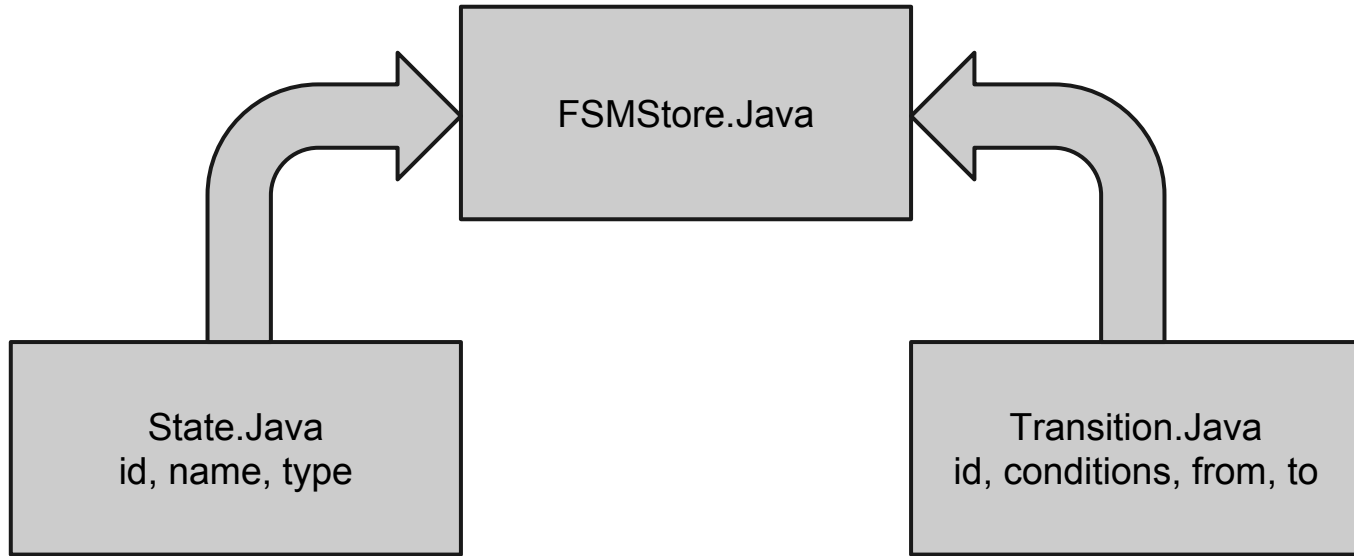


Finite State Machine

Frank, Rudy & Nate

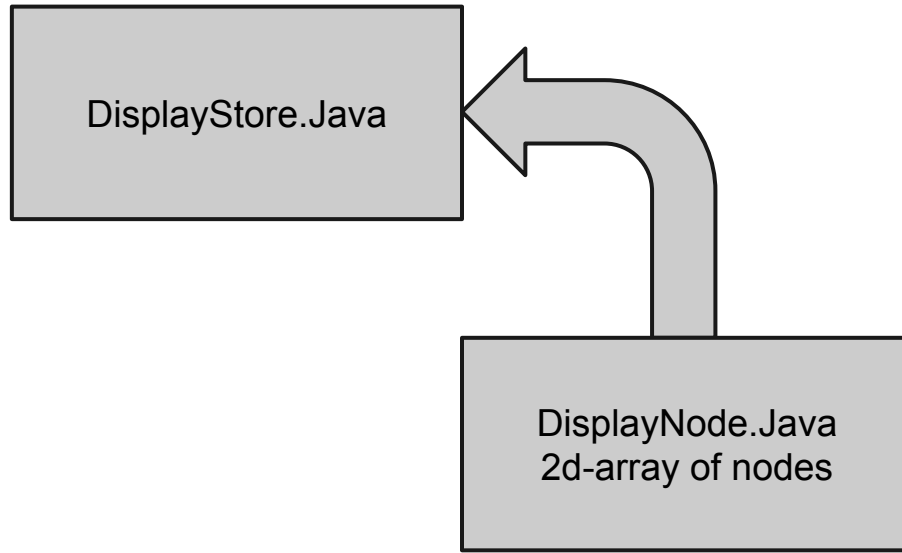
Storing the FSM



FSMStore API

FSMStore	State	Transition
<code>addState(name,type)</code> id <code>addTransition(name,fromID,toID)</code> id <code>removeState(id)</code> <code>removeTransition</code> <code>(id)</code> <code>containsState(id)</code> boolean <code>containsTransition(id)</code> boolean <code>getState(id)</code> State <code>getTransition(id)</code> Transition <code>numStates()</code> & <code>numTransitions()</code>	<code>getID()</code> id <code>getName()</code> name <code>rename(name)</code> <code>setType(type)</code>	<code>getID()</code> id <code>getFromID()</code> fromID <code>getToID()</code> toID <code>addCondition(name)</code> <code>removeCondition(name)</code> <code>numConditions()</code> <code>getLabel()</code>

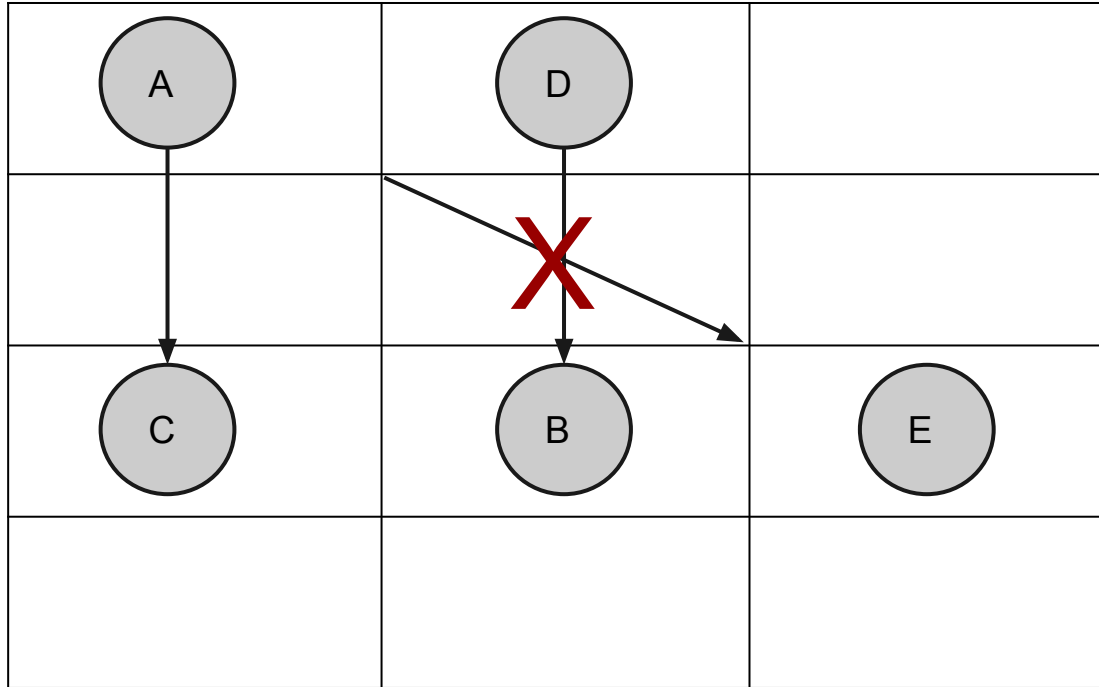
Storing the Grid



DisplayStore API

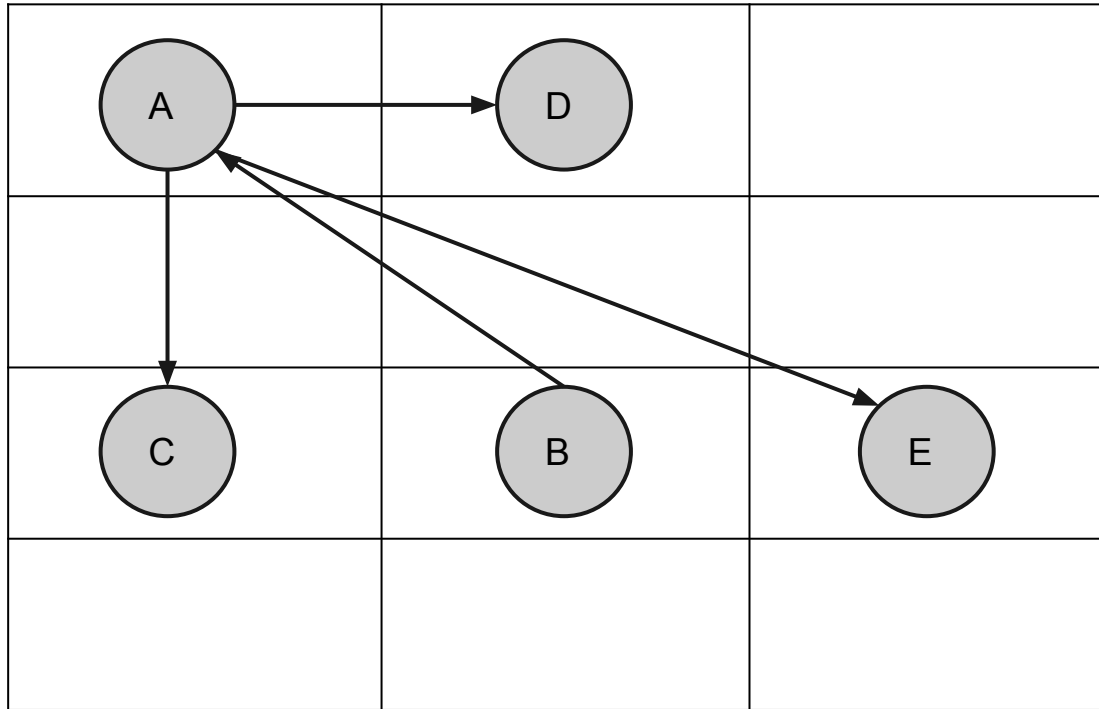
DisplayStore	DisplayNode
<code>getRow(id) → row</code> <code>getColumn(id) → column</code> <code>addState(x,y,id) → id</code> <code>removeState(x,y)</code> <code>moveState(x1,y1,x2,y2) → bool</code> <code>getState(x,y) → id</code>	<code>getID() → id</code> <code>containsState() → bool</code> <code>setNode(id) → id</code> <code>removeNode() → id</code>

Change in Design



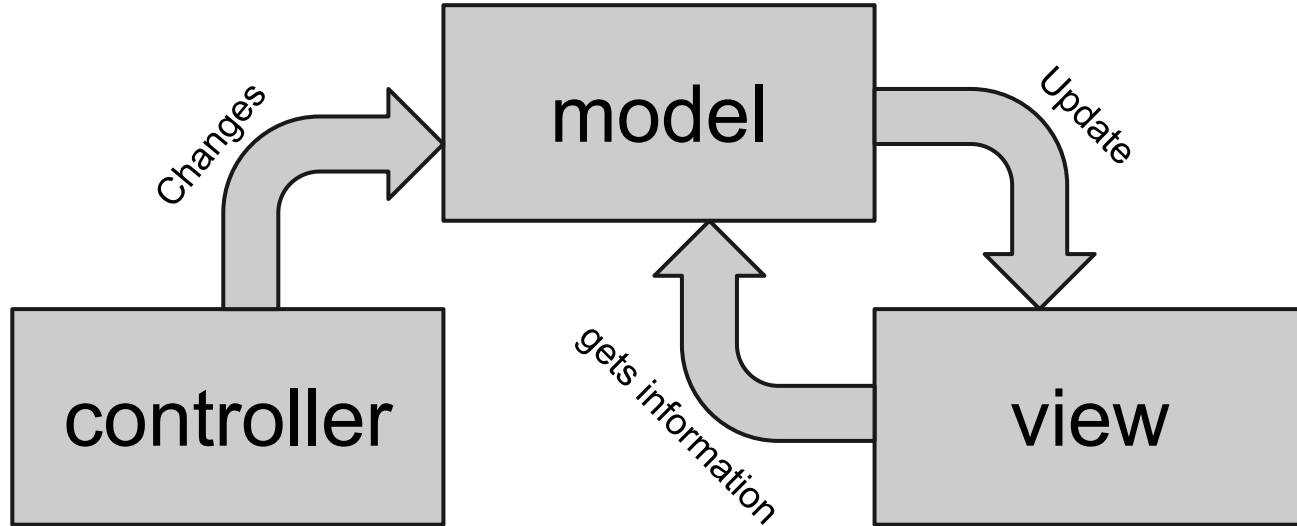
Original Implementation

Change in Design (cont)

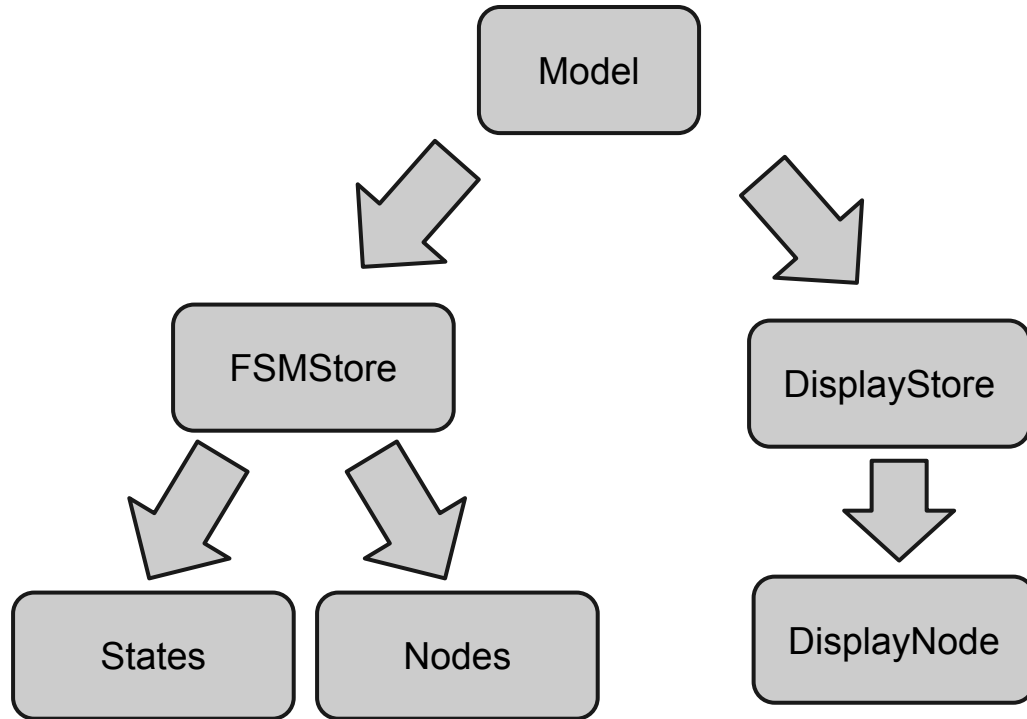


New Implementation

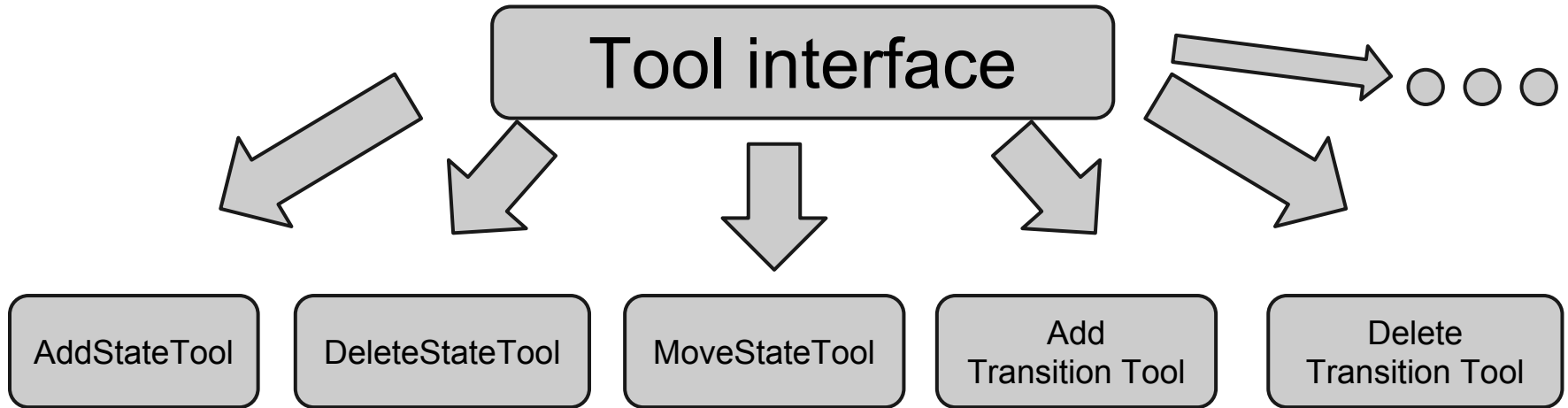
Model-View-Control Layout



Model



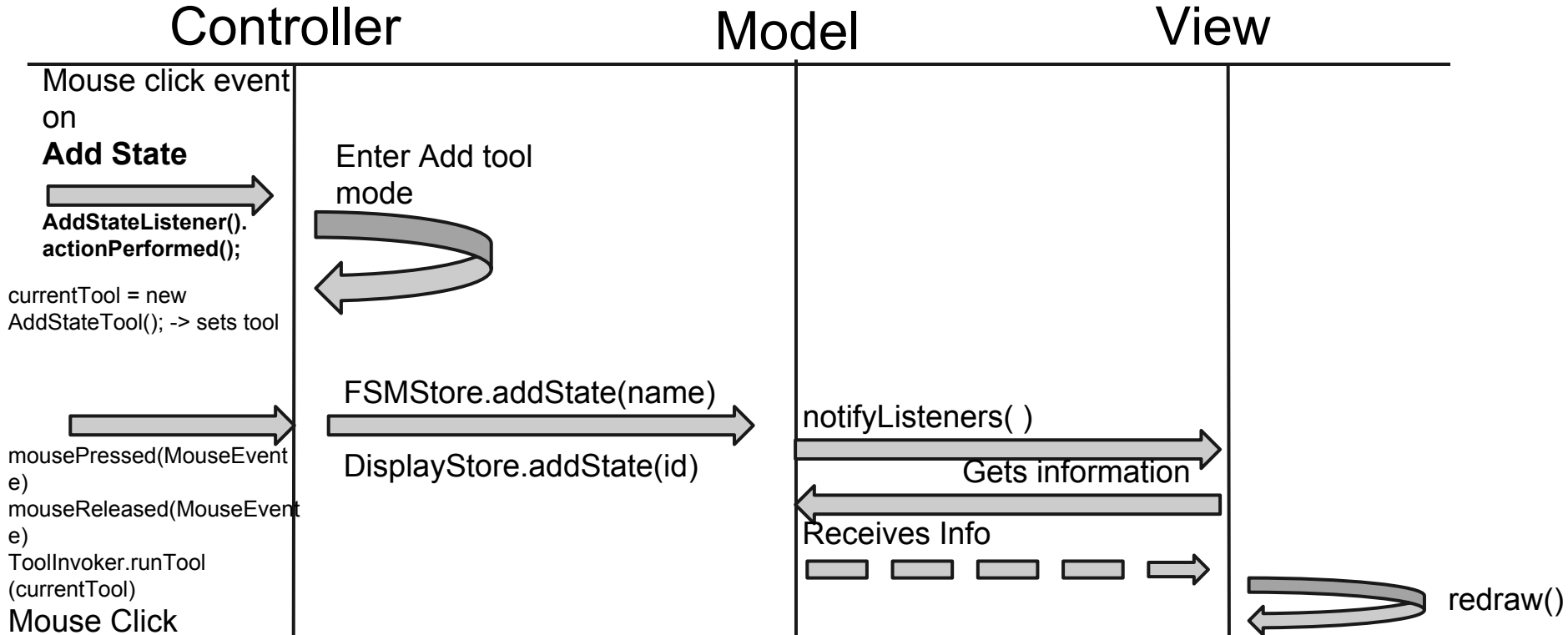
ToolBar - Command Pattern



Tool Implementation

- ToolInvoker
 - ToolInvoker.runTool(currentTool);
- Controller is a MouseListener
 - mousePressed(MouseEvent e)
- Tools are Private Classes within the controller
 - allows for access to mouse info inside of the controller without having to pass around alot of info

Click on Empty Location with create State Tool - Sequence Diagram



Save

Saving the State Machines to files would be implemented by going through the State Machine, taking its data and concatenating it to a string. This string will then be saved to a different file using the `fileWriter`.

Load

Load will take a string, which contains all of the elements pertinent to the class. We then call StringTokenizer on the string and on the element that we will be splicing the string around. We then throw in the “nextElement” into the load of the class below.

```
“;$false,-1;true,23;true,2;false,-1;$”
```

In this example the code first would set \$ as the delimiter, and would pass the “false...-1;” into our DisplayStore. From there it would throw the new string and “;” into StringTokenizer. So the “nextElement” would be “false,-1”, which would be thrown into DisplayNode.

User Interface Diagram



The image shows a user interface for a 'Finite State Machine Viewer'. The window has a title bar with three window control buttons (red, yellow, green) on the left. The title text 'Finite State Machine Viewer' is centered in the title bar. Below the title bar, there is a control area with the following elements:

- State:** A label followed by four buttons: 'Add', 'Delete', 'Move', and 'Toggle Type'.
- Transition:** A label followed by two buttons: 'Add' and 'Delete'.
- Name:** A label followed by a text input field.

Below the control area is a large grid consisting of 5 rows and 8 columns of empty square cells, intended for drawing the state machine diagram.

Demo

YAY!

Any Questions?