```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) {
  }

```

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

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

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

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

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

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

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

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