba2 = mLabsListView.getCheckedItemPositions();

              for (int i = 0; i < mLabsListView.getCount(); i++) {
                if (sba2.get(i) == true) {
                  c.mOfferings.add(mLabsOfferings.get(i));
                  Log.d(TAG, "Added: " + mLabsOfferings.get(i).mSubj
                      + " " + mOfferings.get(i).mCode + ", Type "
                      + mOfferings.get(i).mType + ", Section "
```

```
                        + mOfferings.get(i).mSection
                        + " to Offerings");
                }
              }
            }

            if (mTutsListView != null) {
              sba3 = mTutsListView.getCheckedItemPositions();

              for (int i = 0; i < mTutsListView.getCount(); i++) {
                if (sba3.get(i) == true) {
                  c.mOfferings.add(mTutsOfferings.get(i));
                  Log.d(TAG, "Added: " + mTutsOfferings.get(i).mSubj
                      + " " + mOfferings.get(i).mCode + ", Type "
                      + mOfferings.get(i).mType + ", Section "
                      + mOfferings.get(i).mSection
                      + " to Offerings");
                }
              }
            }

            if (mSemsListView != null) {
              sba4 = mSemsListView.getCheckedItemPositions();

              for (int i = 0; i < mSemsListView.getCount(); i++) {
                if (sba4.get(i) == true) {
                  c.mOfferings.add(mSemsOfferings.get(i));
                  Log.d(TAG, "Added: " + mSemsOfferings.get(i).mSubj
                      + " " + mOfferings.get(i).mCode + ", Type "
                      + mOfferings.get(i).mType + ", Section "
                      + mOfferings.get(i).mSection
                      + " to Offerings");
                }
              }
            }
            if (c.mSubject != null) {
              try {
                mCourseHandle.addCourse(c);
                onBackPressed();
              } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
              }
            }
          }
        });

        mCancelButton.setOnClickListener(new OnClickListener() {
          public void onClick(View v) {
            // Do nothing.
            onBackPressed();
          }
        });
      }
    }

```

```java
/**
 * Offering.java
 * Brock Butler
 * A wrapper class for Offering information
 * Created by James Grisdale on 2013-02-24
 * Copyright (c) 2013 Sea Addicts. All rights reserved.
**/

package edu.seaaddicts.brockbutler.coursemanager;

import java.util.ArrayList;

public class Offering {
  public int mId;     //offering id
  public String mSubj;//faculty name
  public String mCode;//course code
  public int mSection;//section
  public String mType;//type
  public ArrayList<OfferingTime> mOfferingTimes;//list of times offered
  public int mOid; //extra id if needed

  //constructor - initializes arraylist of offering times
  public Offering(){
    mOfferingTimes = new ArrayList<OfferingTime>();
  }
}
```

```java
/**
 * OfferingTime.java
 * Brock Butler
 * A wrapper class for OfferingTime information
 * Created by James Grisdale on 2013-02-24
 * Copyright (c) 2013 Sea Addicts. All rights reserved.
**/

package edu.seaaddicts.brockbutler.coursemanager;

public class OfferingTime {
  public int mOid;  //id associated with offering
  public String mStartTime; //start time of offering
  public String mEndTime; //offering end time
  public String mDay;  //day available
  public String mLocation; //location
}

```

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="10sp"
    android:orientation="vertical"
    android:scrollbars="vertical"
    tools:context=".AddTaskActivity" >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <TextView
            android:layout_width="85sp"
            android:layout_height="wrap_content"
            android:text="@string/add_course_subject" />

        <Spinner
            android:id="@+id/add_course_subjects_spinner"
            android:layout_width="180sp"
            android:layout_height="40sp"
            android:layout_marginLeft="25sp" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <TextView
            android:layout_width="85sp"
            android:layout_height="wrap_content"
            android:text="@string/add_course_code" />

        <Spinner
            android:id="@+id/add_course_codes_spinner"
            android:layout_width="180sp"
            android:layout_height="40sp"
            android:layout_marginLeft="25sp" />
    </LinearLayout>

    <LinearLayout
        android:id="@+id/layout_add_lecs"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20sp"
        android:orientation="horizontal"
        android:visibility="gone" >

        <TextView
            android:layout_width="85sp"
            android:layout_height="wrap_content"
            android:text="@string/add_course_add_lecs" />

        <ListView
            android:id="@+id/add_course_add_lecs"
            android:layout_width="fill_parent"
            android:layout_height="100sp"
            android:layout_marginLeft="25sp"
            android:choiceMode="singleChoice" />
    </LinearLayout>

    <LinearLayout
        android:id="@+id/layout_add_labs"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20sp"
        android:orientation="horizontal"
        android:visibility="gone" >
```

```

            <TextView
                android:layout_width="85sp"
                android:layout_height="wrap_content"
                android:text="@string/add_course_add_labs" />

            <ListView
                android:id="@+id/add_course_add_labs"
                android:layout_width="fill_parent"
                android:layout_height="100sp"
                android:layout_marginLeft="25sp"
                android:choiceMode="singleChoice"
                android:text="@string/add_course_add_labs" />
        </LinearLayout>

        <LinearLayout
            android:id="@+id/layout_add_tuts"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20sp"
            android:orientation="horizontal"
            android:visibility="gone" >

            <TextView
                android:layout_width="85sp"
                android:layout_height="wrap_content"
                android:text="@string/add_course_add_tuts" />

            <ListView
                android:id="@+id/add_course_add_tuts"
                android:layout_width="fill_parent"
                android:layout_height="100sp"
                android:layout_marginLeft="25sp"
                android:choiceMode="singleChoice"
                android:text="@string/add_course_add_tuts" />
        </LinearLayout>

        <LinearLayout
            android:id="@+id/layout_add_sems"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20sp"
            android:orientation="horizontal"
            android:visibility="gone" >

            <TextView
                android:layout_width="85sp"
                android:layout_height="wrap_content"
                android:text="@string/add_course_add_sems" />

            <ListView
                android:id="@+id/add_course_add_sems"
                android:layout_width="fill_parent"
                android:layout_height="100sp"
                android:layout_marginLeft="25sp"
                android:choiceMode="singleChoice"
                android:text="@string/add_course_add_sems" />
        </LinearLayout>

        <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:gravity="center_horizontal"
            android:orientation="horizontal"
            android:layout_marginTop="10sp" >

            <Button
                android:id="@+id/add_course_save_button"
                android:layout_width="100sp"
                android:layout_height="wrap_content"
                android:text="@string/button_save" />

            <Button
                android:id="@+id/add_course_cancel_button"
                android:layout_width="100sp"
                android:layout_height="wrap_content"
                android:layout_marginLeft="20sp"
                android:text="@string/button_cancel" />
        </LinearLayout>

    </LinearLayout>
```

```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <item
        android:id="@+id/add_course"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:onClick="addCourse"
        android:title="@string/add_course_add"/>

    <item
        android:id="@+id/update_master_list"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:onClick="updateMaster"
        android:title="@string/add_course_update_master_list"/>
    <item
        android:id="@+id/menu_show_coursemanager_help"
         android:orderInCategory="100"
         android:showAsAction="never"
        android:onClick="showHelp"
        android:title="@string/menu_show_coursemanager_help"/>

</menu>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/course_manager_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <ListView
        android:id="@+id/course_manager_list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:visibility="gone"
        android:divider="@drawable/grad_course_title" >
    </ListView>

    <TextView
        android:id="@+id/tv_no_courses"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="@string/no_courses"
        android:textColor="#ddd"
        android:textStyle="italic"
        android:textSize="20sp"
        android:gravity="center"
        android:visibility="gone" />
</LinearLayout>
```

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="10sp"
    android:orientation="vertical"
    android:scrollbars="vertical"
    tools:context=".ModifyTaskActivity" >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <TextView
            android:id="@+id/modify_course_subject_textview"
            android:layout_width="85sp"
            android:layout_height="wrap_content"
            android:text="@string/add_course_subject" />

    </LinearLayout>

    <LinearLayout
        android:id="@+id/layout_add_lecs"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20sp"
        android:orientation="horizontal"
        android:visibility="gone" >

        <TextView
            android:layout_width="85sp"
            android:layout_height="wrap_content"
            android:text="@string/add_course_add_lecs" />

        <ListView
            android:id="@+id/add_course_add_lecs"
            android:layout_width="fill_parent"
            android:layout_height="100sp"
            android:layout_marginLeft="25sp"
            android:choiceMode="singleChoice" />
    </LinearLayout>

    <LinearLayout
        android:id="@+id/layout_add_labs"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20sp"
        android:orientation="horizontal"
        android:visibility="gone" >

        <TextView
            android:layout_width="85sp"
            android:layout_height="wrap_content"
            android:text="@string/add_course_add_labs" />

        <ListView
            android:id="@+id/add_course_add_labs"
            android:layout_width="fill_parent"
            android:layout_height="100sp"
            android:layout_marginLeft="25sp"
            android:choiceMode="singleChoice"
            android:text="@string/add_course_add_labs" />
    </LinearLayout>

    <LinearLayout
        android:id="@+id/layout_add_tuts"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20sp"
        android:orientation="horizontal"
```

```xml
            android:visibility="gone" >

            <TextView
                android:layout_width="85sp"
                android:layout_height="wrap_content"
                android:text="@string/add_course_add_tuts" />

            <ListView
                android:id="@+id/add_course_add_tuts"
                android:layout_width="fill_parent"
                android:layout_height="100sp"
                android:layout_marginLeft="25sp"
                android:choiceMode="singleChoice"
                android:text="@string/add_course_add_tuts" />
        </LinearLayout>

        <LinearLayout
            android:id="@+id/layout_add_sems"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20sp"
            android:orientation="horizontal"
            android:visibility="gone" >

            <TextView
                android:layout_width="85sp"
                android:layout_height="wrap_content"
                android:text="@string/add_course_add_sems" />

            <ListView
                android:id="@+id/add_course_add_sems"
                android:layout_width="fill_parent"
                android:layout_height="100sp"
                android:layout_marginLeft="25sp"
                android:choiceMode="singleChoice"
                android:text="@string/add_course_add_sems" />
        </LinearLayout>

        <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:gravity="center_horizontal"
            android:orientation="horizontal"
            android:layout_marginTop="10sp" >

            <Button
                android:id="@+id/add_course_save_button"
                android:layout_width="100sp"
                android:layout_height="wrap_content"
                android:text="@string/button_save" />

            <Button
                android:id="@+id/add_course_cancel_button"
                android:layout_width="100sp"
                android:layout_height="wrap_content"
                android:layout_marginLeft="20sp"
                android:text="@string/button_cancel" />
        </LinearLayout>

    </LinearLayout>
```