```java
/*************************************************************
* Dalton Nofs                                                *
* Login ID: nofs5491                                         *
* CS-102, Summer 2017                                        *
* Programming Assignment 5                                   *
* GpaCalc class:  calculator used to find database gpa         *
*************************************************************/
public class GpaCalc
{
    /*************************************************************
    * Method: calcGpa()                                          *
    * Purpose: Calculate gpa for given database                 *
    *                                                           *
    * Parameters:                                              *
    *         Database: targetDatabase: database to calc gpa     *
    * Returns:                                                  *
    *         double: the calculated gpa                          *
    *************************************************************/
    public double calcGpa(Database targetDatabase) throws IllegalArgumentException
    {
        int totalCredits = 0; // Total credits on not excluded classes
        double creditGpa = 0; // Running total for credit * class grade

        // Check the status of the database
        if(targetDatabase.getDatabaseSize() <= 0)
        {
            throw new IllegalArgumentException("N/A\nDatabase is empty!\n");
        }

        for(int index=0; index<targetDatabase.getDatabaseSize(); index++)
        {
            creditGpa += gatherGpa(targetDatabase.get(index).getRoot());
            totalCredits += gatherCredits(targetDatabase.get(index).getRoot());
        }
        // Calc final database gpa
        return (creditGpa/totalCredits);
    }

    /*************************************************************
    * Method: getClassGrade()                                    *
    * Purpose: figure out what the gpa value from string        *
    *                                                           *
    * Parameters:                                              *
    *         Course: targetCourse: course to find the grade     *
    * Returns:                                                  *
    *         double: the calculated gpa                          *
    *************************************************************/
    double getClassGrade(Course targetCourse) throws IllegalArgumentException
    {
        double courseGrade = 0;
        // Switch for checking the uppercase version of the grade
        switch(targetCourse.getCourseGrade().toUpperCase())
        {
            case "A":     courseGrade = 4.0;
                          break;

            case "A-":     courseGrade = 3.7;
                           break;

            case "B+":     courseGrade = 3.3;
                           break;

            case "B":     courseGrade = 3.0;
                          break;

            case "B-":     courseGrade = 2.7;
                           break;
```

```
68
69              case "C+":      courseGrade = 2.3;
70                              break;
71
72              case "C":      courseGrade = 2.0;
73                              break;
74
75              case "C-":      courseGrade = 1.7;
76                              break;
77
78              case "D+":      courseGrade = 1.3;
79                          break;
80
81              case "D":     courseGrade = 1.0;
82                          break;
83
84              case "F":     courseGrade = 0.0;
85                          break;
86
87              // Catch all non compiant grades
88              default:    throw new IllegalArgumentException("Grade is not correct!");
89          }
90          // Return found grade if exception is not thrown first
91          return courseGrade;
92      }
93
94      /*************************************************************
95      * Method: gatherGpa()    *private*                            *
96      * Purpose: gathers grades from tree                           *
97      *                                                             *
98      * Parameters: TreeNode:        current node                    *
99      * Returns: float:             partal gpa from tree            *
100     *************************************************************/
101     private double gatherGpa(TreeNode<Course> current)
102     {
103         double gpa = 0; // running count of gpa
104         if(current == null) {return gpa;} // if fallen off list
105
106         // Check to see if exclude flag is set
107         if(current.getDatum().getExcludeFlag().toUpperCase().equals("N"))
108         {
109             // Get grade and multiply by the credit count for the top of the gpa calc
110             try
111             {
112                 // add to the total credit count
113                 gpa = (getClassGrade(current.getDatum()) *
114                         current.getDatum().getCreditCount());
115             }
116             catch(IllegalArgumentException exc)
117             {
118                 if(current.getDatum().getCourseGrade().toUpperCase().equals("CR") ||
119                    current.getDatum().getCourseGrade().toUpperCase().equals("I"))
120                 {
121                     // do nothing
122                     throw new IllegalArgumentException(
123                             "N/A\nThe grade for " +
124                             current.getDatum().getCourseNumber() +
125                             " is \"" + current.getDatum().getCourseGrade() +
126                             "\" which is not applicable for GPA calculations!\n");
127                 }
128                 else
129                 {
130                     throw new IllegalArgumentException(
131                             "N/A\nThe grade for " +
132                             current.getDatum().getCourseNumber() +
133                             " is \"" + current.getDatum().getCourseGrade() +
134                             "\" is invalid!\n");
135                 }
```

```
136             }
137         }
138         else{/* do nothing */}
139 
140         // gather the rest of the left till null
141         gpa += gatherGpa(current.getRight());
142         // gather the rest of the right till null
143         gpa += gatherGpa(current.getLeft());
144         return gpa;
145     }
146 
147     /*************************************************************
148     * Method: gatherCredits()    *private*                       *
149     * Purpose: gathers total credits from tree                  *
150     *                                                            *
151     * Parameters: TreeNode:        current node                  *
152     * Returns: float:             num of credits                 *
153     *************************************************************/
154     private int gatherCredits(TreeNode<Course> current)
155     {
156         int totalCredits = 0; // running count of credits
157         if(current == null) {return totalCredits;} // if fallen off list
158 
159         // Check to see if exclude flag is set
160         if(current.getDatum().getExcludeFlag().toUpperCase().equals("N"))
161         {
162             totalCredits += current.getDatum().getCreditCount();
163         }
164         else{/* do nothing */}
165 
166         // gather the rest of the left till null
167         totalCredits += gatherCredits(current.getRight());
168         // gather the rest of the right till null
169         totalCredits += gatherCredits(current.getLeft());
170         return totalCredits; // return after searching left and right
171 
172     }
173 }
```