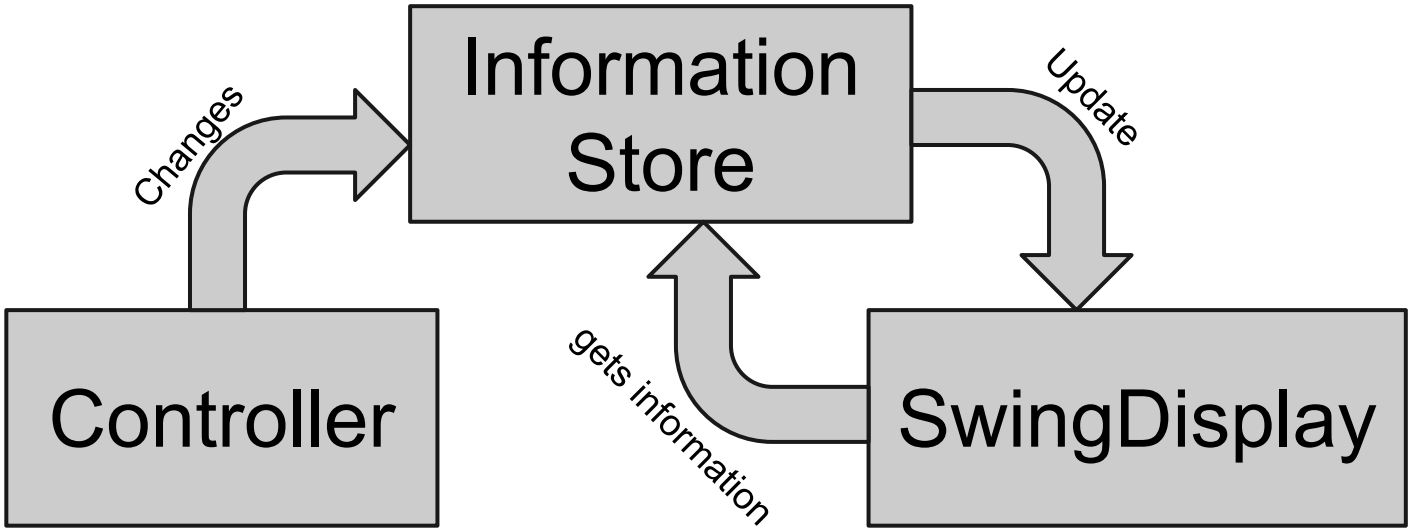
Finite State Machine

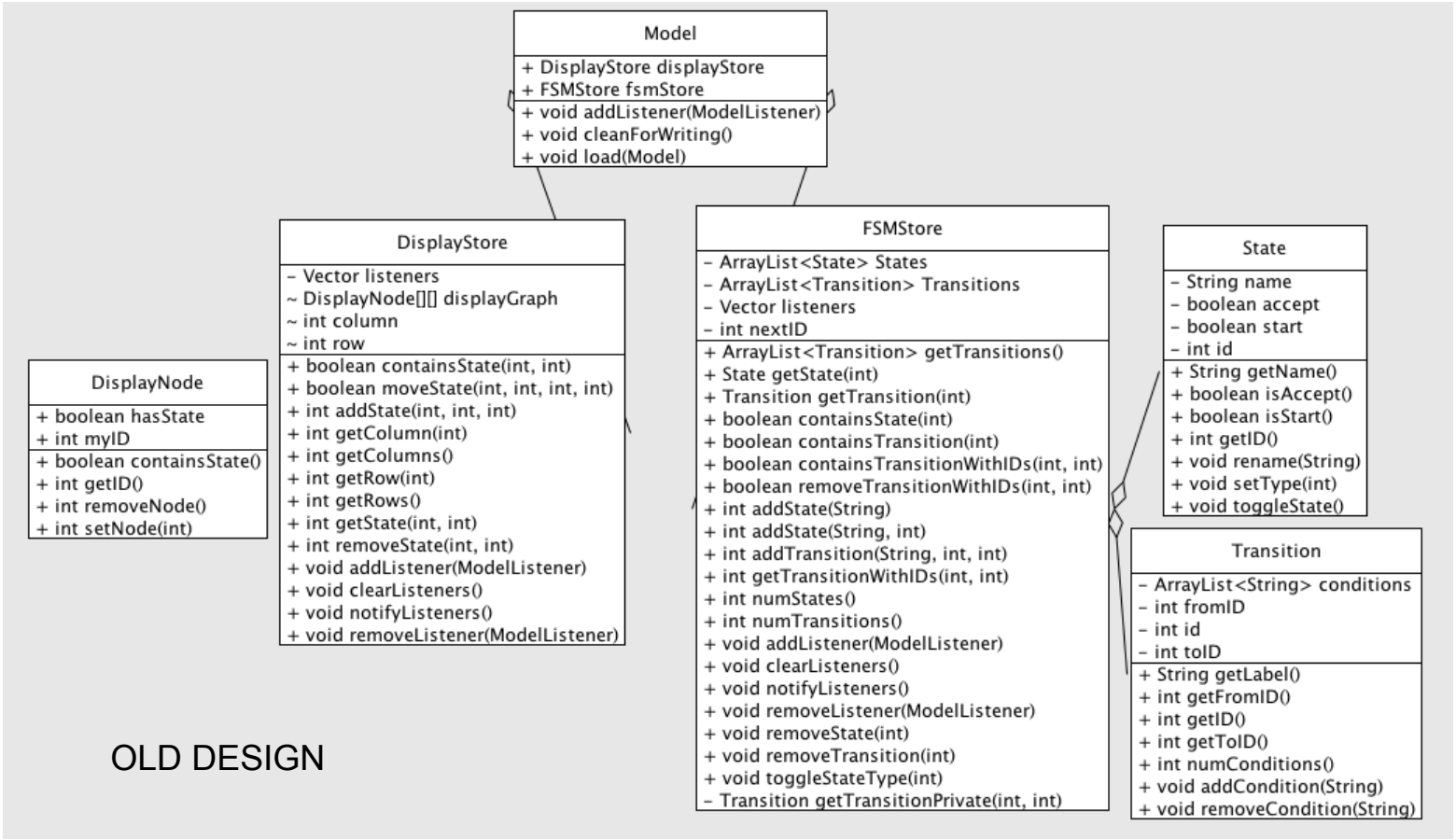
Frank, Rudy & Nate

Updates:

What have we done?

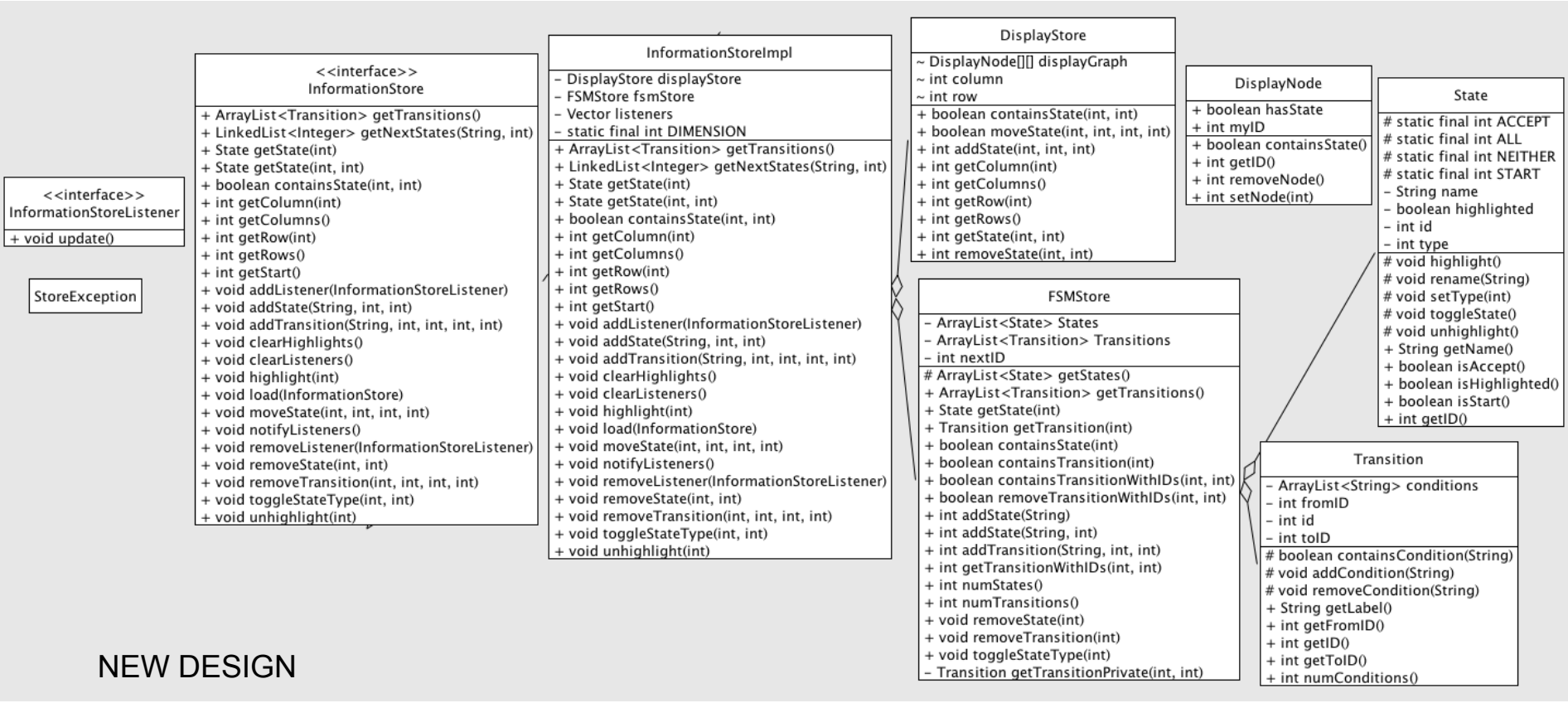
Model-View-Control Layout





Storage Changes: Facade Pattern

(pulling the functionality into the model itself)

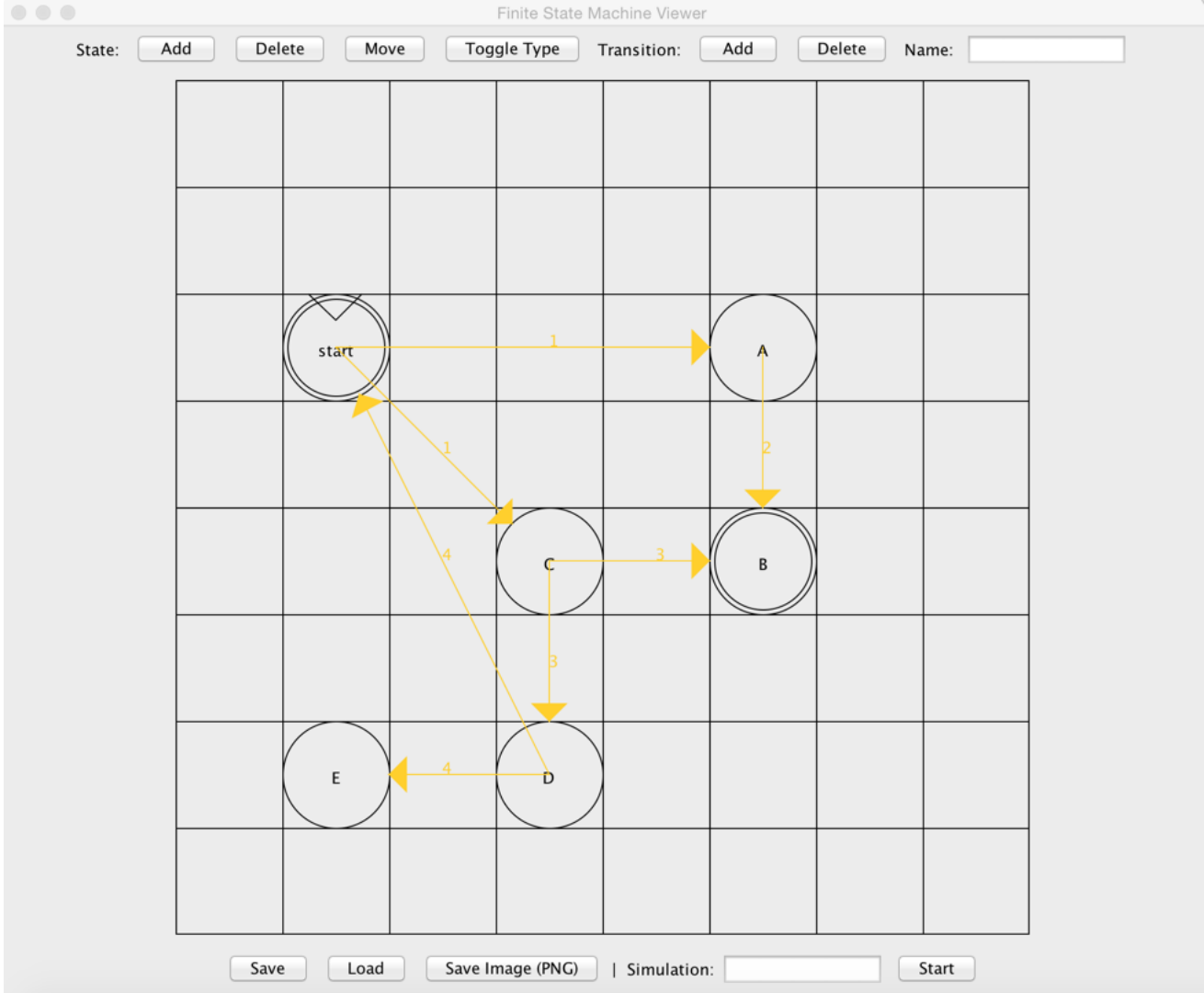


NEW DESIGN

Storage Changes: Facade Pattern

(pulling the functionality into the model itself)

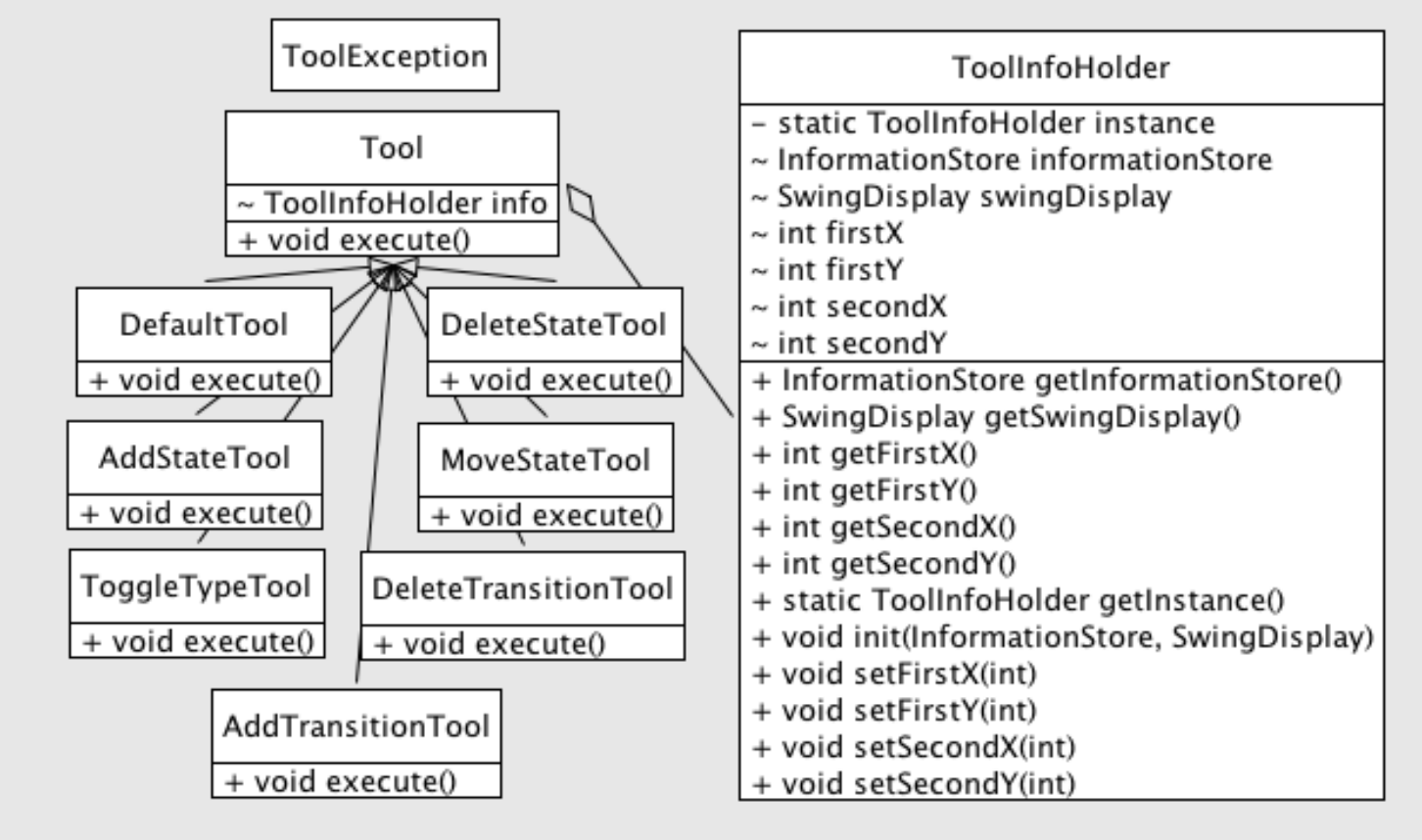
User Interface Diagram



Controller Changes:

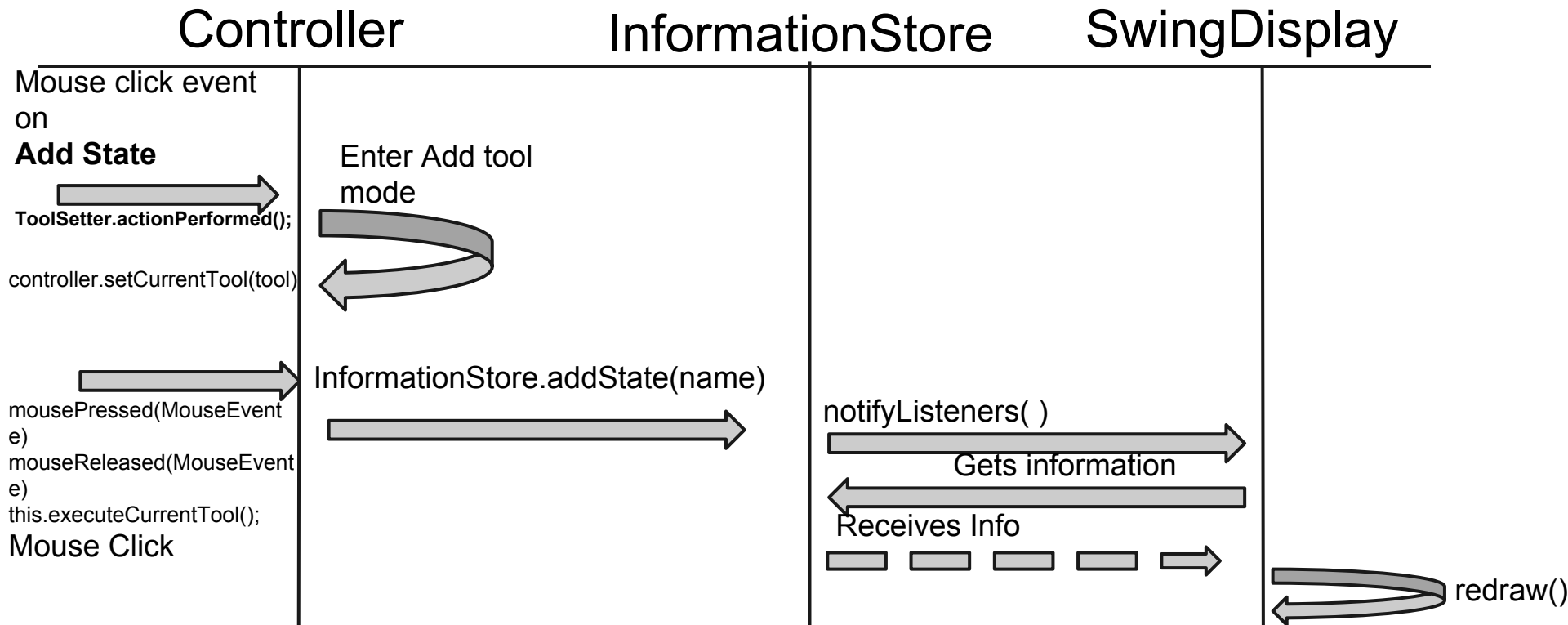
Tool Pattern

- Pulling out Tools to their own package
- ToolInfoHolder to pass along information to tools
  - now a singleton
    - cuts down on construction of tools vastly
- Tool: interface -> abstract class
  - tools only need the super constructor
    - ToolInfoHolder.getInstance() called in Tool class



Tool Changes: New Package (edu.union.fsm.tool)

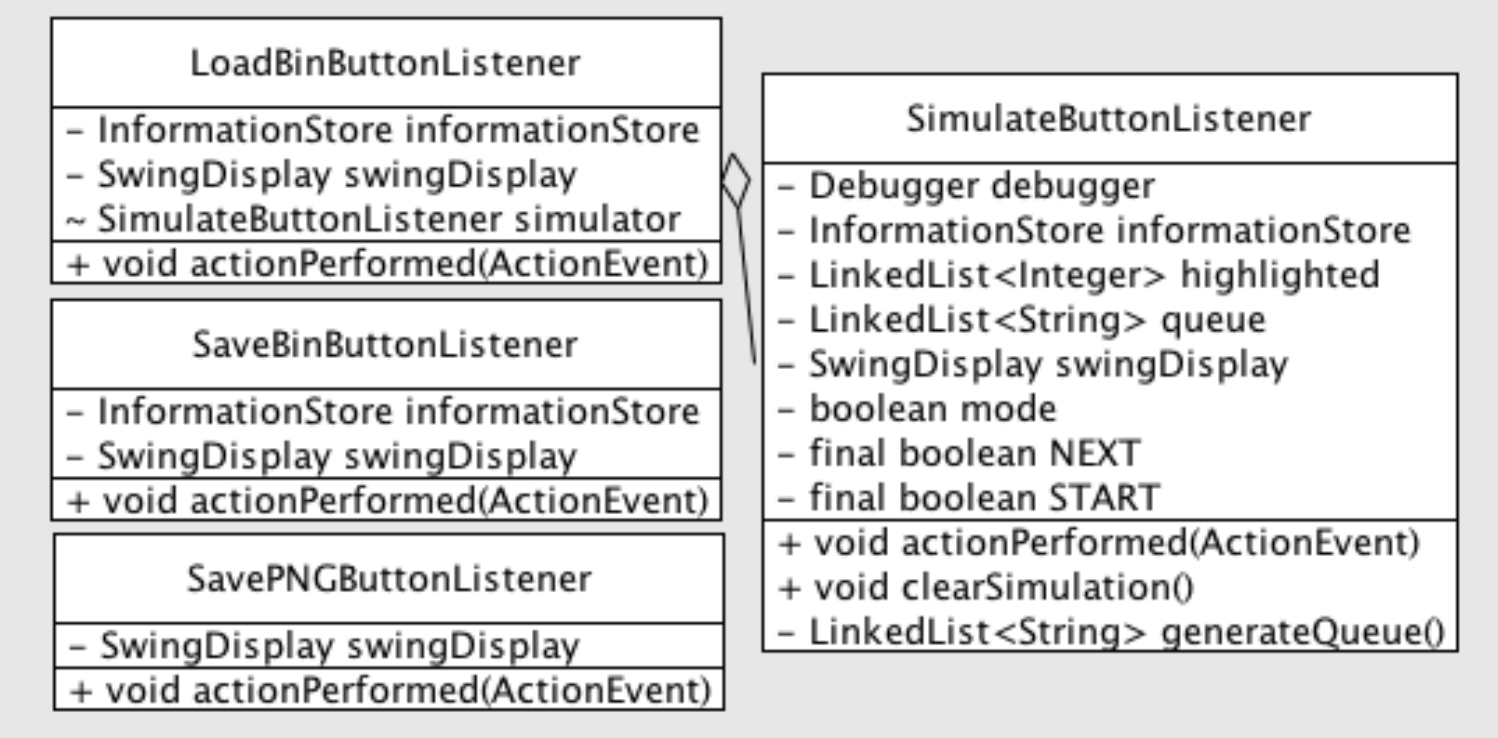
Click on Empty Location with add State Tool - Sequence Diagram



Executing with Exception Handling

```
/**
* Executes the current Tool.
*/
private void executeCurrentTool() {
    try {
        currentTool.execute();
    } catch (ToolException ex) {
        debugger.stackDebug(ex);
        String toPrint = ex.getMessage();
        swingDisplay.displayErrorMessage(toPrint);
    }
}
```

ActionListener Package



Saving and Loading

- Serializable: for saving and loading, forces users to save to .bin files.
- Image: saves png representation of the FSM
- Independent of FSM Program (reusable)
- allows for saving and loading anywhere using a file chooser.

Saving and Loading (cont)

SaveBin
- Object toSave - JFrame toPrompt
+ void saveFile

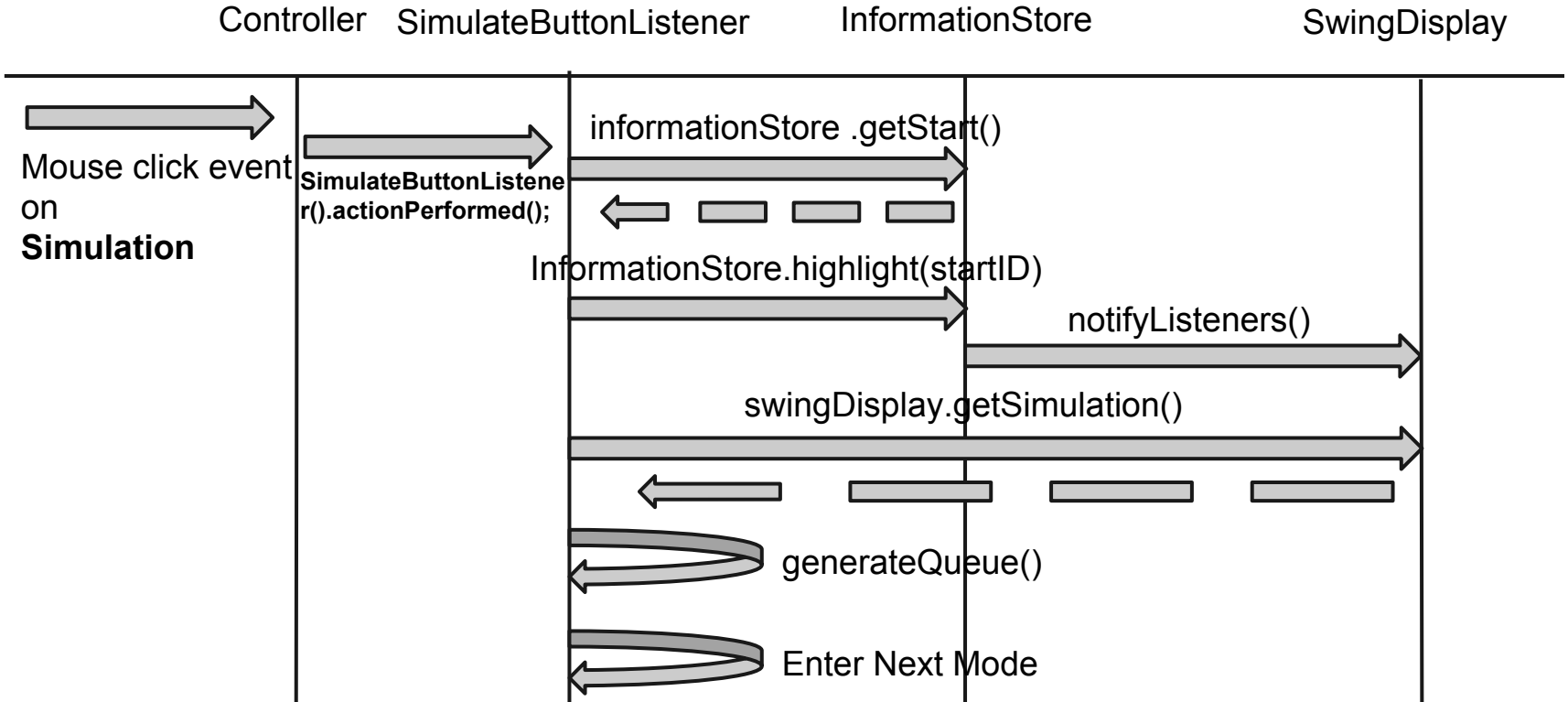
loadBin
- Object toLoad - JFrame toPrompt
+ Object loadFile

SavePNG
- JComponent toSave - JFrame toPrompt
+ void saveFile

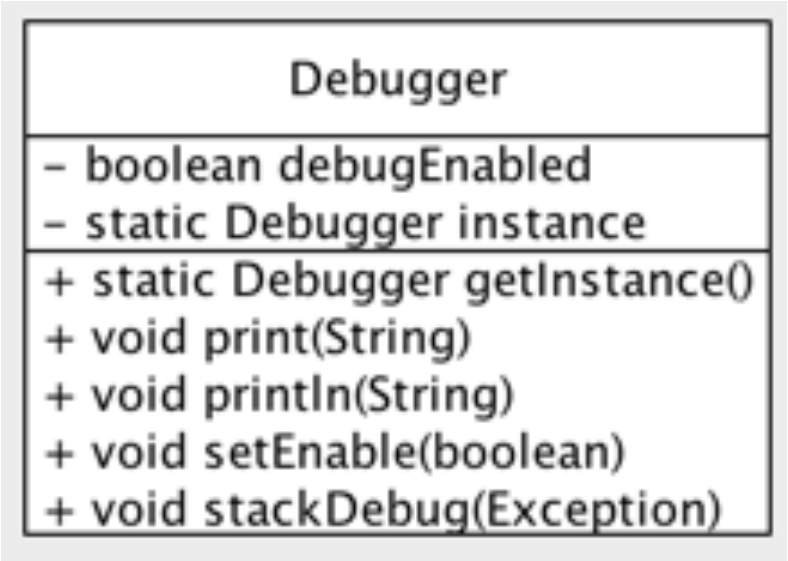
Simulation

- ActionListener
  - on Start, parses the input into a queue, begins by highlighting the start state, then goes into NEXT mode
  - in next mode, it grabs the next transition to from the queue and gets the next states and highlights them
  - prompts using dialog boxes if any invalid traversal occurs or the traversal ends.

Clicking Start Simulation - Sequence Diagram



Debugging using a singleton



```java
/**
* Executes the current Tool.
*/
private void executeCurrentTool() {
    try {
        currentTool.execute();
    } catch (ToolException ex) {
        debugger.stackDebug(ex);
        String toPrint = ex.getMessage();
        swingDisplay.displayErrorMessage(toPrint);
    }
}
```

Demo

YAY!

Any Questions?