

# CMSC 131: Chapter 16 (Supplement)

## GUI Programming Concepts

### Graphical User Interfaces

**Graphical User Interfaces (GUIs):** Programming for user interfaces has become much more important in recent years.

**Text-based:** Input is read from a command line (or input file) through a set of **commands** to the program.

**GUI-based:** The user interacts with a graphical user interface by:

- clicking buttons
- selecting from menus
- dragging and dropping
- sliding scrollbars

**Examples:** Pine a text-based email system vs. **Microsoft Outlook**

<u>Operation</u>	<u>Pine</u>	<u>Outlook</u>
Move to Trash	"s Trash"	Drag to Trash folder icon
Save to file	"e fileName.txt"	Select <b>File</b> → <b>SaveAs</b> from <b>menu</b> and enter file name.
Send Reply	"r"	Click the "Reply" button.

### Issues

The move to GUI-based interfaces raises a number of issues:

**Separating function from interface:** Good programming design requires that we **separate** these elements. It should be possible to:

- **Change the look and controls** of the interface, without altering the underlying functionality.
- **Change the underlying functionality** should not necessarily require changes to the user-interface.

**Event-driven programming:** The standard programming approaches used in text-based interfaces **do not apply** to GUI programming.

```
do {  
    prompt user for input;  
    read and parse the input;  
    perform the required operation;  
} while ( ! finished );
```

## MVC: Separating Interface and Function

To separate interface and function, programmers developed a new **design pattern** for their programs. Almost all GUI programs involve the interaction of three basic entities:

**Model:** The underlying **data** and primitive operations.

**View:** The **visual presentation** of data to the user.

**Controller:** The **commands/graphical controls** (widgets) that are presented to the user, and the effect they have on the model.

When programming a system, each of these should be implemented as **separate objects** (or collections of objects), which **interact** through method calls.

This is called the **Model-View-Controller** design pattern, or **MVC**.

## MVC: Separating Interface and Function

**Example:** Outlook and Pine email systems.

**Model:** Underlying email **messages, headers, folders**. (Same for both)

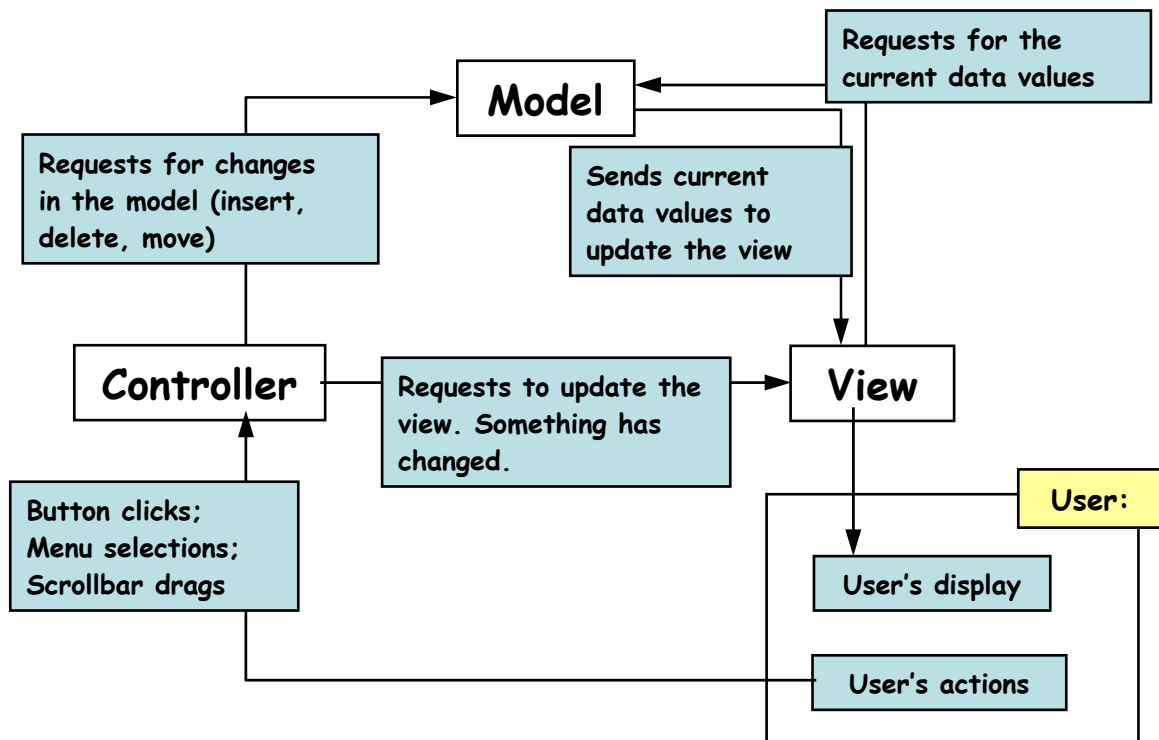
**View: Outlook:** Graphical views. (Image omitted)

**Pine:** Text view.

**Controller: Outlook:** GUI-based. (Image omitted)

**Pine:** Keyboard input.

## Programming in the MVC Model



## Event Driven Programming and Callbacks

When programming with GUIs the classical text-based input loop **does not apply**.

**Callback:** A method that is called when the user performs some action. Possible actions include:

- **mouse button** press or release
- **keyboard key** press
- **mouse has moved** (for dragging)

**How it works:** Your program informs the system:

- Which **input events** you are interested in listening for.
- For each such input event, which **method** of yours is to be called.
  - **mouse click event:** call `myMouseClicked()`
  - **keyboard key pressed:** call `myKeyboard()`
- The **parameters** to these callback methods provide information (e.g. which mouse button was pressed, which keyboard key hit).

# Event Driven Programming and Callbacks

## How it works: the "Event loop"

- Your program **sets up the callbacks** and then **goes to sleep**, and waits for the user to do something.
- When an **event** occurs, the **callback** method you specified is called automatically by the system.
- **Note:** You have **no idea** which of your callbacks will be called next, so you must be ready for any possibility.
- Each callback **updates the model** (the underlying data) appropriately and **returns immediately**, waiting for the next event.

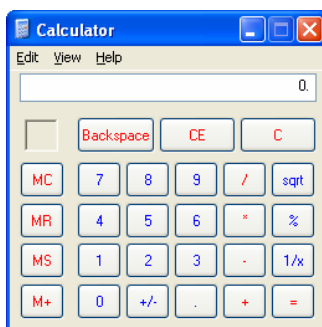
## Example: Simple Integer Calculator

### Simple Calculator:

**State (Model):** Need to save internal information such as the **current value** entered, the **last operation** entered, the contents of the **memory**.

### Events and Callbacks:

- **Digit key:** `digitKey( int value )`
- **Operation key:** `opKey( char op )`  
Operations: '+', '-', '=', ...
- **Clear key:** `clear( )`
- **Backspace key:** `backspace( )`
- ...



## Example: Simple Integer Calculator

**Simple Calculator:** Sample event sequence.

<b>Event:</b>	<b>Effect on State:</b>	<b>Display (view):</b>
<b>Start</b>	Initialize value = 0	<b>0</b>
<b>Digit: 8</b>	value = 8	<b>8</b>
<b>Digit: 3</b>	value = 83 (value = value*10 + 3)	<b>83</b>
<b>Digit: 2</b>	value = 832 (value = value*10 + 2)	<b>832</b>
<b>Backspace</b>	value = 83 (value = value / 10)	<b>83</b>
<b>Op Key: '+'</b>	operand = value (83) save operation code (+) value = 0	<b>0</b>
<b>Digit: 4</b>	value = 4 (value = value*10 + 4)	<b>4</b>
<b>Op Key: '='</b>	value = 87 (value = operand + value)	<b>87</b>