```java
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.Writer;
import java.text.ParseException;
import java.util.Scanner;

/************************************************************
* Dalton Nofs                                               *
* Login ID: nofs5491                                        *
* CS-102, Summer 2017                                       *
* Programming Assignment 5                                  *
* Database class:  storage location of all classes imported  *
************************************************************/
public class Database
{
    LinkedList<Term> courseList = new LinkedList<Term>(); // Storage location for data read from file

    /************************************************************
    * Method: loadDatabase()                                    *
    * Purpose: Read file and store data into courseArray        *
    *                                                              *
    * Parameters:                                               *
    *            String args:     Pram's passed at program start  *
    *                               which notes file location     *
    * Returns: Void:             nothing will be returned       *
    ************************************************************/
    public void loadDatabase(String[] args) throws ArrayIndexOutOfBoundsException, IllegalArgumentException
    {
        Scanner file = null; // File scanner

        try
        {
            file = new Scanner(new File(args[0]));
        }
        catch(ArrayIndexOutOfBoundsException exc)
        {
            UserInterface.sendWarning("No arguments given! Terminating program!",
                    "ERROR");
            System.exit(1);
        }
        catch(FileNotFoundException exc)
        {
            UserInterface.sendWarning(
                    "File could not be opened. Terminating program!","ERROR");
            System.exit(2);
        }

        while(file.hasNext())
        {
            String fileLine = file.nextLine();      // Line read from file
            Scanner pieces = new Scanner(fileLine); // Split the line
            String dateString = "";                 // Temp string for separating the year and semester
            Course tempCourse = new Course();         // Temp course obj for storing imported data until obj
                                                    //    is added to the array

            // setTermTaken, and setExcludeFlag will throw a parse error
            //     if data sent is not in the correct format
            try
            {
                pieces.useDelimiter("/");
                dateString = pieces.next();
                if(dateString.length() != 6)
                {
                    throw new ParseException("Year/Term is wrong length", courseList.size()+1);
                }
                tempCourse.setYearTaken(dateString.substring(0, 4)); // Set year to string char's 0-4 (year)
                tempCourse.setTermTaken(dateString.substring(4, 6)); // Set term taken to the 2 digit semester code
                tempCourse.setCourseNumber(pieces.next());             // Set the course number
                tempCourse.setCreditCount(pieces.nextInt());         // Set the number of credits the class is worth
                tempCourse.setCourseTitle(pieces.next());             // Set the course title
                tempCourse.setCourseGrade(pieces.next());             // Set the course grade
                tempCourse.setExcludeFlag(pieces.next());             // Set the exclude flag
                // Add the new course to the database
                this.addCourse(tempCourse);
            }
```

```
81              catch(ParseException exc)
82              {
83                  UserInterface.sendWarning(exc.getMessage() + "\nError occured on line: "
84                          + fileLine + " : Line is being ignored and not added to array.",
85                          "ERROR");
86              }
87          }
88          if(courseList.isEmpty())
89          {
90              throw new IllegalArgumentException("The database is empty!");
91          }
92          else {/* do nothing */}
93      }
94
95      /************************************************************
96      * Method: storeDatabase()                                    *
97      * Purpose: Read file and store data into courseArray         *
98      *                                                              *
99      * Parameters:                                                *
100     *          String fileName:     stores database to file      *
101     * Returns: Void:             N/A                             *
102     ************************************************************/
103     public void storeDatabase(String fileName)
104     {
105         String buffer = ""; // buffer for file write
106         // local arraycount for cycling through the database
107         int arrayCount = this.getDatabaseSize();
108
109         // Check the status of the database
110         if(arrayCount <= 0)
111         {
112             throw new IllegalArgumentException("Database is empty!");
113         }
114
115         // Loop semesters
116         for(int index=0; index<arrayCount; index++)
117         {
118             buffer += gatherPrint(this.get(index).getRoot());
119         }
120         try
121         {
122             // configure the writer for writing to file
123             Writer writer = new BufferedWriter(new OutputStreamWriter(
124                     new FileOutputStream(fileName), "utf-8"));
125             writer.write(buffer); // try writing to the file
126             writer.close(); // Close the file
127         }
128         catch(IOException exc)
129         {
130             UserInterface.sendWarning("There was an error writing to the file!",
131                     "ERROR");
132         }
133         buffer = ""; // clear the buffer
134         UserInterface.sendMessage("Save successfull!","INFO");
135     }
136
137     /************************************************************
138     * Method: addCourse()                                        *
139     * Purpose: manually add a course to the database             *
140     *                                                              *
141     * Parameters:                                                *
142     *    Course: newCourse:     course to be added                  *
143     * Returns: Void:          nothing to be returned
144     ************************************************************/
145     public void addCourse(Course newCourse)
146     {
147         int insertIndex; // index for course insertion
148         if(courseList.isEmpty())
149         {
150             // Add new course to lower layer
151             Term lowerList = new Term(newCourse.getTermTakenRaw());
152             try
153             {
154             lowerList.add(newCourse);
155             }
156             catch(IOException exc)
157             {
158                 UserInterface.sendWarning("Course already exists!", "ERROR");
159             }
160             // Add new linkedList to upper layer
161             courseList.add(0, lowerList);
```

```
162         }
163         else
164         {
165             //Check to see if term exists, if exists then index is returned
166             insertIndex = addTerm(newCourse);
167             // add new course to lower layer
168             Term lowerList = courseList.get(insertIndex);
169             // Add the course to list
170             addCourse(lowerList, newCourse);
171         }
172     }
173 
174     /*************************************************************
175     * Method: getDatabaseSize()                                  *
176     * Purpose: return the size of the database                  *
177     *                                                            *
178     * Parameters:               N/A                             *
179     * Returns: int:             the size of the database        *
180     *************************************************************/
181     public int getDatabaseSize()
182     {
183         return courseList.size();
184     }
185 
186     /*************************************************************
187     * Method: checkIfTermExists()                                *
188     * Purpose: add course to low level tree                 *
189     *                                                            *
190     * Parameters: LinkedList, Course:        lowerList courseIn   *
191     * Returns: void:            N/A                              *
192     *************************************************************/
193     private void addCourse(Term lowerList, Course courseIn)
194     {
195         try
196         {
197             lowerList.add(courseIn);
198         }
199         catch(IOException exc)
200         {
201             UserInterface.sendWarning("Course already exists!","ERROR");
202         }
203     }
204 
205     /*************************************************************
206     * Method: checkIfTermExists()                                *
207     * Purpose: add course to low level linkedlist                *
208     *                                                            *
209     * Parameters: Course:       courseIn                         *
210     * Returns: int:            index of added term               *
211     *************************************************************/
212     private int addTerm(Course courseIn)
213     {
214         for(int index=0;index<courseList.size();index++)
215         {
216             // Create the 201704 string
217             String tempCourse = courseIn.getTermTakenRaw();
218             if(tempCourse.equals(courseList.get(index).getTerm()))
219             {
220                 /* term already exists so just return index */
221                 return index;
222             }
223             else if ((courseList.get(index).getTerm().compareToIgnoreCase(tempCourse) > 0))
224             {
225                 courseList.add(index, new Term(courseIn.getTermTakenRaw()));
226                 return index;
227             }
228         }
229         // small then all add to end
230         courseList.addLast(new Term(courseIn.getTermTakenRaw()));
231         return (courseList.size()-1);
232     }
233 
234     /*************************************************************
235     * Method: get()                                              *
236     * Purpose: get list position                                *
237     *                                                            *
238     * Parameters: int:            index                         *
239     * Returns: Object:          item stored at index            *
240     *************************************************************/
241     public Term get(int index)
242     {
```

```
243         return(courseList.get(index));
244     }
245 
246     /*************************************************************
247     * Method: remove()                                           *
248     * Purpose: remove a course from database                      *
249     *                                                             *
250     * Parameters: int: Course:    index, obj to be removed       *
251     * Returns: void:            N/A                               *
252     *************************************************************/
253     public void remove(int index, Course target)
254     {
255         try
256         {
257             Term lowerList = courseList.get(index);
258             lowerList.remove(target);
259             if(lowerList.isEmpty())
260                 courseList.remove(index);
261         }
262         catch(IndexOutOfBoundsException exc)
263         { UserInterface.sendWarning("Index is out of bounds!","ERROR");}
264     }
265 
266     /*************************************************************
267     * Method: removeAll()                                          *
268     * Purpose: remove a course from database                      *
269     *                                                             *
270     * Parameters: int: Course:    index, obj to be removed       *
271     * Returns: void:            N/A                               *
272     *************************************************************/
273     public void removeAll()
274     {
275         // de link the old list
276         courseList.removeAll();
277         courseList = new LinkedList<Term>();
278     }
279 
280     /*************************************************************
281     * Method: gatherPrint()    *private*                           *
282     * Purpose: fills buffer with tree info                       *
283     *                                                             *
284     * Parameters: TreeNode:        current node                    *
285     * Returns: String:             compiled buffer from tree       *
286     *************************************************************/
287     private String gatherPrint(TreeNode<Course> current)
288     {
289         String buffer = "";
290         if(current == null) {return buffer;} // if fallen off list
291         // follow format yyyytt/cs-num/cred/title/grade/include
292         buffer += current.getDatum().getTermTakenRaw() + "/" +
293                   current.getDatum().getCourseNumber() + "/" +
294                   current.getDatum().getCreditCount()  + "/" +
295                   current.getDatum().getCourseTitle()  + "/" +
296                   current.getDatum().getCourseGrade()  + "/" +
297                   current.getDatum().getExcludeFlag()  + "\r\n";
298                         // for return in file use \r\n
299         // gather the rest of the left till null
300         buffer += gatherPrint(current.getRight());
301         // gather the rest of the right till null
302         buffer += gatherPrint(current.getLeft());
303         return buffer;
304     }
305 }
```