Project 1

CSE 681 SOFTWARE MODELING & ANALYSIS

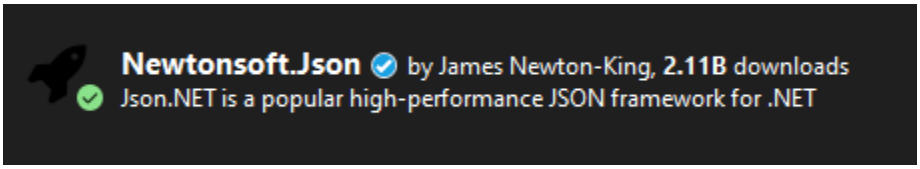
PROF. GREGORY WAGNER

7/28/2022

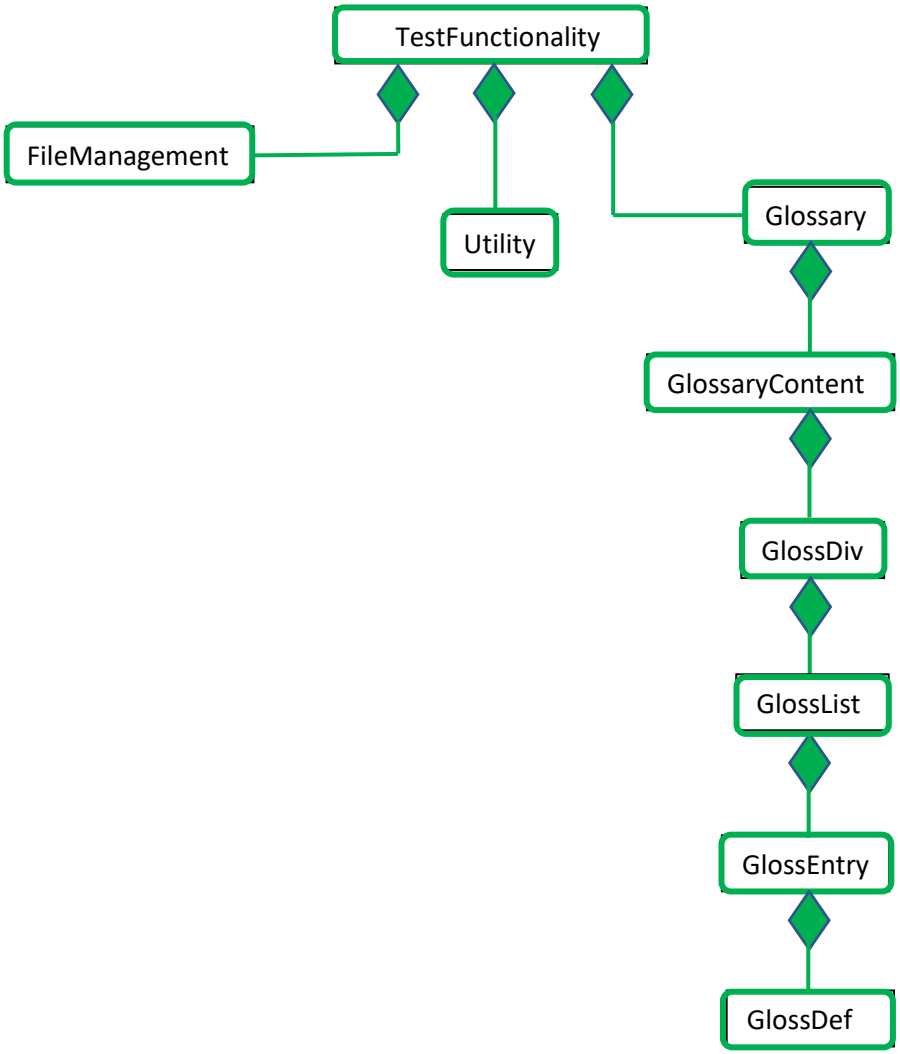
Anthony Redamonti
SYRACUSE UNIVERSITY

Introduction

The following project was written in C# targeting the .NET6 framework in Visual Studio 2022 IDE. The goal of the project is to receive a JSON file from the user and convert it to a Glossary object, which is an instance of a custom class that stores the content of a glossary. Below is a class flowchart. Please download the Newtonsoft.Json package. In Visual Studio, click Project > Manage NuGet Packages. Search for "json." The list of packages should include "Newtonsoft.Json" by James Newton-King.



Class Flowchart:



TestFunctionality Class

The TestFunctionality class composes instances of the FileManagement, Utility, and Glossary classes. It asks the user for a file path to a valid JSON file, then checks the validity of that file using the FileManagement's validateFile method. After the file is validated, it is converted to Glossary object using the utility class's createGlossaryObj method. The content of the glossary object is then displayed using the displayContent method in the glossary object.

It then creates a copy of the glossary object by calling the Utility class's copyGlossaryObjs method. The contents of the copy are edited. The original object contents are then added to the copy. The copy will then contain two distinct GlossDiv objects. The copy's contents are displayed to the console.

Utility Class

The createGlossaryObj method creates a JSON string using the ReadAllText method in the File class. The JSON string is then passed into the DeserializeObject method of the JsonConvert class, which creates a glossary object populated with the content in the JSON file.

The addGlossaryObjs method adds two glossary objects together. The second glossary object stores their sum.

The copyGlossaryObjs method copies the contents from the first glossary object to the second.

FileManagement Class

The FileManagement class is responsible for verifying the validity of the user input. There is one public method, validateFile, which calls three private methods: verifyFileExtension, verifyFileExists, and verifyFileIsNotEmpty. ValidateFile returns 0 on success and a negative value if the file is not valid.

Glossary Class

The Glossary class has a JsonProperty "glossary" which is an instance of the GlossaryContent class. It also has a public method to display the content to the console.

GlossaryContent Class

The GlossaryContent class has a JsonProperty "title" which is a string. It has a JsonProperty "GlossDiv" which is an instance of the GlossDiv class. It also has a list of GlossDiv objects as well as a public method to display the content to the console.

GlossDiv Class

The GlossDiv class has a JsonProperty "title" which is a string. It has a JsonProperty "GlossList" which is an instance of the GlossList class. It also has a public method to display the content to the console.

GlossList Class

The GlossList class has a JsonProperty "GlossEntry" which is an instance of the GlossEntry class. It also has a list of GlossEntry objects. It also has a public method to display the content to the console.

GlossEntry Class

The GlossaryEntry class has the following JsonProperty strings: ID, SortAs, GlossTerm, Acronym, Abbrev, and GlossSee. It has a JsonProperty “GlossDef” which is an instance of the GlossDef class. It also has a public method to display the content to the console.

GlossDef Class

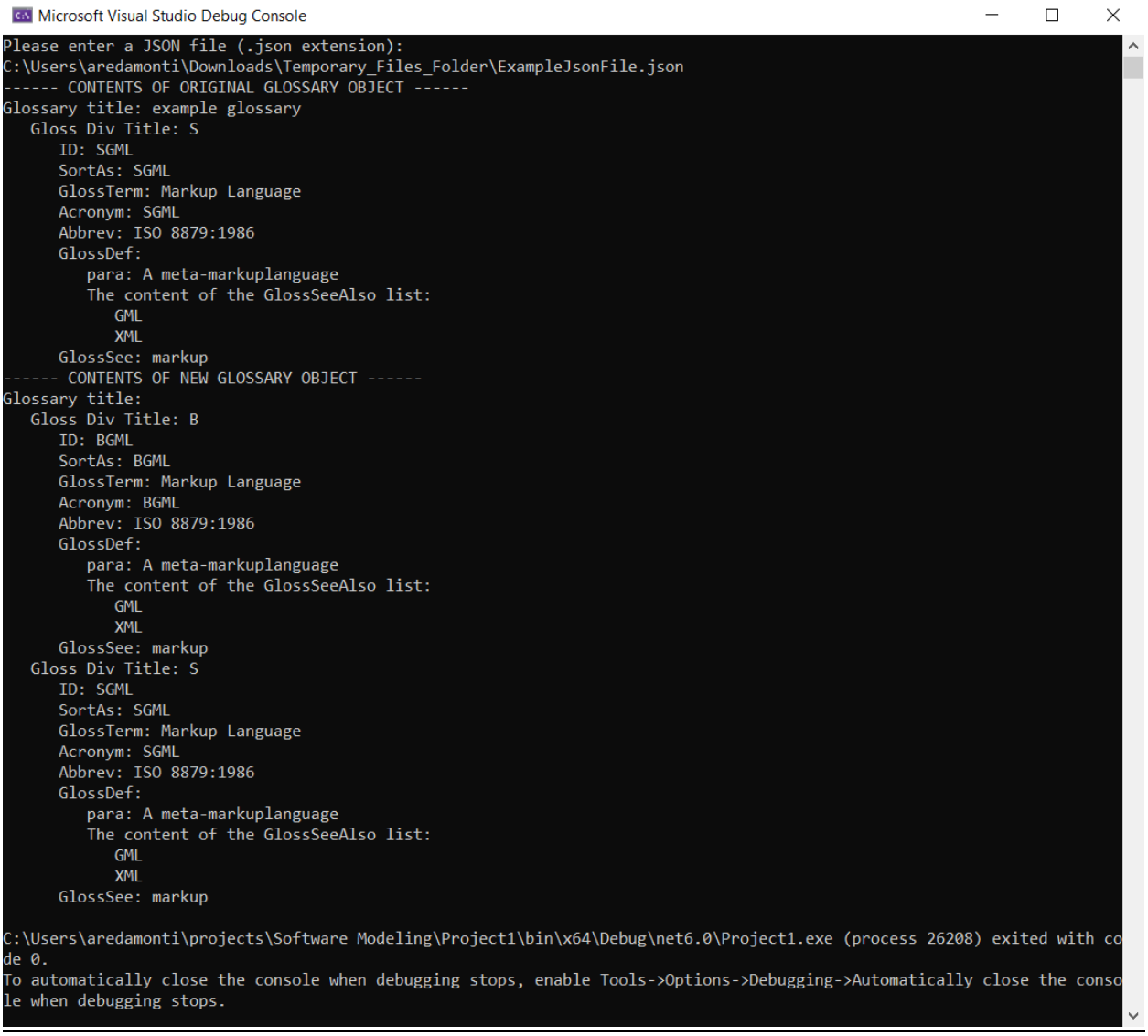
The GlossDef class has a JsonProperty “para” which is a string. It has a JsonProperty “GlossSeeAlso” which is a list of strings. It also has a public method to display the content to the console.

Example Input:

ExampleJsonFile.json:

```
{
	"glossary": {
		"title": "example glossary",
		"GlossDiv": {
			"title": "S",
			"GlossList": {
				"GlossEntry": {
					"ID": "SGML",
					"SortAs": "SGML",
					"GlossTerm": "Markup Language",
					"Acronym": "SGML",
					"Abbrev": "ISO 8879:1986",
					"GlossDef": {
						"para": "A meta-markuplanguage",
						"GlossSeeAlso": ["GML", "XML"]
						},
					"GlossSee": "markup"
				}
			}
		}
	}
}
```

Example Output:



Project1.cs File

```
/*
 * Anthony Redamonti
 * Professor Gregory Wagner
 * CSE 681 Software Modeling
 * Project 1
 * 7/28/2022
 *
 * The following program asks the user for an input JSON file.
 * The JSON file is checked for validity. Its contents are then
 * converted to a string and deserialized into a Glossary object.
 * The contents of the Glossary object are displayed to the console.
 *
 * A copy of the glossary object is then created. Its content is
 * altered. Then the original content is added to the new object.
 * The contents of the new object are displayed to the console.
 *
 * Json File Format:
{
      "glossary": {
            "title": "example glossary",
            "GlossDiv": {
                  "title": "S",
                  "GlossList": {
                        "GlossEntry": {
                              "ID": "SGML",
                              "SortAs": "SGML",
                              "GlossTerm": "Markup Language",
                              "Acronym": "SGML",
                              "Abbrev": "ISO 8879:1986",
                              "GlossDef": {
                                    "para": "A meta-markuplanguage",
                                    "GlossSeeAlso": ["GML", "XML"]
                                    },
                              "GlossSee": "markup"
                        }
                  }
            }
      }
}
 *
 */

using System.Text.Json;
using System.Text.Json.Nodes;
using System.Runtime.Serialization;
using Newtonsoft.Json;

namespace Project1
{
    /// <summary>
    /// Below is the public TestFunctionality class.
    /// It will ask the user for an input JSON file and then validate its content.
    /// The file is then passed to the Utility class to be deserialized into a
    /// glossary object. The contents of the glossary object are displayed to the
    /// console. A new object is created and altered. Then it's added to the
    /// original object and displayed to the console.
```

```
    /// </summary>
    public class TestFunctionality
    {
        // the main method will accept the JSON file from the user.
        static public void Main(String[] stringArgs)
        {
            int error = 1;
            string inputFile = null;
            FileManagement fileManagementObj = new FileManagement();
            Glossary glossaryObj1 = new Glossary();
            Utility utilityObj = new Utility();
            try
            {
                // make sure the user entered something.
                while ((inputFile == null) || (inputFile == "") || (error != 0))
                {
                    Console.WriteLine("Please enter a JSON file (.json extension):
");
                    inputFile = Console.ReadLine();
                    error = fileManagementObj.validateFile(inputFile);
                }

                utilityObj.createGlossaryObj(inputFile, ref glossaryObj1);

                Console.WriteLine("------ CONTENTS OF ORIGINAL GLOSSARY OBJECT -----
-");
                glossaryObj1.displayContent();

                Glossary gossaryObj2 = new Glossary();

                // copy the first glossary object to the new one.
                utilityObj.copyGlossaryObjs(ref glossaryObj1, ref gossaryObj2);

                // edit some properties of the second object.
                gossaryObj2.glossaryContentObj.glossDivList[0].title = "B";

gossaryObj2.glossaryContentObj.glossDivList[0].glossListObj.glossEntryList[0].ID =
"BGML";

gossaryObj2.glossaryContentObj.glossDivList[0].glossListObj.glossEntryList[0].SortAs
= "BGML";

gossaryObj2.glossaryContentObj.glossDivList[0].glossListObj.glossEntryList[0].Acrony
m = "BGML";

                // GlossaryObj2 = GlossaryObj1 + GlossaryObj2.
                utilityObj.addGlossaryObjs(ref glossaryObj1, ref gossaryObj2);

                Console.WriteLine("------ CONTENTS OF NEW GLOSSARY OBJECT ------");
                gossaryObj2.displayContent();
            }
            // catch any exception here.
            catch (Exception ex)
            {
                Console.WriteLine("Exception thrown in TestFunctionality::Main
method: " + ex.Message);
                throw;
            }
```

```
        }
    }

    /// <summary>
    /// The Utility class is used to create new glossary objects from input JSON
files.
    /// They are first converted to string format. Then they are deserialized into a
    /// glossary object. The glossary object passed by reference is updated.
    /// </summary>
    public class Utility
    {
        /// <summary>
        /// Below is the public createGlossaryObj method.
        /// It will convert the contents of the file to a string.
        /// The string will be deserialized and stored inside a Glossary object.
        /// </summary>
        public void createGlossaryObj(string inputFile, ref Glossary glossaryObj)
        {
            try
            {
                // create a JSON string by reading all the text in the file.
                string jsonStr = File.ReadAllText(inputFile);

                // create a glossary object by deserializing the JSON string using
the Glossary Object class structure.
                glossaryObj = JsonConvert.DeserializeObject<Glossary>(jsonStr);

                // add the glossListObj to the glossList.

glossaryObj.glossaryContentObj.glossDivObj.glossListObj.glossEntryList.Add(glossaryO
bj.glossaryContentObj.glossDivObj.glossListObj.glossListObj);

                // add the glossDivObj to the glossary list for this glossary
object.

glossaryObj.glossaryContentObj.glossDivList.Add(glossaryObj.glossaryContentObj.gloss
DivObj);
            }
            // catch any exception here.
            catch (Exception ex)
            {
                Console.WriteLine("Exception thrown in Utility::createGlossaryObj
method: " + ex.Message);
                throw;
            }
        }

        /// <summary>
        /// Below is the public addGlossaryObjs method.
        /// It will add the contents of the first glossary object to the second.
        /// </summary>
        public void addGlossaryObjs(ref Glossary glossObj1, ref Glossary glossObj2)
        {
            try
            {
                for(int i = 0; i < glossObj1.glossaryContentObj.glossDivList.Count;
i++)
                {
```

```
                    bool isInList2 = false;

                    for(int j = 0; j <
glossObj2.glossaryContentObj.glossDivList.Count; j++)
                    {
                        // if the keys of the GlossDivEntries are the same, add the
GlossEntries from Obj1 to Obj2's.
                        if(glossObj1.glossaryContentObj.glossDivList[i].title ==
glossObj2.glossaryContentObj.glossDivList[j].title)
                        {
                            isInList2 = true;

                            // make sure there are no copies of GlossEntries (keys
must be unique).
                            for(int k = 0; k <
glossObj1.glossaryContentObj.glossDivList[i].glossListObj.glossEntryList.Count; k++)
                            {
                                bool isInGlossList2 = false;

                                for(int z = 0; z <
glossObj2.glossaryContentObj.glossDivList[j].glossListObj.glossEntryList.Count; z++)
                                {

if(glossObj1.glossaryContentObj.glossDivList[i].glossListObj.glossEntryList[k].ID ==
glossObj2.glossaryContentObj.glossDivList[j].glossListObj.glossEntryList[z].ID)
                                    {
                                        isInGlossList2 = true;
                                    }
                                }
                                if (!isInGlossList2)
                                {

glossObj2.glossaryContentObj.glossDivList[j].glossListObj.glossEntryList.Add(glossOb
j1.glossaryContentObj.glossDivList[i].glossListObj.glossEntryList[k]);
                                }
                            }
                        }
                    }

                    // if it is not in list 2 then add it to list 2.
                    if (!isInList2)
                    {

glossObj2.glossaryContentObj.glossDivList.Add(glossObj1.glossaryContentObj.glossDivL
ist[i]);
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("Exception thrown in Utility::addGlossaryObjs
method: " + ex.Message);
                throw;
            }
        }

        /// <summary>
        /// Below is the public copyGlossaryObjs method.
```

```
        /// It will copy all the items from the original glossary object into the
        /// new glossary object.
        /// </summary>
        /// <param name="originalGlossObj"></param>
        /// <param name="newGlossObj"></param>
        public void copyGlossaryObjs(ref Glossary originalGlossObj, ref Glossary
newGlossObj)
        {
            try
            {
                // first, clear out any old points in the new glossary object
                if (newGlossObj.glossaryContentObj != null)
                {
                    newGlossObj.glossaryContentObj.glossDivList.Clear();
                }
                // if it's null, then create a new GlossaryContentObj on the heap.
                else
                {
                    newGlossObj.glossaryContentObj = new GlossaryContent();
                    newGlossObj.glossaryContentObj.glossDivObj = new GlossDiv();
                    newGlossObj.glossaryContentObj.glossDivObj.glossListObj = new
GlossList();
                }

                int originalDivListCount =
originalGlossObj.glossaryContentObj.glossDivList.Count();

                // next, copy each element from the original glossary object
                for (int i = 0; i < originalDivListCount; i++)
                {
                    // create a new space in memory for this glossDivEntry
                    GlossDiv newGlossDiv = new GlossDiv();

                    // populate the newGlossDiv object
                    populateGlossDiv(ref newGlossDiv, ref originalGlossObj, i);

                    // add the elements from the glossDivlist to the object.
                    newGlossObj.glossaryContentObj.glossDivList.Add(newGlossDiv);
                }
            }
            // catch any exception here.
            catch (Exception ex)
            {
                Console.WriteLine("Exception thrown in Utility::copyGlossaryObjs
method: " + ex.Message);
                throw;
            }
        }

        /// <summary>
        /// A helper function for the copyGlossaryObjs method. It populates the new
gloss div from the
        /// original gloss object's glossDivList entry.
        /// </summary>
        /// <param name="newGlossDiv"></param>
        /// <param name="originalGlossObj"></param>
        private void populateGlossDiv(ref GlossDiv newGlossDiv, ref Glossary
originalGlossObj, int i)
```

```
        {
            try
            {
                newGlossDiv.title =
originalGlossObj.glossaryContentObj.glossDivList[i].title;

                // create a new gloss list for this entry
                newGlossDiv.glossListObj = new GlossList();
                newGlossDiv.glossListObj.glossEntryList = new List<GlossEntry>();

                int originalNumberOfGlossListEntries =
originalGlossObj.glossaryContentObj.glossDivList[i].glossListObj.glossEntryList.Coun
t;

                // copy each of the GlossListEntries from the original object to the
new object.
                for (int j = 0; j < originalNumberOfGlossListEntries; j++)
                {
                    // allocate new memory on the heap.
                    GlossEntry newGlossEntry = new GlossEntry();

                    // use the helper function to populate the new GlossEntry
object.
                    populateGlossEntry(ref newGlossEntry, ref originalGlossObj, i,
j);

                    // add the new gloss entry to the new gloss list.
                    newGlossDiv.glossListObj.glossEntryList.Add(newGlossEntry);
                }
            }
            // catch any exception here.
            catch (Exception ex)
            {
                Console.WriteLine("Exception thrown in Utility::populateGlossDiv
method: " + ex.Message);
                throw;
            }
        }

        /// <summary>
        /// A helper function for the populateGlossDiv method. It populates the new
gloss entry from the
        /// original gloss object's gloss list entry.
        /// </summary>
        /// <param name="newGlossEntry"></param>
        /// <param name="originalGlossObj"></param>
        private void populateGlossEntry(ref GlossEntry newGlossEntry, ref Glossary
originalGlossObj, int i, int j)
        {
            try
            {
                // copy each of the properties from the original object to the new
object.
                newGlossEntry.ID =
originalGlossObj.glossaryContentObj.glossDivList[i].glossListObj.glossEntryList[j].I
D;
```

```
                newGlossEntry.Abbrev =
originalGlossObj.glossaryContentObj.glossDivList[i].glossListObj.glossEntryList[j].A
bbrev;
                newGlossEntry.Acronym =
originalGlossObj.glossaryContentObj.glossDivList[i].glossListObj.glossEntryList[j].A
cronym;
                newGlossEntry.SortAs =
originalGlossObj.glossaryContentObj.glossDivList[i].glossListObj.glossEntryList[j].S
ortAs;
                newGlossEntry.GlossTerm =
originalGlossObj.glossaryContentObj.glossDivList[i].glossListObj.glossEntryList[j].G
lossTerm;
                newGlossEntry.GlossSee =
originalGlossObj.glossaryContentObj.glossDivList[i].glossListObj.glossEntryList[j].G
lossSee;

                // create new GlossDef and GlossSeeAlso objects.
                GlossDef newGlossDef = new GlossDef();
                newGlossDef.GlossSeeAlso = new List<string>();
                newGlossDef.para =
originalGlossObj.glossaryContentObj.glossDivList[i].glossListObj.glossEntryList[j].g
lossDefObj.para;

                // populate all GlossSeeAlso elements into the newGlossDef object.
                int numberOfGlossSeeAlsoOriginal =
originalGlossObj.glossaryContentObj.glossDivList[i].glossListObj.glossEntryList[j].g
lossDefObj.GlossSeeAlso.Count;
                for (int k = 0; k < numberOfGlossSeeAlsoOriginal; k++)
                {
                    string temp =
originalGlossObj.glossaryContentObj.glossDivList[i].glossListObj.glossEntryList[j].g
lossDefObj.GlossSeeAlso[k];
                    newGlossDef.GlossSeeAlso.Add(temp);
                }

                newGlossEntry.glossDefObj = newGlossDef;
            }
            // catch any exception here.
            catch (Exception ex)
            {
                Console.WriteLine("Exception thrown in Utility::populateGlossEntry
method: " + ex.Message);
                throw;
            }
        }
    }

    /// <summary>
    /// Below is the public Glossary class.
    /// Use the JsonConvert.DeserializeObject<Glossary>(JsonString) method
    /// to populate the properties of an instance of this class. Then use
    /// the displayContent method to view the glossary's content on the console.
    /// </summary>
    public class Glossary
    {
        [JsonProperty("glossary")]
        public GlossaryContent glossaryContentObj { get; set; }
```

```
    // display the content of the Glossary object to the user.
    public void displayContent()
    {
        glossaryContentObj.displayContent();
    }
}

/// <summary>
/// Below is the public GlossaryContent class.
/// It holds important JsonProperties for the instance of the glossary class.
/// It also holds a list of GlossDiv objects.
/// </summary>
public class GlossaryContent
{
    [JsonProperty("title")]
    public string? title { get; set; }

    [JsonProperty("GlossDiv")]
    public GlossDiv glossDivObj { get; set; }

    // display the content of the GlossaryContent object to the user.
    public void displayContent()
    {
        Console.WriteLine("Glossary title: " + title);

        // display the content in each GlossDiv object in the list.
        for(int i = 0; i < glossDivList.Count; i++)
        {
            glossDivList[i].displayContent();
        }
    }

    // a list holding the glossDivEntries.
    public List<GlossDiv> glossDivList = new List<GlossDiv>();
}

/// <summary>
/// Below is the public GlossDiv class.
/// It holds important JsonProperties for the GlossDiv object.
/// </summary>
public class GlossDiv
{
    [JsonProperty("title")]
    public string? title { get; set; }

    [JsonProperty("GlossList")]
    public GlossList glossListObj { get; set; }

    // display the content of the instance of the GlossDiv class.
    public void displayContent()
    {
        Console.WriteLine("   Gloss Div Title: " + title);
        glossListObj.displayContent();
    }
}

/// <summary>
/// Below is the public GlossList class.
```

```
/// It holds an instance of the gloss list object.
/// It also holds a list of gloss entries.
/// </summary>
public class GlossList
{
    [JsonProperty("GlossEntry")]
    public GlossEntry glossListObj { get; set; }

    // a list to store the GlossEntry objects.
    public List<GlossEntry> glossEntryList = new List<GlossEntry>();

    // display the content of the instance of the GlossDiv class.
    public void displayContent()
    {
        for (int i = 0; i < glossEntryList.Count; i++)
        {
            glossEntryList[i].displayContent();
        }
    }
}

/// <summary>
/// Below is the public GlossEntry class.
/// It holds important JsonProperties for an entry in the GlossDiv.
/// </summary>
public class GlossEntry
{
    [JsonProperty("ID")]
    public string? ID { get; set; }

    [JsonProperty("SortAs")]
    public string? SortAs { get; set; }

    [JsonProperty("GlossTerm")]
    public string? GlossTerm { get; set; }

    [JsonProperty("Acronym")]
    public string? Acronym { get; set; }

    [JsonProperty("Abbrev")]
    public string? Abbrev { get; set; }

    [JsonProperty("GlossDef")]
    public GlossDef glossDefObj { get; set; }

    [JsonProperty("GlossSee")]
    public string? GlossSee { get; set; }

    // display the contents of the gloss entry.
    public void displayContent()
    {
        Console.WriteLine("      ID: " + ID);
        Console.WriteLine("      SortAs: " + SortAs);
        Console.WriteLine("      GlossTerm: " + GlossTerm);
        Console.WriteLine("      Acronym: " + Acronym);
        Console.WriteLine("      Abbrev: " + Abbrev);
        Console.WriteLine("      GlossDef:");
        glossDefObj.displayContent();
```

```
            Console.WriteLine("      GlossSee: " + GlossSee);
        }
    }

    /// <summary>
    /// Below is the public GlossDef class.
    /// It holds important JsonProperties for the definition of the gloss entry.
    /// </summary>
    public class GlossDef
    {
        [JsonProperty("para")]
        public string? para { get; set; }

        [JsonProperty("GlossSeeAlso")]
        public IList<string> GlossSeeAlso { get; set; }

        // display the contents of the Gloss Def object.
        public void displayContent()
        {
            Console.WriteLine("         para: " + para);
            Console.WriteLine("         The content of the GlossSeeAlso list: ");
            for (int i = 0; i < GlossSeeAlso.Count; i++)
            {
                Console.WriteLine("             " + GlossSeeAlso[i]);
            }
        }
    }

    /// <summary>
    /// The file management class is responsible for verifying
    /// the user input. The user must enter a valid, non-empty
    /// JSON file.
    /// </summary>
    public class FileManagement
    {
        public int validateFile(string inputFileName)
        {
            int error = 0;

            // call the three private methods used to check for validity.
            error = verifyFileExtension(inputFileName);
            error = verifyFileExists(inputFileName);
            error = verifyFileIsNotEmpty(inputFileName);

            // return the error code if there is one. Error = 0 on success.
            return error;
        }

        // verify the file's extension is .json. If there was an exception, return -
1.
        // If the file extension is not .json, return -2. If the file extension is
.json,
        // return 0.
        private int verifyFileExtension(string inputFileName)
        {
            string extension = null;
            try
            {
```

```
                extension = Path.GetExtension(inputFileName);
            }
            catch(Exception ex)
            {
                Console.WriteLine(ex.Message);
                return -1;
            }
            if(extension != ".json")
            {
                Console.WriteLine("File extension is " + extension + ". Please enter
a .json file.");
                return -2;
            }
            else
            {
                return 0;
            }
        }

        // Verify the file exists. If it does not exist, return -3. If there was an
exception,
        // return -4. If the file exists, return 0.
        private int verifyFileExists(string inputFileName)
        {
            try
            {
                FileInfo fileInfoObj = new FileInfo(inputFileName);
                if (!fileInfoObj.Exists)
                {
                    Console.WriteLine(inputFileName + " does not exist.");
                    return -3;
                }
                else
                {
                    return 0;
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                return -4;
            }
        }

        // Verify that the file is not empty. Use the FileInfo object's length
method to
        // check for the existance of file content. If the file is empty, return -5.
If
        // there was an exception, return -6. If the file is not empty, return 0.
        private int verifyFileIsNotEmpty(string inputFileName)
        {
            try
            {
                FileInfo fileInfoObj = new FileInfo(inputFileName);
                if (fileInfoObj.Length == 0)
                {
                    Console.WriteLine("The file is empty. Please enter a non-empty
file in JSON format.");
```

```
                    return -5;
                }
                else
                {
                    return 0;
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                return -6;
            }
        }
    }
}
```