```java
/**
 * 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
```

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

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

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