# EECS 2030: Programming Exam 2
Fall 2021

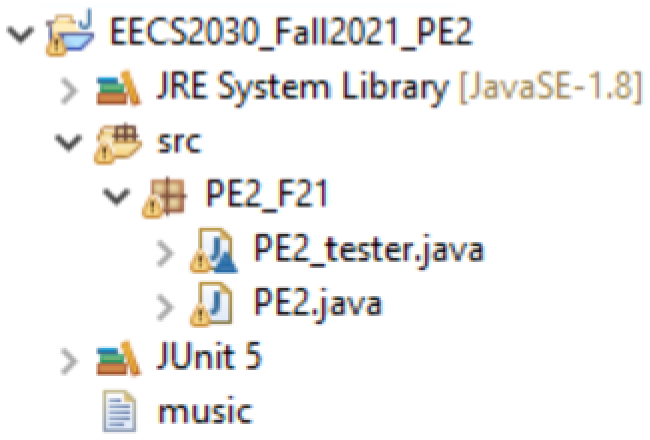
Deadline: Tuesday Dec 7th, 2021 11:59pm

In this programming exam, we are going to assess your understanding of

- Objects relationships (such as inheritance, composition, and aggregation).
- the difference between interfaces, abstract classes, and classes.
- polymorphism
- exception handling
- encapsulation
- java documentation API and the ability to create javaDoc

**Setting up the environment:**

- Download the attached zip file.
- Open eclipse.
- Click on File and select Import
- Choose Existing Projects into Workspace and click Next
- Click on Select Archive File and then Browse.
- Find the zip file that you have already downloaded and click Finish
- Please make sure that you do not already have a project called EECS20_Fall2021_PE2
- Now you should see the following settings in the package explorer.



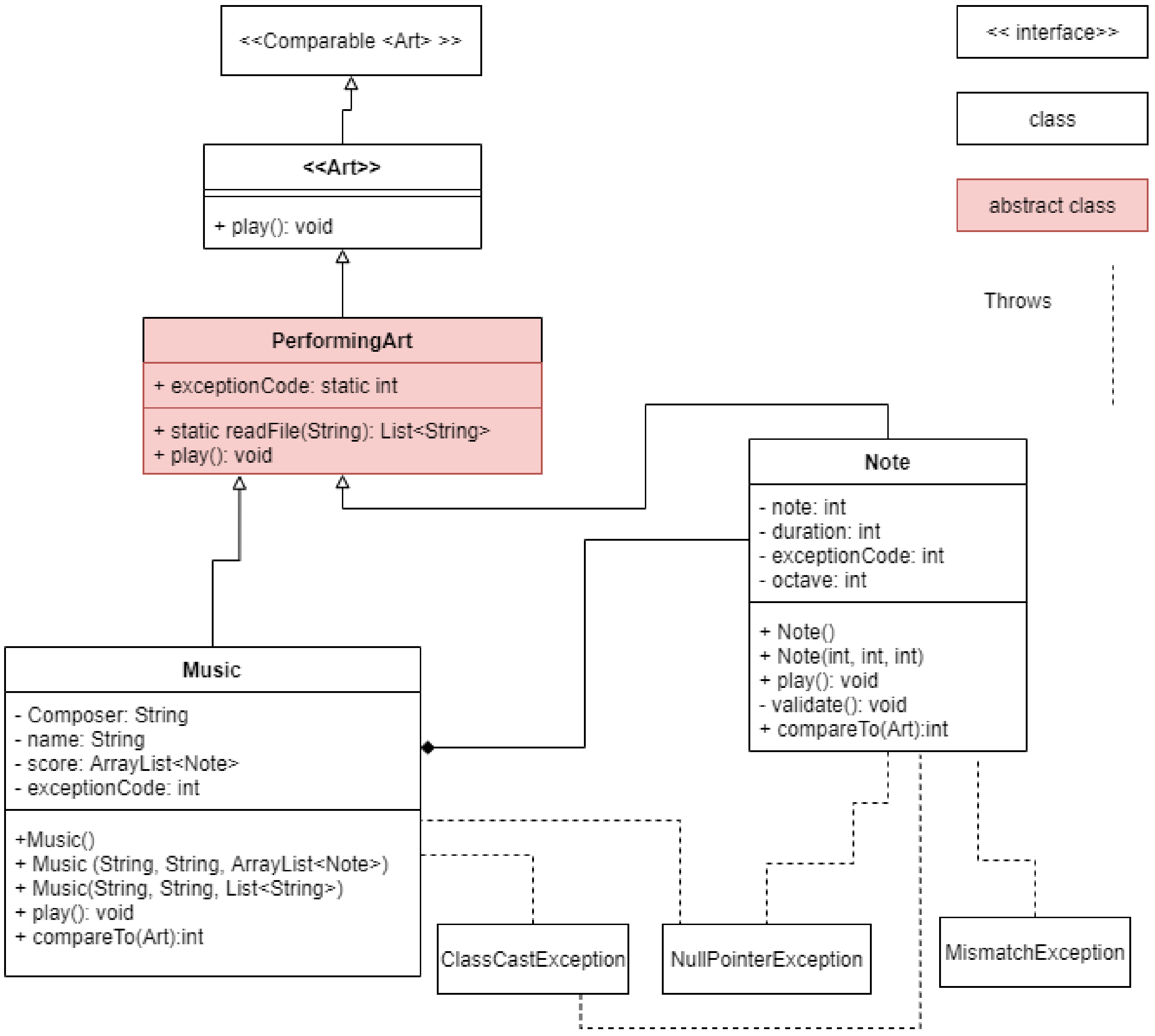
```
EECS2030_Fall2021_PE2
  JRE System Library [JavaSE-1.8]
  src
    PE2_F21
      PE2_tester.java
      PE2.java
  JUnit 5
  music
```

Before you start, I would like to remind you that this assignment is an **exam** that you take home. This means you can neither discuss your solution with anybody including your peers nor get help from them. Therefore, please carefully read the statement at the top of `PE2.java` before you start doing the assignment. When you finished the assignment, you should sign this statement by writing your information (full name and student number) in the gap provided. Please be aware that your assignment WILL NOT be graded if you do not sign the declaration. Also, I would like to inform you that your code will be checked by both a plagiarism detection tool and us to ensure your solution to this problem is unique and not similar to others'.

Refusing to follow the policy stated above, is a violation of academic integrity that will have a serious consequence. Please read about the definition of academic integrity and the penalty that one may face, in case they violate academic integrity in https://lassonde.yorku.ca/student-life/academic-support/academic-honesty-integrity .

**Problem Description**:

For this assignment, you are required to write an object-oriented program that reads a file that contains music notes and play the music. The following UML shows the architecture of this program. The legend of the diagram explains how the objects should be defined. Therefore, if you look at PE2.java, you will not find any useful code, as you are supposed to read this diagram and implement the code in PE2.java.



As can be seen from the diagram above, this program comprised of 8 classes and interfaces, out of which 3 has been already defined by Java. These three are `Comparable` interface, `ClassCastException` and `NullPointerException` class. The rest should be implemented by you. Here you can find the description of some of the classes.

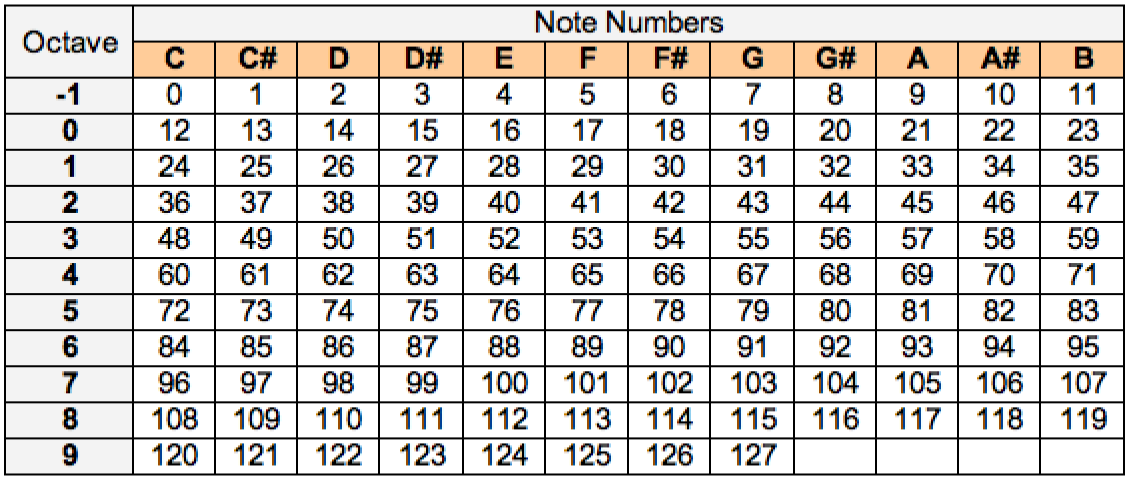
Note:

For simplicity, in this program a note is known by three properties, the `note`, the `duration` in millisecond that the note is played, and the `octave` in which the note is played. A valid value for a note is an integer number in the interval of [0, 127]. A valid value for an octave is a number between -1 and 9. For example, note=60, represent the middle C in the Piano in octave 4. The duration can be any positive integer number, however a small duration my not be practical. For example, duration = 10 ms, is too quick to let human hear the note.
The fourth attribute of class note is `exceptiopnCode` that gets 0, -1 , -2 , -3 in case no exception, `NullPointerException` or `ClassCastException` and `MismatchException` is thrown respectively. The reason that these exceptions are thrown will be discussed later.

`The Constructors`: This class needs two constructors, on default and one overloaded constructor. The overloaded constructor initializes all the attributes using the input parameters and assigns the default value (i.e., zero) to the `exceptionCode` attribute. The default constructor initializes the attributes using the default values.

`compareTo()`: Two notes are said to be the same if both the note and octave are the same. If two notes are not the same, they are compared with their note values only. Therefore, the difference between the note values represents if the note is smaller or greater than the other note. It is expected that the possible exceptions that may be thrown by this method, is handled in the same method. In case `NullPointerException` or `ClassCastException` is thrown the value of `exceptionCode` should change to -1 and -2 respectively.

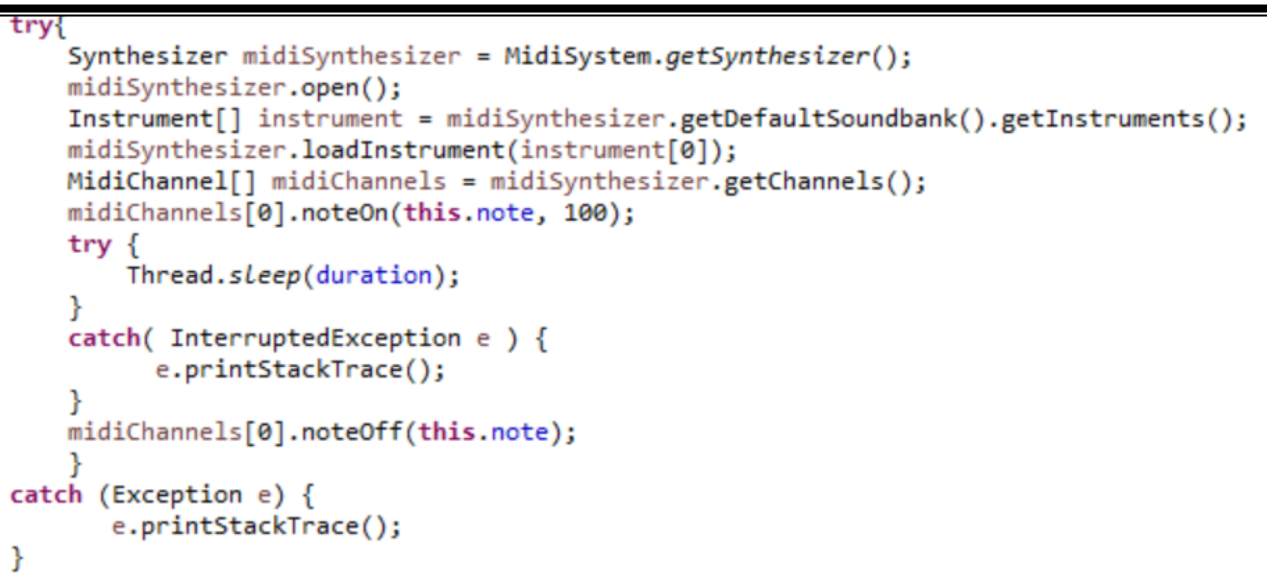
`validate()`: This method validates the note based on the octave value according to Table below. For example, if octave = -1, the valid value for a note in this octave is between zero and 11. This method throws `MismatchException` with “Octave/Note Mismatch” message, in case the octave value does not match the note value. The Exception should be written by you. Please note that this method works on the instance variable of this class and therefore does not need any input arguments.



| Octave | Note Numbers | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | C | C# | D | D# | E | F | F# | G | G# | A | A# | B |
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 0 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 1 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |
| 2 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 3 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| 4 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 |
| 5 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 |
| 6 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| 7 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 |
| 8 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 |
| 9 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | | | | |

*Table 1: Taken from [https://content.instructables.com/ORIG/FWX/NBXG/H4AFZWE7/FWXNBXGH4AFZWE7.png]*

`play()`: The code to play a note is given below. You need to copy the code to this method and add required code to complete this method. This method plays a note in a specific octave with a certain duration. All these information is provided by the instance variables of this class, which are initialized in the constructor.


```
try{
    Synthesizer midiSynthesizer = MidiSystem.getSynthesizer();
    midiSynthesizer.open();
    Instrument[] instrument = midiSynthesizer.getDefaultSoundbank().getInstruments();
    midiSynthesizer.loadInstrument(instrument[0]);
    MidiChannel[] midiChannels = midiSynthesizer.getChannels();
    midiChannels[0].noteOn(this.note, 100);
    try {
        Thread.sleep(duration);
    }
    catch( InterruptedException e ) {
          e.printStackTrace();
    }
    midiChannels[0].noteOff(this.note);
    }
catch (Exception e) {
       e.printStackTrace();
}
```
Before a node is played, it should be validated. This is done by `validate()` method. Remember that this method may throw an exception. Your job is to handle this exception in the `play()` method by setting the `exceptionCode` attribute to -3 and print the message thrown by `validate()` method. In this case no note should be played.

Music:

This class has four instance variables that show the name of the `composer` of this piece of music, the `name` of the music itself, an `arrayList` of all the notes that comprises this music and `exceptionCode` that shows the type of the exceptions that are thrown by the methods of this class.

`The constructors`: This class has three constructors, the default constructor that initializes the attributes as expected by the default values, the first overloaded constructor that gets the name of the composer, music name and the list of the notes to initialize the attributes and the second overloaded constructor that gets the name of the composer, music and a list of strings, where the string contains a note, its duration and octave. This constructor is used when the notes are read from a file.

`Play()`: this method plays all the notes that are stored in the `score` arrayList.

`compareTo()`: This method works slightly different from what we expect from compareTo. It returns zero, if the name and the composer of the music is the same and both the scores are completely the same. If not, -1 is returned. It is expected that the possible exceptions that may be thrown by this method, is handled in the same method. In case `NullPointerException` or `ClassCastException` is thrown the value of `exceptionCode` should change to -1 and -2 respectively.

PerformingArt:

This class has one static variable that represents the state of reading from a file. If a file is read successfully, the value of this attribute is zero, otherwise it is -1;

`readFile()`: This method reads the file, whose name is passed to the method as a string input argument. The method returns an arrayList of String, where each string is one line of the given file. In each line of the file, there are 3 numbers that represent the note, duration, and octave. You can trust the file to follow this format for all its line. The numbers in each line are separated by one or more blanks. In case reading the file is not successful, `exceptionCode` gets -1 otherwise it gets 0.

play(): This method should be able to play any type of performing arts.

Please note that encapsulating data is important in this assignment. Your code will be evaluated on the correct implementation of encapsulation. Since the testing is done automatically, improper implementation of encapsulation might affect the successful execution of other test cases.

When you finished the code, change the notes, duration, and octave in the given file, run the code and enjoy the music that you have composed.

## Marking Scheme:

- [70 points]: Your code is tested with 35 test cases. each test case has 2 points. Only a subset of test cases was provided as we are assessing you on how well you have understood the concept of object-orient3ed programming. Please note that you should not submit a code that contains compiler errors. All these 70 points are lost if the code has compiler error.
- [10 points]: This is awarded to you if you have used polymorphism correctly such that the code reuse is apparent in your implementation.
- [20 points]: For inner documentation, javaDoc and proper code style.

## Checklist:

Before you submit, check this checklist to ensure that you have done all we asked you to do.

- ☐ I have finished implementing all the classes and interfaces.
- ☐ I have signed the statement on academic honesty.
- ☐ I have removed/commented out the code that I have added in `main()` method, if applicable.
- ☐ I have tested my code with more JUnit test cases, which I have written myself as I am aware that a small sample of test cases have been provided.

## What and how to Submit:

- You only submit one file, which is the completed PE2.java file.
- Submit your solution in eClass, where you have downloaded these instructions.