Honors Capstone Reflection

Following the American Psychological Association's Guidelines

Hannah Laws

Southwest Baptist University

For my Capstone, I chose to research the topic of MusicXML and to create a Java program that involves writing and manipulating a MusicXML file. When I began my research, I knew very little about the topic except for a small intro given to me by Dr. Lu in my Database Management class. The first step I took involved looking into XML itself, as MusicXML is a type of XML file.

XML stands for Extensible Markup Language. A markup language is something that uses items called tags (notated <tag>) to classify information within a document and is a way of storing structured data (TechTerms, 2022). In XML, you can define your own tags, as opposed to predefined tags that are found in the commonly known markup language, HTML. Defining tags makes for a more flexible and precise way of storing and accessing specific data. XML is readable by many programs, which makes the exporting and sharing of its data easy. On top of that, XML files are typically very human-readable, meaning that it is easy for humans to interpret since the element names are in plain English.

MusicXML is a specific type of XML that was created by Micheal Good in 2004 (Wikipedia, 2022). It has tags that are related to different elements of a music score. It is specifically used to export and share music scores so that they can be read by many different musical notation softwares, such as Finale and MuseScore (MakeMusic, 2022). All XML files contain a Document Type Definition (DTD) which references a DTD file including information about how the document must be structured and the elements that it is allowed to contain. This information includes the root element, the elements that each element must contain, and the type of data that each element must be of (W3Schools, 2022).

MusicXML files, like all XML files, are structured in a hierarchy. The most difficult part about representing music in a hierarchical fashion is determining what the root element should

be. A real music score has both horizontal and vertical categorizations because one simultaneously reads the musical horizontally via measures and vertically via the parts of the score (voice part, instrument part, etc.). So, there are two root elements that a MusicXML file can have: score-timewise and score-partwise, where score-timewise is representation by measures and score-partwise is representation by parts (W3C Community Group, n.d.). The score-timewise measure elements then contain part elements, and the score-partwise elements then are made up of measure elements. It is most common to use score-partwise, especially with pieces that have many instrument parts or voice parts. Because of this, I decided to look at scores that used score-partwise and write my MusicXMLproject file using score-partwise as the root element.

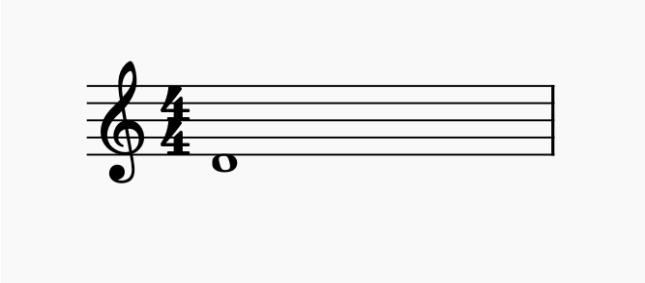
There are many different elements that can be inside the root score-partwise element. Part elements within have unique IDs that are defined near the beginning of the file. These are then referenced later when creating each part's measures. Each measure has an element that keeps track of the measure number. Within the measure element, there is an attributes element, which contains information about the time signature, the key, and the clef. The measure element also contains the note element, which has information about the pitch, the pitch quality (sharp, flat, natural), and note duration. There are many other elements within MusicXML that can produce very complicated scores and pieces of music, but these are the main elements that I used in my project.

Once I felt like I had a good understanding of the elements of MusicXML, I successfully wrote a "Hello World!" program (this does not actually output the words, "Hello World!", but instead, it includes the most basic elements of MusicXML). Below is the code and the output for this program. In the file you can see many of the different elements discussed above.

helloWorld.musicxml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML 4.0 Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="4.0">
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>0</fifths>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>D</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
    </measure>
  </part>
</score-partwise>
```

Output in MuseScore



Next, I downloaded several large scores via MusicXML and compared the file to what the score actually looked like in MuseScore, a free and widely-used musical notation software. This led me to understand many of the more complicated subtleties about MusicXML. After I did this, I decided to produce a short piece of music by writing a MusicXML file. I wrote a simple melody with two parts, Soprano and Bass, and I based it off of a short piece of music I wrote for my Music Theory II class. It proved to be quite a challenge to get the score to do exactly what I wanted it to.

Once I had created this file, I spoke to my Capstone supervisor, Dr. Lu, about what he thought I should do moving forward. He gave me the idea of creating a program to transpose my

MusicXML file. In doing this, I would be able to get more experience manipulating MusicXML files and get an even better understanding of the hierarchy. Not only that, but I could truly combine my knowledge of programming algorithmically and music theory. So, I decided to run with this idea.

For my final project, I created a Java program to transpose. My first challenge was getting the program to access the MusicXML file so that it could read my file's current notes. I did some research and decided to use DocumentBuilderFactory, a Java object that defines a factory API which allows applications to get access to a parser that produces trees from XML documents. (Oracle, 2020). Once this is created, you can define a document and then parse it using DocumentBuilder. Because MusicXML is a type of XML file, this class worked for MusicXML, too. From there, I was able to create NodeLists which pull specific elements in my file by specifying an element name. By the ability to get child nodes from these lists, I was able to loop through each element list and access each pitch, pitch alter,

My next challenge was creating an algorithm that could change the note name and alter. To better understand what I needed to do before I began coding, I wrote a file of pseudocode. I found that there are many special cases, usually involving transpositions around E and F or B and C. This is because there is only a half step between these notes, instead of a whole step such as the one between D and E. Because of this, my algorithm included many if and else statements. Below is a screenshot of the beginning of my pseudocode file, where I tried to understand the first steps of transposition.

```
pseudocode for actual transposing

user input for what key they want to transpose to (or number of half steps up/down?)
-ask transpose up or down

check current key (fifths element)
-if positive int, in sharps
-if negative int, in flats
-if 0, then in they key of C

make an array with the circle of fifths and their keys =
    ex. hSteps = 3
        direction = down

    loop(pitches.length)


    0       1       2       3       4       5           6           7               8       9         10        11

[[c, 0], [g, 1], [d, 2], [a, 3], [e, 4], [b, 5], [f#/gb, 6/-6], [c#/db, 7/-5], [ab, -4], [eb, -3], [bb, -2], [f, -1]]

3 half steps: key of G to B flat ---array index 1 to 10 (12), fifths: 1 to -2
              key of A to C ---array index 3 to 0, fifths: 3 to 0
              key of Db to E ---array index 7 to 4, fifths: -5 to 4

//say it's in the key of B Flat currently (fifths = -2)
//user wants it to be in key of G (fifths = 1)
-change fifths element to user input
-


if alter = -1, then flatted (down a step, could be a natural)
if alter = 1, then sharped (up a step, could be a natural)
```

When I began coding in Java, I broke up the pseudocode into three static methods:

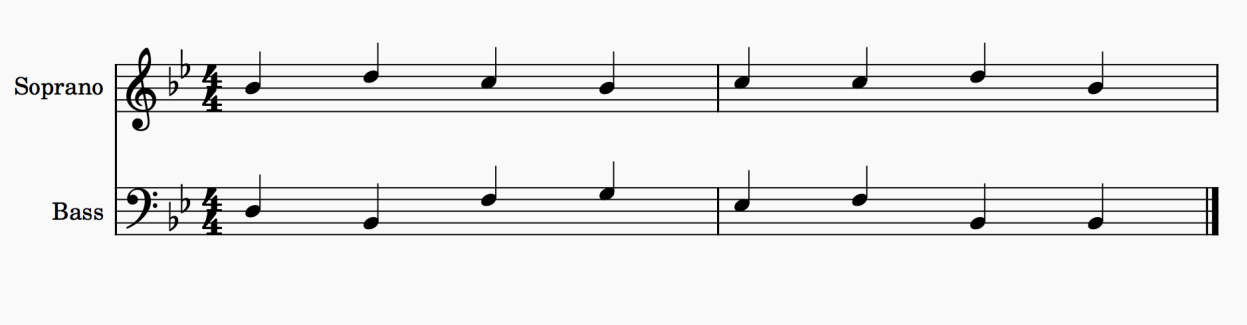
- getNewFifths(String currentFifths, String direction): This method returns a String and is responsible for changing the key signature. It takes in the current element “Fifths” which corresponds to the key signature, as well as the direction so that it can transpose up or down.
- getNewStep(String currentNote, String direction): This method is responsible for just changing the note name in transposition. It takes in the current note name (step element) and the direction of transposition and returns a String of the new note name.
- getNewNote(String step, String alter, String octave, String direction): This method is responsible for getting all of the transposed note information, such as the new note name (step), alter (quality, such as flat, sharp, or natural), and octave. It takes in the current

values of each of these and the direction of transposition and calculates the change based on these. It calls the getNewStep() method to get the new note name for each note. The main method takes user input to get the desired direction of transposition (up or down) and the desired number of half steps moved in that direction. Therefore, each note is read one by one and transposed, calling each of the above static methods in a loop.

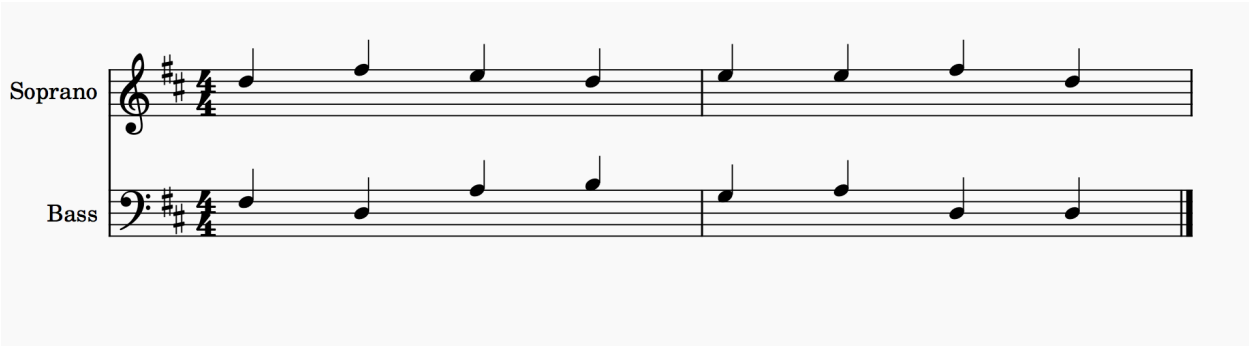
My final challenge was getting my program to write the new, transposed notes back to a file. After some more research, I ended up creating a static method called writeXML() using the Java class TransformerFactory to create a Transformer. A Transformer instance can transform the original node tree into a resultant node tree (Oracle, 2020). Once all of the elements were transposed and saved to the resultant tree, the transformer finally writes to a specified source using the transform method. In my final program, the user can specify the file to save the transposed piece to, so you can choose to save it to the original file or to a completely new file to keep the original version intact.

After much frustration, a lot of testing, and hours rewriting, my Java transpose file finally worked as it was supposed to. I tested the transposition by several different steps and into every key. Below are some screenshots of the results of this testing. As you can see, I began with a melody in the key of B-flat major and transposed it into the key of G major.

Before:



After:



It was so exciting getting to manipulate what looks like a simple file and get to see and hear it turn into a piece of music. I overall absolutely loved the experience of learning about something that tied together my loves of computer science and music. To view my work, please visit https://github.com/hannah-laws/MusicXMLCapstone!

Sources

MakeMusic. (2022). MusicXML:The standard open format for exchanging digital sheet music. MakeMusic. https://www.musicxml.com/

Oracle. (2020). Class DocumentBuilderFactory. Java Platform, Standard Edition 7 API Specification. https://docs.oracle.com/javase/7/docs/api/javax/xml/parsers/DocumentBuilderFactory.html

Oracle. (2020). Class Tramsformer. Java Platform, Standard Edition 7 API Specification. https://docs.oracle.com/javase/7/docs/api/javax/xml/transform/Transformer.html

TechTerms. (2022). Markup Language Definition. TechTerms. https://techterms.com/definition/markup_language

W3C Community Group. (n.d.). MusicXML 4.0 Tutorial. W3. https://www.w3.org/2021/06/musicxml40/tutorial/hello-world/

W3Schools. (2022). XML DTD. W3Schools. https://www.w3schools.com/xml/xml_dtd.asp

Wikipedia. (14 January 2022). MusicXML. Wikipedia. https://en.wikipedia.org/wiki/MusicXML