CMSC 132:
Object-Oriented Programming II

GUI II

Department of Computer Science

University of Maryland, College Park

Overview

- Event-driven programming
- Model-View-Controller (MVC) Pattern
- GUI elements
- Java GUI classes
- JTables

Graphical User Interface (GUI)

- User interface
  - Interface between user and computer
  - Both input and output
  - Affects usability of computer
- Interface improving with better hardware
  - Switches & light bulbs
  - Punch cards & teletype (typewriter)
  - Keyboard & black/white monitor (text)
  - Mouse & color monitor (graphics)

Graphical User Interface (GUI)

- Goal
  - Present information to users clearly & concisely
  - Make interface easy to use for users
  - Make software easy to implement / maintain for programmers
- Design issues
  - Ease of use
  - Ease of understanding
  - Ability to convey information
  - Maintainability
  - Efficiency

Event-driven Programming

- Normal (control flow-based) programming
  - Approach
    - Start at main()
    - Continue until end of program or exit()
- Event-driven programming
  - Unable to predict time & occurrence of event
  - Approach
    - Start with main()
    - Build GUI
    - Await events (& perform associated computation)

Event-driven Programming in Java

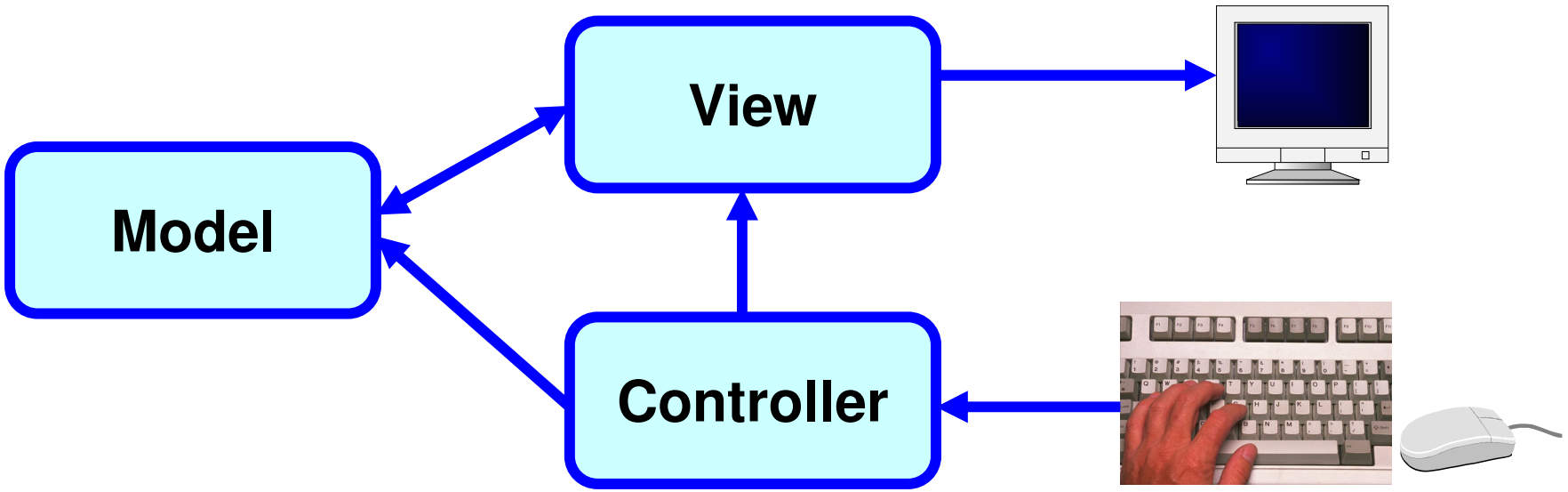
- During implementation
  - Implement event listeners for each event
  - Usually one event listener class per widget
- At run time
  - Register listener object with widget object
  - Java generates event object when events occur
  - Java then passes event object to event listener
- Example of observer design pattern

Event-driven Programming in Java

- **Example listeners & actions causing event**
  - **ActionEvent ⇒ clicking button in GUI**
  - **CaretEvent ⇒ selecting portion of text in GUI**
  - **FocusEvent ⇒ component gains / loses focus**
  - **KeyEvent ⇒ pressing key**
  - **ItemEvent ⇒ selecting item from pull-down menu**
  - **MouseEvent ⇒ dragging mouse over widget**
  - **TextEvent ⇒ changing text within a field**
  - **WindowEvent ⇒ closing a window**

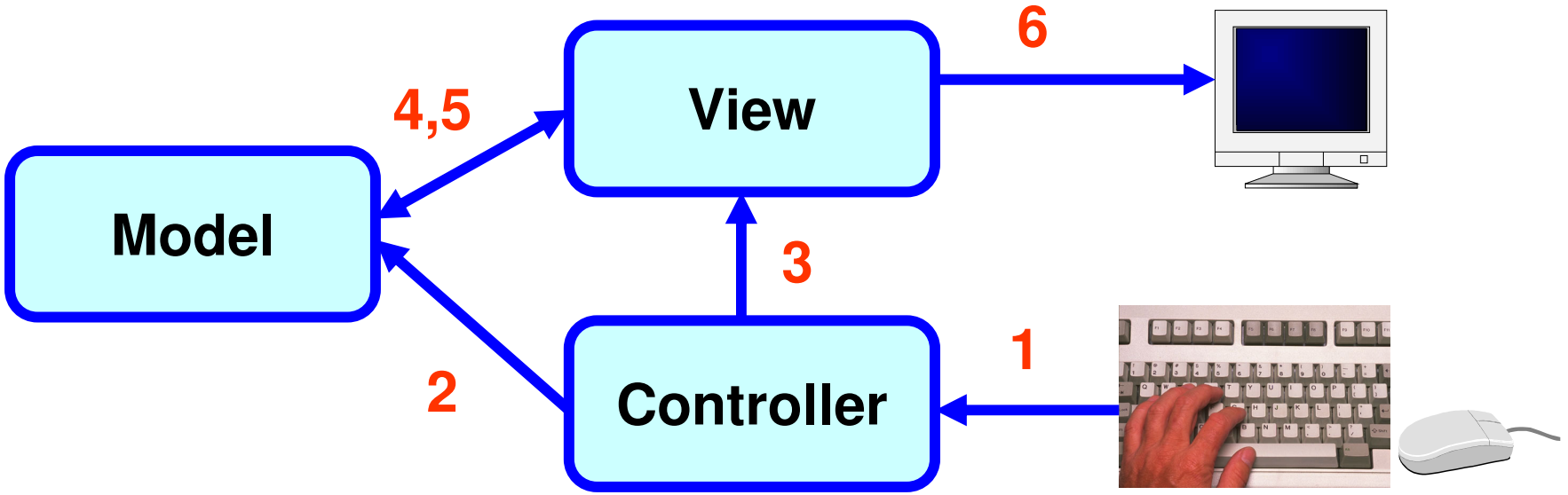
Model-View-Controller (MVC) Pattern

- Developed at Xerox PARC in 1978
- Separates GUI into 3 components
  - Model ⇒ application data
  - View ⇒ visual interface
  - Controller ⇒ user interaction



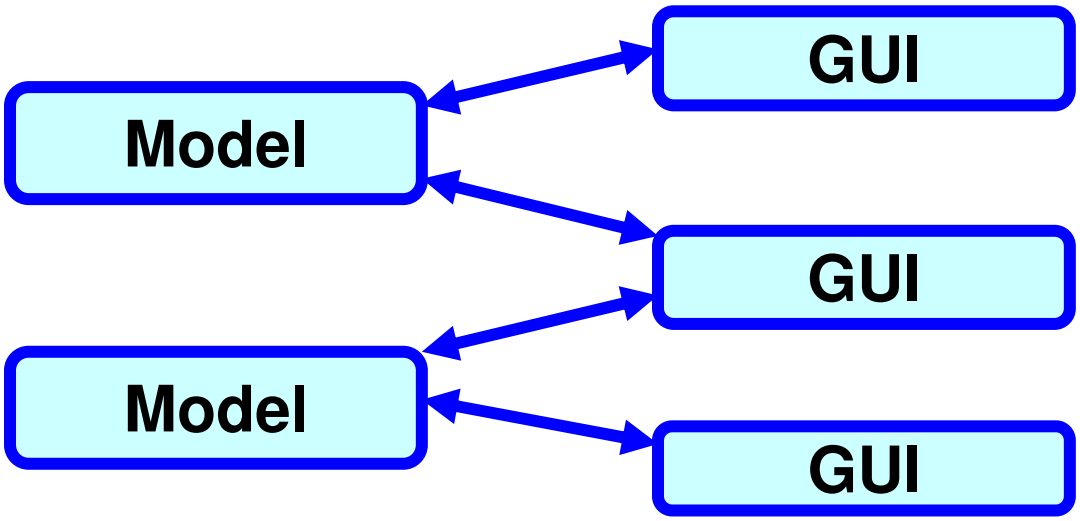
MVC Interaction Order

1 User performs action, controller is notified
2 Controller may request changes to model
3 Controller may tell view to update
4 Model may notify view if it has been modified
5 View may need to query model for current data
6 View updates display for user



MVC Pattern – Advantages

- Separates data from its appearance
  - More robust
  - Easier to maintain
- Provides control over interface
- Easy to support multiple displays for same data



MVC Pattern – Model

- Contains application & its data
- Provide methods to access & update data
- Interface defines allowed interactions
- Fixed interface enable both model & GUIs to be easily pulled out and replaced
- Examples
  - Text documents
  - Spreadsheets
  - Web browser
  - Video games

MVC Pattern – View

- Provides visual representation of model
- Multiple views can display model at same time
  - Example: data represented as table and graph
- When model is updated, all its views are informed & given chance to update themselves

MVC Pattern – Controller

- Users interact with the controller
- Interprets mouse movement, keystrokes, etc.
- Communicates those activities to the model
- Interaction with model indirectly causes view(s) to update

The Model View Controller Song

Lyrics and music by James Dempsey.

Model View, Model View, Model View Controller

MVC's the paradigm for factoring your code, into functional segments so your brain does not explode.

To achieve reusability you gotta keep those boundaries clean, Model on the one side, View on the other, the Controller's in between.

Model View - It's got three layers like Oreos do.

Model View creamy Controller

Model objects represent your applications raison d'ere. Custom classes that contain data logic and et cetra.

You create custom classes in your app's problem domain, then you can choose to reuse them with all the views, but the model objects stay the same

Principles of GUI Design

- **Model**
  - Should perform actual work
  - Should be independent of the GUI
    - But can provide access methods
- **Controller**
  - Lets user control what work the program is doing
  - Design of controller depends on model
- **View**
  - Lets user see what the program is doing
  - Should not display what controller thinks is happening (base display on model, not controller)

Principles of GUI Design

- **Model is separate**
  - Never mix model code with GUI code
  - View should represent model as it really is
    - Not some remembered status
- **In GUI's, user code for view and controller tend to mingle**
  - Especially in small programs
  - Lot of the view is in system code

Do you have a good model?

- Could you reuse the model if you wanted to port the application to:
  - a command-line textual interface
  - an interface for the blind
  - an iPod
  - a web application, run on the web server, accessed via a web browser

JTables

- Every table gets its data from an object implementing the TableModel interface
- JTable class allow us to display the contents of the TableModel object.
- DefaultTableModel is a class that implements the TableModel
- There are different ways we can use tables to display data.
- Examples
  - See TableExamples.zip