```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";
```

```java
    //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,"
```

```java
            + 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) {}
    }

    /* 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
    }
  }
```