Please replace this paragraph with your Assignments 1 and 2 unchanged but including grading and comments. 1/28/2021 7:03 AM

# Assignment 3

Implement the next release of your term project (preferably, or start a new one if you have to). You will incorporate generics. The same instructions as in Assignment 2 apply to this completed Word document, the gray text, the 5 page limit, appendices, JUnit tests, and a ReadMe file.

## 3.1 SUMMARY DESCRIPTION, UPDATED AS NECESSARY THIS

*EVALUATION CRITERION (i) APPLIES*

One- or two-paragraph overall description of your proposed term project. Color in red the parts different from Assignment 2.

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 based on the provided context.

## 3.2 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 which are new vs. old.

### 3.2.1 Implement a class that contains a particular grouping of Chords and tracks the relationships of these objects to a tonal center (NEW REQUIREMENT)

There must be a container for a cell of a sequence of chords and a method of tracking how chords in the given sequence relate to one another. The classic example of this is being able to classify a ii-V-I chord progression and the tonal center to which this instance belongs.

### 3.2.2 Implement a class that contains chord sequences and how these sequences relate to one another (NEW REQUIREMENT)

Where requirement 3.2.1 might represent a musical phrase, this requirement might represent an entire song and provide the capacity to track modulations (key changes) or key centers visited and obviously contain the individual Chord, Interval and Note data or each ChordSequence therein.

### 3.2.3 Deliver chord sequence and/or chord progression information to the user with Roman Numeral notation (NEW REQUIREMENT)

It’s utterly important that the system maintain clear communication standards. Notes are lower case strings, Chords may be represented by upper case strings and sequences of chords and progressions must be represented in terms of the roman numerals as they relate to the given tonal centers. The system needs to support this.

### 3.2.4 Your title replaces this. (OLD / NEW REQUIREMENT)

Your response replaces this.

### 3.2.5 ….

## 3.3 I/O EVIDENCE OF ACCOMPLISHING THE REQUIREMENTS LISTED ABOVE

*EVALUATION CRITERION (ii) APPLIES*

Provide an example of actual input / output corresponding to the requirements above

Your response replaces this.

### Input File(s)

File ….

### Input / Output

Console I/O:

### Output File(s)

File …

## 3.4 YOUR DIRECTORY

Show a screenshot of your directory. This should include a parallel directory of JUnit tests where applicable—package-by-package, class-by-class, and method-by-method, except for trivial and inapplicable ones.

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## 3.5 YOUR UPDATED CLASS MODEL AND CLARIFICATION OF HOW THE EXECUTION WORKS

*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 and clear. Indicate clearly in your class model where you applied generics. To do this use tools, PowerPoint, or a combine models as in [this example](https://docs.google.com/spreadsheets/d/1vBmDVtWWh3EX0oehFFLRU0P6eR-fn4d0qVg1-XOUooM/edit?usp=sharing) (which you **are** free to cut and paste from). Insert indications in red to show where generics apply.

Your response replaces this.

## 3.6 WHERE GENERICS ARE IMPLEMENTED

*EVALUATION CRITERION (iii) APPLIES*

### 3.6.1 Class model fragment showing generic class THIS

Explain where and how you applied *generic classes* in your class model.

Generics prove useful for the concept of having a sequence of chords, a collection of Chord objects of any type where the type extends the Chord abstract class. Such a class, ChordSequence in the Builders package was created and is composed of Objects extending Chord, Interval objects and a Note object that indicates the tonal center of the ChordSequence

Diagram

Description automatically generated

### 3.6.2 Code (including test code), input (if applicable), and output showing generics

Explain why the use of *generics* is appropriate here.

The use of Upper Bounded Generics in this class is appropriate as the class needs the flexibility to handle any subtype of the Chord Abstract Class and as this application grows, so will the types that extend Chord – not infinitely but substantially as a matter of fact. The use of Generics allows for effortless grouping of these subtypes in this necessary substructure, a ChordSequence, we might think of as a phrase of a song and eliminate the need for casting these subtypes unless a very type-specific operation is needed.

package Builders;  
  
import AbstractStructures.Chord;  
import javafx.util.Pair;  
  
import java.util.ArrayList;  
import java.util.Collections;  
  
*/\*\*  
 \** ***@author*** *Michael Kramer  
 \* <p>  
 \* CS622 Spring 1, 2022 Advanced Programming Techniques  
 \* <p>  
 \* The purpose of this class is to provide a framework for the concept of a  
 \* harmonic phrase (such as ii-V-I)  
 \*/*public class ChordSequence<E extends Chord> {  
  
 private ArrayList<E> sequence = new ArrayList<>(0); // ArrayList needed for dynamic size  
  
 private ArrayList<Interval> progression = new ArrayList<>(0); // i.e. represents ii-V-I ---> [d, g, c]  
  
 private Note tonalCenter;  
  
 private int size;  
  
 // Constructors, overloaded  
 public ChordSequence(){}  
  
 */\*\*  
 \* The purpose of this method is to construct a ChordSequence object from  
 \* two chords and a given tonal center (i.e. key -> ex: d minor is the  
 \* ii chord of C tonal center)  
 \* <p>Precondition: Two objects extending Chord exist</p>  
 \* <p>Postcondition: A ChordSequence object is created from the  
 \* two Chords given as arguments and the Note object defining tonal center</p>  
 \*/* public ChordSequence(E firstChord, E secondChord, Note tonalCenter){  
  
 Collections.*addAll*(this.sequence, firstChord, secondChord);  
 this.size += 2;  
 this.tonalCenter = tonalCenter;  
 }  
  
 */\*\*  
 \* The purpose of this method is to construct a ChordSequence object from  
 \* three chords and a given tonal center (i.e. key -> ex: d minor is the  
 \* ii chord of C tonal center)  
 \* <p>Precondition: Three objects extending Chord exist</p>  
 \* <p>Postcondition: A ChordSequence object is created from the  
 \* three Chords given as arguments and the Note object defining tonal center</p>  
 \*/* public ChordSequence(E firstChord, E secondChord,  
 E thirdChord, Note tonalCenter){  
  
 Collections.*addAll*(this.sequence, firstChord, secondChord, thirdChord);  
 this.size += 3;  
 this.tonalCenter = tonalCenter;  
 this.setProgression();  
 }  
  
 */\*\*  
 \* The purpose of this method is to construct a ChordSequence object from  
 \* four chords and a given tonal center (i.e. key -> ex: d minor is the  
 \* ii chord of C tonal center)  
 \* <p>Precondition: Four objects extending Chord exist</p>  
 \* <p>Postcondition: A ChordSequence object is created from the  
 \* four Chords given as arguments and the Note object defining tonal center</p>  
 \*/* public ChordSequence(E firstChord, E secondChord,  
 E thirdChord, E fourthChord,  
 Note tonalCenter){  
 Collections.*addAll*(this.sequence, firstChord, secondChord, thirdChord, fourthChord);  
 this.size += 4;  
 this.tonalCenter = tonalCenter;  
 }  
  
 */\*\*  
 \* The purpose of this method is to return the ArrayList sequence  
 \* <p>Precondition: A ChordSequence object exists constructed with  
 \* one of the non empty constructors for substantive return</p>  
 \* <p>Postcondition: An ArrayList of objects E extends Chord are returned</p>  
 \*  
 \** ***@return*** *An ArrayList of objects E extends Chord are returned  
 \*/* public ArrayList<E> getSequence() {  
 return sequence;  
 }  
  
 */\*\*  
 \* The purpose of this method is get the Note representing the tonalCenter  
 \* of this ChordSequence  
 \* <p>Precondition: A ChordSequence has been constructed with a tonalCenter  
 \* passed as argument to constructor</p>  
 \* <p>Postcondition: The Note object is returned</p>  
 \*  
 \** ***@return*** *The Note object is returned  
 \*/* public Note getTonalCenter(){return tonalCenter; }  
  
 public ArrayList<Interval> getProgression() {  
 return progression;  
 }  
  
 */\*\*  
 \* The purpose of this method is to return a chord in the sequence at  
 \* a an index provided as a parameter  
 \* <p>Precondition: A ChordSequence exists</p>  
 \* <p>Postcondition: The Chord at the given index is returned</p>  
 \*  
 \** ***@param*** *index the index of the desired Chord  
 \** ***@return*** *the Chord at the given index passed as an argument  
 \*/* public E getChord(int index)  
 throws IndexOutOfBoundsException{  
 E chordTarget;  
 if(index >= this.size || index < 0){  
 throw new IndexOutOfBoundsException("Error: index must be in range 0-" + (this.size - 1));  
 }  
 else{  
 chordTarget = this.sequence.get(index);  
 }  
 return chordTarget;  
 }  
  
 */\*\*  
 \* The purpose of this method is to return the size of the sequence array  
 \* i.e. how many Chords are in the sequence  
 \* <p>Precondition: A ChordSequence is constructed with types E extend Chord  
 \* objects passed</p>  
 \** ***@return*** *the size of ChordSequence as an int  
 \*/* public int getSize() {  
 return size;  
 }  
  
 */\*\*  
 \* The purpose of this method is to set the Interval structure of the  
 \* progression in relation to the passed tonalCenter. Since these are  
 \* Chords we'll like to render these intervals as roman numerals in the  
 \* UI, where uppercase is utilized for Chords with a major third and  
 \* lower case is utilized for Chords with a minor third.  
 \* <p>Precondition: A ChordSequence is being constructed with a constructor  
 \* that takes arguments</p>  
 \* <p>Postcondition: A ChordSequence is constructed and the Interval sequence  
 \* of the roots of each object of type E extends Chord in relation to  
 \* the Note tonalCenter passed is set</p>  
 \*/* public void setProgression(){  
 for(int i = 0; i < this.sequence.size(); i++){  
 Note note = this.sequence.get(i).getRoot();  
 Interval interval = new Interval(new Pair<>(this.tonalCenter, note));  
 this.progression.add(interval);  
 }  
 }  
}

////////TESTS/////////

package Builders;  
  
import AbstractStructures.Chord;  
import ThreeNoteStructures.MajorTriad;  
import ThreeNoteStructures.MinorTriad;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import java.util.ArrayList;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
*/\*\*  
 \** ***@author*** *Michael Kramer  
 \* <p>  
 \* CS622 Spring 1, 2022 Advanced Programming Techniques  
 \* <p>  
 \* The purpose of this class is to test the construction a class methods of  
 \* the Chord Sequence Test, a generic class where type E extends Chord  
 \*/*class ChordSequenceTest {  
  
 private ChordSequence chordSequence;  
  
 @BeforeEach  
 void setUp(){  
 // I Chord  
 String[] data = new String[]{"c", "e", "g"}; // create input data  
 // Create a ChordBuilder instance from input data called rawData  
 Note[] notes = new Note[data.length];  
 for (int i = 0; i < data.length; i++){  
 try{  
 notes[i] = new Note(data[i]);  
 } catch (InvalidNoteException e){  
 System.*out*.println(e);  
 }  
 }  
 ChordBuilder rawData = new ChordBuilder(notes);  
 MajorTriad one = new MajorTriad(rawData);  
 one.setRoot(rawData.getNotes()[0]);  
 one.setThird(rawData.getNotes()[1]);  
 one.setFifth(rawData.getNotes()[2]);  
 one.setQuality("major triad");  
 one.setInversion("root position");  
  
 // ii Chord  
 String[] data1 = new String[]{"d", "f", "a"}; // create input data  
 // Create a ChordBuilder instance from input data called rawData  
 Note[] notes1 = new Note[data1.length];  
 for (int i = 0; i < data1.length; i++){  
 try{  
 notes1[i] = new Note(data1[i]);  
 } catch (InvalidNoteException e){  
 System.*out*.println(e);  
 }  
 }  
 ChordBuilder rawData1 = new ChordBuilder(notes1);  
 MinorTriad two = new MinorTriad(rawData1);  
 two.setRoot(rawData1.getNotes()[0]);  
 two.setThird(rawData1.getNotes()[1]);  
 two.setFifth(rawData1.getNotes()[2]);  
 two.setQuality("minor triad");  
 two.setInversion("root position");  
  
 // V Chord  
 String[] data2 = new String[]{"g", "b", "d"}; // create input data  
 // Create a ChordBuilder instance from input data called rawData  
 Note[] notes2 = new Note[data2.length];  
 for (int i = 0; i < data2.length; i++){  
 try{  
 notes2[i] = new Note(data2[i]);  
 } catch (InvalidNoteException e){  
 System.*out*.println(e);  
 }  
 }  
 ChordBuilder rawData2 = new ChordBuilder(notes2);  
 MajorTriad five = new MajorTriad(rawData2);  
 five.setRoot(rawData2.getNotes()[0]);  
 five.setThird(rawData2.getNotes()[1]);  
 five.setFifth(rawData2.getNotes()[2]);  
 five.setQuality("major triad");  
 five.setInversion("root position");  
  
 this.chordSequence = new ChordSequence(two, five, one, one.getRoot());  
 }  
  
 */\*\*  
 \* The purpose of this method is to test the method constructor where  
 \* three objects of generic type E extends Chord are passed plus one Note  
 \* argument representing the tonalCenter is passed  
 \* <p>Precondition: A ChordSequence object was created from three  
 \* objects of type E extends Chord and one Note object for tonalCenter</p>  
 \* <p>Postcondition: The Object passed as the third argument was properly  
 \* classified, it's root should correspond with the tonalCenter of this  
 \* ChordSequence</p>  
 \*/* @Test  
 void TestSequenceBuild(){  
  
 Chord one = chordSequence.getChord(2);  
  
 *assertTrue*(one instanceof MajorTriad);  
 *assertEquals*("c", one.getRoot().toString());  
 *assertEquals*("e", ((MajorTriad) one).getThird().toString());  
 *assertEquals*("g", ((MajorTriad) one).getFifth().toString());  
 *assertEquals*("root position", ((MajorTriad) one).getInversion());  
 *assertEquals*("major triad", ((MajorTriad) one).getQuality());  
 *assertEquals*(one.getRoot(), this.chordSequence.getTonalCenter());  
 }  
  
 */\*\*  
 \* The purpose of this method is to test the progression (ex: ii-V-I) is  
 \* properly set for the chords in the setUp function. Here we query the  
 \* int values so (ex: ii-V-I) should correspond to [2, 7, 0] or  
 \* [major second, perfect fifth, unison/octave] without .getIntValue()  
 \* <p>Precondition: A ChordSequence was created with Chords that have  
 \* a ii-V-I relationship</p>  
 \* <p>Postcondition: The ii-V-I relationship is reflected in the  
 \* int values of the Interval relationships/classifications  
 \* of each chords root to the tonalCenter passed to the ChordSequence object</p>  
 \*/* @Test  
 void TestProgressionSet(){  
 ArrayList<Integer> progression = new ArrayList<>(3);  
 for(int i = 0; i < chordSequence.getProgression().size(); i++){  
 Interval interval = (Interval) chordSequence.getProgression().get(i);  
 progression.add(interval.getIntValue());  
 }  
 *assertEquals*("[2, 7, 0]", progression.toString());  
 }  
  
 */\*\*  
 \* The purpose of this method is to ensure an IndexOutOfBoundsException  
 \* is thrown should an attempt to get a Chord out of the range of the  
 \* chordSequence.sequence ArrayList attribute is called  
 \* <p>Precondition: A ChordSequence of length 3 has been instantiated </p>  
 \* <p>Postcondition: An IndexOutOfBoundsException is thrown when  
 \* chordSequence.getChord(int not in range 0-2) is called </p>  
 \*  
 \** ***@return*** *\*/* @Test  
 void TestIndexOutOfBoundsThrown(){  
 Exception thrown = *assertThrows*(IndexOutOfBoundsException.class, () -> {  
 this.chordSequence.getChord(3);});  
 *assertEquals*("Error: index must be in range 0-" + (this.chordSequence.getSize() - 1), thrown.getMessage());  
   
 Exception thrown1 = *assertThrows*(IndexOutOfBoundsException.class, () -> {  
 this.chordSequence.getChord(-1);});  
 *assertEquals*("Error: index must be in range 0-" + (this.chordSequence.getSize() - 1), thrown1.getMessage());  
 }  
   
}

## 3.7 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. Please contact your facilitator in advance if you want to request another transmission process (e.g., github).

## 3.8 INSTRUCTOR’S EVALUATION

