11/11/2015

Jessie velano/James Gabiana

99108605/11981342

Assignment 3

BCPR301

# Class diagram

**See** “KivyClassDiagram – ScreenManager.docx”

# design patterns

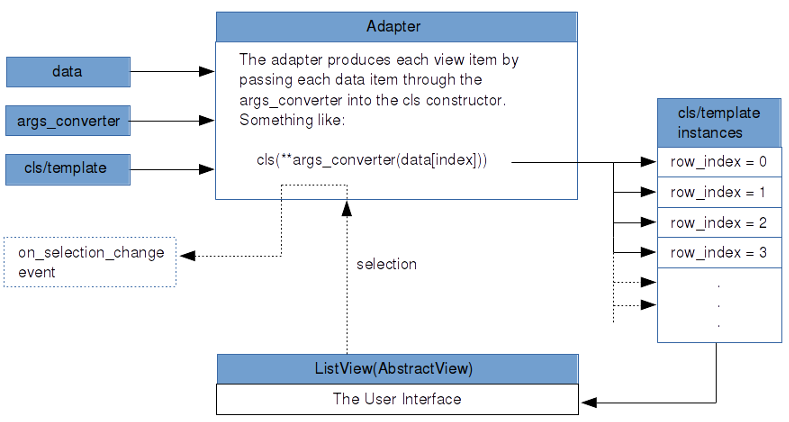
1. **Adapter Pattern**

The use of this pattern is discovered and has been identified inside a folder named adapters

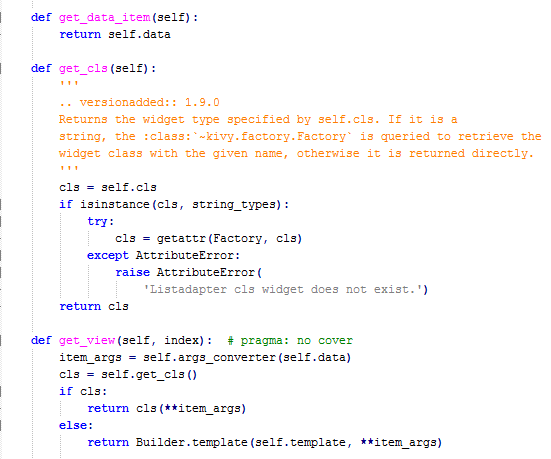
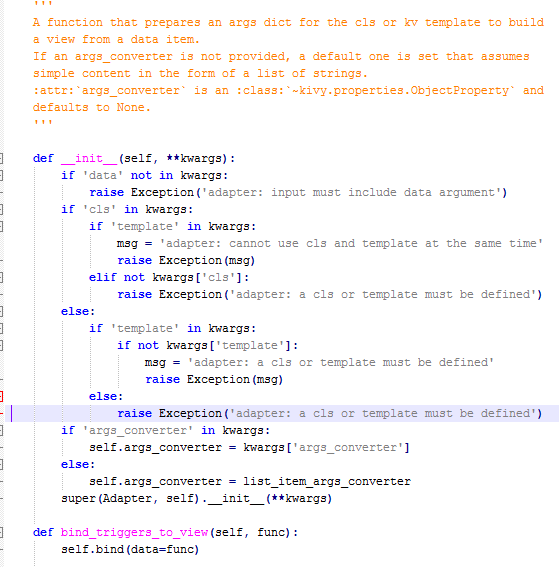
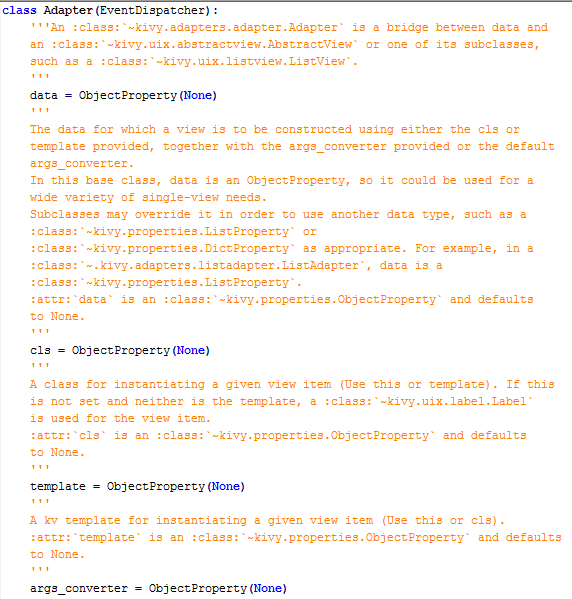
(“Kivy-1.9.0-py3.4-win32-x86\kivy34\kivy\adapters”). The folder contains several python files which encapsulates different classes that are interconnected to each other. The structure of the said files is:



In this diagram, the Adapter class has a subclass called ListAdapter and SimpleListAdapter. And the ListAdapter class has a subclass called DictAdapter that is a more advanced and a flexible subclass. Here is a diagram provided inside the kivy program:



Here is the code of the Adapter Class:



The adapter plays a mediating role between the user interface and your data. It manages the creation of the view elements for the model using the args\_converter to prepare the contructor arguments for your cls/template view items.

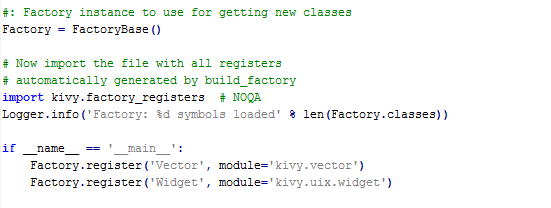
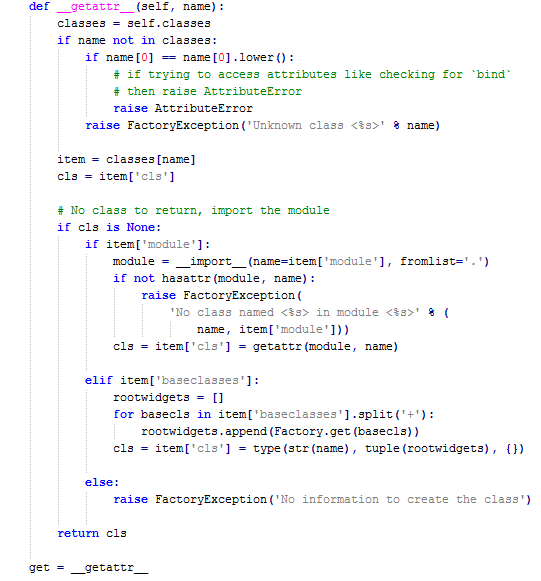
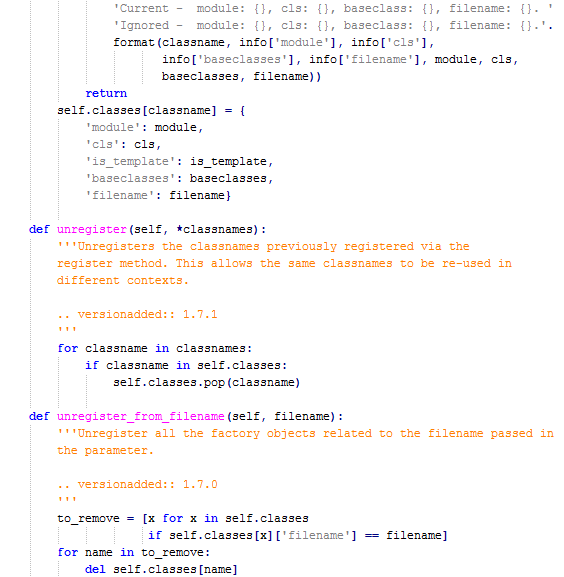
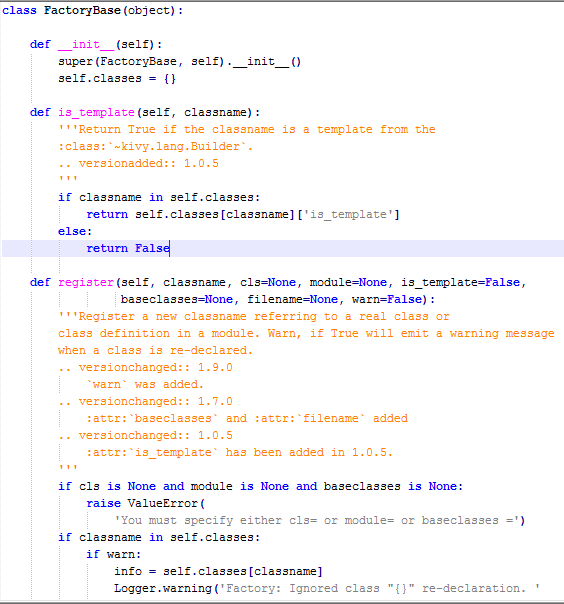
Kivy adapters are modelled on the [Adapter design pattern](http://en.wikipedia.org/wiki/Adapter_pattern). Conceptually, they play the role of a ‘controller’ between you data and views in a [Model-View-Controller](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) type architecture.

1. **Factory Pattern**

Another design patter that has been used by the kivy program is the Factory Pattern. This class has been found inside a folder named kivy (“Kivy-1.9.0-py3.4-win32-x86\kivy34\kivy”) and the file’s name is factory.py.

A **factory** is an [object](https://en.wikipedia.org/wiki/Object_(computer_science)) for [creating other objects](https://en.wikipedia.org/wiki/Object_creation) – formally a factory is simply an object that returns an object from some method call, which is assumed to be "new”. More broadly, a [subroutine](https://en.wikipedia.org/wiki/Subroutine) that returns a "new" object may be referred to as a "factory", as in *factory method* or *factory function*. This is a basic concept in OOP, and forms the basis for a number of related [software design patterns](https://en.wikipedia.org/wiki/Software_design_pattern).

Here is the code of the said file:



The factory can be used to automatically register any class or module and instantiate classes from it anywhere in your project. It is an implementation of the [Factory Pattern](http://en.wikipedia.org/wiki/Factory_pattern).Here is a sample photo of the factory method registering heaps of modules:

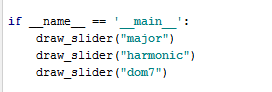


# implemented functionality

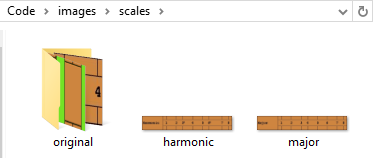
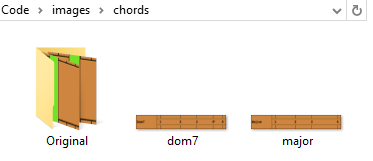
**Implemented Requirements:**

1. Draw any of the scale and chords slides.
2. Draw any 5 different scale and chord slides.
3. Must be able to display the slider in the Music Slider Interface when selected.

All requirements stated above are related to each other, they share a common goal that is to draw and display the created chord and scale slider. When the user runs the “DrawSlider.py” the program will create a number of slider based on how many times the user calls the function and what chord or scale is he creating.



When a user runs the code above. He will be able to create the major slider for both Chords and Scale. But when he runs it with “Harmonic” as a parameter, it will only create the harmonic slider in the Scale group because there are no Harmonic slider for the Chords group. The same goes with dom7. The program will only create a slider of dom7 in the Chords group and not in the Scale group.



**PEP8 Code Check:**

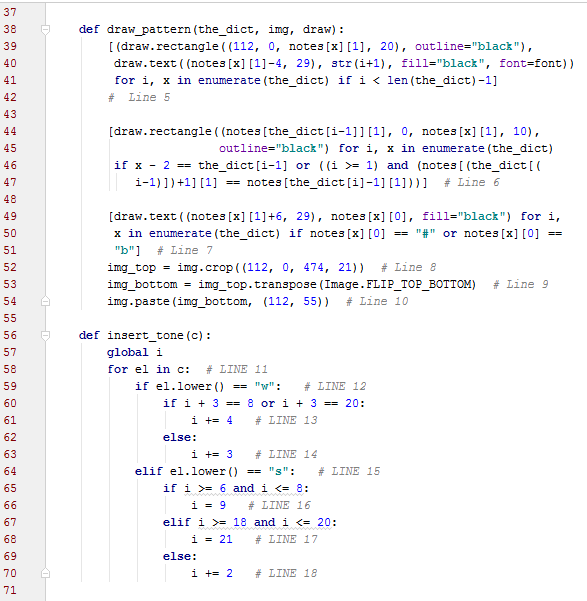
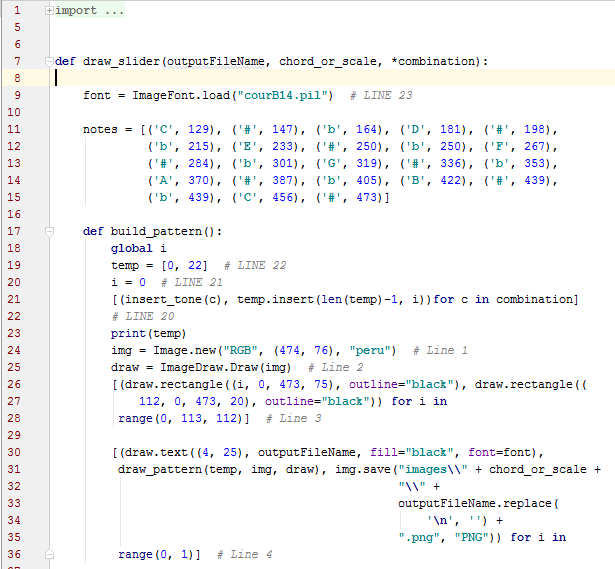


# amount of code

**Implemented Requirements:**

1. Draw any of the scale and chords slides.
2. Draw any 5 different scale and chord slides.
3. Must be able to display the slider in the Music Slider Interface when selected.

**22 Logical Lines of Code (LLOC) in total**



# Evaluation

**External Characteristics:**

* Correctness
  + The program is free from errors in its specification, design, and implementation. The instructions are carefully read and the implementation process has been followed correctly to avoid faults.
* Efficiency
  + The program created is proven to be efficient because of the minimal Logical Lines of Code (LLOC). By doing that, it saves space occupying the memory and by that, reducing the execution time.
* Accuracy
  + The program built is free from error and the output of this program is correct and it follows the given requirement of the assignment. From the code to the output the correctness is assured.
* Robustness
  + The robustness of this program is tested through testing it by entering invalid inputs where the program identifies it as error and does not perform or display a slider. This kind of test will assure that the program can handle invalid inputs properly.

## **Effectiveness of my code:**

### Zen of Python “Sparse is better than dense”

* We completed our python program in just 13 lines of code which is very efficient in terms of the size of the program. It will only consume less size than codes that didn’t use the “list comprehension expressions”. This is a python coding technique that uses two square brackets to enclose several different tasks. This technique will only produce one logical line of code (LLOC) which is the best way to do in order to follow the requirement of this assignment knowing that we’re only allowed to have 20 logical lines of code (LLOC).

### Zen of Python “Although practicality beats purity”

* In order to complete the program within 20 lines of code with full features, I have to break some rules, so I put more than one task in one logical line of code (LLOC). By doing this, the code looks cluttered and hard to read but by following the commas per each tasks, the reader will be able to understand what a particular task is performing.

### Zen of Python “Simple is better than complex”

* Because of the limited number of logical lines of code (LLOC) we are required to write, my code is short and is easier to understand, simple logics not complex.

## **Improvements**

### Zen of Python “Readability counts”

* Because I put multiple tasks on one line of code, and also had to follow PEP8 format, readability becomes a little bit messy so the value of Zen of Python “Readability Counts” reduces. Also, when checking the code in PEP8, I need to compromise and add new line when it goes beyond the suggested length of a line of code.

#### Recommendation

* + Assign only one task to be done in one line of code just like PEP8 recommends. Each line of code should only have one responsibility, and try to make separate modules if necessary to improve readability and lessen the complexity of the code.

### DRY

* Some line of code is duplicated which violates the DRY principle.

#### Recommendation

* + I could have used Behavioural Pattern or Structural Pattern to make my code more flexible and remove duplication specially when drawing rectangles.

# Self-Marking

1. Diagram showing the structure of the framework (**1 /1 mark**)
2. Identification of 2 design patterns used in the framework (**2/2 marks**)
3. Amount of required Functionality successfully implemented (**7 /7 marks**)
4. Amount of code (**4/5 marks**)
5. Evaluate the effectiveness of your work, and make recommendations accordingly. (**5 /5 marks**)