CST8132 Lab 5 Guitar Simulation II

Due Date

See your Lab Blackboard for Due date

Description

The purpose of this Assignment is to continue to develop our thinking about Java Objects and how they communicate, as well as the main Object-Oriented concepts and Java, including Javadoc. The activity will be enhancing the guitar simulation from Lab 3 as described in the Detailed Steps section below. **You are required** to write appropriate **Javadoc** comments in your implementation, and generate the Javadoc pages to be included in your project folder for submission (See below for the list of java files in which you need to put Javadoc comments). **You are required** to ensure your Java code is written according to acceptable coding conventions, for example the Sun/Oracle Java Coding Conventions or the Google Java Coding Conventions. **You are required** to author your own version of a UML class diagram for the Guitar, Wire, Peg, GuitarFrame, and RightHand classes.

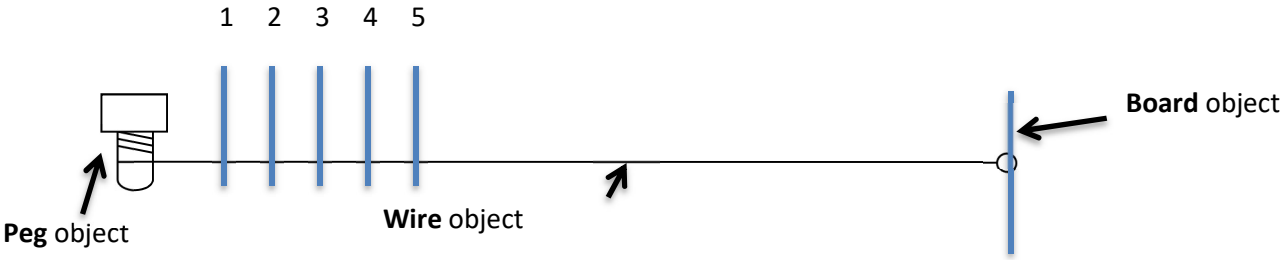
The main modifications you will make to your solution to Lab 3 are

- Use the Singleton design pattern (See Hybrid 5) to handle the **Board** property of the **Wire**s
- Enhance the guitar simulation with frets and strumming behavior
- Create a **RightHand** class that implements a Java interface

Recall that a Java interface is simply a set of abstract methods and perhaps some final constants (because all methods in an interface are abstract, the keyword abstract is left out). When a class implements an interface, it means the instances of that class are guaranteed to provide all of the methods in the interface. In this guitar simulation, there are some Java interfaces provided to you:

- **GuitarPlayerRightHand**: this interface lists the methods that the right hand will need to be able to do. You will write Java code for the **RightHand** class, which will implement this interface. Your **RightHand** object will be able to do everything that a **GuitarPlayerRightHand** can do.
- **GuitarPlayerLeftHand**: this interface lists the methods that the left hand will need to be able to do. The **LeftHand** class is provided for you, and it implements the **GuitarPlayerLeftHand** interface, or in other words, it can do everything a **GuitarPlayerLeftHand** can do.

The new guitar simulation will include the ability to *fret* the **Wire**s. A fret is a metal ridge on the **Board** under the **Wire**s such that normally the **Wire**s don't touch. However, when a **Wire** is pushed against a fret (say fret number 2) and then **pluck**ed, the **Wire** is now effectively shorter, and it now makes a sound as if the **Wire**'s tension had increased by **2** (if it was fret number **2**). The tension doesn't increase, but the sound made now depends on the **fretNum** property of the **Wire**. See the diagram below:



You will be provided with a library with parts of the enhanced solution, especially the parts that involve knowledge of music, like **LeftSideOfGuitar** and **LeftHand**. It is not intended for you to need to study music or have musical knowledge, or guitar knowledge (other than what this document directly tells you).

Our enhanced guitar simulation will be capable of playing songs, such as John Lennon's "Imagine". **ImagineTest** (from Blackboard) is a class that contains the **main** method for our simulation, similar to **SimpleSong** from Lab 3.

Detailed Steps

1. Create a new Java Project and create a **guitar** package in it. All of your Java code for this assignment will belong to this package.
2. As in Lab 3, download and "add library" **edu.ac.guitarlib5** (from Blackboard) into your project.
3. Create a new **GuitarFrame** class so that it supports the Singleton Design Pattern:
    a. Make the **GuitarFrame** constructor **private** so that no other class can instantiate it directly
    b. Add a **board** property of type **Board** (edu.ac.guitarlib.Board) that is instantiated to a new single **Board** object.
    c. Add a **static** method **getInstance** that returns a reference to the one **GuitarFrame** instance you added in the previous step
    d. Add a method **getBoard** that returns a reference to **board**.
    e. Questions you should be able to answer:
        i. Why did we make the constructor private?
        ii. Why must the method **getInstance** be static?
        iii. How can a **Wire** get a reference to the one and only one **Board** object?
4. Make the following changes to the Wire class:
    a. add a constructor that takes as a parameter just a **tension** int for its default **tension** value, so that a Wire can be created without passing a reference to a **Board** object (which will be retrieved using the Singleton design pattern)
    b. set the **board** property by making the appropriate method call involving the **Board** class
    c. add a **fretNum** property which represents the fret that the string is temporarily being held against, defaulting to 0. When a Wire is temporarily held against a fret, the sound

will be different. While it is held against the fret, the sound will be as if the tension were tighter by the same number as the fret number. So when the **Wire** invokes the **Board**'s **soundNote** method, the parameter passed to the method will now be **tension+fretNum**.

   d. add a **fret(int fretNum)** method that will set the **fretNum** property of the string to some non-negative integer. When this method is called, it will mean that the Wire is being held against that fret given by the **fretNum**. When this method is called with **fretNum** of **0**, it will mean the **Wire** is not held against a fret, and subsequently the sound will be given by **tension+0**, which is what we expect.

5. Make the following changes to the Guitar class:
   a. Remove the **Board** property from the **Guitar** class. We will be using the Singleton design pattern so that the **Wire** objects can retrieve a reference to the one and only one **Board** object.
   b. Add a **fret(wireNum, fretNum)** behavior to the guitar, so that the guitar will call the selected **Wire**'s **fret** method, passing that method the appropriate fret number.
   c. Add a behavior to the guitar called **hammerOn(wireNum,fretNum)** which will fret the **Wire** at the **fretNum** and **pluck** it at the same time (that is, one method call immediately after the other, with no **pause** in between). This guitar behavior will be needed for the Imagine performance.
6. Create **RightHand** class that implements **GuitarPlayerRightHand**. Study the **GuitarPlayerRightHand** interface (let Eclipse "Add unimplemented methods" and mouse over the method to display javadoc) and implement the required methods.
   a. The **RightHand** class will have a **guitar** property of type **Guitar** and this property will be set by the **RightHand** constructor (we can think of this as putting the guitar in the right hand).
   b. The RightHand class needs to implement all of the methods specified by the GuitarPlayerRightHand interface. In other words, when you run your program simulation, your **RightHand** object needs to be able to do everything that a **GuitarPlayerRightHand** can do.
7. At this point **ImagineTest** class may give you an error **LeftHand(guitar)** undefined. What interface should **Guitar** implement? Fix this error.
8. You should now have everything needed to compile and run **ImagineTest.**java
9. Go back over your code that you wrote and fix any indentation problems, and generally be sure your code is as readable as possible. Be sure your code complies to the Sun/Oracle Java Coding Conventions or the Google Java Coding Conventions.

IMPORTANT: Document your code using the Javadoc system by writing the Javadoc comments for every class, interface, instance variable, and method, and generate the Javadoc for your project. The files for which you need to write Javadoc comments are:

- Guitar.java
- GuitarFrame.java
- Wire.java

- Peg.java
- RightHand.java

Submission

The submission process for this assignment is the same as before, but be sure your Javadoc output directory, and UML class diagram file (format: jpg, png, or PowerPoint), is included:

1. demonstrate your simulation to your lab instructor.
2. submit a zip archive of your deliverables folder: Lastname_Firstname_CST8132_Lab5.zip

Grading Scheme

Demonstration required: 5 marks (without demonstration you assignment receives an overall grade of zero)

Properly written Java code for the simulation described above: 5 marks

Properly written Javadoc comments and Javadoc output: 5 marks

UML class diagram for the following Classes: Guitar, Wire, Peg, GuitarFrame, RightHand: 5 marks