Please replace this paragraph with your Assignments 1 and 2 unchanged but including grading and comments. 2/2/2021 4:45 PM

# Assignment 4

Implement the next release of your term project or start a new one if necessary. You will incorporate—at a minimum—the *saving and retrieval of objects*, the use of *lambdas*, and the use of *streams*. You can substitute the use of JavaFX for one of these three concepts if you wish. Also, substitutions for any of these concepts by another advanced techniques are acceptable if you have already used them: please clear this with your facilitator.

The same instructions as in Assignment 3 apply to this completed Word document regarding the gray text, the 5 page limit, appendices, JUnit tests, and a ReadMe file.

## 4.1 SUMMARY DESCRIPTION THIS

*Evaluation criterion (i) applies*

One- or two-paragraph overall description of your proposed term project. Color red the parts changed from Assignment 2, if any.

This project concerns a system for organizing and categorizing harmony (chords, a collection of musical notes). To use this system, called *HarmonyMuse*, a musician inputs either a collection of notes or a chord quality with an accompanying complexity and a context for the sonority of the chord. *HarmonyMuse* analyzes the given input and provides a performance solution or harmonic analysis based on the provided context.

## 4.2 ADDITIONAL REQUIREMENTS (FEATURES) IMPLEMENTED IN THIS RELEASE

*Evaluation criterion (i) applies*

Title and one or two sentences per requirement. Don’t repeat requirements implemented for prior assignments unless they are necessary to provide context—in which case, make it clear they are old.

### 4.2.1 Serialize a Sequence of Chords for network transmission. (NEW)

HarmonyMuse should serialize a Sequence of Chords (and subsequent sub structures) to JSON for easy transmission over the network.

### 4.2.2 Implement deserialization for files containing system native objects that may be transmitted over a network. (NEW/OLD)

HarmonyMuse should deserialize JSON such that Objects can be manipulated, filtered, and otherwise transformed and modulated by system logic.

### 4.2…. more as needed

## 4.3 I/O SUPPORTING THE NEW REQUIREMENTS LISTED ABOVE

*Evaluation criterion (ii) applies*

Provide an example of input / output showing the new features of your application.

### Input

private ChordSequence chordSequence;  
private final String filename = "test";  
private final WriteToJSON writer = new WriteToJSON();  
private final TriadClassifier tc = new TriadClassifier();  
private final ReadFromJSON reader = new ReadFromJSON();  
  
*/\*\*  
 \* The purpose of this method is to initialize Chords of different types  
 \* and store these in a ChordSequence for JSON serialization and deserialization  
 \*/*@BeforeEach  
void setUp(){  
 Note c;  
 Note fSharp;  
 Note eNat;  
 Note a;  
 try{  
 c = new Note("c");  
 MajorTriad C = new MajorTriad(c);  
 fSharp = new Note("f#");  
 MinorTriad Fsharp = new MinorTriad(fSharp);  
 eNat = new Note("e");  
 AugmentedTriad E = new AugmentedTriad(eNat);  
 a = new Note("a");  
 DiminishedTriad A = new DiminishedTriad(a);  
 this.chordSequence = new ChordSequence(C, Fsharp, E, A, c);  
  
 }catch (InvalidNoteException e){  
 System.*out*.println(e);  
 }  
}

### Input / Output

Text

Description automatically generated

## 4.4 YOUR DIRECTORY

Show a screenshot of your directory. Include your “.dat” files (where objects are written). This should include JUnit tests—class-by-class, and method-by-method, except for trivial and inappropriate ones.

Your response replaces this.

## 4.5 DESIGN

*Evaluation criterion (i) applies*

Supply a main use case, the class model, and the sequence diagram corresponding to the use case. These should be consistent. Indicate in red your class model where you applied object read, object write, streams and lambdas. Excellent assignments will typically include the use of Java FX (speak to your facilitator first if you wish to use alternative API’s) and event-driven programming.

Your response replaces this.

## 4.6 CODE SNIPPETS

*Evaluation criterion (iii) applies*

### 4.6.1 Code showing object read and write THIS

package FileHandling;  
  
  
import Builders.ChordSequence;  
import com.google.gson.Gson;  
import com.google.gson.GsonBuilder;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.io.Writer;  
  
*/\*\*  
 \** ***@author*** *Michael Kramer  
 \* <p>  
 \* CS622 Spring 1, 2022 Advanced Programming Techniques  
 \* <p>  
 \* The purpose of this class is to serialize app native Objects to JSON  
 \*/*public class WriteToJSON {  
  
 */\*\*  
 \* The purpose of this method is to serialize and write to file a ChordSequence  
 \* object  
 \* <p>Precondition: A ChordSequence object has been instantiated and  
 \* a valid filename passed to the system</p>  
 \* <p>Postcondition: The ChordSequence is serialized and written to  
 \* a file filename.json</p>  
 \*  
 \** ***@param*** *filename a valid filename String  
 \** ***@param*** *chordSequence a ChordSequence object  
 \*/* public void writeSequenceToJSON(String filename, ChordSequence chordSequence){  
 try (Writer writer = new FileWriter("data\\" + filename +".json")) {  
 Gson gson = new GsonBuilder().setPrettyPrinting().create();  
 gson.toJson(chordSequence, writer);  
 }catch (IOException | NullPointerException e){  
 System.*out*.println(e);  
 }  
 }  
}

package FileHandling;  
  
import AbstractStructures.Chord;  
import Builders.ChordSequence;  
import Classifiers.TriadClassifier;  
import ThreeNoteStructures.\*;  
import com.google.gson.Gson;  
import com.google.gson.GsonBuilder;  
import com.google.gson.reflect.TypeToken;  
import org.junit.jupiter.api.Assertions;  
  
import java.io.FileReader;  
import java.io.IOException;  
import java.lang.reflect.Type;  
import java.util.Arrays;  
import java.util.stream.Stream;  
  
*/\*\*  
 \** ***@author*** *Michael Kramer  
 \* <p>  
 \* CS622 Spring 1, 2022 Advanced Programming Techniques  
 \* <p>  
 \* The purpose of this class is to deserialize JSON objects from file and  
 \* get them into Object forms that can be manipulated by the system  
 \*/*public class ReadFromJSON<T extends Chord> {  
  
 */\*\*  
 \* The purpose of this method is to deserialize a ChordSequence object from  
 \* file  
 \* <p>Precondition: a file exists of the passed filename param with  
 \* a serialized ChordSequence Object</p>  
 \* <p>Postcondition: The ChordSequence is read back into memory and returned</p>  
 \*  
 \** ***@param*** *filename the file to be read  
 \*/* public ChordSequence<Chord> readChordSequenceFromJSON(String filename){  
 Gson gson = new GsonBuilder().setPrettyPrinting().create();  
  
 try {  
 // you have to give it a concrete type  
 Type type = new TypeToken<ChordSequence<ConcreteTriad>>(){}.getType();  
 ChordSequence<Chord> deserializedSequence = gson.fromJson(new FileReader("data\\" + filename + ".json"), type);  
  
 return deserializedSequence;  
 } catch (IOException e){  
 System.*out*.println(e);  
 }  
 return null;  
 }  
  
 */\*\*  
 \* The purpose of this method is to convert a ChordSequence object that has  
 \* just been successfully read from file into a Stream of Chords for  
 \* manipulation  
 \* <p>Precondition: A ChordSequence has successfully been deserialized</p>  
 \* <p>Postcondition: a Stream of Chords is returned</p>  
 \*  
 \** ***@param*** *deserializedSequence is a ChordSequence to be returned as a Stream  
 \* of Chords  
 \*/* public Stream<Chord> deserializedJSONToChordStream(ChordSequence<Chord> deserializedSequence) {  
 TriadClassifier tc = new TriadClassifier();  
 Chord[] chordsFromFile = new Chord[deserializedSequence.getSize()];  
 for (int i = 0; i < deserializedSequence.getSize(); i++) {  
 Assertions.*assertTrue*(deserializedSequence.getChord(i) instanceof ConcreteTriad);  
 if (deserializedSequence.getChord(i).getQuality().equals(tc.getTriadQualities()[0])) {  
 DiminishedTriad diminishedTriad = new DiminishedTriad(deserializedSequence.getChord(i).getRoot());  
 chordsFromFile[i] = diminishedTriad;  
 } else if (deserializedSequence.getChord(i).getQuality().equals(tc.getTriadQualities()[1])) {  
 MinorTriad minorTriad = new MinorTriad(deserializedSequence.getChord(i).getRoot());  
 chordsFromFile[i] = minorTriad;  
 } else if (deserializedSequence.getChord(i).getQuality().equals(tc.getTriadQualities()[2])) {  
 MajorTriad majorTriad = new MajorTriad(deserializedSequence.getChord(i).getRoot());  
 chordsFromFile[i] = majorTriad;  
 } else if (deserializedSequence.getChord(i).getQuality().equals(tc.getTriadQualities()[3])) {  
 AugmentedTriad augmentedTriad = new AugmentedTriad(deserializedSequence.getChord(i).getRoot());  
 chordsFromFile[i] = augmentedTriad;  
 }  
 }  
 Stream<Chord> chordSequenceStream = Arrays.*stream*(chordsFromFile);  
 return chordSequenceStream;  
 }  
}

4.6.2 Code Showing stream() and Lambdas (separate int0 4.5.3 and 4.5.4 if you wish) THIS

*/\*\*  
 \* The purpose of this method is to test that a deserialized ChordSequence  
 \* can be converted to a Java Stream of Chords via the ReadFromJSON.deserializedJSONToChordStream method  
 \* and filtered out by quality  
 \* <p>Precondition: a json file of serialized chord sequences exists</p>  
 \* <p>Postcondition: the chords of specified quality are filtered out  
 \* of the Stream created from the object read back into memory</p>  
 \*/*@Test  
void TestChordStreamFilterInstanceof(){  
 System.*out*.println("\nWriteReadFromJSONTest.TestChordStreamFilterInstanceof");  
 ChordSequence deserializedSequence = reader.readChordSequenceFromJSON(this.filename);  
 Stream<Chord> chordStream = reader.deserializedJSONToChordStream(deserializedSequence);  
 chordStream.filter(i -> i instanceof AugmentedTriad).forEach(i -> System.*out*.println(i));  
}  
  
*/\*\*  
 \* The purpose of this method is simply to read a ChordSequence back into  
 \* memory and print each Chord in the Sequence to the console by way of  
 \* creating a Stream of Chords with the ReadFromJSON.deserializedJSONToChordStream  
 \* method  
 \* <p>Precondition: a json file with a serialized ChordStream exists</p>  
 \* <p>Postcondition: The ChordSequence is deserialized and converted to  
 \* a Stream of Chords which can be operated of as a Stream of Objects</p>  
 \*/*@Test  
void TestChordStreamForEach(){  
 System.*out*.println("\nWriteReadFromJSONTest.TestChordStreamForEach:");  
 ChordSequence deserializedSequence = reader.readChordSequenceFromJSON(this.filename);  
 Stream<Chord> chordStream = reader.deserializedJSONToChordStream(deserializedSequence);  
 chordStream.forEach(i -> System.*out*.println(i));  
}

Your response replaces this.

## 4.7 YOUR CODE

*Evaluation criterion (iii) applies*

Unless your facilitator arranges another method, copy your Eclipse project to your file system, zip it, and attach it to your Blackboard response. Please contact your facilitator in advance if you require an exception.

Your response replaces this.

## 4.8 Evaluation

