```java
import java.io.File;
import java.io.FileNotFoundException;
import java.text.ParseException;
import java.util.LinkedList;
import java.util.Scanner;

/************************************************************
* Dalton Nofs                                               *
* Login ID: nofs5491                                        *
* CS-102, Summer 2017                                       *
* Programming Assignment 3                                  *
* 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)
        {
            System.out.println("No arguments given!");
            System.exit(1);
        }
        catch(FileNotFoundException exc)
        {
            System.out.println("File could not be opened.");
            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
            }
            catch(ParseException exc)
            {
                System.out.println(exc.getMessage() + "\nError occured on line: " + fileLine +
                                   " Line is being ignored and not added to array.");
            }
            // Add the new course to the database
            this.addCourse(tempCourse);
        }
        if(courseList.isEmpty())
```

```
81          {
82              throw new IllegalArgumentException("The database is empty!");
83          }
84          else {/* do nothing */}
85      }
86
87      /*************************************************************
88      * Method: addCourse()                                        *
89      * Purpose: manually add a course to the database             *
90      *                                                             *
91      * Parameters:                                                *
92      *    Course: newCourse:     course to be added                    *
93      * Returns: Void:          nothing to be returned
94       * @throws Exception *
95      *************************************************************/
96      public void addCourse(Course newCourse)
97      {
98          int insertIndex; // index for course insertion
99          if(courseList.isEmpty())
100         {
101             // Add new course to lower layer
102             Term lowerList = new Term(newCourse.getYearTaken()+
103                     newCourse.getTermTakenRaw());
104             lowerList.append(newCourse);
105             // Add new linkedList to upper layer
106             courseList.add(0, lowerList);
107         }
108         else
109         {
110             //Check to see if term exists, if exists then index is returned
111             insertIndex = addTerm(newCourse);
112             // add new course to lower layer
113             Term lowerList = courseList.get(insertIndex);
114             // Add the course to list
115             addCourse(lowerList, newCourse);
116         }
117     }
118
119     /*************************************************************
120     * Method: getArrayPosition()                                 *
121     * Purpose: Return data a array position requested             *
122     *                                                             *
123     * Parameters:                                                *
124     *    String position:     index to return                        *
125     * Returns: Course:  data stored in the position requested    *
126     *************************************************************/
127     public Course getArrayPosition(int position, int secondIndex) throws ArrayIndexOutOfBoundsException
128     {
129         if(position <= courseList.size())
130         {
131             Term returnCourse = courseList.get(position);
132             return returnCourse.get(secondIndex);
133         }
134         // Throw an error that tells asker that they picked an invalid array position
135         else{throw new ArrayIndexOutOfBoundsException();}
136     }
137
138     /*************************************************************
139     * Method: getArraySize()                                     *
140     * Purpose: Read file and store data into courseArray         *
141     *                                                             *
142     * Parameters:               N/A                              *
143     * Returns: int:             the size of the array            *
144     *************************************************************/
145     public int getArraySize()
146     {
147         return courseList.size();
148     }
149
150     /*************************************************************
151     * Method: getArrayCount()                                    *
152     * Purpose: Get the size of course List                       *
153     *                                                             *
154     * Parameters: int:           index                           *
155     * Returns: int:            number of elements in list       *
156     *************************************************************/
157     public int getArrayCount(int index)
158     {
159         return (courseList.get(index).size());
160     }
161
```

```
162     /*************************************************************
163     * Method: checkIfTermExists()                                *
164     * Purpose: add course to low level linkedlist                 *
165     *                                                              *
166     * Parameters: LinkedList, Course:        lowerList courseIn    *
167     * Returns: void:             N/A                               *
168     *************************************************************/
169     private void addCourse(Term lowerList, Course courseIn)
170     {
171         for(int index=0;index<lowerList.size();index++)
172         {
173             String tempCourse = ((Course) lowerList.get(index)).getCourseNumber();
174             if(tempCourse.equals(courseIn.getCourseNumber()))
175             {
176                 System.out.print("Course Already Exsists!");
177                 return;
178             }
179             // check to see if current index is smaller
180             else if(tempCourse.compareToIgnoreCase(courseIn.getCourseNumber()) > 0)
181             {
182                 lowerList.add(index, courseIn);
183 
184                 return;
185             }
186             else { /* do nothing */ }
187         }
188         // Its the lowest, add to bottom
189         lowerList.append(courseIn);
190     }
191 
192     /*************************************************************
193     * Method: checkIfTermExists()                                *
194     * Purpose: add course to low level linkedlist                 *
195     *                                                              *
196     * Parameters: Course:       courseIn                           *
197     * Returns: int:            index of added term                 *
198     *************************************************************/
199     private int addTerm(Course courseIn)
200     {
201         for(int index=0;index<courseList.size();index++)
202         {
203             // Create the 201704 string
204             String tempCourse = courseIn.getYearTaken()+courseIn.getTermTakenRaw();
205             if(tempCourse.equals(courseList.get(index).getTerm()))
206             {
207                 /* term already exists so just return index */
208                 return index;
209             }
210             else if ((courseList.get(index).getTerm().compareToIgnoreCase(tempCourse) > 0))
211             {
212                 courseList.add(index, new Term(courseIn.getYearTaken()+
213                         courseIn.getTermTakenRaw()));
214                 return index;
215             }
216         }
217         // small then all add to end
218         courseList.addLast(new Term(courseIn.getYearTaken()+
219                             courseIn.getTermTakenRaw()));
220         return (courseList.size()-1);
221     }
222 
223     /*************************************************************
224     * Method: get()                                               *
225     * Purpose: get list position                                 *
226     *                                                              *
227     * Parameters: int:            index                          *
228     * Returns: Object:          N/A                               *
229     *************************************************************/
230     public Term get(int index)
231     {
232         return(courseList.get(index));
233     }
234 
235     /*************************************************************
236     * Method: remove()                                           *
237     * Purpose: remove a course from database                      *
238     *                                                              *
239     * Parameters: int:            index,index                    *
240     * Returns: void:            N/A                               *
241     *************************************************************/
242     public void remove(int index, int index2)
```

```
243     {
244         try
245         {
246             Term lowerList = courseList.get(index);
247             lowerList.remove(index2);
248             if(lowerList.size()==0)
249                 courseList.remove(index);
250         }
251         catch(IndexOutOfBoundsException exc)
252         { System.out.println("Index is out of bounds!");}
253     }
254 }
```