Java Vs. Python: Student Course Registration System

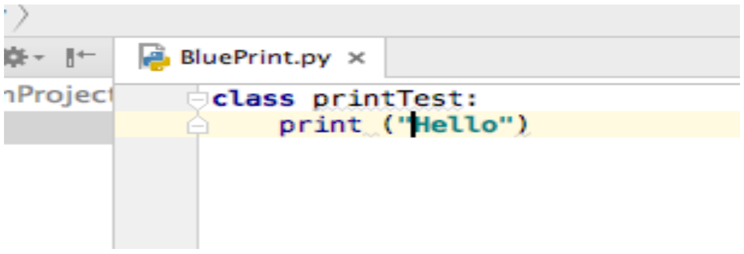
Thomas Norman

I. Introduction

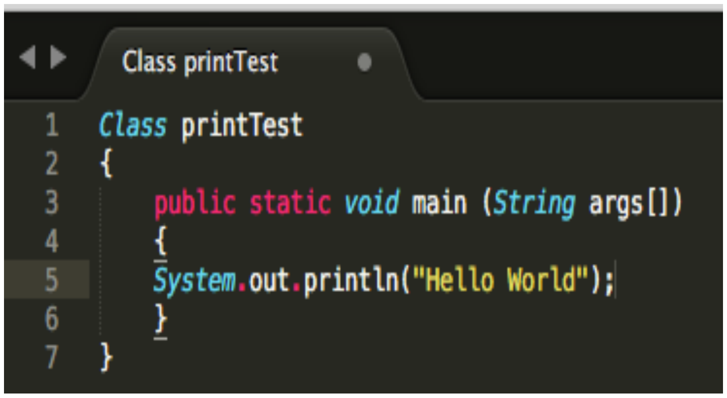
As a long time Java programmer, I've become rather set in my ways when it comes to syntax, techniques and resource utilization. I'm constantly learning new things, but the basics have become mostly second nature for me. I've been familiar with Python for quite some time, seeing the code and knowing of people that have used it primarily in certain applications, but I've never been one to jump it to it and get hands on. My time has been spent with Java, C++ and HTML/CSS. As far as productivity is concerned, Python presented me with a welcome break from the long hours of coding just simple syntax to make up a bigger component of the application. I've quickly realized that Python is everything that you put into it. You can achieve more complicated tasks by utilizing its resources in tandem. This isn't to say that you can start Python and in one day achieve the high levels of Java programming, but you can definitely notice the differences rather quickly

II. Language

The first and most noticeable difference between the languages is declaration. Java, being statically typed, needs to have all variables declared. Python needs no declaration due to its dynamic nature. Regardless of each programmer's opinion, when looking for a language that can assist with heavy scripting at a fast pace, Python absolutely delivers. The first thing I appreciated about Python, especially in coding this program, was the ability to go from a design idea to coding in mere seconds. I had used pen and paper to write down some of my ideas, assets and relationships. I made a use case diagram and a few other design tools and I was able to immediately put them into practice in PyCharm, my Python IDE of choice. Rather then going through a Java IDE, creating a package, classes and methods, I just started testing strings and calculations in Python. As shown in the example below, testing the print nature of a string is simple and quick while removing about 3 lines of code from the Java Equivalency:



```
class printTest:
    print ("Hello")
```



```
Class printTest
{
    public static void main (String args[])
    {
    System.out.println("Hello World");
    }
}
```

While many situations see a reduction of actual lines of code, most of the time you see the majority of the relief being taken up by the body of code rather than each specific line. Being explicit is a necessity in Java, when defining subclasses but in Python you are able to build the class as you code without having to work reductively and explicitly define a super element. Without getting into the idea of databases, which were used fairly heavily in this application, the simple act of reading file input is drastically unburdened by Python, reducing 10-15 lines of Java code to 2-4 in python. As far as backend coding is concerned, Python obviously takes a lot of the strain off of the programmer in cases such as these.

III. The Application

For this application, I designed the main infrastructure to simply show relationships between actors and actions in a simple and immediate way. I wanted a main login interface that would differentiate between a student and a faculty member, taking the actor to their appropriate home page and presenting them with necessary actions within their permissions. Obviously, a student would not be able to add or delete a course to/from the main university course schedule, but only in regards to their individual student schedule. I wanted to display rather basic security protocols for this exercise, simply encrypting the password and shielding it from view without relying heavily on extensive hashing or anything of the like.

Once the actor is presented with their appropriate homepage, I wanted the possible actions to be displayed in a user friendly and linear manner. I chose Tab Panes to display tables, schedules and other appropriate database Schema, reacting to button actions as well as tab actions, giving the user the option to operate back and forth. By the end of the accompanying walkthrough, you can see a student add or delete from their schedule, as well as check schedule status, while the faculty member can login, view their respective course schedule for instruction, add or delete from the registry and view changes in real time. The end of any and all actions presents the actor with the option to return to home or logout completely.

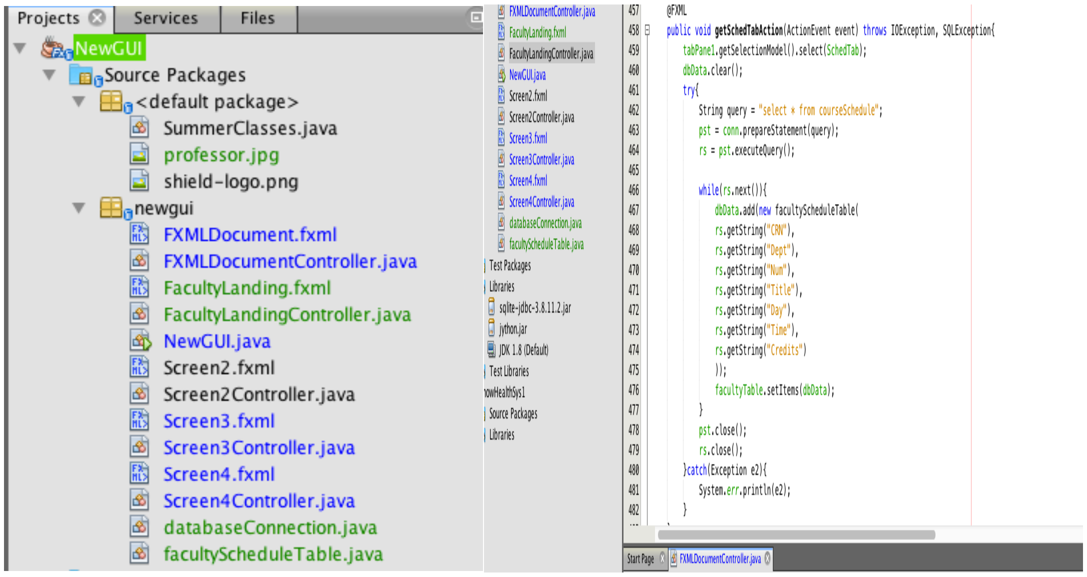
IV. The GUI

Once I began working on the GUI with JavaFX, I began to witness my first real difficulties. Integrating the two languages with one another, whole-heartedly, became an issue with design and usage. With every task completed, a new hurdle was presented and before too long, the hurdles began to recursively affect past successes. This is where I began to refactor and start from scratch. With my newfound realizations, I began to use Python to code my main scripts for test purposes, and once completed, I would move to Java, finish the current component of the GUI, and then convert my python tests over to Java-approved syntax in the way that it would work best with my GUI practices. While this did not result in an application that used both languages, it did result in about the same amount of hours worth of work in each language. One way to think about it is sketching preliminary ideas in pencil, several different ways before committing to pen and color.

For the login screen, I used python to simply associate asset labels/tags with prompt text and user input, before worrying about security. This helped simplify relationships between assets. I didn't use Python much for security practices, as that's something that I just decided on and scripted in Java. My next task in Python was to associate certain actions and assets with the student user and others with the faculty user. This helped me when I later started operating between separate GUI windows, so that I could keep up with associations as I coded, rather than having to finish one completely before moving to another. I utilized the test names from Python and kept the assets the same in Java, that way my blueprint matched my final product on the backend. I would use my Python tests to show myself associations between button actions for each user and their respective results. My main goal with all of this was to use what I'd learned in Python to help make the Java side of the process more streamlined with less wasted hours and resources. If I couldn't use Python the way I wanted to within the application itself, I would at least make sure it contributed to the overall design and use of the application.

As far as the GUI aspects were concerned within Java, I made a window for the main login, which would continue to a student window or a faculty window. I then made each dashboard presentable through the use of a Tab Pane. This container would have numerous tabs for each fresh screen, each of which would also correspond to a button on the left side of the screen. Once the actions were completed, the final confirmation screen would arrive, allowing the user to return home or Logout. This process kept the number of windows low enough, and not having to use processor speed and memory to load another fresh window with each action and change. Essentially, the databases would save and display changes, while the GUI windows would not change until the user was ready to complete all actions. I did this merely to cut down on the pull of computer resources.

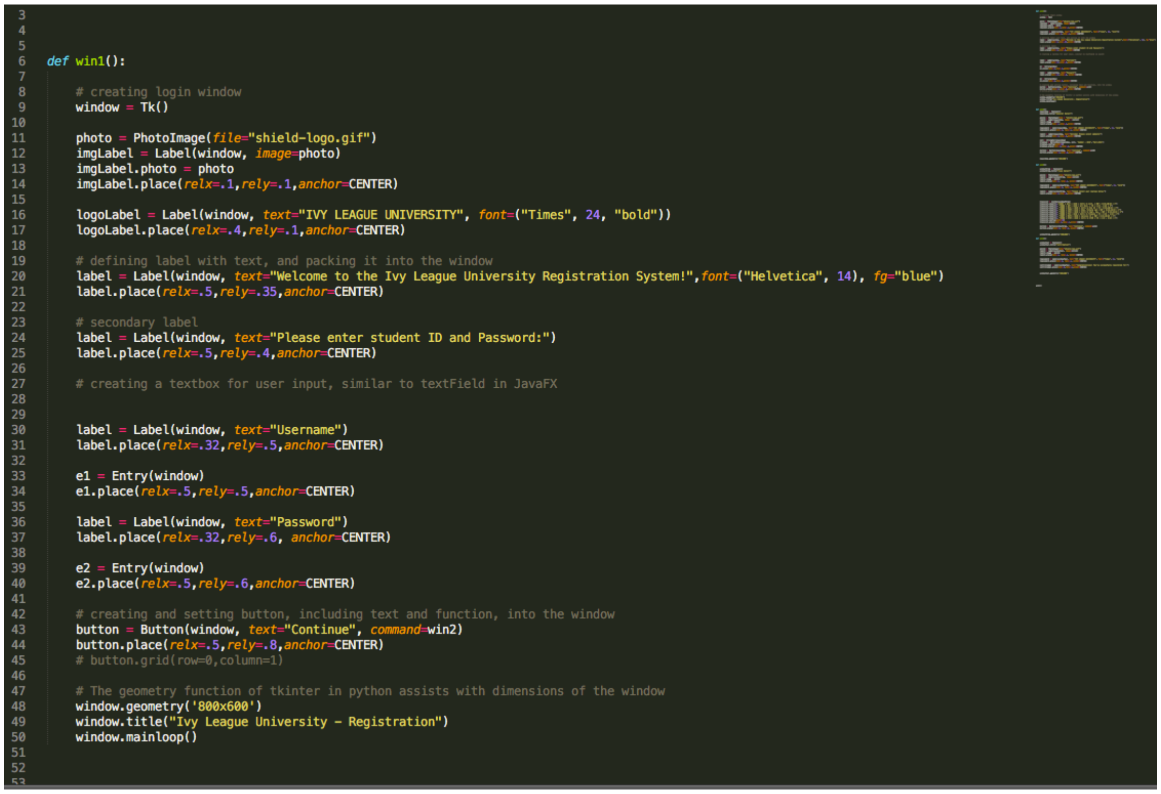
To show the difference in file hierarchy and code equivalency between the languages, the screenshots below give a good example:



```java
    @FXML
    public void getSchedTabAction(ActionEvent event) throws IOException, SQLException{
        tabPane1.getSelectionModel().select(SchedTab);
        dbData.clear();
        try{
            String query = "select * from courseSchedule";
            pst = conn.prepareStatement(query);
            rs = pst.executeQuery();

            while(rs.next()){
                dbData.add(new facultyScheduleTable(
                rs.getString("CRN"),
                rs.getString("Dept"),
                rs.getString("Num"),
                rs.getString("Title"),
                rs.getString("Day"),
                rs.getString("Time"),
                rs.getString("Credits")
                ));
                facultyTable.setItems(dbData);
            }
            pst.close();
            rs.close();
        }catch(Exception e2){
            System.err.println(e2);
        }
```

These screenshots are an example of files and syntax necessary to build the Gui and the contents within Java. The screen to the right is the syntax for only the faculty window/fxml file. It is up to 480 lines of code in order to layout the screen and fill the content/commands. The python equivalence of one window, shown below, is obviously much more basic, but it displays the layout of a window of the same size with the same base contents. It merely shows how much dense code can be eliminated.



```python


def win1():

    # creating login window
    window = Tk()

    photo = PhotoImage(file="shield-logo.gif")
    imgLabel = Label(window, image=photo)
    imgLabel.photo = photo
    imgLabel.place(relx=.1,rely=.1,anchor=CENTER)

    logoLabel = Label(window, text="IVY LEAGUE UNIVERSITY", font=("Times", 24, "bold"))
    logoLabel.place(relx=.4,rely=.1,anchor=CENTER)

    # defining label with text, and packing it into the window
    label = Label(window, text="Welcome to the Ivy League University Registration System!",font=("Helvetica", 14), fg="blue")
    label.place(relx=.5,rely=.35,anchor=CENTER)

    # secondary label
    label = Label(window, text="Please enter student ID and Password:")
    label.place(relx=.5,rely=.4,anchor=CENTER)

    # creating a textbox for user input, similar to textField in JavaFX


    label = Label(window, text="Username")
    label.place(relx=.32,rely=.5,anchor=CENTER)

    e1 = Entry(window)
    e1.place(relx=.5,rely=.5,anchor=CENTER)

    label = Label(window, text="Password")
    label.place(relx=.32,rely=.6, anchor=CENTER)

    e2 = Entry(window)
    e2.place(relx=.5,rely=.6,anchor=CENTER)

    # creating and setting button, including text and function, into the window
    button = Button(window, text="Continue", command=win2)
    button.place(relx=.5,rely=.8,anchor=CENTER)
    # button.grid(row=0,column=1)

    # The geometry function of tkinter in python assists with dimensions of the window
    window.geometry('800x600')
    window.title("Ivy League University - Registration")
    window.mainloop()


```

V. The Databases

Between the hours spent working in Python and Java to complete the basic functions of the application, I began taking account of all of my assets associated with text fields and actors, and I started creating the database tables. I created my main database to embed within the application, and made all distinctions between table schemas. One Table would be for the student schedule and one for the faculty schedule. The main goal of the database schema was to make sure it mirrored the tables in the GUI exactly. That way every table column/row, text field and input area would correspond to an area specifically. Once ensuring all of the table assets and database entries matched, the simple task of updates needed to be addressed. In this instance, I used a database helper to read in the data, apply it to the table and take any changes from the table (prompted by a button action) to save into the database, updating with every change. This way all changes could be viewed in real time by pushing the user to a current view of the data.

VI. Installation

In order for a user to utilize this application, without the use of a fully functional .exe package, they would simply need java with JavaFx capabilities. I tried to ensure cross compatibility for the application itself so that it could run on a system with very light resources, low processing power and heavily utilized memory systems. The original goal was to successfully get the application packaged into an installable and executable windows program as well as a mac DMG file, but if not, at least I would keep the necessary components to be installed as low as possible and easy to acquire/utilize.

VII. Takeaways

Though my initial idea for integrating Python into this project drastically changed, I feel my takeaway is just as accomplished, if not more so, than it would have been originally. The goal of this project was to learn Python and display that knowledge somehow. I could have done a very simple python project, of which I did several while learning the basics of the language (such as a calculator with a GUI and list of options), but I could have done so with very little research and effort in terms of language utilization. While learning how to integrate the languages in this project, albeit unsuccessfully, I learned more than I intended to in order to rectify errors and problems as they came up. My hurdles turned into learning experiences rather quickly, which is the nature of being a learned programmer. When I decided to use the Python language as a template/blueprint for the overall application, I was able to find a true strength with the language, especially since I already had the Java programming experience.

Cross converting languages became a huge asset for me after completing this project. I feel like now, even if I have problems getting the languages to work well together (which is my new challenge), I definitely feel more comfortable converting code and scripts, or even translating ideas to others who may have more experience in a different language, which I feel will be a huge asset in the years to come. Ultimately, after finishing this application, I've taken the time to learn much more about Python and even add it to my resume as I've taken on multiple online learning courses with Python and looking into several projects utilizing the language on GitHub. I may not have dipped into the language as quickly and seriously as I would have without this project, so this experience was a catalyst in taking on a full fledged new programming language.