CMSC 132: Object-Oriented Programming II

Graphic User Interface (GUI)

Department of Computer Science
University of Maryland, College Park
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
Graphical User Interface (GUI)

Design issues

- Ease of use
- Ease of understanding
- Ability to convey information
- Maintainability
- Efficiency
GUI Topics

- Event-driven programming
- Model-View-Controller (MVC) Pattern
- GUI elements
- Java GUI classes
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 Handling in Action

Events

Registered Event Handlers

Can handle an event of type $e_1$

Dispatcher

Execution Environment

State = $S_0$
GUIs are Event-Driven Software

User Events

User events invoke event handlers

Event Handlers

E_1
changeFontSizeActionPerformed (java.awt.event.ActionEvent evt)

E_2

E_3
newDocActionPerformed (java.awt.event.ActionEvent evt)

E_4

E_5
fileSaveActionPerformed (java.awt.event.ActionEvent evt)

User events invoke event handlers
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 & GUls 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
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

- **Controller & view tend to mingle in GUIs**
  - 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
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’être. Custom classes that contain data logic and et cetera.

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...