Lab5.main & .parseNote

① **Avoid catching generic Exceptions**. This can mask errors in your code. For example, if you know that a block of code may produce a FileNotFound exception, but you catch a generic Exception instead, you may miss the fact that NullPointerExceptions are also occurring

② **Try to catch exceptions as close to the source as possible.** Although catching exceptions at a high level does keep a program from crashing completely, it can also make localizing a bug very difficult. Instead, during the debug stage, allow your program to crash, then find the source of the exception and fix it or catch it close to the source.

③ **Provide meaningful feedback when handling exceptions**. Don't just print a stacktrace – explain why the error is occurring based on what is happening in your program at that point. For example, instead of catching a FileNotFound exception and printing a stack trace, print "Could not open the note file. File "+filename+" not found."

④ **Use Scanner or Buffered Reader**. To simplify reading text-based files, use built-in classes that can handle operating-system dependent line endings (e.g. Windows uses "\r\n", Linux uses "\n", Mac used "\r" before switching to a Unix core (and thus "\n"), etc.)

⑤ **Add notes while scanning file.** To simplify reading the file, add the notes directly while you read the file. Then there is no need for an extra variable (even a local one).

⑥ **Simplify if.** Many if constructs can be simplified. e.g. `if( <expr> ) { /* do nothing */ } else { myMethod();}` can be simplified to `if(!<expr>) { myMethod(); }`

⑧ **Handle line format errors correctly.** This method throws exceptions instead of returning null if an error occurs. It should detect the errors, print a warning message, and return null. (The warning message can also be printed in the main method.)

-1 Some errors caught, but not all

-2 No errors caught

⑨ -2 **Close file.** Remember to close the file when you are done with it! This is best done in a finally block, or with a try/using block. (-1 if null point exception on close)

Guitar

⑨ **Check range of input to Guitar(int, float) constructor.** The API specifies that this constructor will check the input and, if the input is wrong, set it to default value and printing a warning message.

-3 – no checks at all

-1 – Always use default value if an incorrect value is supplied.

-1 – No error message printed.

-1 – No default values supplied.

⑩ -1 **Avoid magic numbers.** Instance variables are available for both DEFAULT_DECAY_RATE and DEFAULT_SAMPLE_RATE.

⑪ -0 **Preferred Guitar() implementation.** You can increase maintainability by reusing the Guitar(int, float) method to implement the Guitar method. Just use this code:

```
public Guitar() {
   this(DEFAULT_SAMPLE_RATE, DEFAULT_DECAY_RATE);
}
```

⑫ -1 **Use appropriate primitive types.** The number of samples must be an integer. In the end, you will have a list of samples, and its size is an integer. The purpose of the numberOfSamples variable is to hold this size until the list is filled. (Also: Don't use wrappers unless needed. They imply a function call every time you read or write their value. Not a huge cost, but why do it if you don't have to?)

⑬ -2 **Must initialize `notes` list.** The instance variable List<Note> notes; is not initialized above, so both constructors need to initialize it. If one of them doesn't, a call to addNote will produce a null-pointer exception. See also ⑪ for a nice way to avoid this kind of error.

⑭ -2 **Call constructor from constructor, but not using new.** Using the "new" operator inside a constructor will call another constructor, but the person calling you will not get that object. They will get the object that was created before the code of your constructor was called. To call a constructor from another constructor, use the syntax shown in ⑪.

Everywhere

⑮ -2 **Program to interfaces rather than concrete classes where possible**. Instead of creating a reference variable `ArrayList<String> lines = new ArrayList<String>();`, use a reference to a List: `List<String> lines = new ArrayList<String>();`. This will give you the flexibility to change the implementation later if you need to.