CS2163 Java Homework 10 requirement

Homework 10 needs to be finished on the Greenfoot platform. Homework 10 does not involve Eclipse.

**Follow the steps below to finish homework 10:**

* work on these four exercises in textbook: 10.45, 10.46, 10.48 and 10.49 (exercise 10.47 is NOT included). It requires you to create a new scenario, and name this scenario “johndoe-mp3”. Replace johndoe with your name.
* provide the solutions for all these 4 exercises in one Greenfoot scenario. You need to create a new scenario and start working on the solutions from scratch.
* for the mp3 file that is to be played, you can get an existing mp3 file, or generate your own mp3 file using software Audacity, or using the sound recorder in Greenfoot.
* refer to section 10.6 that explains class GreenfootSound and how to use this class and control audio playing.
* for exercise 10.49, you need to display the volume either in a vertical bar or a horizontal bar. This volume bar is for display volume only, and nothing should happen if you click on the bar, because it is not supposed to respond to user clicking on the bar. The lowest value is 0, and the highest value is 100. According to class GreenfootSound, its API method getVolume () indicates that the volume is between 0 and 100. Alone with the bar, you also need to display the current volume level in numeric value, such as number 97 in the picture below. A sample horizontal volume bar and a vertical bar are listed below, and you don’t need to implement the dark background in these two sample pictures, when you implement the volume bar in your homework.



, or this vertical bar:



* the step for increasing and decreasing volume should be set to 2 or 5. The initial volume level should be set to 50, which is right in the middle of the volume.
* **the efficiency of your solution is very important**. When the program runs, the initial volume should be set to 50, with corresponding volume bar and volume number displayed. After that, an efficient solution will update the volume display bar and change the volume of the song, only when user has clicked the up or down volume button. Therefore, if you solution keep re-drawing the volume bar, or keep re-displaying the volume number, or keep re-setting the volume level of the song, even when user has not clicked on any button, then that is not an efficient solution, and you should avoid providing such an inefficient solution.
* the solutions of these four exercises need to co-exist in one scenario, and you don’t need to comment out any finished exercise when you are debugging the next exercise.
* zip the entire scenario folder “johndoe-mp3”, and rename the zip file as “***JohnDoeHw10.zip***”, where JohnDoe needs to be replaced by your first and last name
* submit the zip file ***JohnDoeHw10.zip*** to Moodle “homework 10 drop box”.

**Additional requirements:**

* there should be no static data defined in any of the classes in your homework.
* all data in all classes must be private.
* if you declare a reference object, this object must be initialized with a new statement.
* Any class that you define needs to have source code in their act() methods, i.e., you cannot define a class and you don’t place any source code in its act() method.
* You cannot place all source code in one class and leave all other classes with an empty act() method in each class.
* If you violate any of these additional requirements, you will receive 0 in this homework.

**Hint: follow the “singleton” design pattern**

One key issue in this homework it to apply the “singleton” design pattern. Singleton design pattern means you only have one instance created of a class type, and in this homework, you should create only one instance of the GreenfootSound class, which is the only music piece we are manipulating with play/pause button, and the volume up and down button. Then to manipulate this only instance among different classes/files, you need to resort to the encapsulation principle, and create public getter and/or setters for this object, so that other classes can reply on the public getter/setter to access this object.

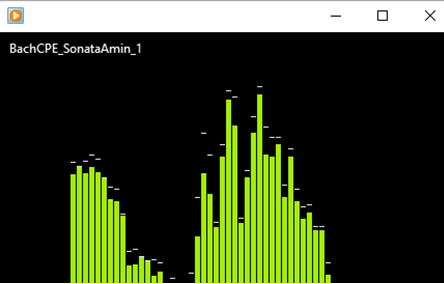
One common mistake in this homework I have seen is: student creates separate instances of the GreenfootSound objects in different classes, and students fail to notice that these instances are independent of each other, for example, one instance of GreenfootSound object is to be controlled by the play/pause button, and the other instance is to be controlled by the volume up button, etc. Then the consequence is that: when you click the up button, you increase the volume of the GreenfootSound object, but the object whose volume you just increased has nothing to do with the GreenfootSound object that is being used by the play/pause button, which is a typical violation of the “singleton” design pattern.

The “singleton” principle also applies to other classes, such as, you should only create one instance for the play/pause button, and only one instance for the bar display, and only one instance for volume up button, and you should share one current volume variable among all different classes. If you have multiple variables indicating the sound object’s current volume in difference classes, then you need to synchronized them, which means that if you increase or decrease the volume in one class, you need to populate the change to other classes where they also have a variable indicting the current volume level.

Notice: in class GreenfootSound, you should use its method [**setVolume**](https://www.greenfoot.org/files/javadoc/greenfoot/GreenfootSound.html#setVolume(int))(int level) to change the volume. When you call this method, you should plug in the desire volume level between 0 and 100.

**Question: In this homework, why we should not call the** [**getVolume**](https://www.greenfoot.org/files/javadoc/greenfoot/GreenfootSound.html#getVolume())()**method in class GreenfootSound**?

**Answer**: because when any audio clip is played, the actual volume level changes at different segments of the sound track, as indicated by the vertical bar display of the volume in the picture below. The horizontal line is the time, and the vertical line is the actual volume of the sound tract.



Therefore, we cannot use the getVolume() method to retrieve the current average volume level, and since you cannot use method getVolume() to obtain the average volume level of the sound track being played, then you need to declare a private data member to store the current average volume level. And when the volume increase or volume decrease button is pressed, you need to modify the data that stored the current average volume, and then use method [**setVolume**](https://www.greenfoot.org/files/javadoc/greenfoot/GreenfootSound.html#setVolume(int))(int level) to set the modified average volume to the sound track. Also, pay attention to the valid data range of the volume that you can set.

**After finishing this homework, how to verify the correctness of your submitted zip files:**

1. Download the zip files you have uploaded to Moodle homework drop box.
2. Unzip the zip file to a different local folder in your computer, other than the original local folder where the zip files are generated.
3. Run the Greenfoot project from the unzip folder, and make sure it compiles and runs correctly.
4. If your submitted zip file in the Moodle drop box

**cannot be downloaded, or** **cannot be unzipped,** or **cannot compile,** or **cannot run,**

then you need to figure out the reason and fix the error, and then submit the corrected zip file to the Moodle drop box. Then start this verification process again until you can download, unzip, compile and run successfully. To upload a corrected zip file to a Moodle drop box, you need to delete the previous submitted zip file from the Moodle drop box first.

In the first page of file “chap1-schedule.docx”, you can find the instructions on how to zip and unzip files.

**Grading components:**

* Each one of the four exercises has 5 points, thus this homework has a total of 20 points

**For any submitted zip file that still has syntax error and it cannot compile or run in Greenfoot or has runtime error, it will receive ZERO point**. No re-submission is allowed after the homework has been graded.

Please click the Moodle homework drop box to see the due day of this homework.

When coding in Eclipse and Greenfoot, please read document “RulesForIndentAndAlignCode.docx” in Moodle folder “chap 1”, and follow all the rules in code alignment and indentation.