Fields, Methods, and Random Numbers Practice

Music Player Lab

# Main Objectives

* Decide which fields are important to store for a class.
* Manipulate fields and update data as appropriate based on user inputs.
* Work with pre-made methods.
* Write methods to break code into smaller chunks and use those methods from other classes.
* Work with Math.random and other basic math operators.
* Optional Goal: Practice with Arrays.

# Prior Knowledge

* Building objects and placing actors on the world in Greenfoot.
* Familiarity with fields.
* Familiarity with method calling (particularly methods of other objects).
* Familiarity with writing methods.
* Familiarity with the char primitive type (not on the APCSA exam).
* Familiarity with basic primitive-array creation and modification.

# Greenfoot Background

* This lab does require minimal knowledge of Greenfoot.
* It will use the GreenfootSound API as its primary usage of Greenfoot.
* The other Greenfoot knowledge students would need are:
  + Adding objects to the screen.
  + Using the mousePressed methods from Actor
  + Using the setImage methods from Actor
* Important note: For an Actor to use a user created method in a subclass of World, like the MusicPlayer subclass, the Actor will need to typecase getWorld() into the appropriate World type due to Polymorphism.

# Getting Started

Install the Greenfoot IDE: <https://www.greenfoot.org/download>

* Greenfoot went through many changes beginning with the 3.0.0 release. Version 2.4.2, although older, tends to be one of the most stable versions but is missing a few of the newer additions.

In your Music Player folder run the project file with the Greenfoot icon to begin.

# Demonstration

View a sample demonstration of what the Music Player lab could look like here:

<https://www.greenfoot.org/scenarios/23664>

# Instructions

The lab can be broken into two separate concepts, one without arrays which could be completely fairly early on in a student’s learning or one with arrays, without much changes to the original lab’s goals otherwise.

Differences: Without an array the sound files names are: Song (0).mp3 … Song (5).mp3. The only change from one song to another is a number which can be kept as an int field. If using an array then the names of the sound files themselves can be kept in tact and the array can be populated with the actual names of the files, with no pattern to the naming of the files. Students will need to keep track of which track number they are on still but will need to look up the song to play in the array.

## Minimal Instructions for more Student Problem Solving:

* Back – When pressed it should tell the world to move one song backwards.
* Forward – When pressed it should tell the world to move one song forwards.
* Stop – When pressed it should tell the world to stop playing the current music. This should also make the Play button switch back to the play image if it was on pause.
* Shuffle – When pressed it should change its mode to be in shuffle mode or to not be in shuffle mode, whichever is opposite of what it current was.
* Play – When pressed it should switch to the pause image, or back to the play image.
* MusicPlayer – Will need to populate the screen with the appropriate buttons. It will also need to keep track of, at minimum, the current track number and the current GreenfootSound object that represents the music playing. It will need methods that the actors can use to play, pause, stop, and change the tracks forward and backwards. It will need to be able to change tracks when asked to and play the appropriate song. It will need to make sure that when changing tracks that the tracks are chosen correctly – if shuffled then a random next track – if not shuffled then the correct next/previous track in a cyclical nature. For example, when hitting previous when on the first track, the MusicPlayer should play the final song in the tracks. Finally, the current track number should be displayed somewhere on the screen and updated appropriately.

## More In-Depth Instructions for Starting Students:

This lab uses the GreenfootSound class to be able to load a music file up and be able to play, pause, and stop the sound itself. The static method Greenfoot.playSound does not have these capabilities.

### Working with the MusicPlayer World (Basic Level)

* The MusicPlayer needs to keep track of, at minimum, two fields.
  + The currently running GreenfootSound object.
  + The current track number (int).
* The constructor for the MusicPlayer:
  + should start the track number at a logical starting place (likely, the first number of the music tracks.)
  + The constructor should also build a GreenfootSound object and place that object into the corresponding field. The file names for the songs are in the format of   
    “Song (0).mp3”, “Song (1).mp3”, … “Song (5).mp3” . When building the GreenfootSound object, look closely at the corresponding API to build the object correctly.
  + Finally, each of the visual buttons needs to be placed on the screen.
* The methods for the MusicPlayer:
  + void next() – The goal of this method is two-fold. First, the track number itself needs to update to the next numbered track. However, the track number can never go beyond the final track number in the Songs directory. Make sure to set the track number back to zero if it gets too big. Second, a new GreenfootSound should be built and put in the corresponding field. Before switching tracks, you may want to stop the old song…
  + void back() – The goal of this is the same as next, except the track needs to go one backwards instead of one forwards. Which track number will need to be considered separately this time?
  + void play() – The goal here is to tell the GreenfootSound field to begin playing.
  + void pause() – The goal here is to tell the GreenfootSound field to pause.
  + void stop() – The goal here is to tell the GreenfootSound field to stop.

### Working with the Actors (Basic Level)

* Each of the Actors is similar. The main goal of each actor is wait until the mouse has been pressed on them. In the act method, keep asking this question until it eventually is true. Then, based on the Actor type, call the corresponding world method created above.
  + Special note: getWorld().someMethod( ) is a standard method call for an Actor. Something like getWorld().getWidth() for example is used relatively often. However, getWorld() returns a regular World type, and regular World types will not have the next(), back(), play(), pause(), or stop() methods. Your code will need to force Java to recognize that the getWorld() you use does have those methods. To do this, you will need to typecast your getWorld() call specifically into the World type which does have these methods (MusicPlayer). This is a topic called Polymorphism and is expected to be learned for the APCSA exam.  
    Syntax: MusicPlayer musicWorld = (MusicPlayer)getWorld();  
     musicWorld.next();  
     -or-

( (MusicPlayer)getWorld() ).next();

Working with the World and Actors Together (More Advanced Level)

* When the tasks above are finished a basic MusicPlayer should be usable. However, you should run the MusicPlayer and try the different options until you find some details that seem off.
* The Play button would be better if it could toggle back and forth from a Play icon to a Pause icon when it is clicked. Add a boolean field to the Play button class and switch the boolean back and forth each time the Play button is pressed. Switch the image of the Play button to match the current value of the boolean correctly.
* New Problem: When the Stop button is pressed, the Play button needs to be set back to the Play icon rather than the Pause icon. Create a method for the Play button which will update it directly to the Play icon and the correct boolean togger value. In the stop() method in the World, call this new method on your Play object (which will now need to be a field) inside the stop method.
* New Problem: When the next or previous buttons are pressed, should the music play automatically or not? What code could be used so that when the next method or the back method of the world is called that the world determines if the songs should play immediately or stop? Hint, if the World has a Play field, then the Play class can have a method which returns if the music is playing or not at the moment.
* Addition: There is a Shuffle and NonShuffle Icon which suggests making a Shuffle button class, similar to the other Actors. When the Shuffle is pressed, the next songs should be generated randomly and the Shuffle icon should switch to the correct image. When it is pressed again, the next songs should return to playing in order (one up or one day for forward and back). What additions to your code would be necessary to add this in?

# Extensions and Modifications

* Students should feel free to add their own songs – however, caution the legality of where they obtain the music they use and also the appropriateness of the songs they choose.
* Students who have learned other data structures (ArrayList, Stacks, Queues, etcetera), can be given the task of keeping track of the previous songs played. This would allow the back button to always go to the last songs, in order, even if shuffle mode is turned on.
* Students could be tasked with adding a scrolling text bar showing the current song title, but with a limited number of characters at a time. Each act the text on the screen could be updated to showing the song title shifted one to the right of where it was before (scrolling text).
* A real challenge would be to have students learn to use a different music API than the GreenfootSound type, so they can keep track of exactly when a song was paused. The song can then be rewound/fast forward a certain amount of seconds. If the same students have also learned how to store information into files they could keep track of which song is currently, and at what time mark, and have the song be started at the same location when the program is restarted next. This could convert the MusicPlayer into an AudioBookPlayer.

