BÁO CÁO

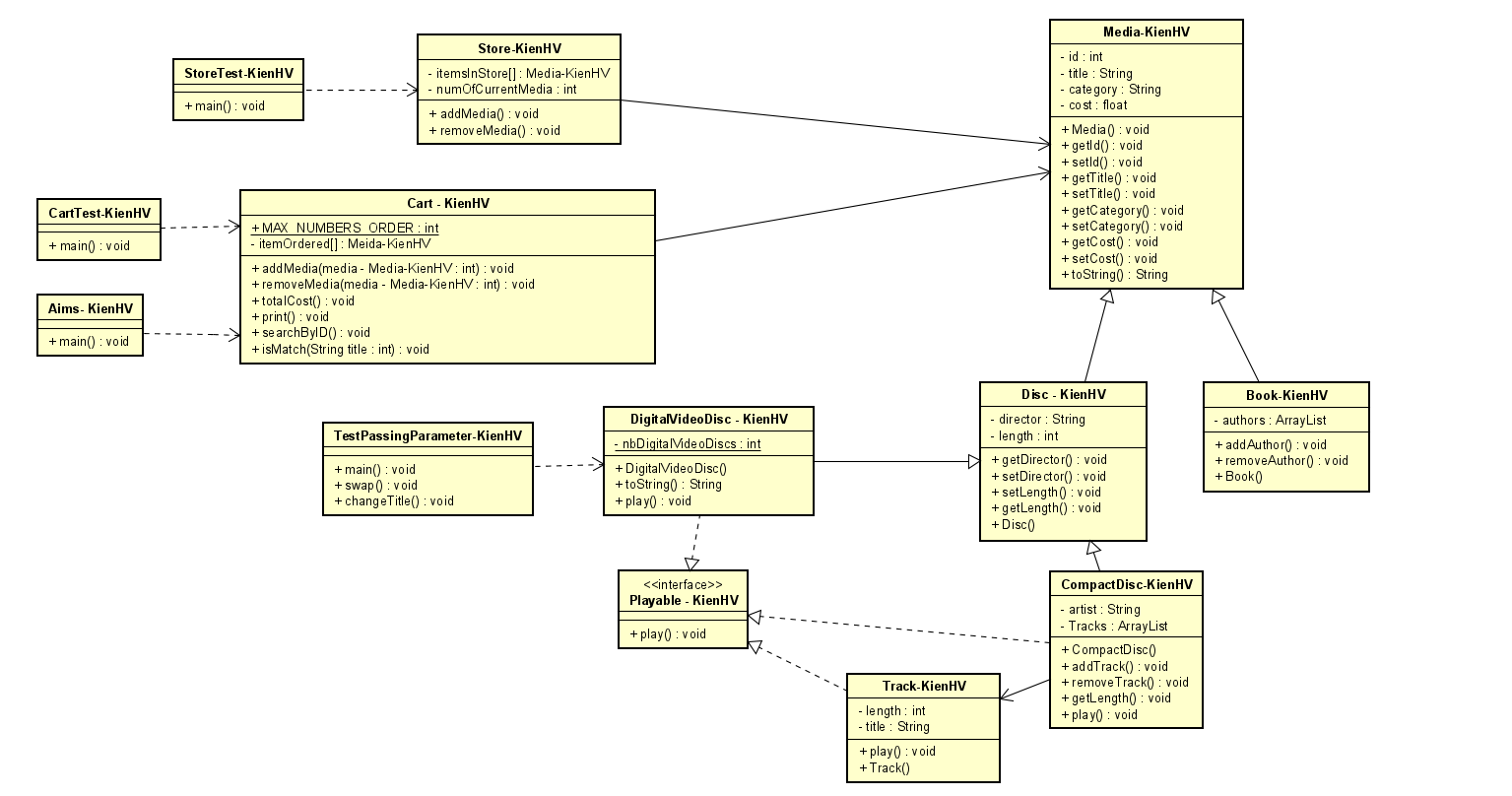
Object-Oriented Programming

Họ và tên: Hoàng Văn Kiên

MSSV: 20205089

**Lab 05: GUI Programming**

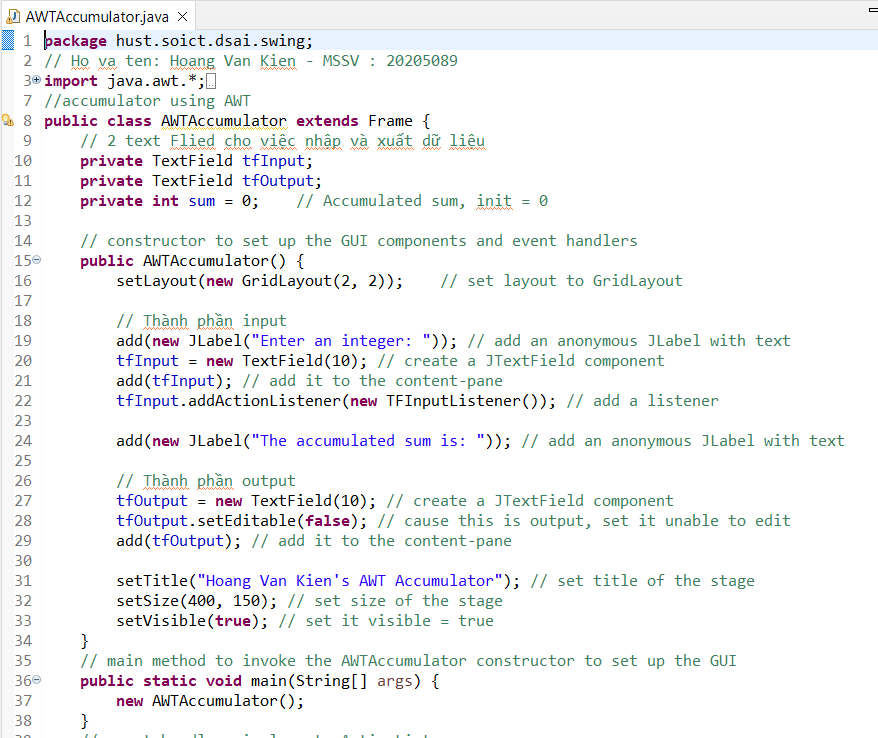
# **Update UML**



# **Swing component**

## **AWTAccumulator**

### Create class AWTAccumulator



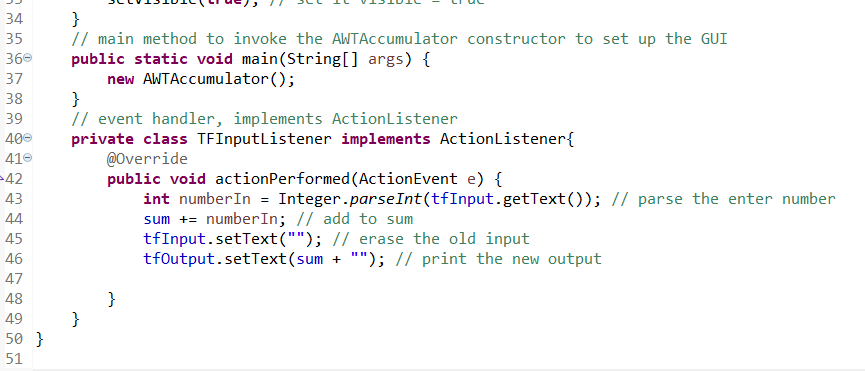
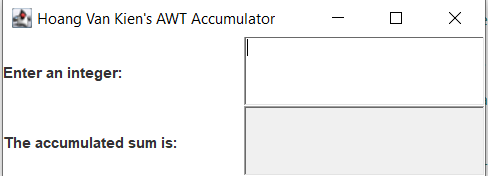


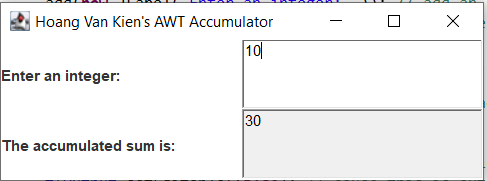
Figure 1. Source code of AWTAccumulator

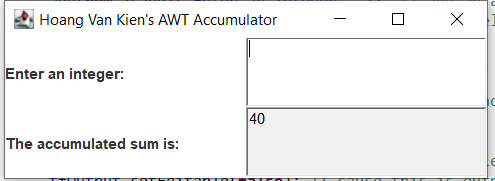
### Giải thích

* Trong AWT, bộ chứa cấp cao nhất là Frame , được kế thừa bởi lớp ứng dụng.
* Trong hàm tạo, chúng tôi thiết lập các thành phần GUI trong đối tượng Khung và xử lý sự kiện:
* Ở dòng 13, bố cục của khung được đặt là GridLayout
* Trong dòng 15, chúng tôi thêm thành phần đầu tiên vào Frame của mình , một Nhãn ẩn danh
* Trong dòng 17-19, chúng tôi thêm một thành phần TextField vào Frame của chúng tôi , nơi người dùng sẽ nhập các giá trị. Chúng tôi thêm một trình nghe lấy thành phần TextField này làm nguồn, sử dụng một lớp bên trong có tên.
* Trong dòng 21, chúng tôi thêm một Nhãn ẩn danh khác vào Khung của chúng tôi
* Trong dòng 23 – 25, chúng tôi thêm một thành phần TextField vào Frame của chúng tôi , nơi tổng giá trị đã nhập sẽ được hiển thị. Thành phần được đặt thành chỉ đọc trong dòng 24.
* Trong dòng 27 – 29, tiêu đề & kích thước của Khung được đặt và khả năng hiển thị của Khung được đặt thành true, hiển thị Khung cho chúng ta.
* Trong lớp người nghe (dòng 36 - 44), phương thức actionPerforming() được triển khai, phương thức này xử lý sự kiện khi người dùng nhấn “Enter” trên TextField nguồn .
  + Trong dòng 39-42, số đã nhập được phân tích cú pháp, được thêm vào tổng và văn bản của TextField đầu ra được thay đổi để phản ánh tổng mới.
* Trong phương thức main() , chúng ta gọi hàm tạo AWTAccumulator để thiết lập GUI
  + 1. **Kết quả**

****

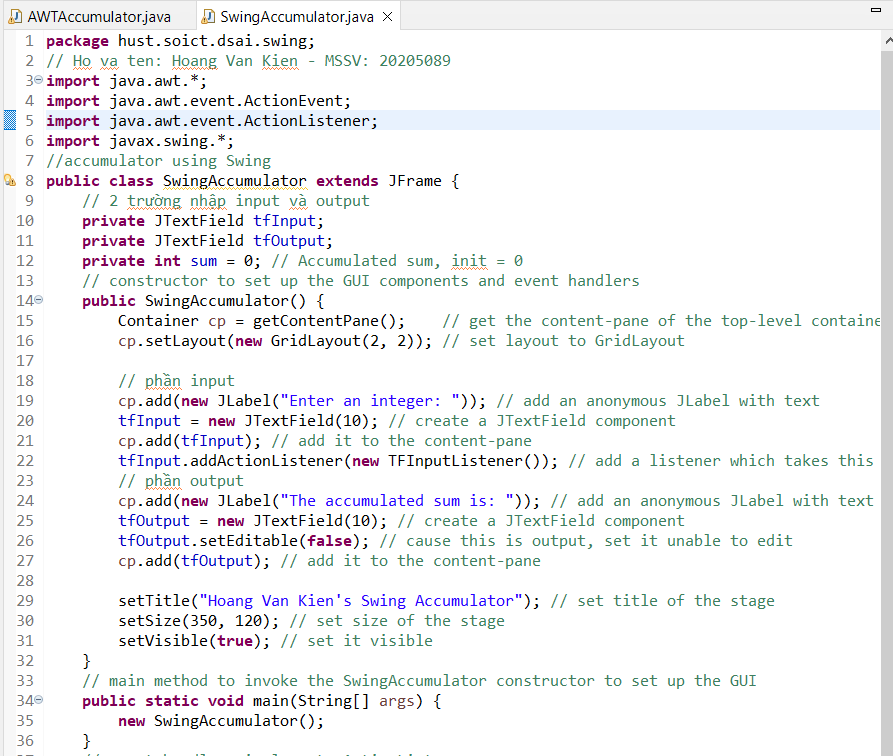
**Nếu nhập tổng là 10 và 30 thì sẽ ra 40**

****

****

## **SwingAccumulator**

### Create class SwingAccumulator with the source code as below:



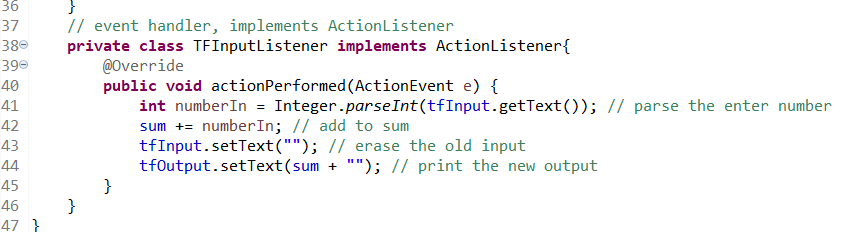
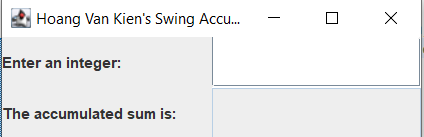


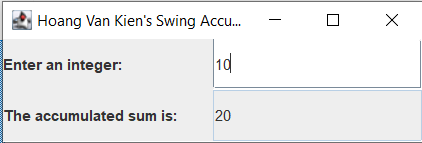
Figure 3. Source code of SwingAccumulator

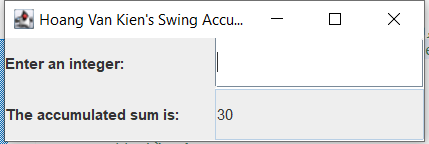
### Giải thích

* Trong Swing, bộ chứa cấp cao nhất là JFrame được kế thừa bởi lớp ứng dụng.
* Trong hàm tạo, chúng tôi thiết lập các thành phần GUI trong đối tượng JFrame và xử lý sự kiện:
* Không giống như AWT, các JComponents sẽ không được thêm trực tiếp vào bộ chứa cấp cao nhất (ví dụ: JFrame , JApplet ) vì chúng là các thành phần nhẹ. Các JComponents phải được thêm vào cái gọi là ngăn nội dung của vùng chứa cấp cao nhất. Ngăn nội dung trên thực tế là một java.awt.Container có thể được sử dụng để nhóm và bố trí các thành phần.
* Ở dòng 15, chúng ta có khung nội dung của vùng chứa cấp cao nhất.
* Ở dòng 16, bố cục của khung nội dung được đặt là GridLayout
* Trong dòng 18, chúng tôi thêm thành phần đầu tiên vào ngăn nội dung của mình, một JLabel ẩn danh
* Trong dòng 20-22, chúng tôi thêm một thành phần JTextField vào ngăn nội dung của chúng tôi, nơi người dùng sẽ nhập các giá trị. Chúng tôi thêm một trình nghe lấy thành phần JTextField này làm nguồn.
* Trong dòng 24, chúng tôi thêm một JLabel ẩn danh khác vào ngăn nội dung của chúng tôi
* Trong dòng 26 – 28, chúng tôi thêm một thành phần JTextField vào ngăn nội dung của chúng tôi, nơi tổng giá trị đã nhập sẽ được hiển thị. Thành phần được đặt thành chỉ đọc trong dòng 27.
* Trong dòng 30 – 32, tiêu đề & kích thước của JFrame được đặt và khả năng hiển thị của Khung được đặt thành true, hiển thị JFrame cho chúng tôi.
* Trong lớp người nghe (dòng 39 - 47), mã xử lý sự kiện giống hệt như AWTAccumulator .
* Trong phương thức main() , chúng ta gọi hàm tạo SwingAccumulator để thiết lập GUI
  + 1. **Kết quả**
* **Màn hình chính**

****

* **Khi chạy chương trình: Tổng của 10 và 20**

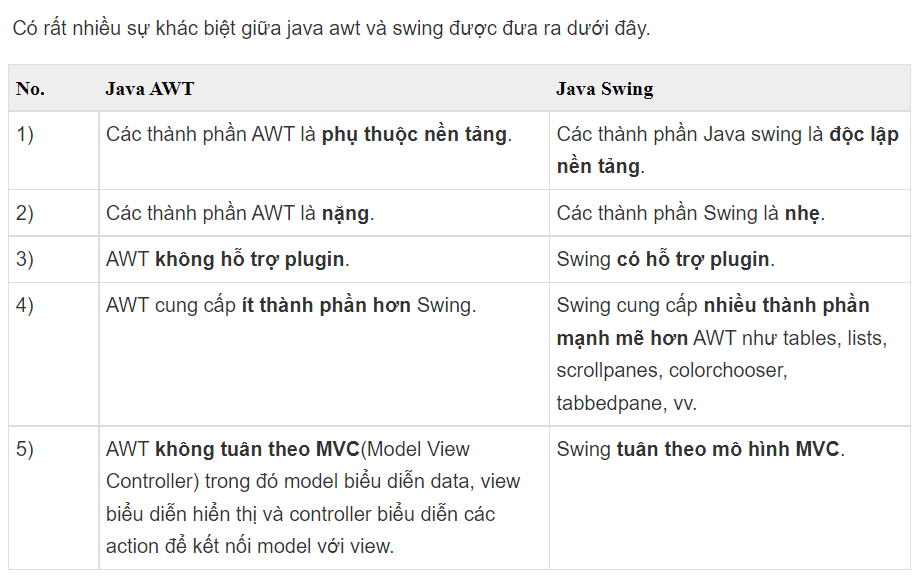
****

****

## **So sánh các phần tử Swing và AWT**

Lập trình với AWT và Swing khá giống nhau (các yếu tố giống nhau bao gồm vùng chứa/thành phần, xử lý sự kiện). Tuy nhiên, có một số điểm khác biệt mà bạn cần lưu ý:

* Các container cấp cao nhất trong Swing và AWT
* Tên lớp của các thành phần trong AWT và tên lớp tương ứng trong Swing



# Organizing Swing components with Layout Managers

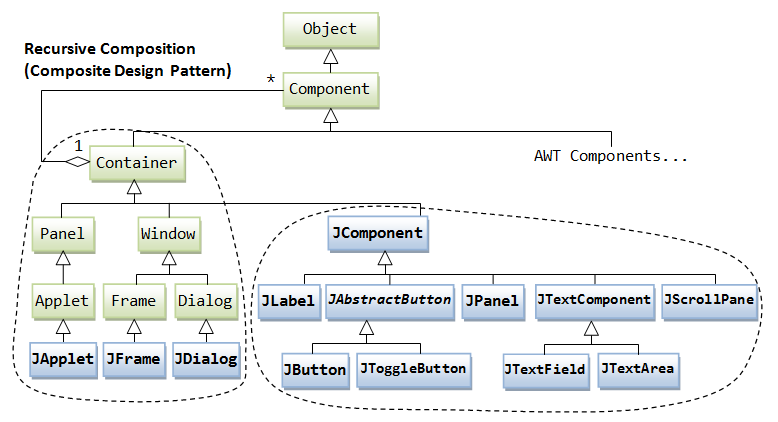


Figure 4. AWT and Swing elements

## Swing top-level and secondary-level containers

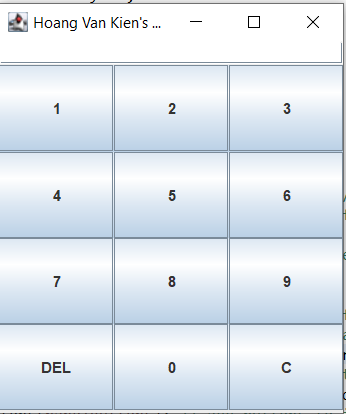
A container is used to hold components. A container can also hold containers because it is a (subclass of) component. Swing containers are divided into top-level and secondary-level containers:

* Top-level containers:
  + JFrame: used for the application's main window (with an icon, a title, minimize/maximize/close buttons, an optional menu-bar, and a content-pane)
  + JDialog: used for secondary pop-up window (with a title, a close button, and a content-pane).
  + JApplet: used for the applet's display-area (content-pane) inside a browser’s window
* Secondary-level containers can be used to group and layout relevant components (most commonly used is JPanel)

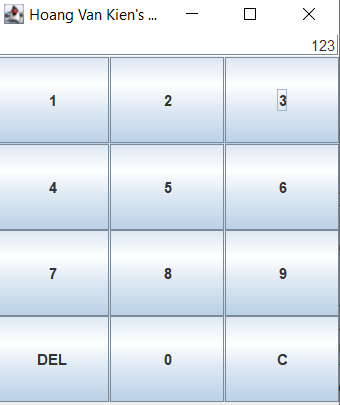
## Using JPanel as secondary-level container to organize components

### Create class NumberGrid

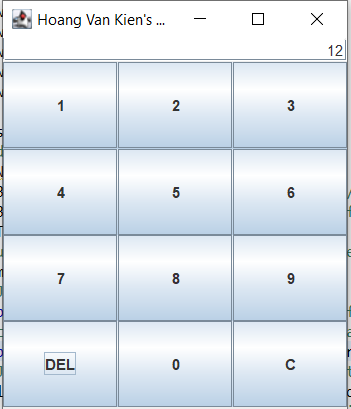
**Kết quả:**



Kết quả sau khi nhập ví dụ 123



Kết quả khi bấm Del:



**Kết quả khi bấm C:**

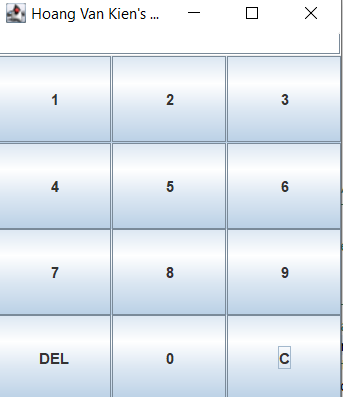


Figure 5. NumberGrid

\



Figure 6. NumberGrid source code(1)

The class has several attributes:

* The btnNumbers array for the digit buttons
* The btnDelete for the DEL button
* The btnReset for the C button
* The tfDisplay for the top display

In the constructor, we add two components to the content pane of the JFrame:

* A JTextField for the display text field
* A JPanel, which will group all of the buttons and put them in a grid layout

### Adding buttons

We add the buttons to the panelButtons in the addButtons() method:

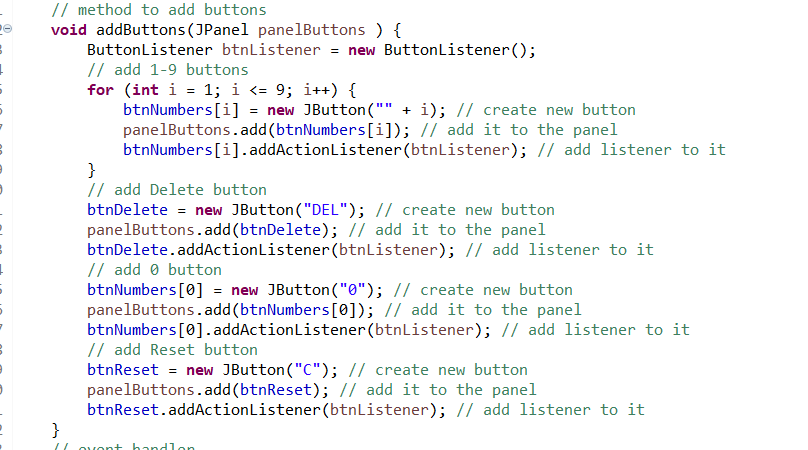
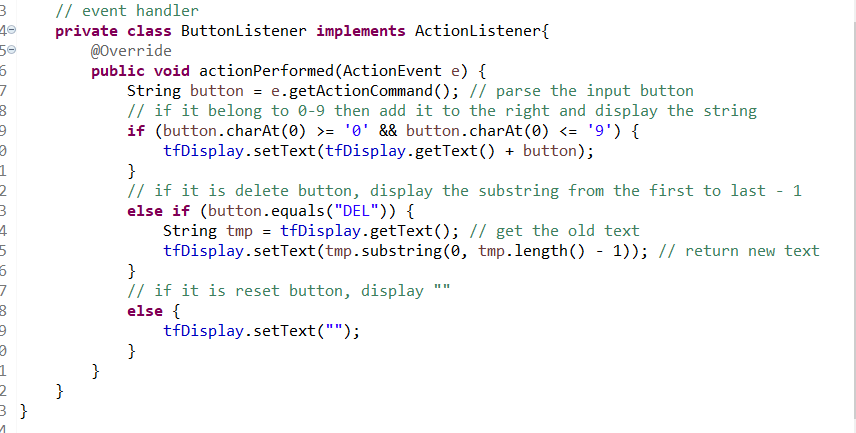


Figure 7. NumberGrid source code(2)

### Complete inner class ButtonListener



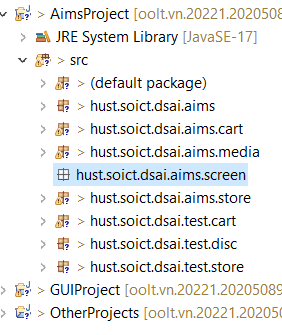
**Figure 8. NumberGrid source code(3)**

* **A digit button: a digit is appended to the end**
* **DEL button: delete the last digit**
* **C button: clears all digits**

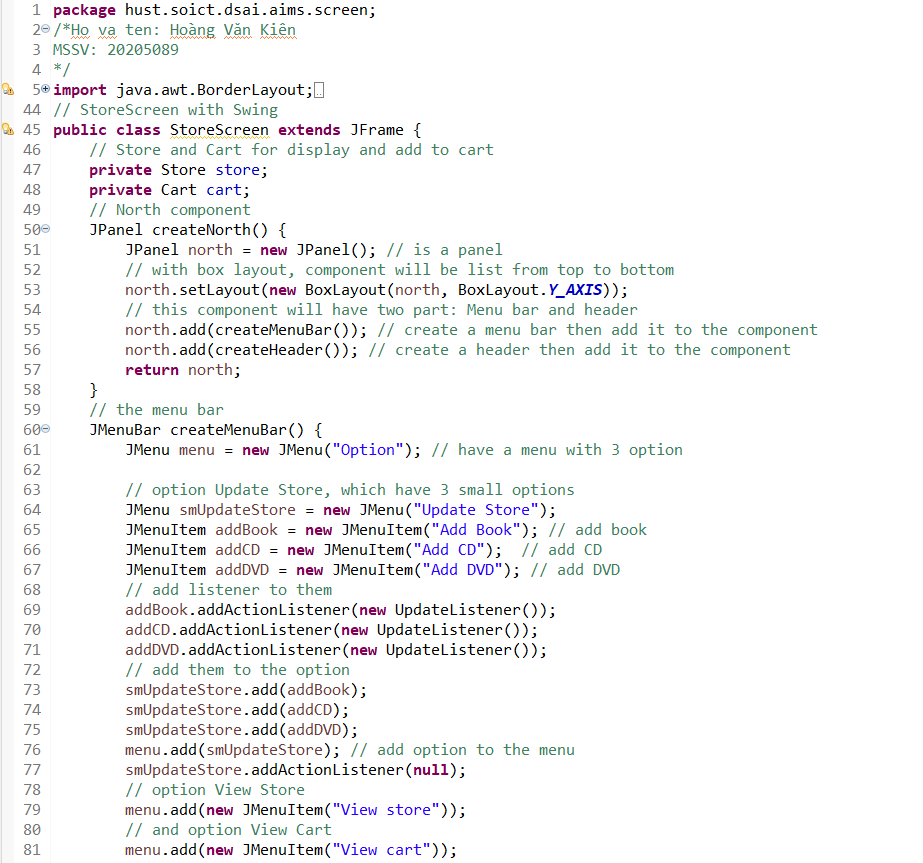
# Create a graphical user interface for AIMS with Swing

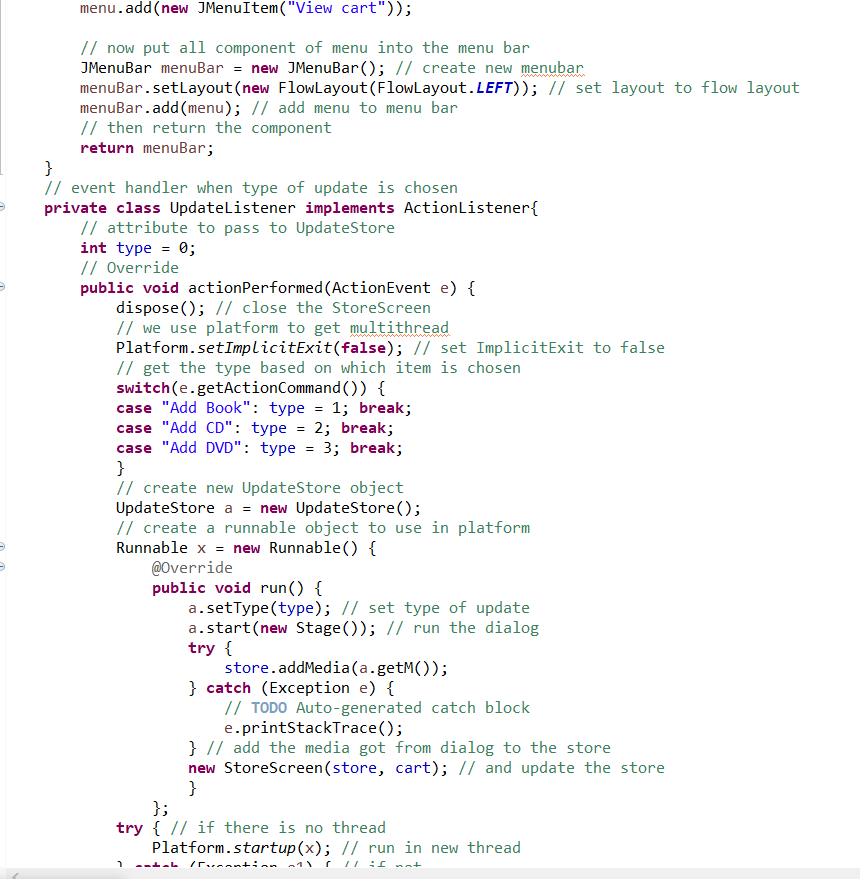
For the AIMS application, we will implement three screens:

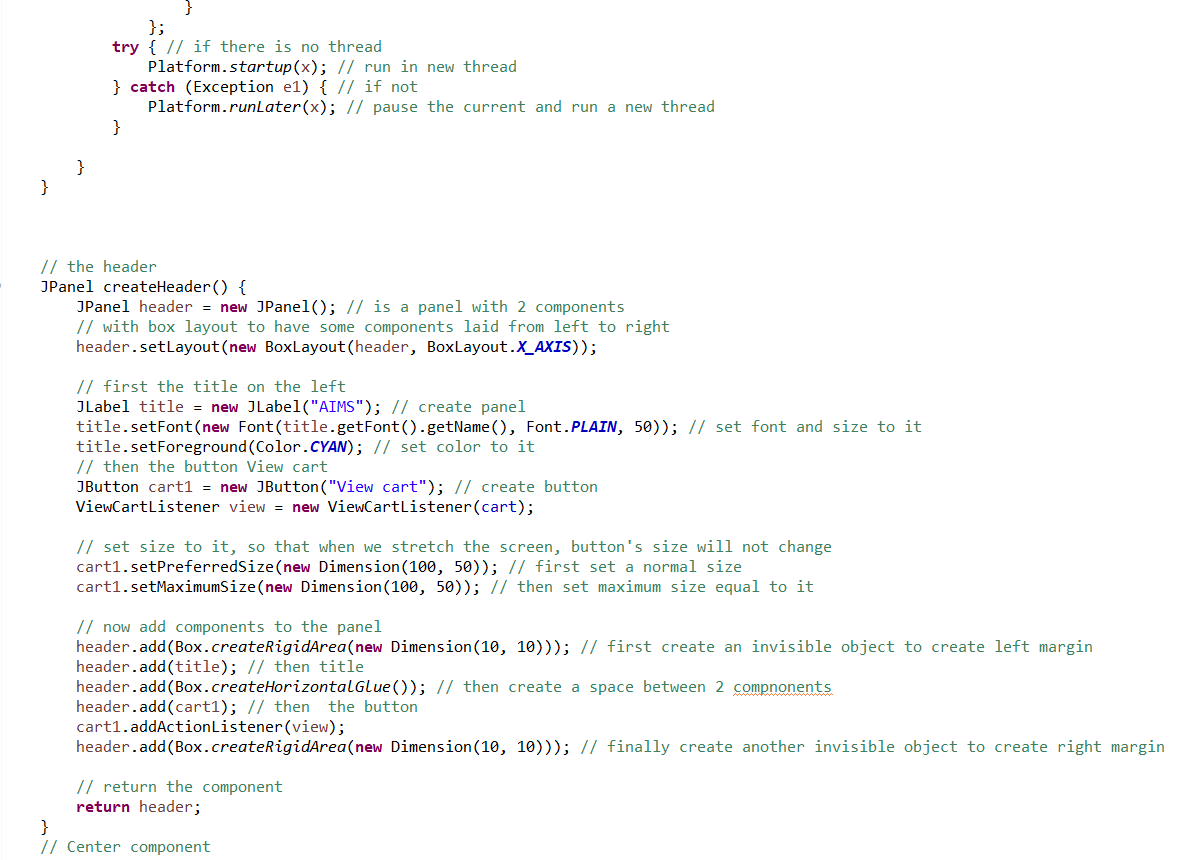
* The “View Store” screen using Swing
* The “View Cart” screen using JavaFX
* The “Update Store” screen using either Swing or JavaFX depending on you



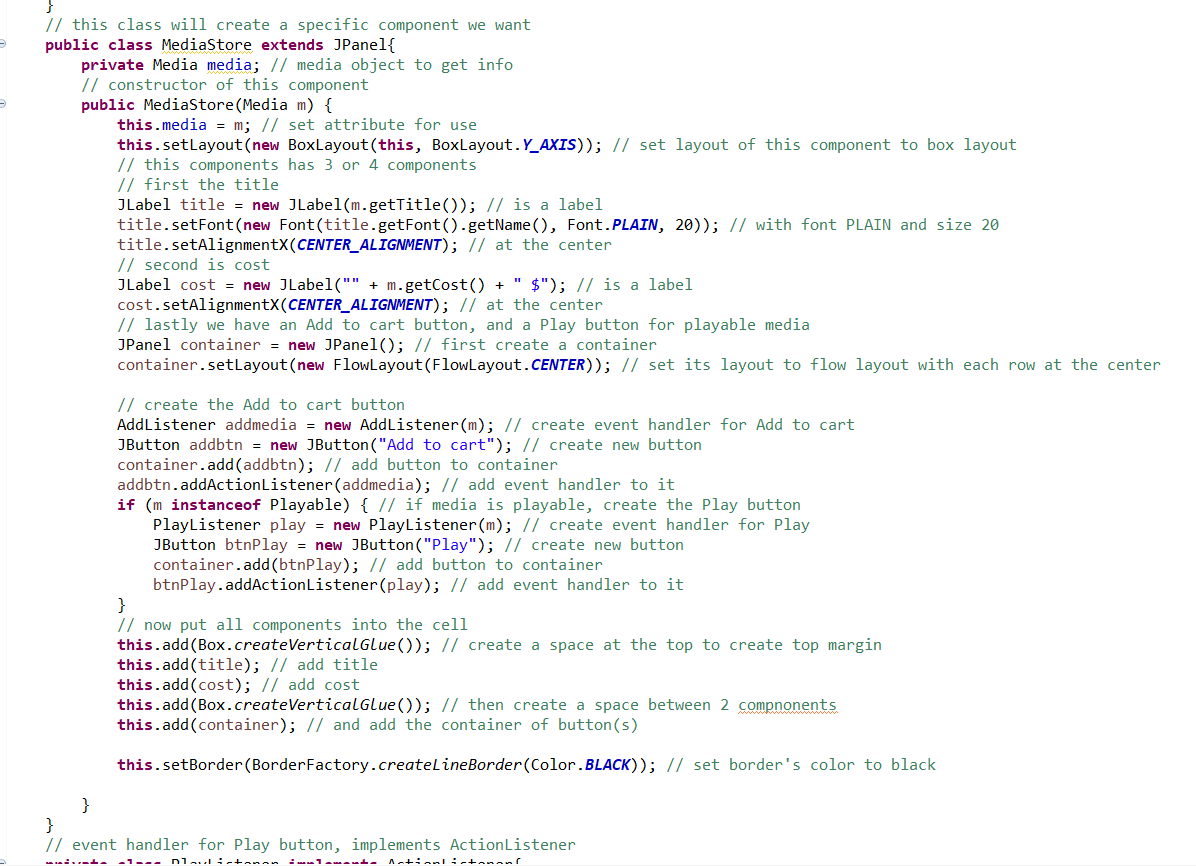
Code StoreScreen.java







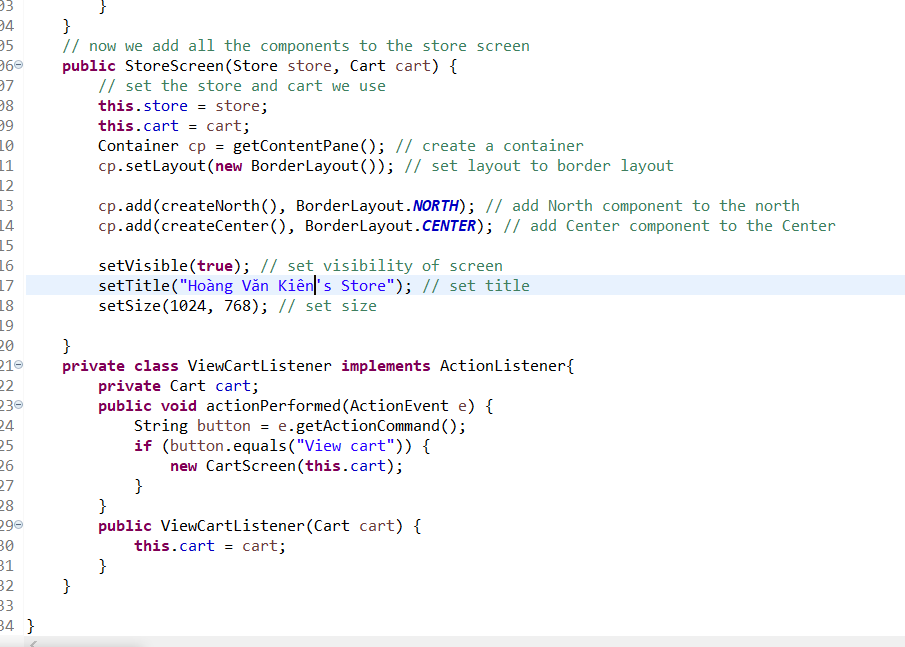
**--Media Store**



* Code AddListener



--Store Screen



## **View Store Screen Kết quả**

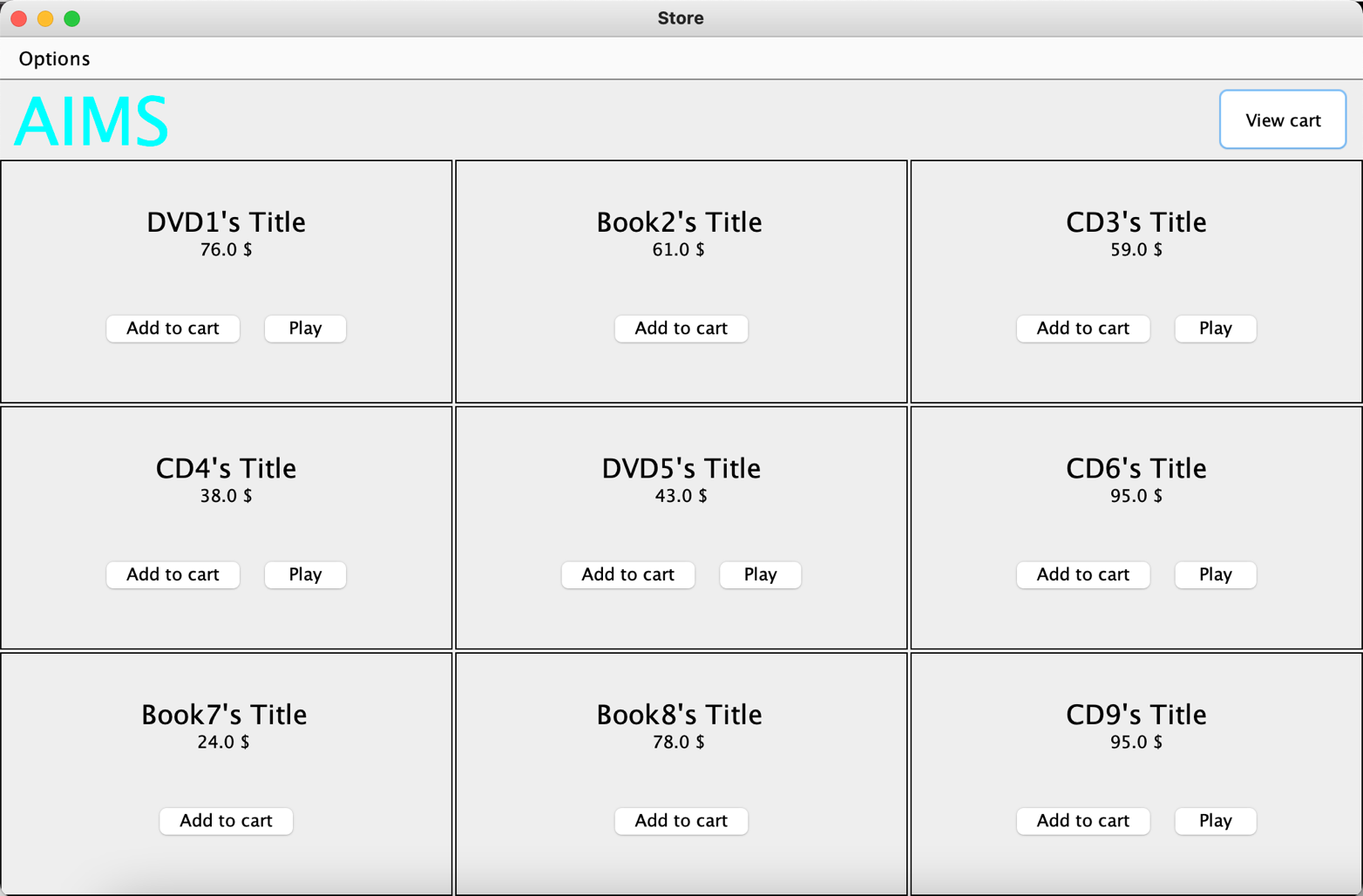
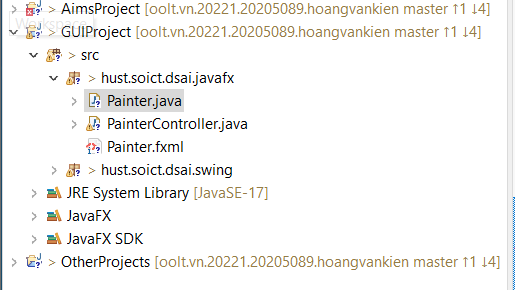


Figure 9. View Store Screen

# JavaFX API

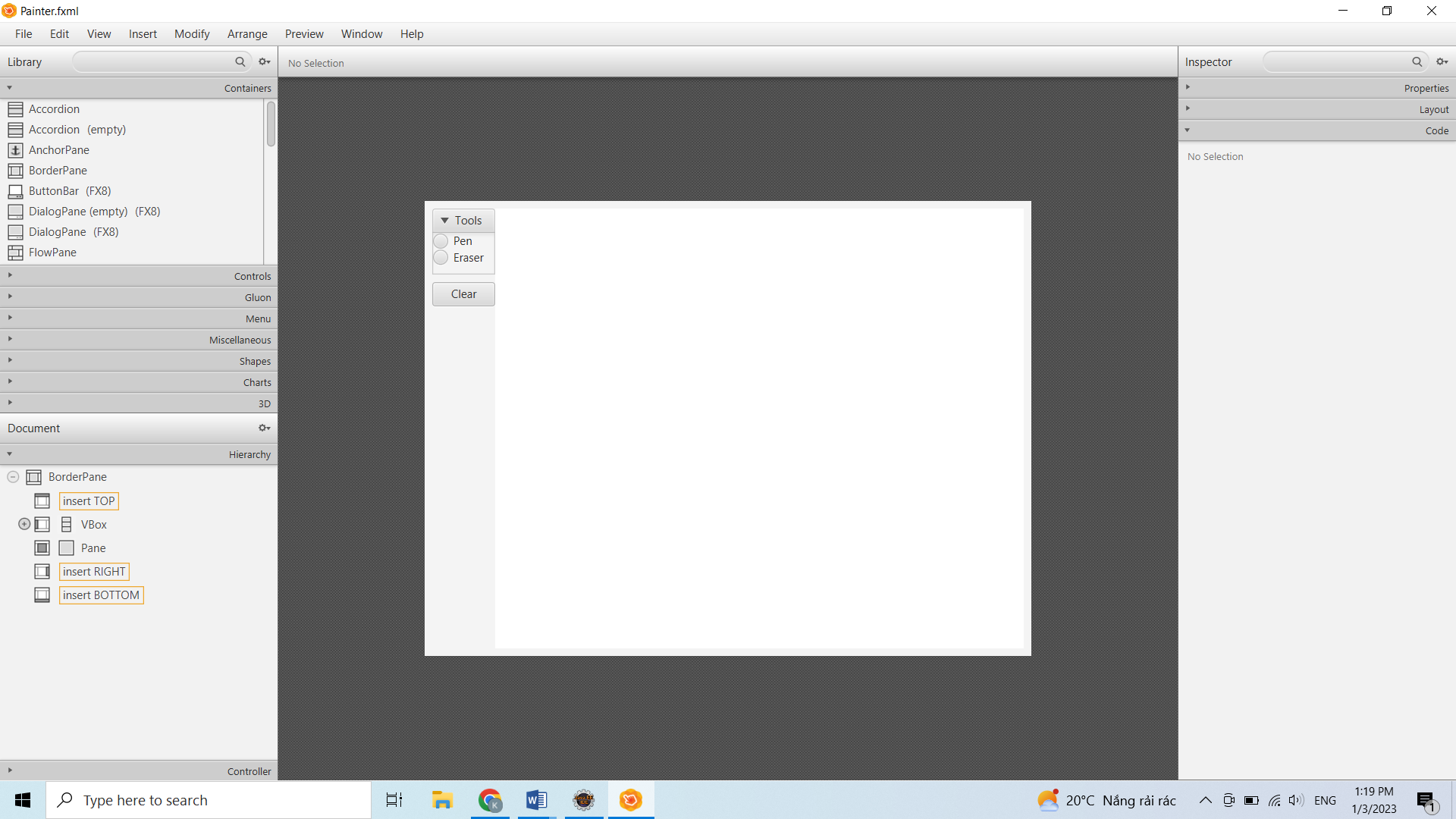
## **Create the FXML file**

### Create and open the FXML file in Scene Builder from Eclipse

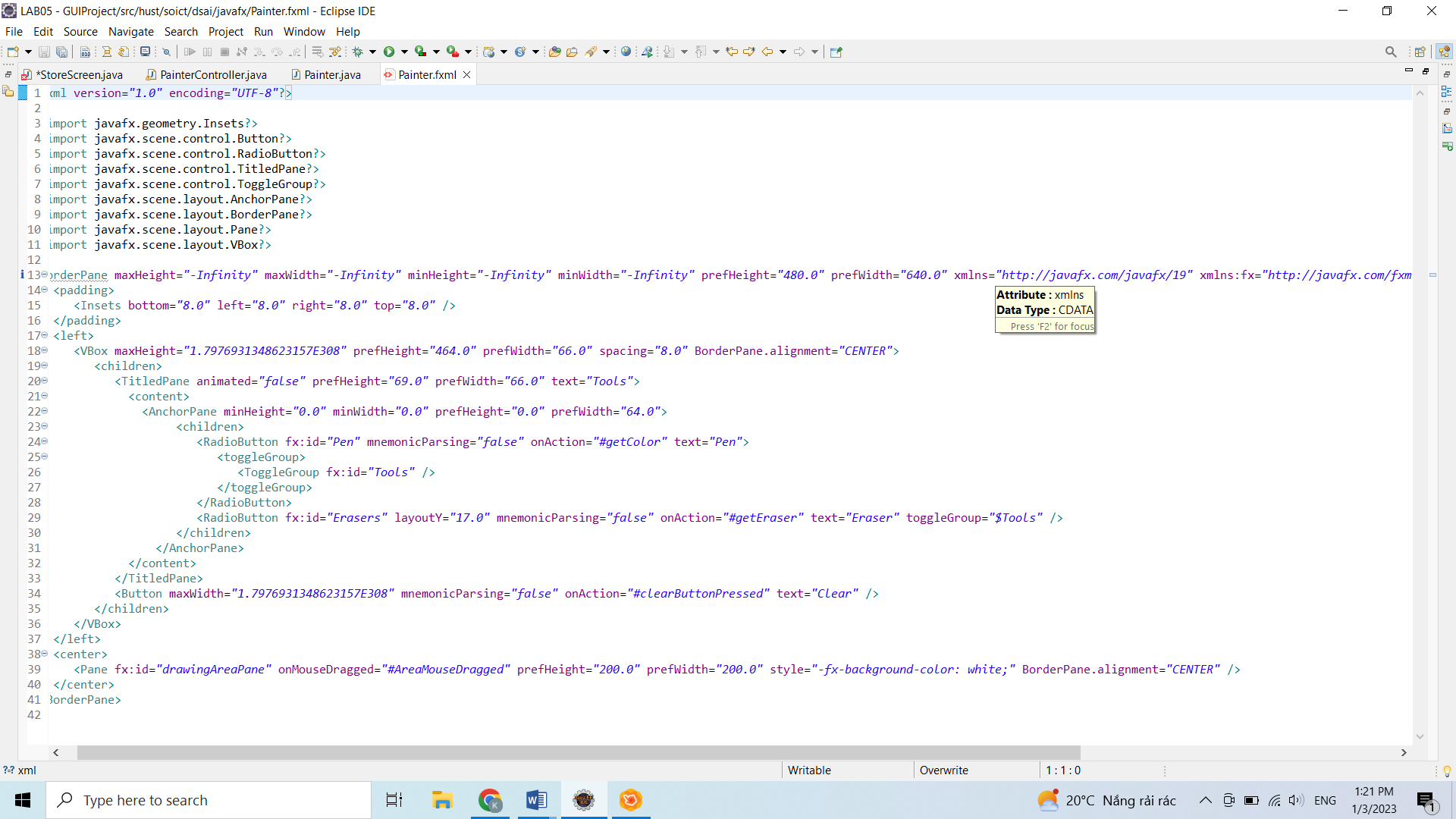


### Building the GUI

**Kết quả mà hình Gui:**

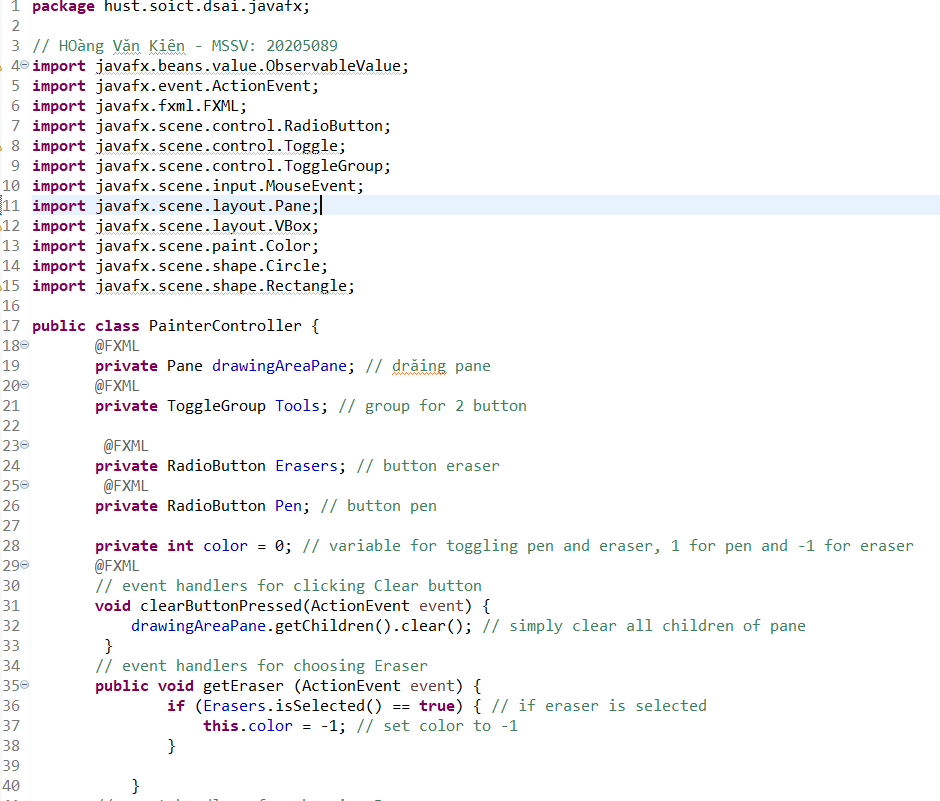


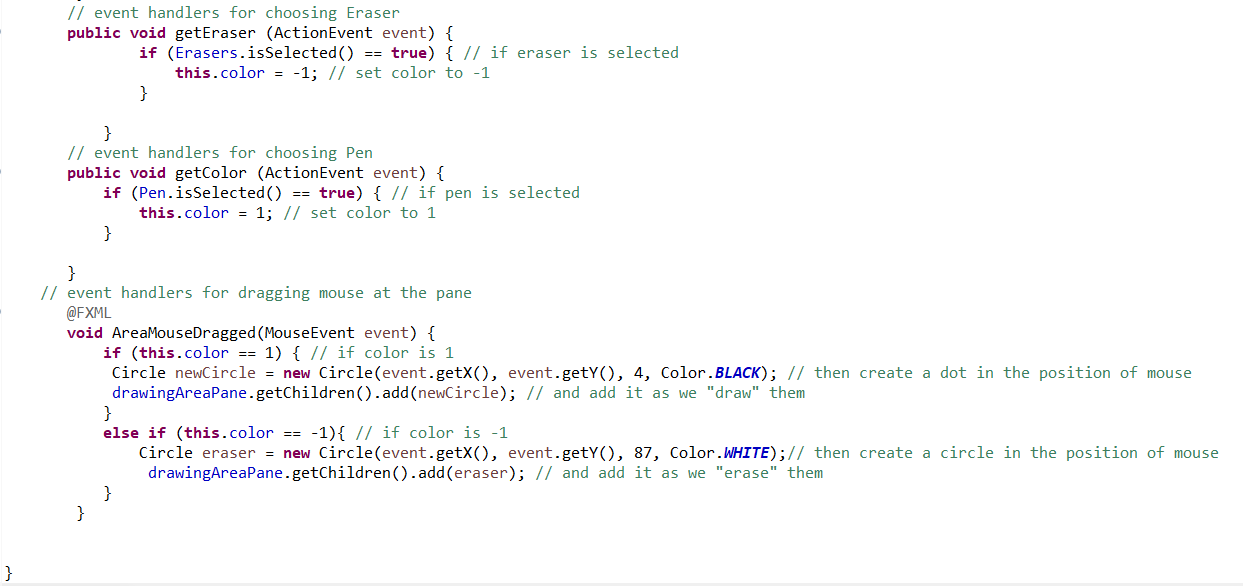
**Kết quả đoạn code file Painter.fxml:**



## **Create the controller class**

**Kết quả code PainterController.java:**

****

****

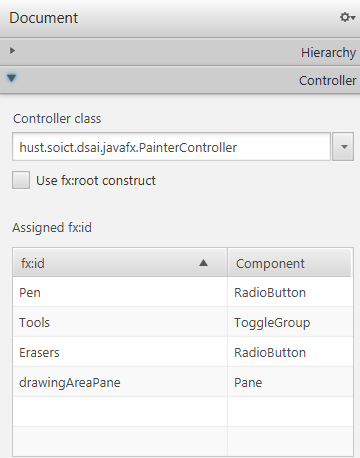
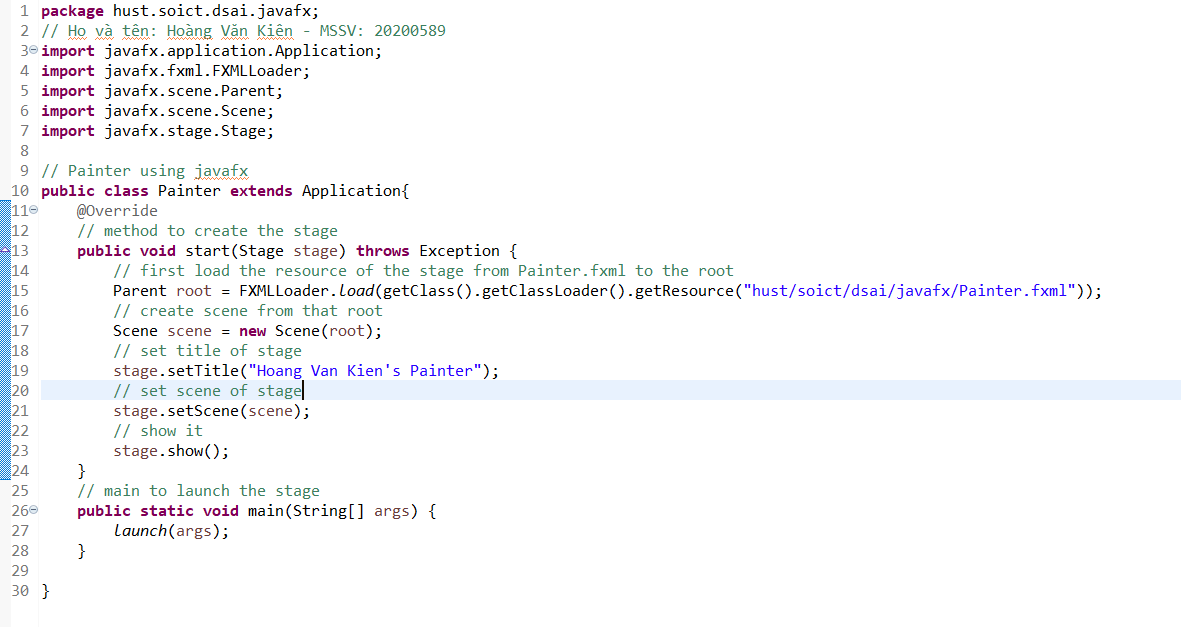


Figure 33. Specify the controller for the FXML file in Scene Builder

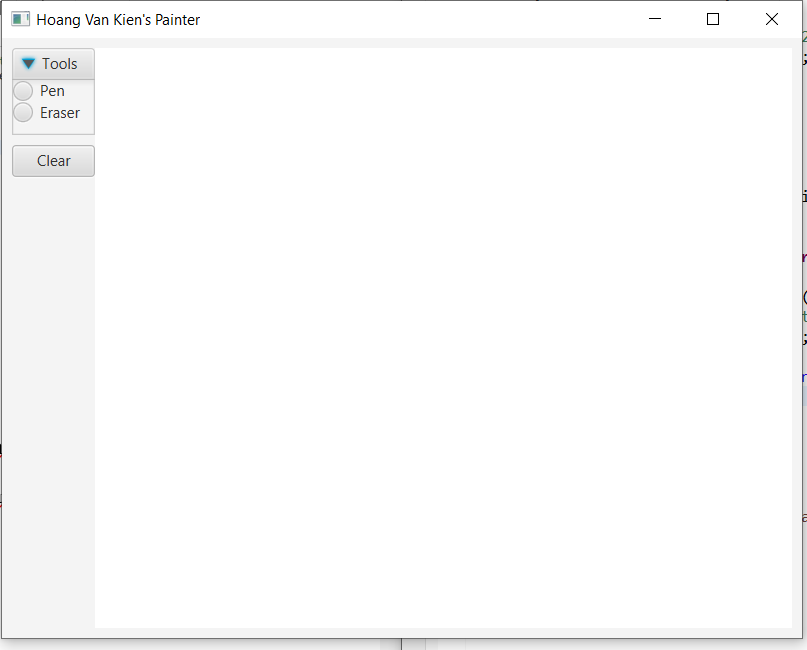
## Create the application

**Kết quả file Painter.java:**

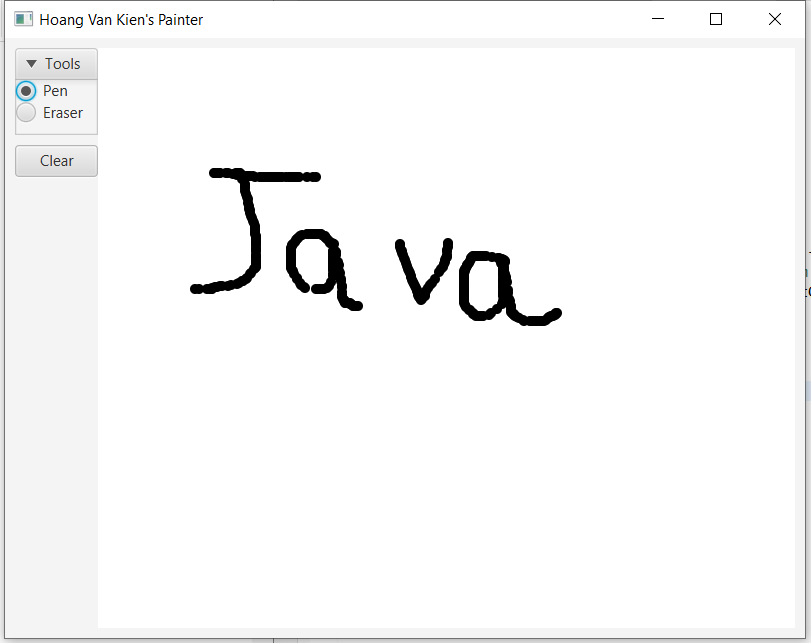


## **Kết quả app Painter**

* **Màn hình giao diện:**



* **Kết quả thử vẽ PEN**

****

* **Kết quả khi sử dụng Eraser**

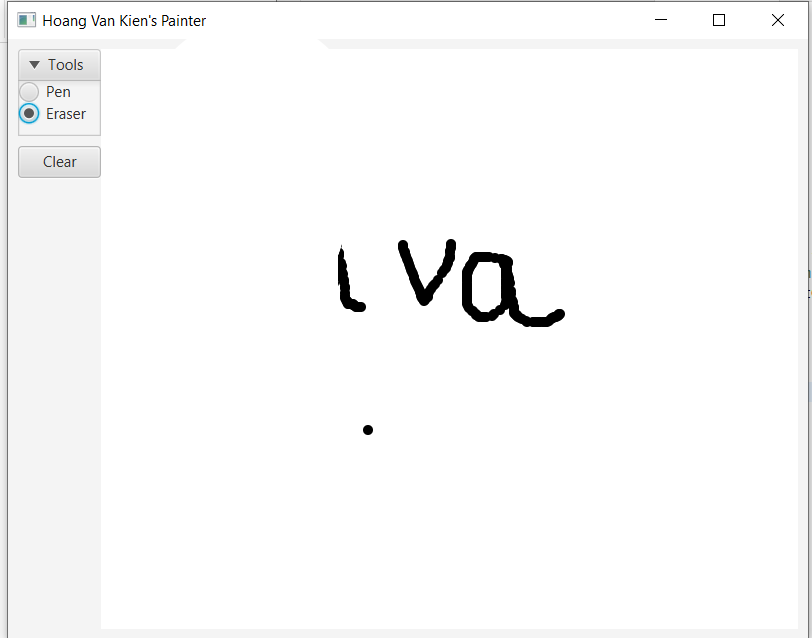
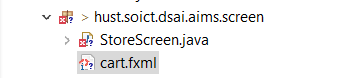
****

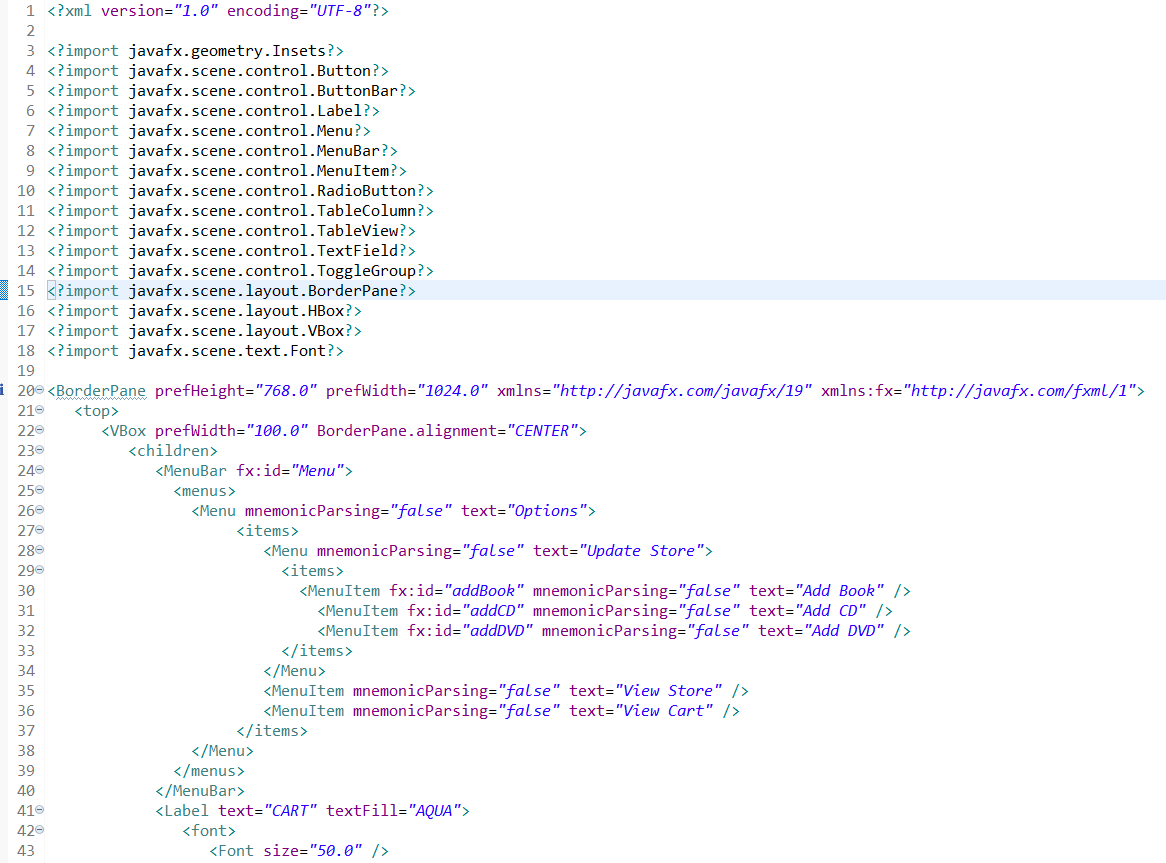
Figure 35. Painter with Eraser

# Setting up the View Cart Screen with ScreenBuilder

**Note**: Kết quả như sau:



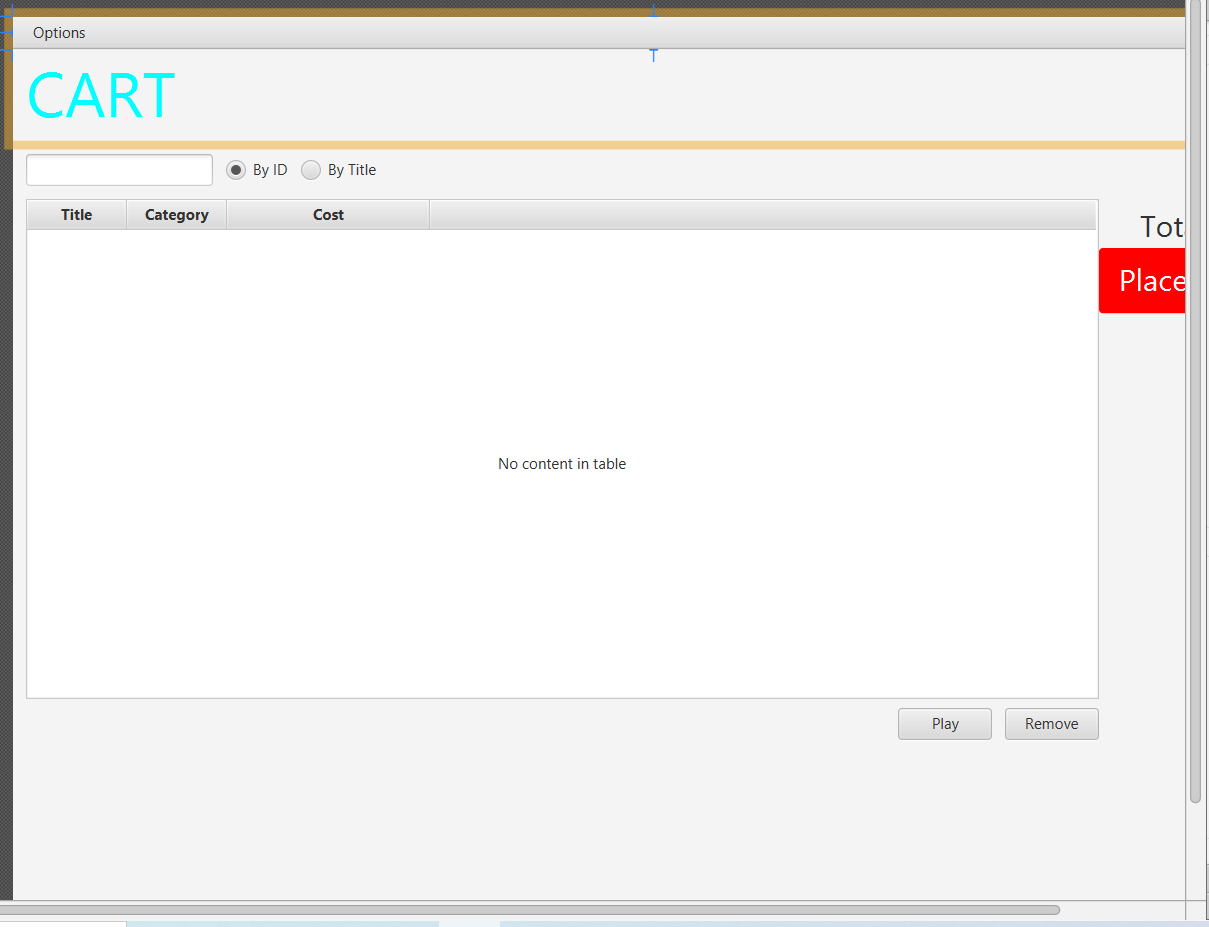
**Souce code sau khi kéo thả file cart.fxml:**

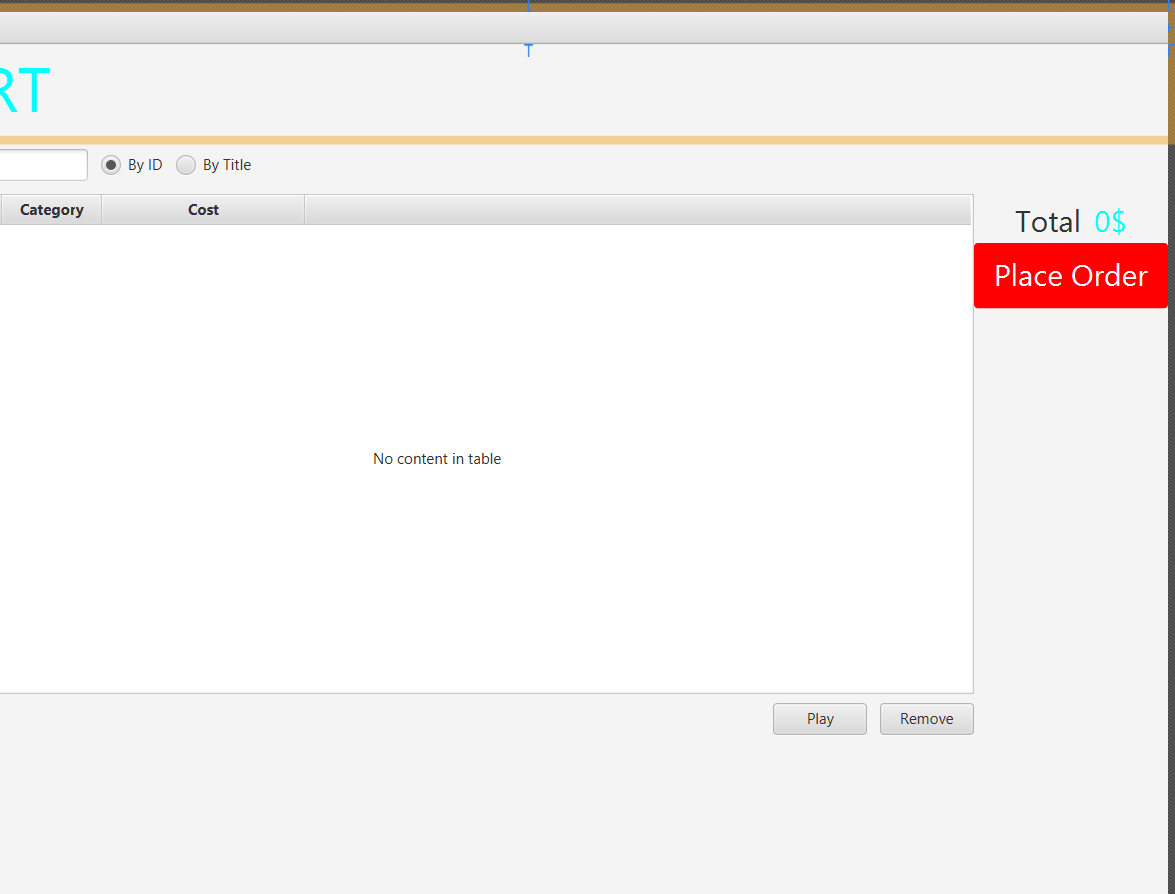
****

****

****

* **Kết quả sau khi design a View Cart Screen trên SceneBuilder như sau:**

****

****

# Integrating JavaFX into Swing application – The **JFXPanel** class

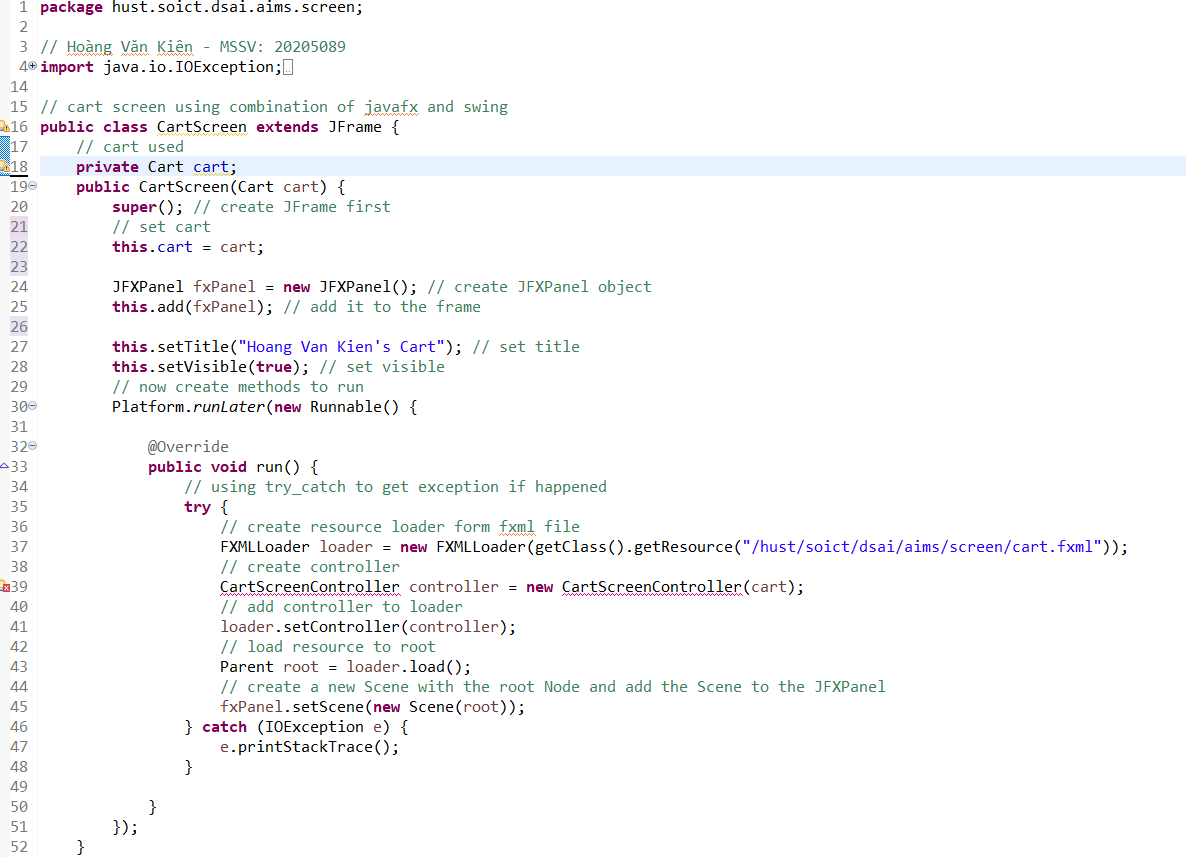


Figure 38. Source code for CartScreen

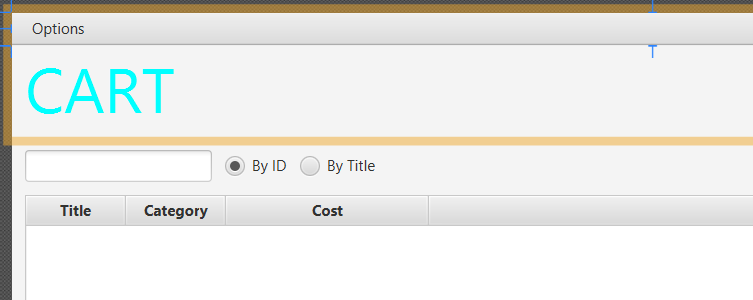
Chúng tôi tạo một lớp CartScreen mở rộng JFrame , giống như lớp StoreScreen . Bên trong hàm tạo của nó, chúng tôi thực hiện các bước sau:

* Vhúng tôi đã thiết lập một JFXPanel trong JFrame của chúng tôi .
* Chúng tôi tải Nút gốc từ tệp FXML và tạo đối tượng bộ điều khiển của nó (chúng ta sẽ đến phần triển khai thực tế của bộ điều khiển trong phần tiếp theo).
* Chúng ta tạo một Scene mới với Node gốc và thêm Scene vào JFXPanel .
* Hiện tại lớp CartScreen.java đang bị lỗi vi chưa có lớp CartScreenController.java

# View the items in cart – JavaFX’s data-driven UI

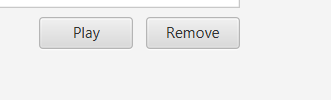
.Open the FXML file “**cart.fxml**” earlier and select the TableView element. Set its fx:id property to **tblMedia**. Similary, set the fx:id property for the columns:

* The Title column: **colMediaTitle**
* The Category column: **colMediacategory**
* The Cost column: **colMediaCost**

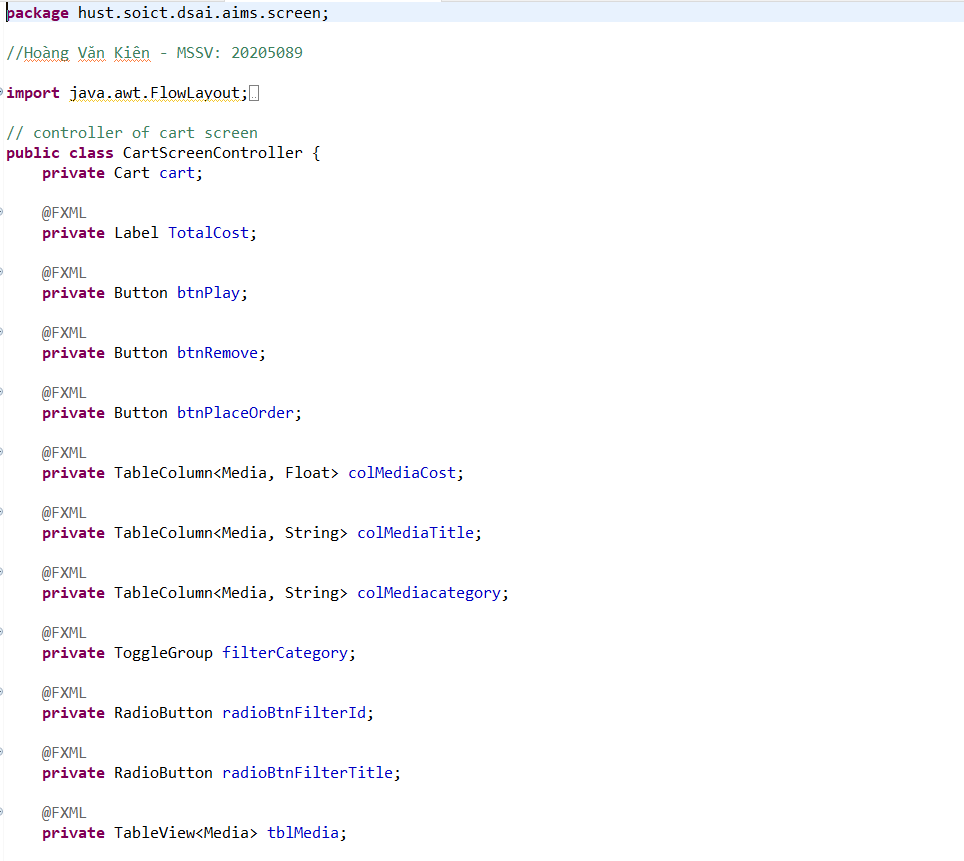


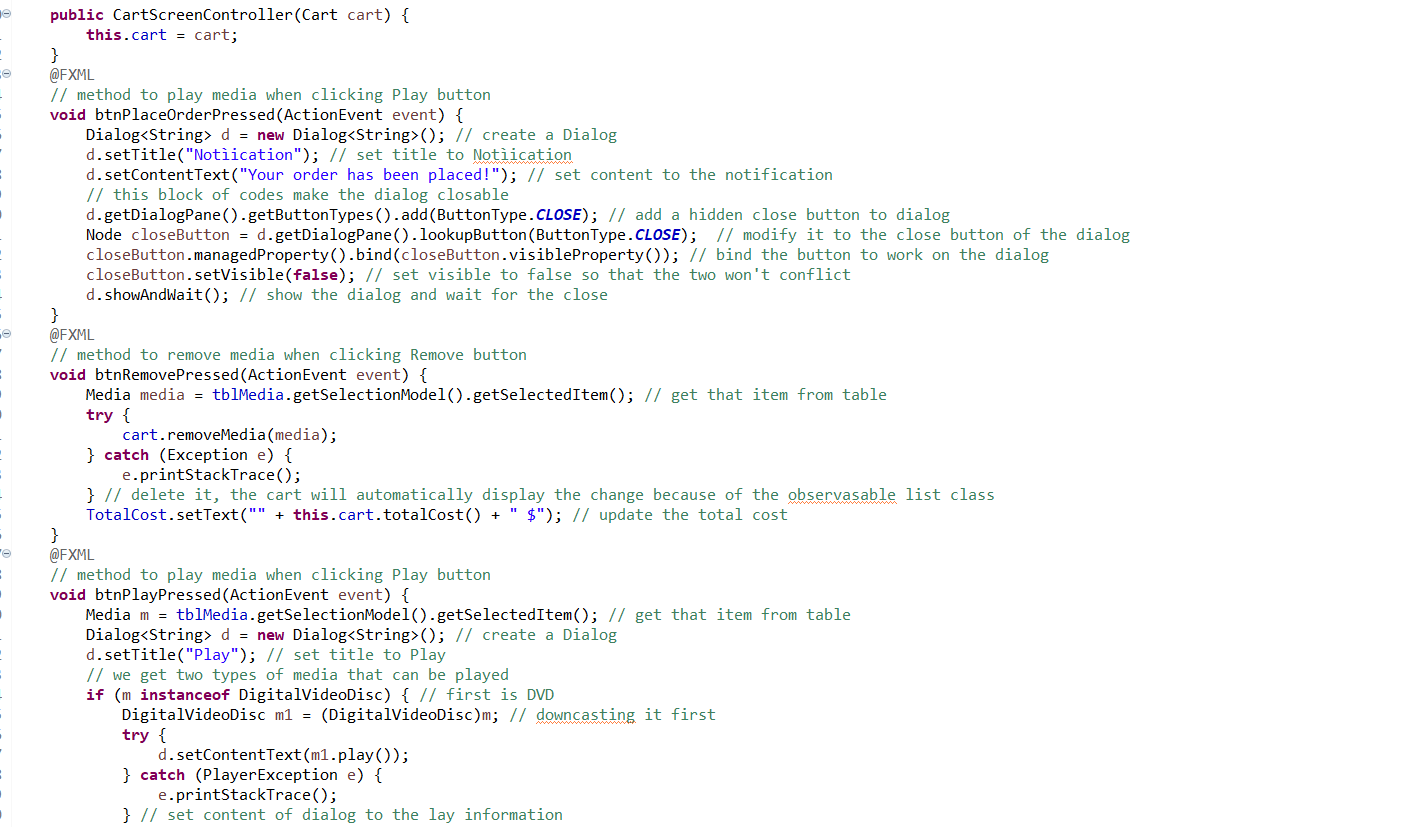
# Updating buttons based on selected item in **TableView** – **ChangeListener**

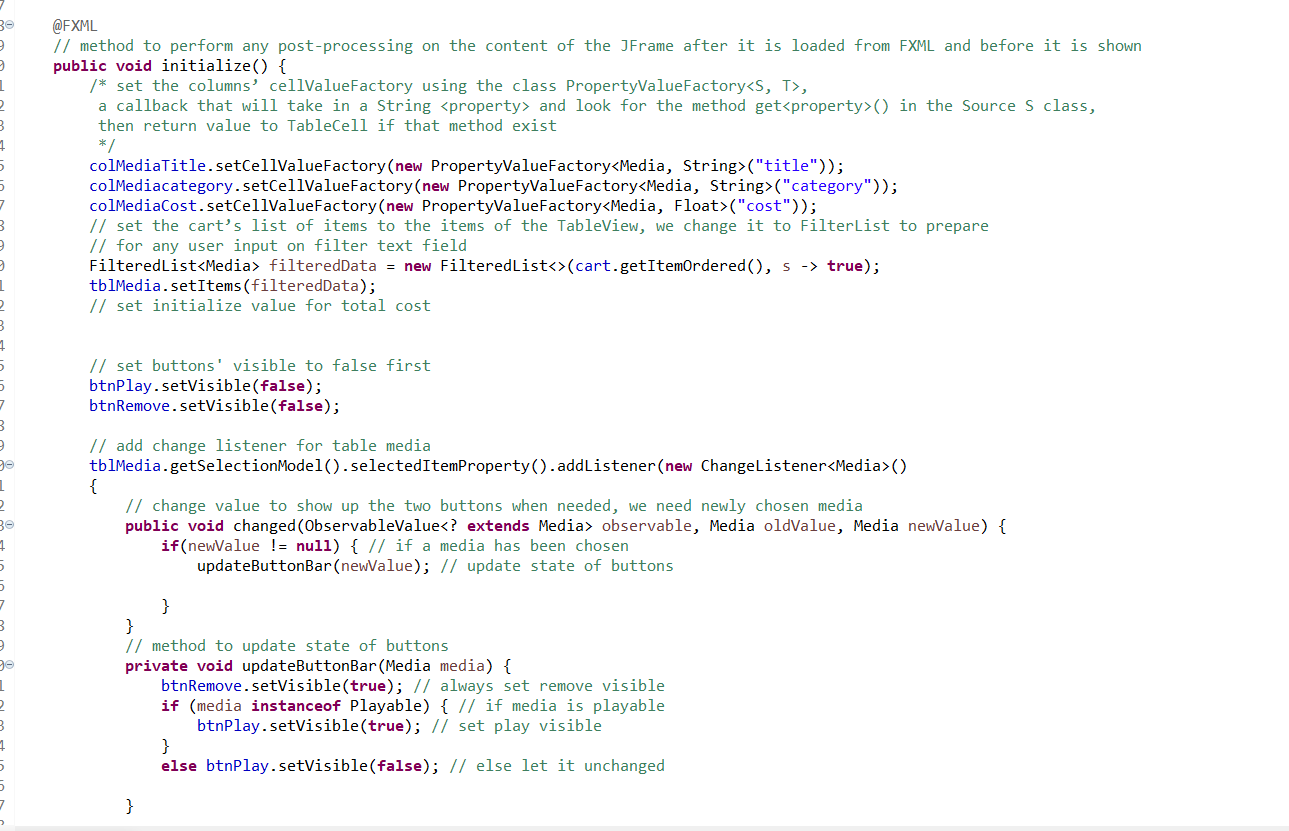
* The “Play” Button: **btnPlay**
* The “Remove” Button: **btnRemove**

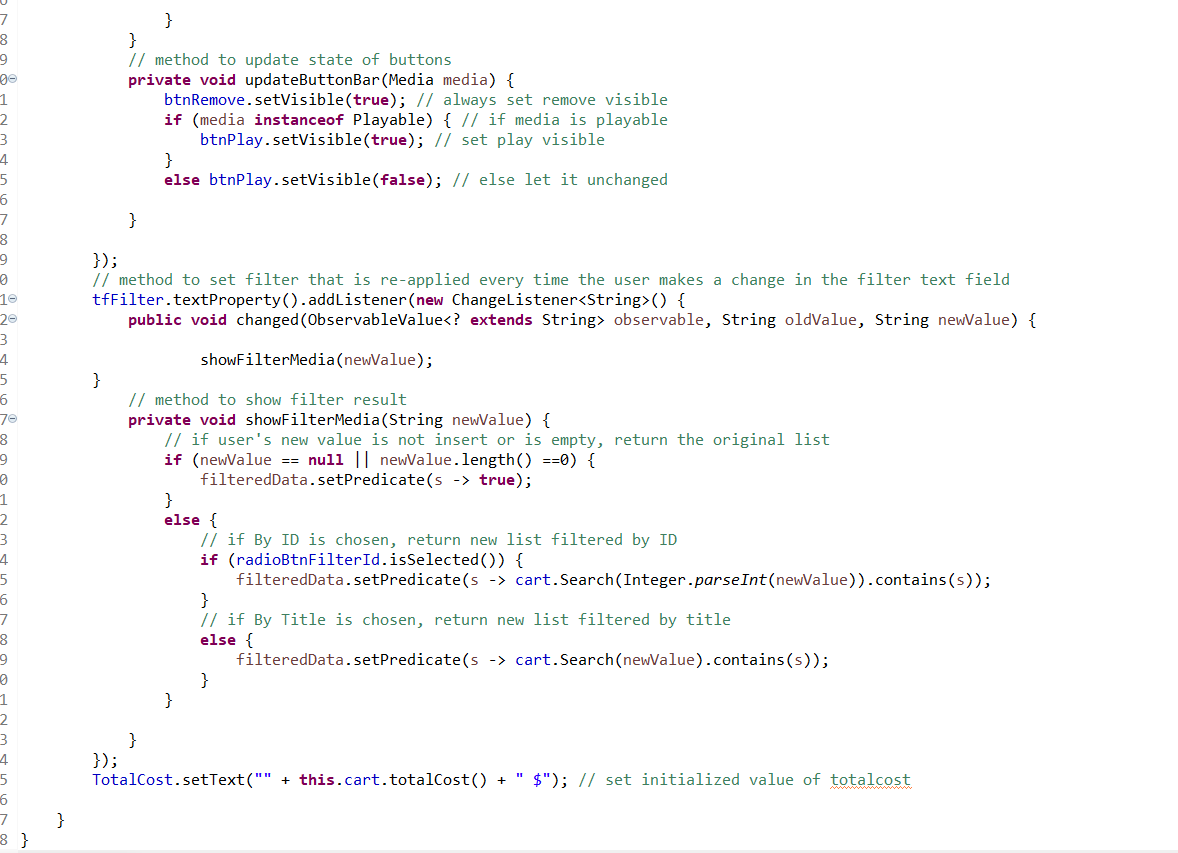


**Code file CartScreenController.java**

****

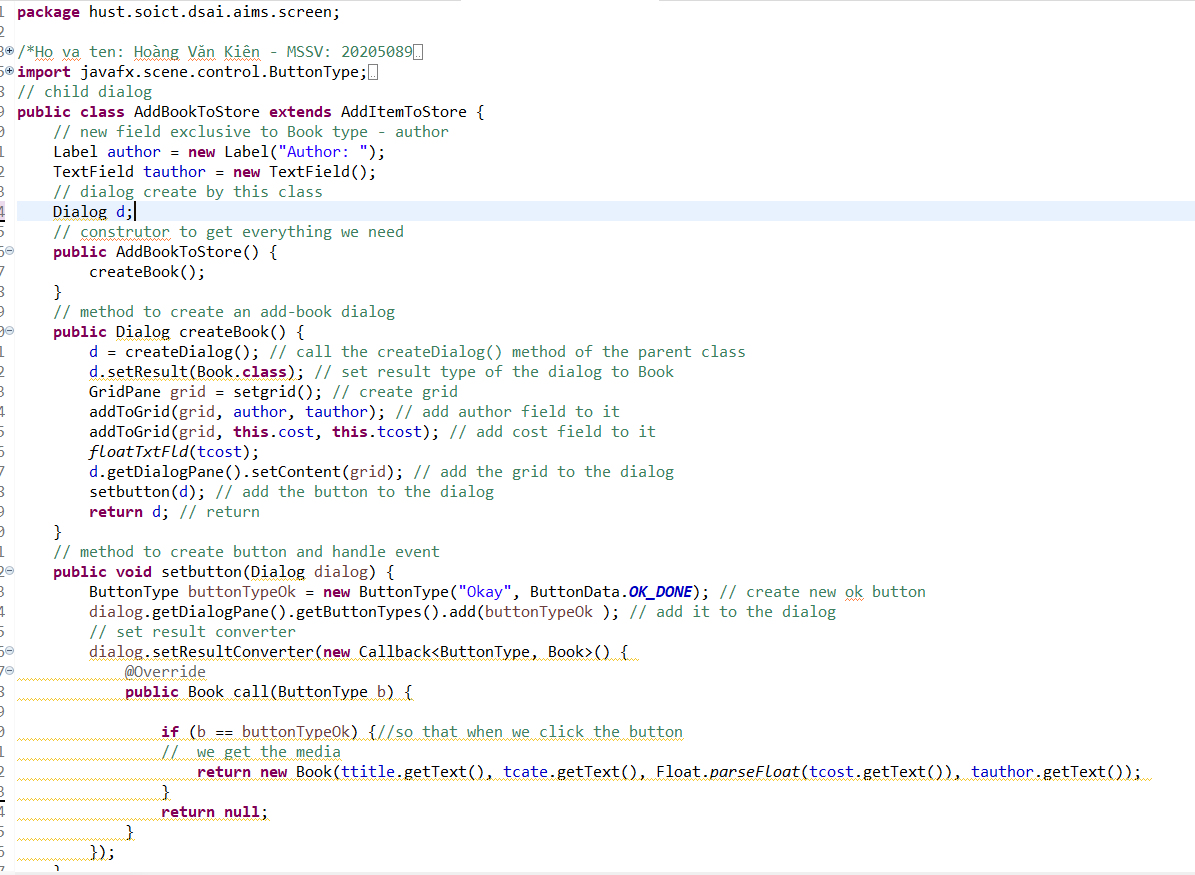
****

****

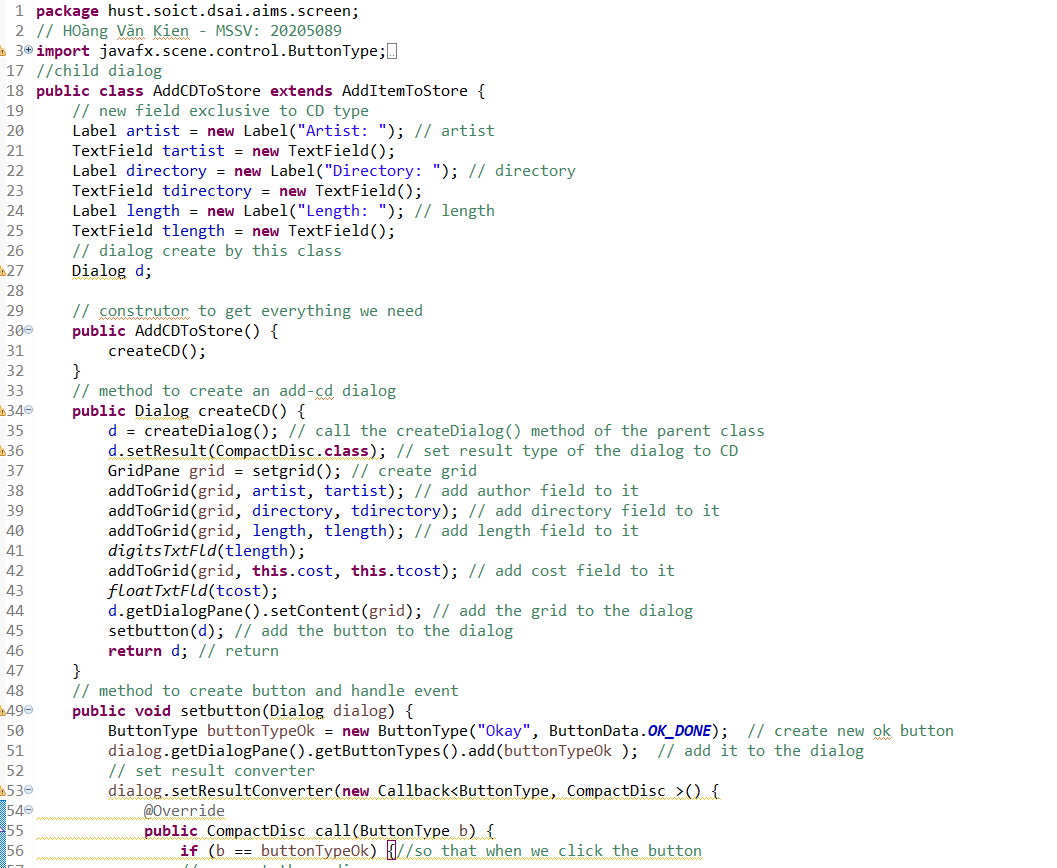
****

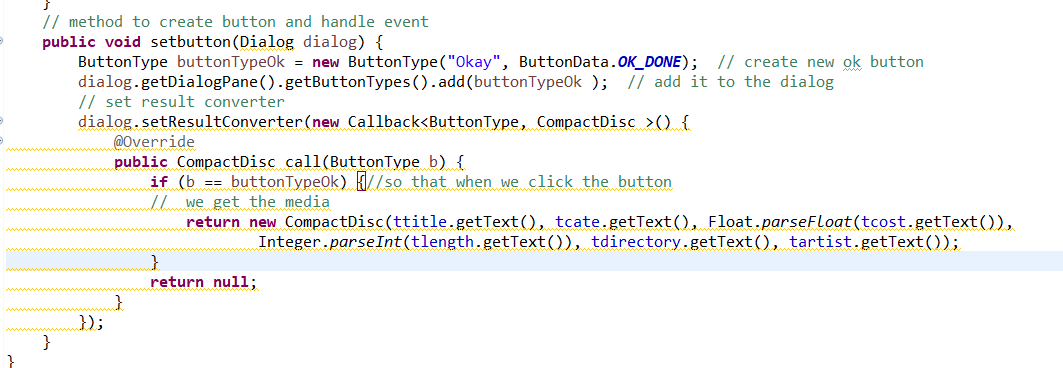
# Filter items in cart – **FilteredList**

* The TextField: **tfFilter**
* The RadioButton “By ID”: **radioBtnFilterId**
* The RadioButton “By Title”: **radioBtnFilterTitle**
* Code AddBookToStore

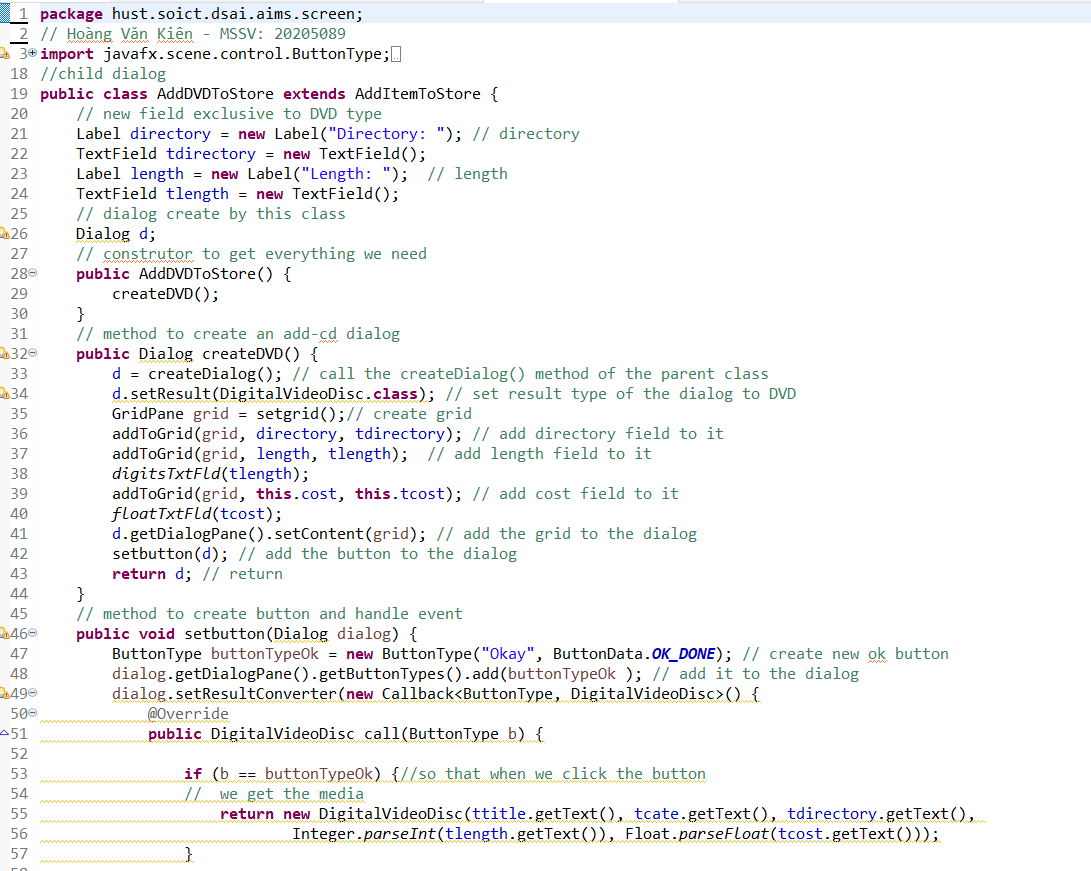


* Code AddCDToStore

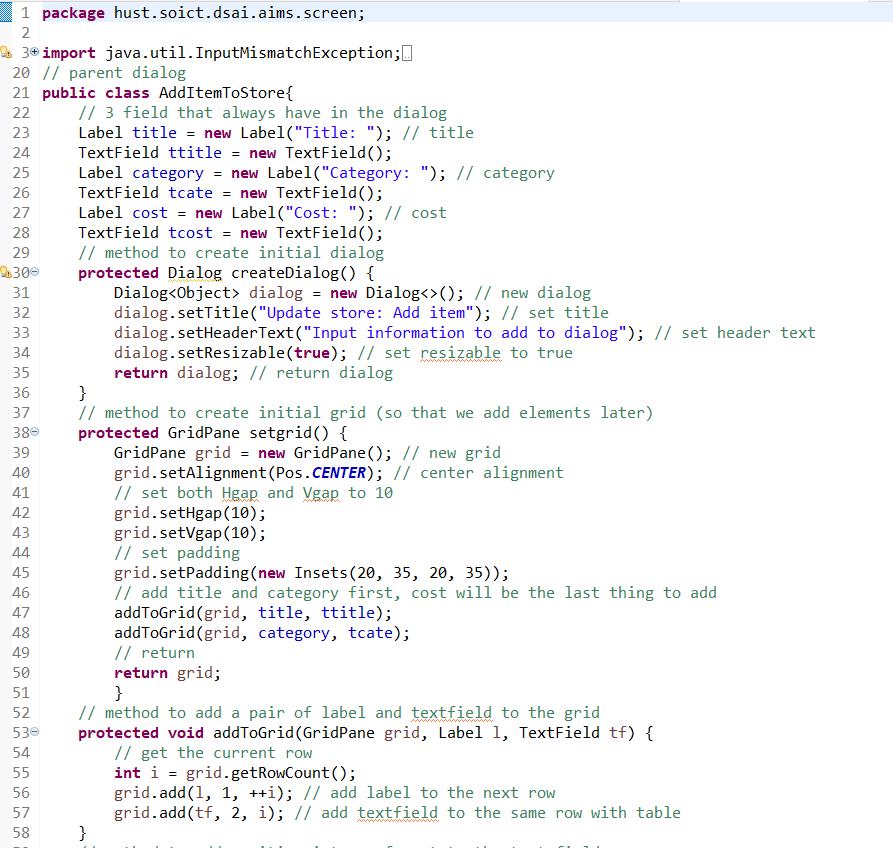




* Code AddDVDToStore



* Code AddItemToStore



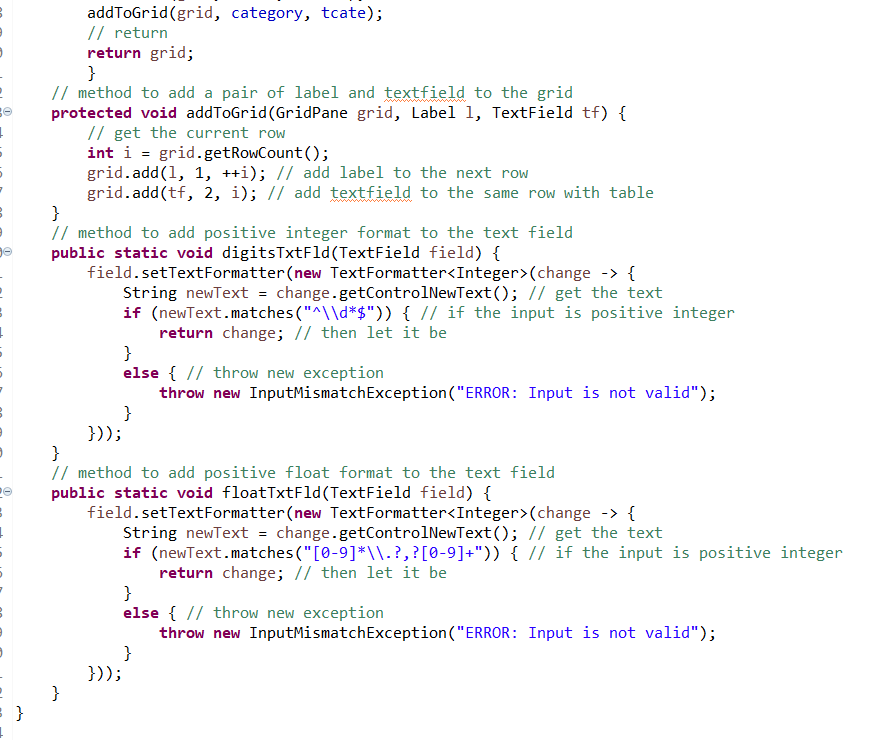
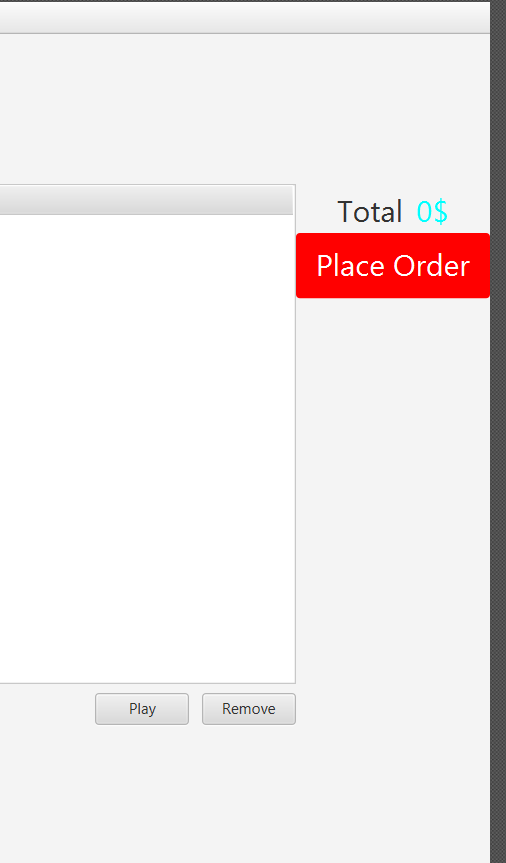


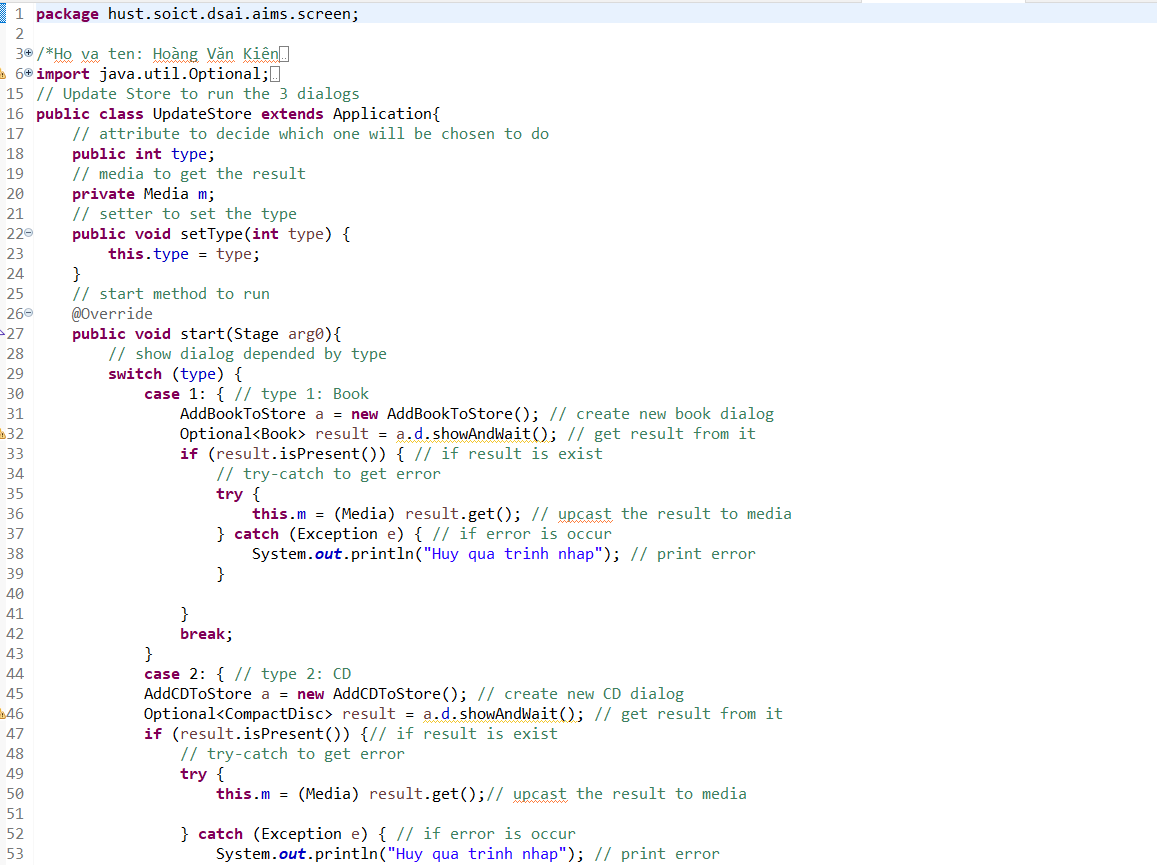
Figure 45. Adding ChangListener for tfFilter in initialize()

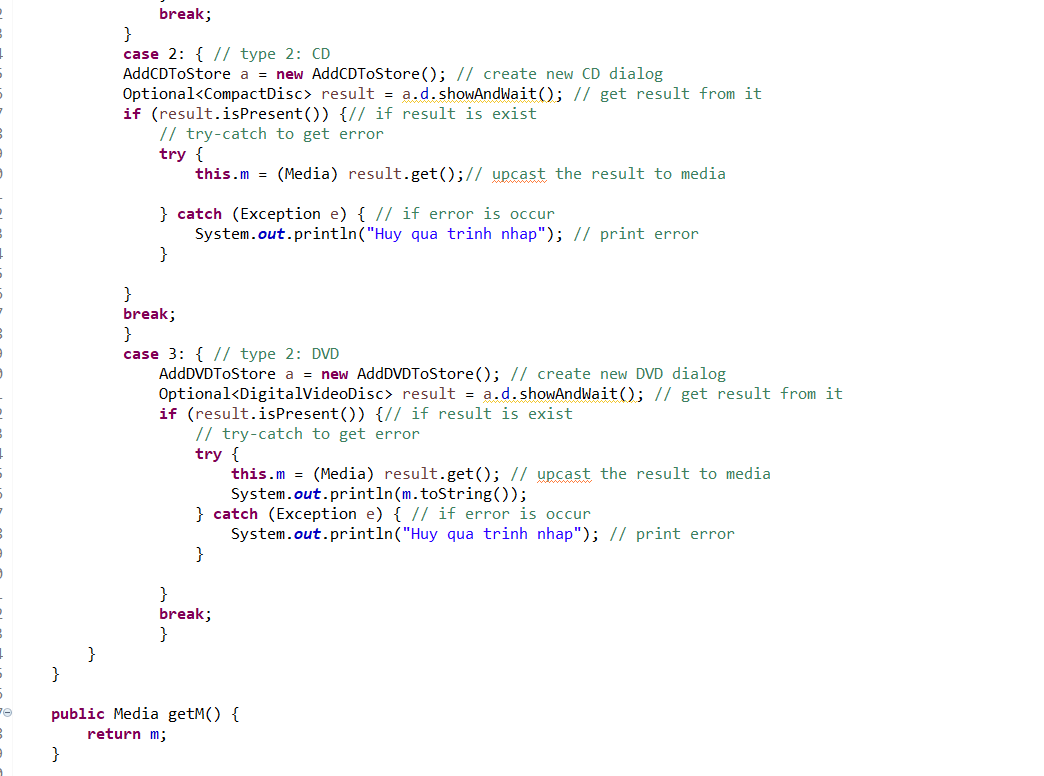
Please implement by yourself the showFilteredMedia() method. **Hint**: You might need to change the source code in previous exercises. Wrap the ObservableList in a FilteredList and set a new Predicate for the FilteredList each time you need to apply a new filter.

# Complete the Aims GUI application

Complete the remaining UI of Aims to make a functioning GUI application

* Cart Screen:
  + “Place order” Button
  + “Play” Button
  + The total cost Label - should updated along with changes in the current cart (add/remove).
  + MenuBar
  + 
* Code UpdateStore:





# Create a class which inherits from **Exception**

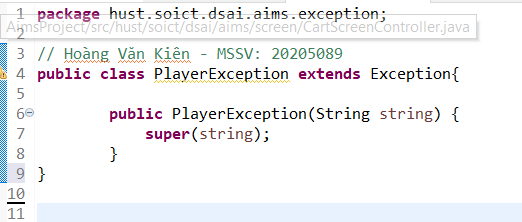
The **PlayerException** class represents an exception that will be thrown when an exceptional condition occurs during the playing of a media in your **AimsProject**.

## Create new class named **PlayerException**

- Enter the following specifications in the New Java Class dialog:

* Name: **PlayerException**
* Package: **hust.soict.[globalict||dsai].aims.exception**
* Access Modifier: **public**
* Superclass: **java.lang.Exception**
* Constructor from Superclass: checked
* **public static void main(String [] args):** do not check
* All other boxes: do not check





## Raise the **PlayerException** in the **play()** method

- Update **play()** method in **DigitalVideoDisc** and **Track**

* For each of **DigitalVideoDisc** and **Track**, update the **play()** method to first check the object's length using **getLength()** method. If the length of the **Media** is less than or equal to zero, the **Media** object cannot be played.
* At this point, you should output an error message using **System.err.println()** method and the **PlayerException** should be raised.

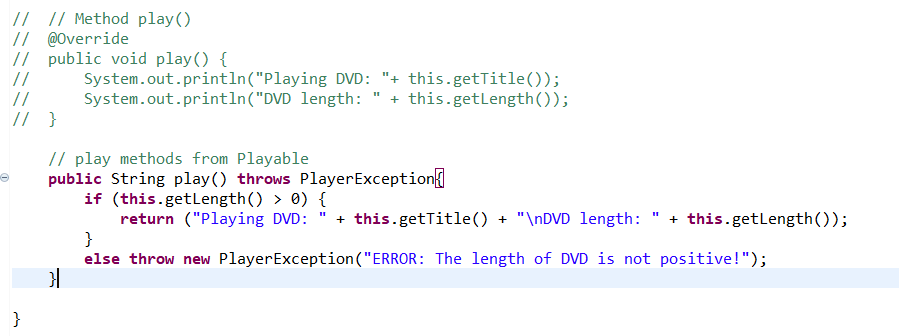
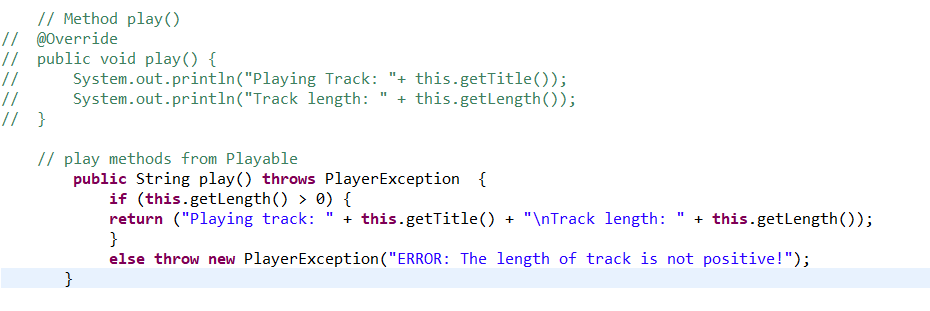


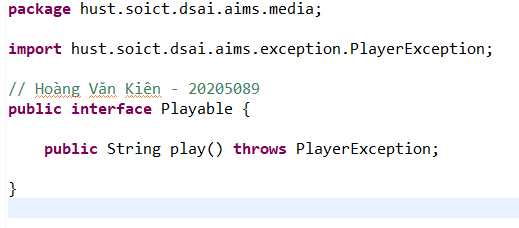
Figure 47. Code for method play() of DigitalVideoDisc

**- play() method of Track**

****

## Update **play()** in the **Playable** interface

**Code interface của Playable:**



## Update **play()** in **CompactDisc**

- The **play**() method in the **CompactDisc** is more interesting because not only it is possible for the **CompactDisc** to have an invalid **length** of 0 or less, but it is also possible that as it iterates through the tracks to play each one, there may have a track of length 0 or less

- First update the **play**() method in **CompactDisc** class to check the length using **getLength**() method as you did with **DigitalVideoDisc**

- Raise the **PlayerException**. Be sure to change the method signature to include **throws** **PlayerException** keywords.

- Update the **play()** method to catch a **PlayerException** raised by each **Track** using block **try-catch.**

**Code của CompactDisc:**

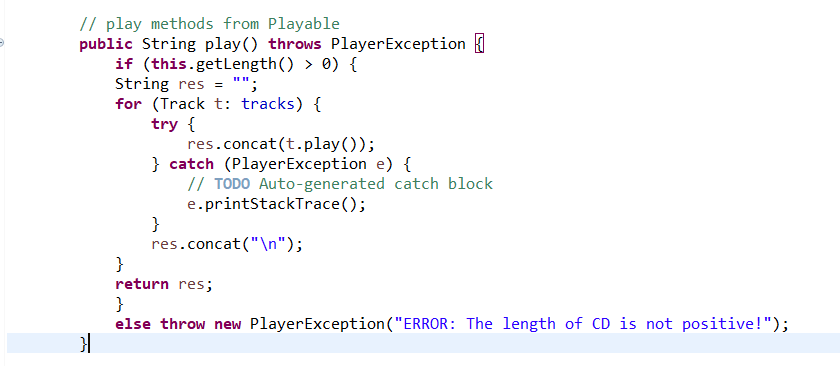
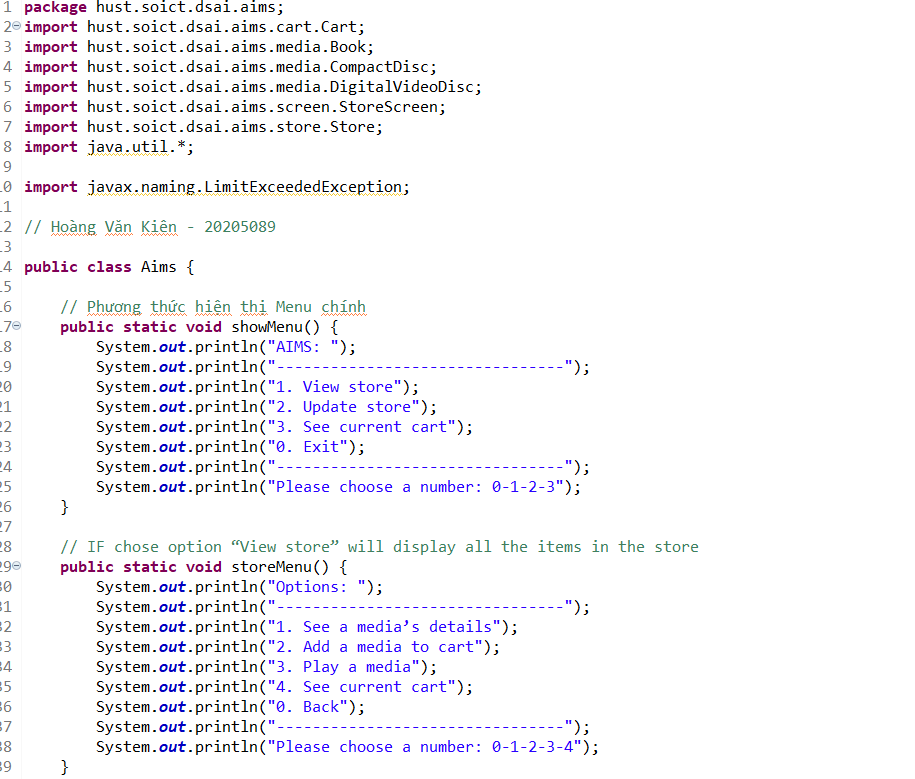
****

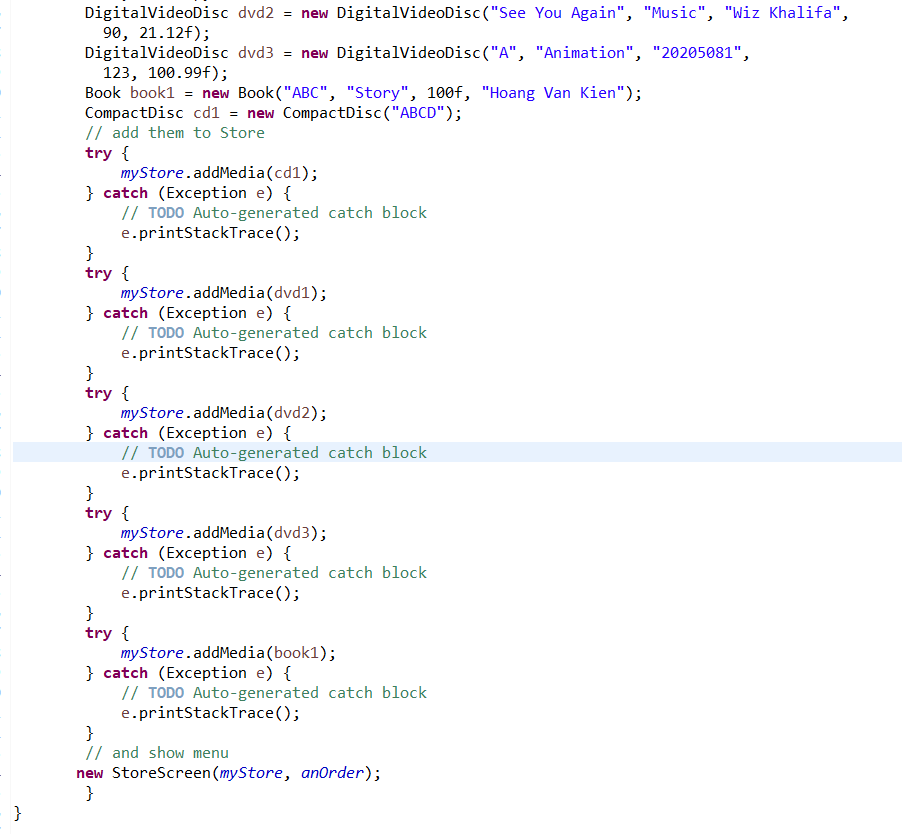
Figure 48. Code for method play() of CompactDisc

# Update the **Aims** class

**Code update Aims class:**

****

****

****