```
1  import java.io.BufferedWriter;
2  import java.io.File;
3  import java.io.FileNotFoundException;
4  import java.io.FileOutputStream;
5  import java.io.FileWriter;
6  import java.io.IOException;
7  import java.io.OutputStreamWriter;
8  import java.io.Writer;
9  import java.text.ParseException;
10 import java.util.Scanner;
11
12 /************************************************************
13 * Dalton Nofs                                               *
14 * Login ID: nofs5491                                        *
15 * CS-102, Summer 2017                                       *
16 * Programming Assignment 4                                  *
17 * Database class:  storage location of all classes imported  *
18 ************************************************************/
19 public class Database
20 {
21     LinkedList<Term> courseList = new LinkedList<Term>(); // Storage location for data read from file
22
23     /************************************************************
24     * Method: loadDatabase()                                    *
25     * Purpose: Read file and store data into courseArray        *
26     *                                                              *
27     * Parameters:                                               *
28     *            String args:     Pram's passed at program start  *
29     *                               which notes file location     *
30     * Returns: Void:             nothing will be returned       *
31     ************************************************************/
32     public void loadDatabase(String[] args) throws ArrayIndexOutOfBoundsException, IllegalArgumentException
33     {
34         Scanner file = null; // File scanner
35
36         try
37         {
38             file = new Scanner(new File(args[0]));
39         }
40         catch(ArrayIndexOutOfBoundsException exc)
41         {
42             System.out.println("No arguments given! Terminating program!");
43             System.exit(1);
44         }
45         catch(FileNotFoundException exc)
46         {
47             System.out.println("File could not be opened. Terminating program!");
48             System.exit(2);
49         }
50
51         while(file.hasNext())
52         {
53             String fileLine = file.nextLine();      // Line read from file
54             Scanner pieces = new Scanner(fileLine); // Split the line
55             String dateString = "";                 // Temp string for separating the year and semester
56             Course tempCourse = new Course();         // Temp course obj for storing imported data until obj
57                                                     //    is added to the array
58
59             // setTermTaken, and setExcludeFlag will throw a parse error
60             //     if data sent is not in the correct format
61             try
62             {
63                 pieces.useDelimiter("/");
64                 dateString = pieces.next();
65                 if(dateString.length() != 6)
66                 {
67                     throw new ParseException("Year/Term is wrong length", courseList.size()+1);
68                 }
69                 tempCourse.setYearTaken(dateString.substring(0, 4)); // Set year to string char's 0-4 (year)
70                 tempCourse.setTermTaken(dateString.substring(4, 6)); // Set term taken to the 2 digit semester code
71                 tempCourse.setCourseNumber(pieces.next());             // Set the course number
72                 tempCourse.setCreditCount(pieces.nextInt());         // Set the number of credits the class is worth
73                 tempCourse.setCourseTitle(pieces.next());             // Set the course title
74                 tempCourse.setCourseGrade(pieces.next());             // Set the course grade
75                 tempCourse.setExcludeFlag(pieces.next());             // Set the exclude flag
76                 // Add the new course to the database
77                 this.addCourse(tempCourse);
78             }
79             catch(ParseException exc)
80             {
```

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

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

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