To GUI or Not to GUI:

On how we teach introduction to programming

Antonio Sanchez, Bo Mei
Texas Christian University

“In intelectum autem nihil est, nisi prius fuerit in sensu”
There is nothing in the understanding that was not before in the senses, and therefore to exercise the senses well about to perceive the differences is the way to lay the grounds for all wisdom”

John Amos Comenius (1592-1670)

Abstract - Active learning comes from experience and practice, and this is the approach we use when we teach programming. We follow Comenius dictum and so we appeal to the senses first. We apply a Model View Controller or MVC design pattern using an object-oriented language or OOL with a Graphic User Interface or GUI approach. We have been using this approach for more than 20 years with good results. In the paper we review the literature that deals with GUIs in general. Using Java, first we deal with developing the View, so the concepts of JFrames, JButtons and many Components are presented, along with several layout managers. The next step is to develop a simple Controller. Here we introduce the if statement for dealing with the set of listeners available. The Controller extends the View and the application begins to show some interactivity. The next step is to do Model, the Controller constructs an instance of the Model class passing its address as a parameter. It is in the final part of the course that we introduce collections to provide a very rich application. In conclusion, we have found that this approach allows us to keep our students interested through their first course in programming

Keywords: Digital Programming, GUI, IDE, Java, MVC, OOL

1 Introduction

It is always a challenge to teach the first course in programming to students that have never thought in terms of a detail set of discrete steps to solve a problem. The main difficulty lies in presenting the syntax and semantics of an abstract language and to learn to model a problem to obtain a valid program that solves the problem at hand. To circumvent this challenge, there have always been plenty of approaches suggested. These approaches are all the way from teaching games [16] to a rigorous mathematical approach [1,8], which includes selecting a syntactical forgiving language such as Python, Basic or HTML. Yet at the end, students complain about the language and what they perceive the lack of generality if they do not have the necessary set of software libraries. After so many years of teaching the first introduction to programming course, we feel we have come up with a better approach if we just follow Comenius dictum, that is appeal to the student senses. Using a programming architectural pattern such as the Model View Controller, initially proposed by T. Reenskaug in 1979 [10,23], we can teach students to focus on one aspect of the design at a time, first the View, then the Controller and finally the Model. We feel this approach should be taken into consideration. Two other important aspects ought to be mentioned here, the use of an object-oriented language or OOL and programming using a graphic user interface or GUI. The course and its implementation are discussed in this paper. It is important to note that the approach discussed in this paper is not necessarily oriented towards those students that already had a programming course before, since they might find the course a bit slow-paced and not algorithmically oriented.

1.1 Graphic Using Interface

Let us first review briefly the literature on teaching programming. The first thing we consider was the selection of a language that would fit the description detailed above. Among all the possibilities, we selected Java, which is an object-oriented language, easily adaptable to design GUI’s and amicable enough to deal with the MVC pattern. There are many other languages also well suited to interact with the Internet such as C++, C#, and Python. Java has gone from been an interesting research language to become part of a programming culture in the world [20]. Furthermore, Java is the most used language as reported in the Tiobe
Index [21]. Having selected the language, the next step is to select a good textbook. There are some many good books about teaching digital programming using the Java language. In the references, we provide the name of a few of them. Among them, we have used Hortsman [11,12,13] and Deitel [7], preferably the object-oriented version. Worth to mention is an older good introductory Java book that we used for a while was the one by Winston [24]. However, like many authors, they use the same approach regardless of the language. That is to say they could be teaching Fortran or Pascal, the idea of using objects from our perspective is that they use the objects constructed and show them visually and for that you need a GUI. Let us mention here that such an idea is not new. As a matter of the fact, it has been discussed by many [4,8,9] even in Internet blogs, as reported in [14,15]. Furthermore students have complained that they are not given the chance to design practical real life applications of their own [25], we feel that OOL and MVC allows for that.

1.2 Object Oriented Programming

To teach OOL programming concepts, we definitely need a graphic user interface, which is one aspect that makes the learning appealing. There have been very few books that teach programming with this perspective. A very early book was *programming Java* by Decker, R., Hirshfield, S. [6] The book had two editions and promoted the use of the Applets and html `<applet>` tags. We used it with a great degree of success, then we moved to *An Introduction to Programming with Java Applets* by E. S. Boese [3] that dealt mostly with JApplets of the `javax.swing` library. The book was very appealing, however as time went by, JApplets were considered by many browsers insecure and so the plug-in stopped being provided. Finally in Java 9, Oracle decided to deprecate Applets. But to be fair, such a decadence was not taken as hard as one might think, since users have moved from web browsing to run applications in a device like a laptop or a smartphone. This is the approach we have taken, so teaching programming took us back to develop local applications. There are few books that deal with this approach. We have recently found Conrod, P. Tylee, L. *Learn Java GUI Applications* [5] as a reasonable alternative. However, the book suffers from the same problem of a regular programming language book, that is it introduces too many abstract concepts a bit early to allure new students. This is where we differ in our approach. Using the MVC approach, we introduce most of the concepts on a need-to-know basis, just as Boese, Decker, and Hirshfield did. The important thing to mention here is that while learning to construct the *View*, students learn both the syntax and the semantics of OOLs.

1.3 Integrated Development Environment

Another important aspect to talk about is the Integrated Development Environments or IDEs available for development. In the case of java there are many options, to name a few Eclipse, IntelliJ, and NetBeans [27,28,29]. Initially, we began using Code Warrior [30] and Apple Xcode [27], but now we have moved to Eclipse [28] which seems to be one the most popular ones. NetBeans initially supported by Sun has now moved to Apache [26]. In any case, we feel that students learn the language along with the IDE since it has a natural GUI appeal.

2 Description of the course

Following Comenius dictum and in view of the fact that nowadays it the way users interact with the Internet using local applications, although search engines are still much to find information; however, there are even local applications that do that. This is why we feel the use of a GUI with an MVC approach is the most appealing way to get students engaged. Our course is divided in three main units, the *View*, the *Controller* and the *Model*. There are three exams throughout the semester, one for each one of these units of the MVC, later in the paper we discuss the type of questions administered. The eight assignments follow the units mentioned: there are three assignments dealing with the development of the *View*, two assignments dealing with extending the *View* into a *Controller* class and three labs that use the full MVC once the *Model* class with its handshake mechanism with the *Controller* is covered. Every semester we define a theme application and develop its various components, as matter of example consider the development a sandwich order application as presented in *Figure 1*.

![Figure 1. The sandwich order JFrame application](image-url)
2.1 The View

The first part of the course deals with learning to use the IDE the basic syntax of Java, as well as the static main method required to trigger any local class. In this unit we also only cover how to display of the View without any interaction. The use of the console to provide feedback for the developer is presented to the students using the statement System.out.println("Feedback for the developer"). Here the concepts of class, objects, methods, as well as how they belong to inherited classes, i.e. a JLabel within a JFrame. Constructors and simple methods are presented using javax.swing widgets such as JFrame, JButtons, JList, JComboBoxes and JTextArea, etc. as well as the display of images as instances of the ImageIcon class to be used in JLables or JButtons. A discussion of Layouts using various JPanels is also given, along with that of fonts, colors, and scroll panes; this covers the View unit of the course. At end of the unit students are subject to an evaluation. In the exam students are tested on their capacity to distinguish the syntactical and semantic difference between objects and classes, how to use methods to the construct a class and how to call a method in Java to obtain a certain action.

The exam provides a View based on a set of JPanels each using an specific Layout; students are required to provide a possible Java code that achieves such a View, as suggested by Figure 2 then they are required to program a simple view for a simple problem. Since they do not have online access the Java API, students are provided with a subset of the API that includes the most useful methods of the classes covered in the unit.

2.2 The Controller

In the second unit of the course, we introduce conditionals, the if and if/else statements are discussed in conjunction with Listeners and basic math functions using the Java Math class; along we introduce the eight Java primitives in comparison to non-primitive types. Figure 2 presents an example that invites students to learn the difference between the integer operations divide / remainder % in contrast with the floating point divide. Primitive and Non-Primitive types are covered along with a discussion on casting, using Integer.parseInt() and Double.parseDouble(). Programming and calling methods is discussed here, notably when using if, if/else and switch statements along with the return statements in methods. Again we emphasize here that programming concepts are introduced only on a need to know basis. In this unit, we cover the various listeners.

Figure 2. Sample Question of View

Students are required to select the correct method

![Figure 2. Sample Question of View](image)

The relation between View and Controller is implemented in Java using the extends clause; students use their previous assignment as the View and are required to program the Controller. Simply stated the code looks as follows:

```java
public class Lab5 extends Lab3 implements ActionListener
```

The exam of this unit includes a set of if and switch questions to evaluate the understanding of these programming statements. Students are provided with a View and a set of desired interaction requirements, then

Figure 3. A Change Maker JFrame application

![Figure 3. A Change Maker JFrame application](image)
they are required to program a Controller class extending the provided View, as shown in Figure 4.

![Figure 4. Controller Sample Question](image)

2.3 The Model

The last unit of the course deals with the concepts of collections, arrays, and loops. By now students have tackled the various listeners available in Java and have understood the purpose of extending a class, so they are ready to create a Model class and to include it in the important performance methods; this is similar to many wrapper classes in Java. The main purpose of the Controller is to construct the Model and call its methods. A simple example of this is the statement Lab6Model $m = \text{new Lab6Model(this)}$; in this case we introduce the \texttt{dot(\)} notation to access the objects residing in the Controller. Another way, also covered in the course, is the construction of the model without passing the address of the calling class i.e. \texttt{Lab6Model $m = \text{new Lab6Model()}$}; in this case, method calls must pass their parameters as part of the method arguments. Figure 5 shows the result when the full MVC pattern is completed. This includes the use of images and sounds as well as creating new JFrames as suggested in the figure.

![Figure 5. Full MVC model when an order is selected and payment is requested](image)

In final two labs we deal with collections and arrays. For the example discussed above we extend the application to allow for multiple orders. The various forms of loops are covered. Figure 6 shows and example of such endeavor. Since one-dimensional arrays can be made out of a non-primitive class, students are taught to create a class that will serve as the basis for the collection as suggested by the code in example below:

```java
public class OrderDetail {
  private String drink;
  private String sandwich;
  private String side;
  private String size;
  private boolean grilled;
  private double price;
  
  public OrderDetail() {
    // constructor
  }

  public String getDrink() { return drink; }
  public void setDrink(String n) { drink = n; }
  public String getSandwich() { return sandwich; }
  public void setSandwich(String n) { sandwich = n; }
  public String getSide() { return side; }
  public void setSide(String n) { side = n; }
  public String getSize() { return size; }
  public void setSize(String n) { size = n; }
  public boolean getGrilled() { return grilled; }
  public void setGrilled(boolean n) { grilled = n; }
  public double getPrice() { return price; }
  public void setPrice(double n) { price = n; }
```

---

Figure 6. An example of a class used for storing order details.
private objects with public getters/setters are discussed in the course, along with a simple index or pointer element as suggested by the code:

```java
public OrderDetail[] orders = new OrderDetail[20];
public int orderIndex = 0;
```

![Figure 6. Multiple Order Display using an array of a collection](image)

Miscellaneous Java concepts such as static, final are introduced in the last part of the course, including basic treatment of exception handling using try/catch/finally. It is also important to mention that whenever an assignment is given, students provided with some starting experiment code that allows them to practice and test the general concept of the assignment and provide the answer to a set of required questions. The final exam provides students with a View and a Controller class, here students are required to develop a Model that uses a collection to be used in conjunction with a one dimensional array, as shown in Figure 7.

![Figure 7 MVC sample question](image)

### 3 Results

The course has been taught at our university using the approach discussed here for more than 20 years, the initial developers of the course were R. Rinewalt and J. Comer. One of the authors of the paper has been teaching the course for more than 15 years. Initially the course began with Applets, then we moved to program with JApplets using the basic html web browser code. Since most browsers have deprecated the Applet plug-ins, we are now using local applications running...
JFrames. Although we know the `javax.swing` will eventually be deprecated too, we will begin to use the newer GUI classes available in JavaFX[18].

The overall comment of the students has been that although the course is difficult and time consuming, the GUI and MVC approach makes it enticing and appealing; added to this they comment that the experiment code given to the students with each lab is very helpful. Here are some positive verbatim comments[22]:

"The labs were extremely helpful for learning Java"  
"Having to spend the time playing with the code and make mistakes allowed me to learn"  
"The experimental code allowed me to better understand what was needed from me"  
"This is a good introduction course that taught me a significant amount of programming"  
"The labs were challenging but also very educational and interesting"  
"Engaging projects"

Indeed there are also some bad comments from students [22]:

"I wish the course was offered as a lab instead of a lecture so that the instructor had more time to help."

"It would have been helpful if we spent a little more time going over the very basics of coding at the start of the class. I had no coding experience before, so this would have been nice."

The overall grade distribution for the course is as follows: 40% for eight lab assignments and 20% for each exam. The distribution of the grades in the last 15 years in the course taught by one of the authors has been as follows: 24% A's, 34% B's, 19% C's and the rest.

Furthermore, an important validation to our teaching approach has been reported recently by Ball [2], in their very interesting paper, they have found that students with GUI assignments rather than just text-based ones, enjoy more on the course and score higher.

5 References


Indeed teaching programming to freshmen is a challenge mostly since many have not decided on a computer science major, the key is to keep them engaged. We feel that our approach has definitely helped. The reason for it lies in the fact that nowadays students are so familiar with using applications when they interact with their devices; they are motivated to develop them on their own as practical real life applications. After this course, we move to a second course in programming where we deal with files, string processing, threads and socket applications basically using the Java API. It is not until the third course that we introduce Abstract Data Types or ADT’s when we teach Data Structures.


Software reviewed (JVM and IDE’s)


