CMSC 131: Chapter 12 (Supplement)
Arrays II and MVC/GUI

Arrays of Objects

Array of Objects: The base type of an array can be a class object.

Example: Array of Strings.

```
String[ ] greatCities = new String[5];
greatCities[2] = new String( "Paris" );
greatCities[0] = "Tokyo";
greatCities[4] = "Beltsville";
greatCities[1] = greatCities[2];
int k = 4;
int x = greatCities[k].length( );
char c = greatCities[2].charAt( 2 );
```

Arrays of Objects

Initializers: Array initializers can be used with class base types as well. The elements of the initializer can be **expressions** (not just constants).

Example: Array of **Strings**

```
String[ ] moreCities = { "New York", "Boston", "Kathmandu" };
```

Example: Initializing using a non-constant **expression**.

```
String[ ] cityState = {
          moreCities[0] + ", NY",   moreCities[1] + ", MA" };
```

Example: Array of **Dates** (constructor is given month, day, year)

```
Date[ ] birthDays = {
          new Date( 2, 12, 1809 ), new Date( 2, 11, 1731 ) };
```

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**

Operation	Pine	Outlook
Move to Trash	"**s Trash**"	**Drag** to Trash folder icon
Save to file	"**e fileName.txt**"	Select **File→SaveAs** from **menu** 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