Homework # 2

due Monday, September 18, 10:00 PM

In Homework # 1, you implemented the Note ADT. For this assignment, you will build on them to implement a Song ADT, a sequence of notes and use this in a program to play a juke box.

1 Concerning the Song ADT

The Song ADT is a variant of the Sequence ADT from the textbook (Section 3.3 on pages 145–158 (142–155, 3rd ed.)). The data structure is used in the same way, but some methods are different:

insert Instead of addBefore and addAfter, there is a single method insert that functions the same as addBefore except where there is no current item, in which case it inserts at the end, rather than at the beginning.

insertAll Instead of addAll, there is a method insertAll that inserts another sequence before the current item (if any) or at the end (otherwise). The first item (if any) of the inserted sequence will become the current item. If the inserted sequence is empty, there is no effect.

hasCurrent Instead of isCurrent, there is a method hasCurrent which has precisely the same behavior.

We give you a partial implementation to start with.

We also have some methods specifically for songs:

Song(String, int, int) Initialize the song’s name, “beats per minute” (BPM) and the initial capacity rather than use default values. The name cannot be null and the BPM must be in the range [20, 1000]. Don’t use these specific values in your code—make sure to use the named constants. The initial capacity must not be negative. There are also constructors that take fewer arguments. It is simplest if all such constructors call this constructor with default values using Java’s this(...) syntax.

get/set Name/BPM We have the normal getters and setters for the name and BPM. The setters must check their parameters.

getDuration The duration of a song is the sum of the duration of all its notes.

stretch(double) Stretch all notes in the song by the given amount. NB: recall that notes are immutable and so you must assign new notes into the array. It’s allowed to leave the Song partially stretched if a problem is found partway through. You can also let an exception thrown by a nested call propagate back to your caller rather than throw your own exception.

transpose(int) Similarly, transpose all the notes in the song.

You are encouraged to write your own toString method which you will find useful when debugging the ADT, but you are not required to define such a method, nor are there any requirements for what the result should look like.

Unlike the lab exercise, null note values are permitted. Nulls should count toward the size, and a null note can be the current note.

Unlike the textbook, we recommend that you do not use System.arraycopy. While it does indeed run faster than doing a loop, it’s not very clear what it is doing. Use a “for” loop for clarity.
2 Concerning Invariants and Assertions

The sequence-specific data structure makes use of two integer fields, a boolean field and an array field. There are certain configurations that make no sense. Thus it is recommended to define and check object invariants. See page 126 (3rd ed. p. 123) in the textbook. We also have song-specific fields, namely the name and BPM; these also have restrictions on their values.

There are conventions and Java language features to help you codify and test the invariant. For this homework, you should implement the class invariant as a private boolean method named _wellFormed(). Then the beginning of every public method should have code as follows:

    assert _wellFormed() : "Invariant failed at start";

and at the end of any public method that changes any field, there must be the following line, right before the end:

    assert _wellFormed() : "Invariant failed at end";

We have placed these lines in the code in the skeleton file for your convenience. Do not remove them!

An invariant may be expensive to check. Therefore in Java, assertions are turned off by default. For that reason, an assertion should not include some side-effect that needs to happen. Don’t rely on an assertion to check (say) that a parameter is good.

As always, you should turn assertions on while running the main test suite. With the command line, this is done by passing the flag -ea to the java executor. In Eclipse, assertions are enabled by adding -ea (don’t forget the hyphen!) as a “VM argument” on the “Arguments” tab of the run configuration.

3 Concerning the JukeBox Program

We provide most of a program that reads in songs from files and plays them. Two sections of the code are omitted; these require you to use the Song ADT that you are implementing.

4 Files

In the git repository for this assignment, we provide the solution to Homework # 1 (in a JAR file) and the following files:

- **src/TestSong.java** JUnit test cases. Do not modify this file.
- **src/TestInvariantChecker** Test Song._wellFormed
- **src/edu/uwm/cs351/Song.java** A skeleton implementation of the ADT.
- **src/edu/uwm/cs351/Jukebox.java** Incomplete program to play songs.