Bài 7:
Kiến trúc MVC

Mục tiêu bài học

- 1.Sự ra đời của mô hình MVC
- 2.Mô hình MVC trong swing

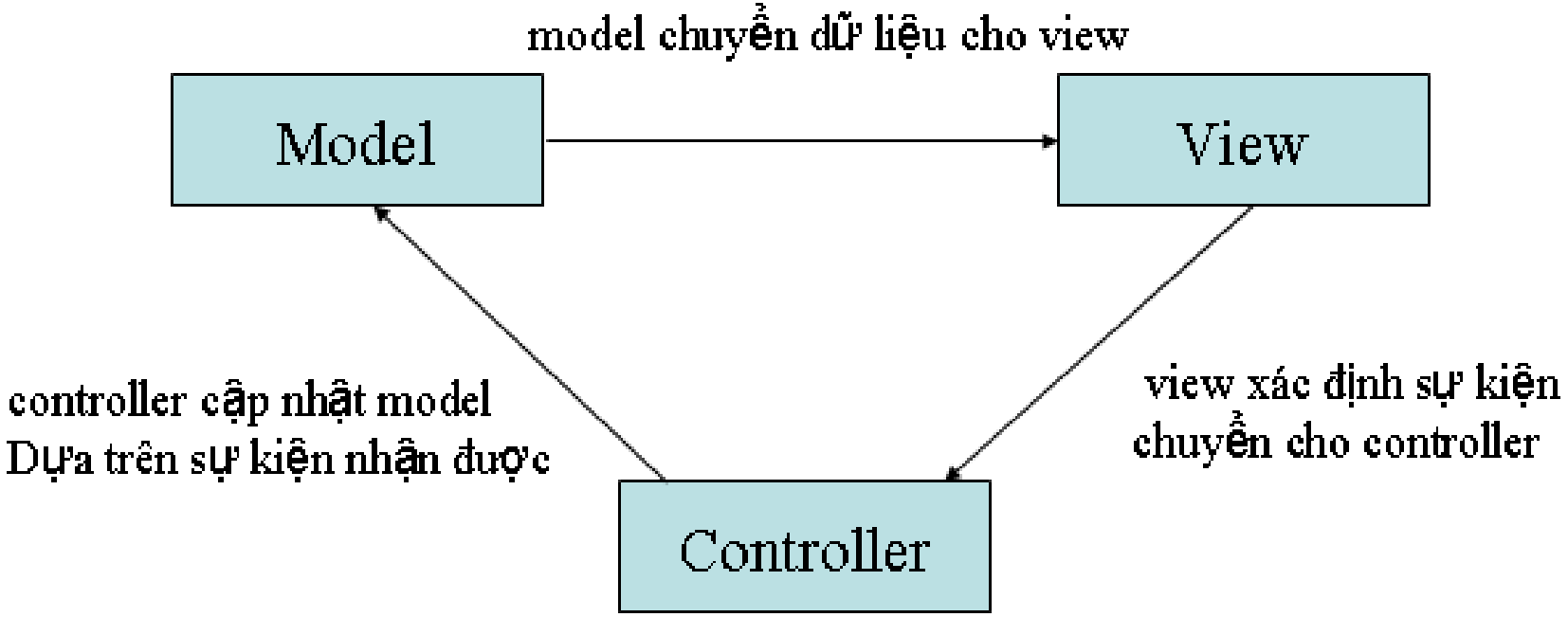
Kiến trúc MVC

- Bắt đầu vào những năm 70 của thế kỷ 20, tại phòng thí nghiệm Xerox PARC ở Palo Alto, sự ra đời của giao diện đồ họa và lập trình hướng đối tượng cho phép lập trình viên làm việc với những thành phần đồ họa như những đối tượng đồ họa có thuộc tính và phương thức riêng của nó. Không dừng lại ở đó, những nhà nghiên cứu ở Xerox PARC còn đi xa hơn khi cho ra đời cái gọi là ***kiến trúc MVC*** (viết tắt của Model – View – Controller).

Kiến trúc MVC

- MVC được dùng một cách rộng rãi trong nhiều hệ thống phần mềm hướng đối tượng , bất kể được viết bằng ngôn ngữ hướng đối tượng nào. Và MVC được biết đến như là một thiết kế giao diện người dùng hướng đối tượng khá tốt

Kiến trúc MVC



Hình 1: Mô hình MVC chuẩn

Kiến trúc MVC

- Model (Mô hình)
- Model chứa đựng trạng thái dữ liệu của mỗi thành phần. Có những mô hình khác nhau cho những thành phần khác nhau. Ví dụ, model của một scrollbar có thể chứa thông tin về vị trí hiện tại có thể điều chỉnh được "thumb", giá trị nhỏ nhất và giá trị lớn nhất của nó, và độ rộng của thumb (quan hệ với dãy giá trị). Một menu lại khác, nó có thể đơn giản chỉ chứa một danh sách các menu item mà người dùng có thể chọn.

Kiến trúc MVC

- Model chịu trách nhiệm nắm giữ tất cả các thể hiện của trạng thái của thành phần. Ví dụ, những giá trị như trạng thái nhấn hoặc không nhấn của một button, và dữ liệu ký tự của một thành phần text cũng như thông tin về việc nó có cấu trúc như thế nào. Một Model có thể chịu trách nhiệm với việc giao tiếp một cách gián tiếp với View và Controller. Gián tiếp có nghĩa là Model không biết View và Controller của nó - nó không duy trì hoặc tìm kiếm các dẫn xuất đến chúng. Thay vào đó, Model sẽ gởi đi những khai báo hoặc những broadcast (những gì chúng ta biết là những sự kiện).

Kiến trúc MVC

- View (Hiển thị)
- View liên quan đến việc bạn nhìn thấy thành phần trên màn hình như thế nào. Đây chính là thành phần "look". Một ví dụ dễ thấy hiển thị có thể khác nhau như thế nào, hãy nhìn vào một ứng dụng Windows trong hai platform GUI khác nhau. Hầu hết frame của cửa sổ có một thanh tiêu đề nằm trên đỉnh của cửa sổ. Tuy nhiên, thanh tiêu đề có thể có một nút close ở phía bên trái như trong platform MacOS, hoặc nó có thể có một nút close ở phía bên phải như trong platform Windows. Đó là những ví dụ những kiểu hiển thị khác nhau của cùng một kiểu đối tượng cửa sổ.

Kiến trúc MVC

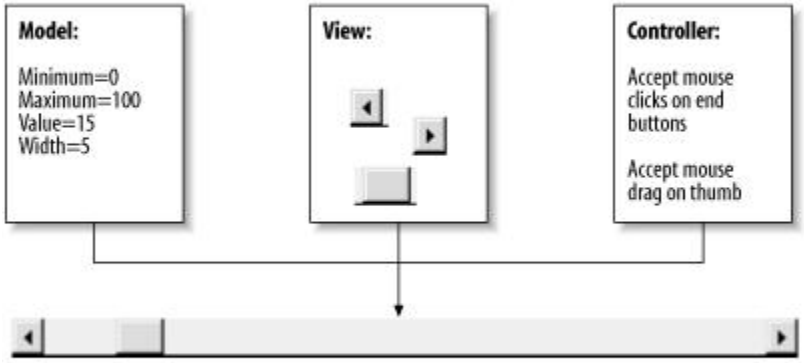
- View xác định việc hiện thị trực quan của Model của thành phần. View chịu trách nhiệm giữ việc thay thế nó trên màn hình luôn được cập nhật, nó có thể nhận những thông điệp gián tiếp từ Model hoặc những thông điệp từ Controller.

Kiến trúc MVC

- Controller (Điều khiển)
- Controller là phần kết nối với giao tiếp người dùng để ra lệnh cho thành phần tương tác như thế nào với những sự kiện. Những sự kiện đến từ nhiều trường hợp - ví dụ, một click chuột, sự kiện bàn phím bắt đầu một lệnh menu cụ thể, hoặc thậm chí một lệnh vẽ lại một phần của màn hình.
- Controller chịu trách nhiệm xác định thành phần nào tác động lại bất kỳ sự kiện nào từ thiết bị input như bàn phím và chuột. Controller chính là "feel" của thành phần, và nó cũng xác định hành động nào được thực thi khi thành phần được sử dụng. Controller có thể nhận thông điệp từ View, và những thông điệp gián tiếp từ Model.

Kiến trúc MVC

- Hình dưới chỉ ra Model, View, Controller làm việc với nhau như thế nào để tạo ra một scrollbar. Model nắm giữ những thông tin về giá trị min và max. View xác định chính xác vẽ scrollbar như thế nào và vẽ ở vị trí nào. Cuối cùng, Controller chịu trách nhiệm xử lý các sự kiện chuột. Kết quả là một scrollbar mang đầy đủ chức năng của MVC



Kiến trúc MVC

- **<u>Ví dụ 1</u>: GUI Component đơn giản là Checkbox**
- Checkbox có thành phần:
  - Model để quản lý trạng thái của nó là check hay uncheck
  - View để thể hiện nó với trạng thái tương ứng lên màn hình
  - Controller :để xử lý những sự kiện khi có sự tương tác của người sử dụng hoặc các đối tượng khác lên Checkbox.

Kiến trúc MVC

- **Ví dụ 2:** Trên JList
- Model: mô hình dữ liệu cho JList
- View: là giao diện hiển thị các mục chọn
- Controller: xử lý sự kiện mỗi khi các mục trong JList được chọn

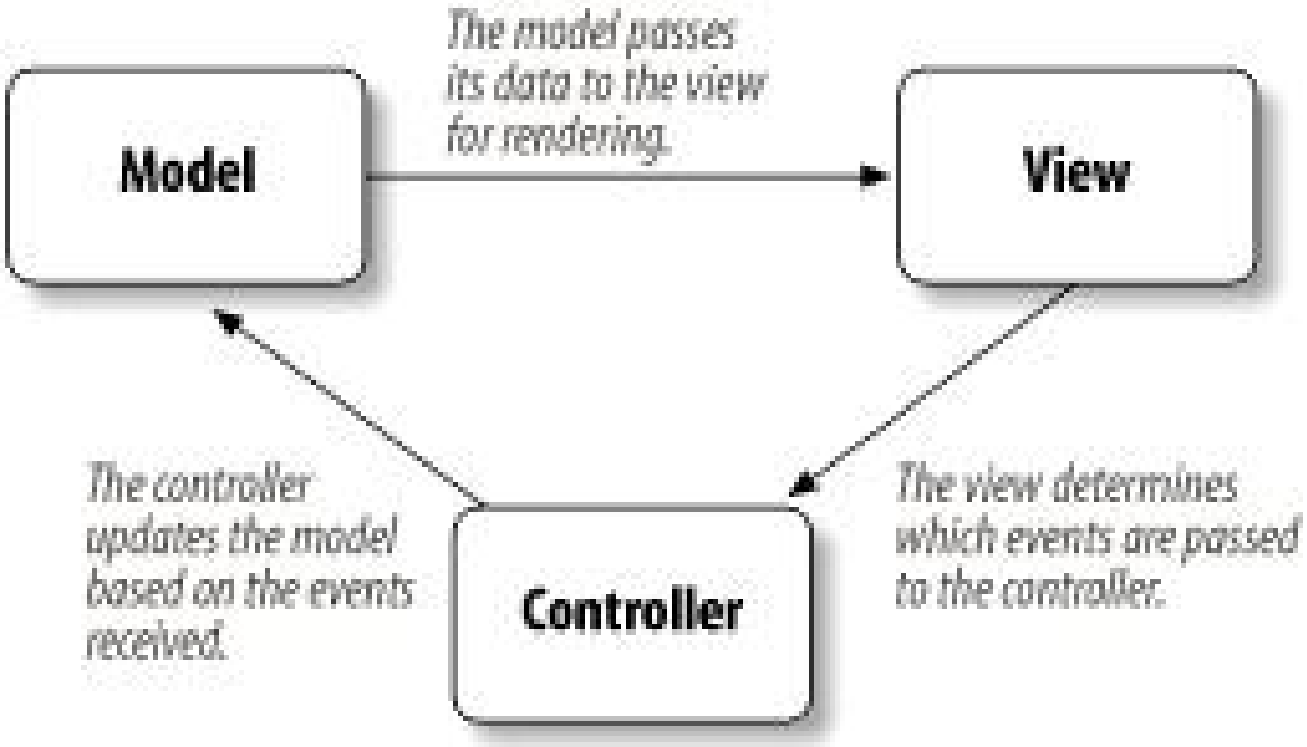
Sự tương tác trong MVC

- Với MVC, mỗi yếu tố Model, View, Controller yêu cầu những dịch vụ của những yếu tố khác để giữ bản thân nó tiếp tục được cập nhật.
- Ví dụ, giả sử rằng chúng ta có một checkbox được check trong giao diện. Nếu Controller xác định người dùng thực hiện một click chuột, nó có thể gởi một thông điệp cho View. Nếu View xác định rằng click xảy ra trên checkbox, nó gởi một thông điệp cho Controller.Controller cập nhật lại Model dựa trên sự kiện nhận được.

Sự tương tác trong MVC

- Model sau khi cập nhật bản thân nó và thông báo một thông điệp, sẽ được nhận bởi View, để thông báo với View rằng nó phải cập nhật lại bản thân nó dựa trên trạng thái mới của Model. Tại đây có một vòng quay liên tục được lặp lại. Theo cách này, một Model không hạn chế một View hoặc Controller cụ thể, điều này cho phép chúng ta có nhiều View và Controller khác nhau thực thi một Model.

Sự tương tác trong MVC



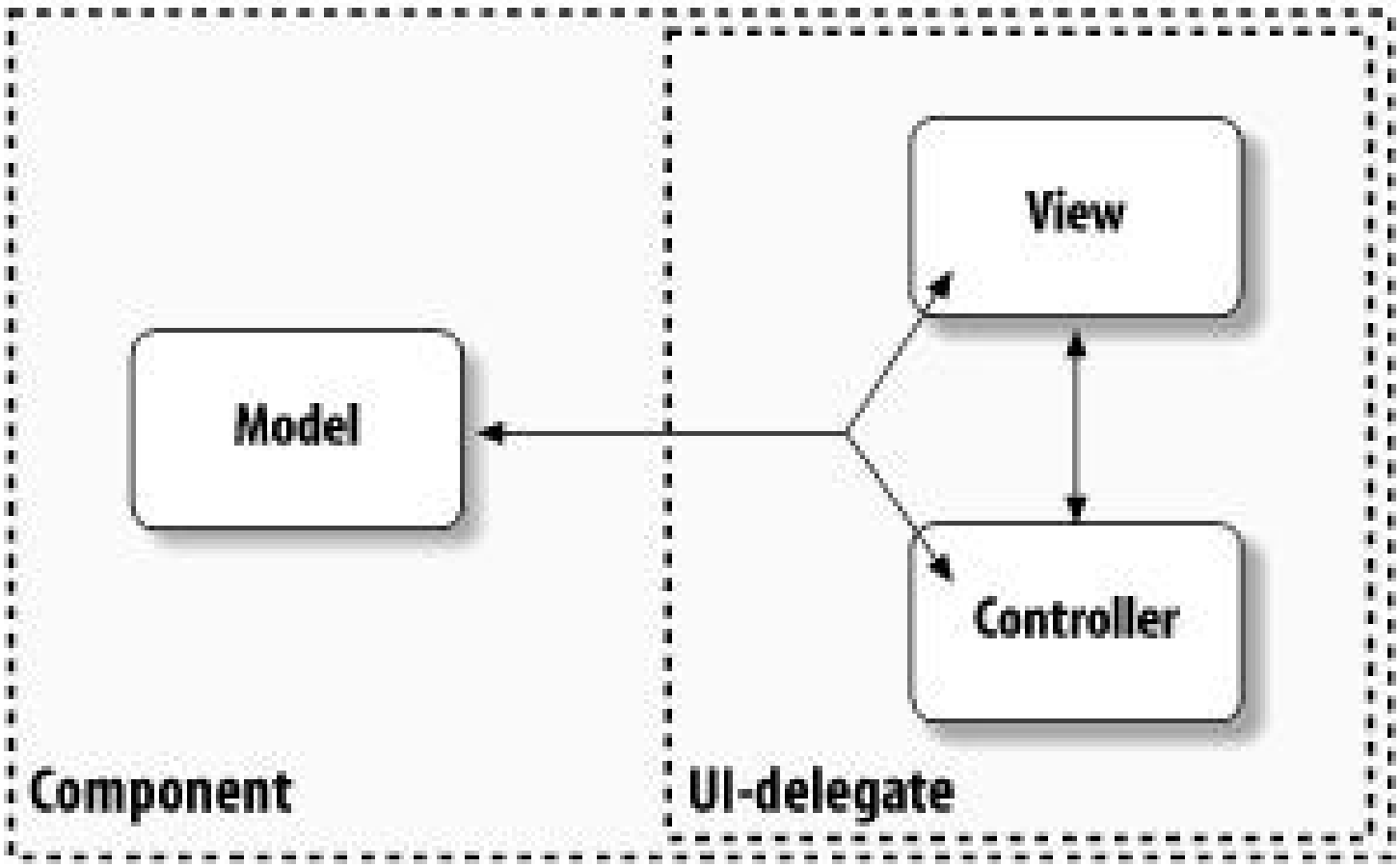
Sự tương tác trong MVC

- **Ví dụ**: GUI Component là Checkbox
- Khi người sử dụng nhấn chuột vào Checkbox
  - Thành phần **Controller** của Checkbox sẽ xử lý sự kiện này, yêu cầu thành phần **Model** thay đổi dữ liệu trạng thái. Sau khi thay đổi trạng thái, thành phần Model phát thông điệp đến thành phần View và Controller
  - Thành phần **Controller** nhận được thông điệp do Model gởi tới sẽ có những tương tác phản hồi với người sử dụng nếu cần thiết
  - Thành phần **View** của Checkbox nhận được thông điệp sẽ cập nhật lại thể hiện của Checkbox, phản ánh chính xác trạng thái Checkbox do Model lưu giữ.

MVC trong Swing

- Swing thật sự sử dụng một kiểu riêng đơn giản của thiết kế MVC gọi là Model - Delegate. Thiết kế này kết hợp View và Controller vào trong một yếu tố đơn là UI delegate, sẽ vẽ thành phần trên màn hình và xử lý những sự kiện GUI. Kết hợp khả năng đồ họa và xử lý sự kiện là một chút dễ dàng trong Java, khi hầu hết việc xử lý sự kiện đã được thực hiện cẩn thận trong AWT. Như bạn có thể mong đợi, việc giao tiếp giữa Model và UI delegate sau khi kết hợp được chỉ ra ở hình dưới

MVC trong Swing



MVC trong Swing

- Hãy nhìn lại: mỗi thành phần Swing chứa một Model và một UI delegate. Model chịu trách nhiệm cho việc nắm giữ thông tin về trạng thái của thành phần. UI delegate chịu trách nhiệm nắm giữ thông tin về việc vẽ thành phần trên màn hình như thế nào. Thêm vào đó, UI delegate (tập hợp với AWT) tương tác lại với những sự kiện riêng biệt để truyền lại thông qua thành phần.

MVC trong Swing

- Chú ý việc chia lại của Model và UI delegate trong thiết kế MVC là một sự thuận lợi cực kỳ. Một diện mạo duy nhất của kiến trúc MVC là khả năng hiển thị nhiều tầng (multi tie) cho một cùng Model.
- Ví dụ, nếu bạn muốn hiển thị cùng dữ liệu trong một biểu đồ và trong một bảng, bạn có thể dựa trên việc hiển thị của hai thành phần trên cùng một mô hình dữ liệu duy nhất. Theo cách này, nếu dữ liệu cần được thay đổi, bạn có thể chỉ sửa nó ở một nơi - View sẽ cập nhật lại bản thân chúng tương ứng. Theo cùng cách như thế, việc chia ra những thành phần đại diện từ Model sẽ cho người dùng thêm tiện lợi của việc chọn lựa một thành phần nào để xem mà không hề ảnh hưởng đến dữ liệu của nó. Bằng việc sử dụng hướng đi này, trong kết hợp với thiết kế lightweight, Swing có thể cung cấp mỗi thành phần với những cảm quan pluggable của nó

GUI-state Model và Application-data Model

- Tất cả component trong Swing đều sử dụng một trong hai mô hình là GUI-state hoặc Application-data.
  - **GUI-state**: lưu giữ trạng thái của component, ví dụ như button có được nhấn xuông hay không, phần tử nào được chọn trong một list,...
  - **Application-data**: chứa dữ liệu để hiển thị lên component. Dữ liệu này thường nằm ở mức ứng dụng như một table, danh sách phần tử của list.

GUI-state Model và Application-data Model

- Không phải tất cả component đều phân biệt rõ ràng mô hình được sử dụng. Ví dụ như JSlider và JprogressBar. Bạn có thể nhận thấy điều này trong bảng liệt kê mô hình mà các component của Swing sử dụng sau:

GUI-state Model và Application-data Model

Component	Model Interface	Model Type
JButton	ButtonModel	GUI
JToggleButton	ButtonModel	GUI/data
JCheckBox	ButtonModel	GUI/data
JRadioButton	ButtonModel	GUI/data
JMenu	ButtonModel	GUI
JMenuItem	ButtonModel	GUI
JCheckBoxMenuItem	ButtonModel	GUI/data
JRadioButtonMenuItem	ButtonModel	GUI/data
JComboBox	ComboBoxModel	data
JProgressBar	BoundedRangeModel	GUI/data
JScrollBar	BoundedRangeModel	GUI/data
JSlider	BoundedRangeModel	GUI/data

GUI-state Model và Application-data Model

Component	Model Interface	Model Type
JTabbedPane	SingleSelectionModel	GUI
JList	ListModel	data
JList	ListSelectionModel	GUI
JTable	TableModel	data
JTable	TableColumnModel	GUI
JTree	TreeModel	data
JTree	TreeSelectionModel	GUI
JEditorPane	Document	data
JTextPane	Document	data
JTextArea	Document	data
JTextField	Document	data
JPasswordField	Document	data

Thông báo sự thay đổi của Model

- Việc thông báo khi model của các component thay đổi là một điều cần thiết để cập nhật lại các trạng thái, giao diện của các đối tượng liên quan. Swing cung cấp hai cách thức để làm điều này:
  - **Lightweight**: thông báo sự thay đổi trạng thái của model. Sự kiện được sử dụng cho kiểu thông báo này là ChangeEvent. Cách này được dùng cho những component đơn giản và có thể quản lý nhiều thay đổi của model trong một sự kiện.

Thông báo sự thay đổi của Model

- Ví dụ:

```
checkbox.addChangeListener(new ChangeListener() {
    public void stateChanged(ChangeEvent e) {

        JCheckBox checkbox=(JCheckBox)e.getSource();
        DefaultButtonModel model = (DefaultButtonModel)checkbox.getModel();
        System.out.println("Selected: "+model.isSelected());
    }
});
```

Thông báo sự thay đổi của Model

- **Stateful**: Sử dụng cho các component phức tạp và cho mỗi sự thay đổi cụ thể.

Model	Listener	Event
ListModel	ListDataListener	ListDataEvent
ListSelectionModel	ListSelectionListener	ListSelectionEvent
ComboBoxModel	ListDataListener	ListDataEvent
TreeModel	TreeModelListener	TreeModelEvent
TreeSelectionModel	TreeSelectionListener	TreeSelectionEvent
TableModel	TableModelListener	TableModelEvent
TableColumnModel	TableColumnModel-Listener	TableColumnModel-Event
Document	DocumentListener	DocumentEvent
Document	UndoableEditListener	UndoableEditEvent

Thông báo sự thay đổi của Model

- **Ví dụ**

```
String items[] = {"One", "Two", "Three", "Four"};
JList<String> list = new JList<String>(items);
this.add(list);

list.addListSelectionListener
       (new ListSelectionListener() {
   public void valueChanged(ListSelectionEvent e) {

      JList list=(JList)e.getSource();
      ListSelectionModel model=(ListSelectionModel)list.getSelectionModel();

      if (!e.getValueIsAdjusting()) {
         System.out.println("Selection index: " + model.getLeadSelectionIndex());
      }
   }
});
```

DEMO

- Thiết kế giao diện như hình dưới

- Model: Dữ liệu hiển thị cho TextArea
- View: Giao diện hiển thị khi nhấn nút “Show”
- Controller: Xử lý sự kiện khi nút “Show” được nhấn

DEMO

```
2  public class ShowDemo extends JFrame implements ActionListener {
3      JTextArea jta;
4      JButton jbt;
5      public ShowDemo(){
6          setSize(300,150);
7          jta = new JTextArea(2,20);
8          jbt = new JButton("Show");
9          jbt.addActionListener(this); // Bat su kien
10         getContentPane().add(jta);
11         getContentPane().add(jbt);
12         setVisible(true);
13     }
14 // Phuong thuc xu ly su kien khi nut duoc nhan
15
16     public void actionPerformed(ActionEvent e) {
17         if (e.getSource()==jbt){
18             jta.setText("FPT Polytechnic");
19         }
20     }
21
22     public static void main(String[] args) {
23     // TODO Auto-generated method stub
24         new ShowDemo();
25     }
26 }
27
```

XIN CẢM ƠN!