CS 106A, Lecture 24
Graphical User Interfaces (GUIs) part 2

reading:

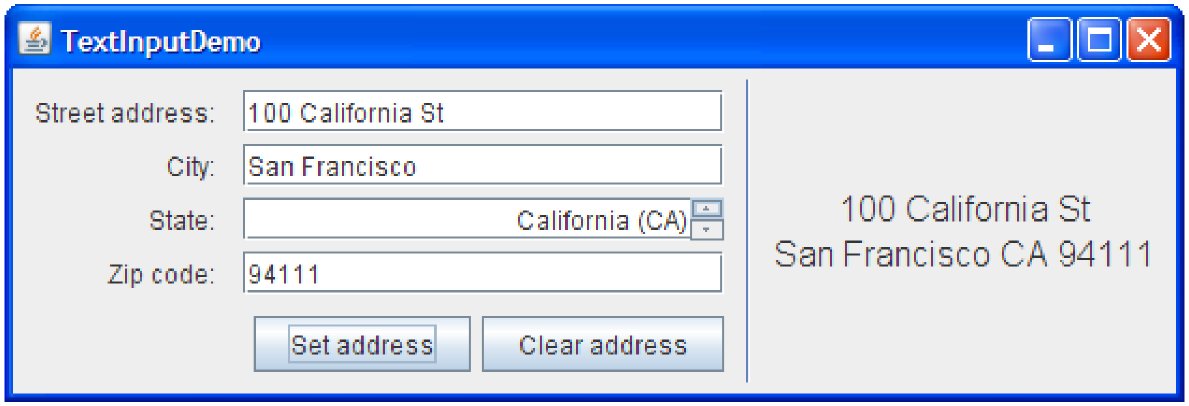
Art & Science of Java, Chapter 10

Lecture at a glance

- Today we will cover more about GUIs.
  - We will see new **components** such as checxboxes and radio buttons.

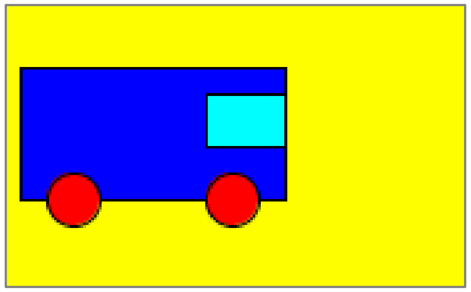
- We will also learn how to mix 2D graphics/animation with GUIs.
  - An animated graphical program is one that contains a **canvas**.

- Lastly we will discuss the idea of "**model**" and "**view**" separation.
  - This design pattern guides us in decomposing an OO GUI problem.


TextInputDemo

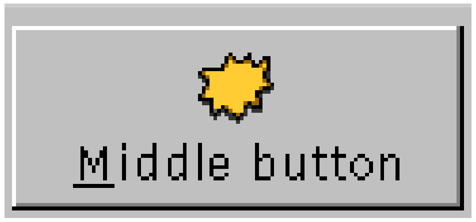
Street address:	100 California St
City:	San Francisco
State:	California (CA)
Zip code:	94111

100 California St
San Francisco CA 94111



Icon

a picture that can appear inside a component



- `ImageIcon` ***`name`*** `= new ImageIcon("`***`filename`***`");`

- in `JButton`, `JLabel`, `JRadioButton`, `JCheckBox`, etc...
  - **constructor** that takes an `ImageIcon`
  - or, ***`jb`*** **`.setIcon(`** ***`icon`*** **`)`** `;`

- example:

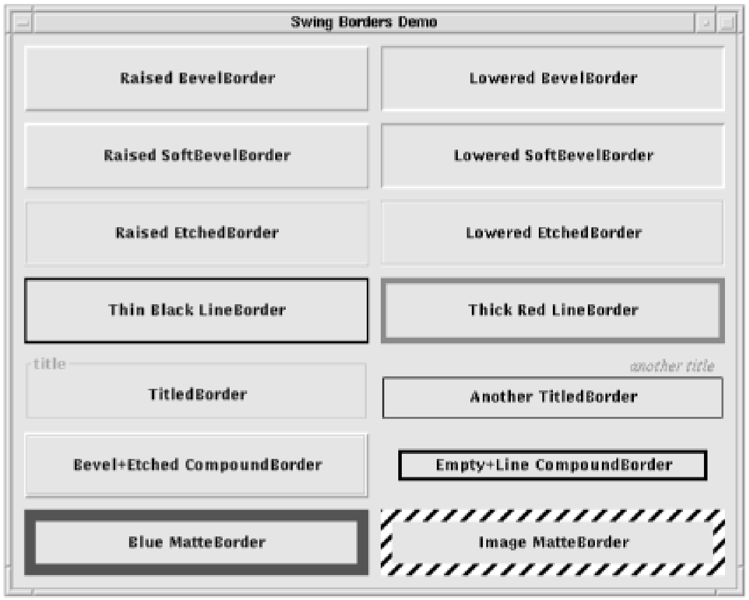
```
ImageIcon icon = new ImageIcon("res/smiley.gif");
myButton.setIcon(icon);
```

Borders

```
component.setBorder(
    BorderFactory.createBorderType(params));
```

– Where ***BorderType*** is:

- `BevelBorder`, `EtchedBorder`, `LineBorder`, `MatteBorder`, `TitledBorder`, …



– Example:

```
// set a 4-px-thick red border around the button
myButton.setBorder(
    BorderFactory.createLineBorder(Color.RED, 4));
```

– (see BorderFactory docs)

JTextField

a single-line input control for typing text values

Method	Description

`new JTextField("`***`text`***`")`
`new JTextField(`***`columns`***`)`
	Create new text field of given size

`jtf``.addActionListener(this);`	causes action events to occur when the user presses Enter on the field

`jtf``.getActionCommand()`
`jtf``.setActionCommand("`***`cmd`***`");`
	set/return a string to identify the action events that will occur in this field

`jtf``.getText()`
`jtf``.setText("`***`text`***`");`
	set/return text in the field

Events on text fields

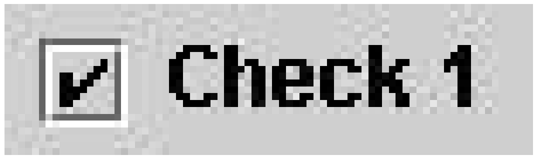
- By default, no event will occur if you press Enter on a `JTextField`.
  - To change this, add your program as an action listener to the field.
  - You must also set its **action command** to a unique string.

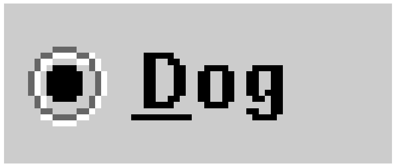
```
myTextField.addActionListener(this);
myTextField.setActionCommand("bingo");
...

public void actionPerformed(ActionEvent event) {
    if (event.getActionCommand().equals("bingo")) {
        ...
    }
}
```

George Washington

JCheckBox, JRadioButton

a toggleable yes/no value (checkbox)
or a way choose between options (radio)





Method	Description

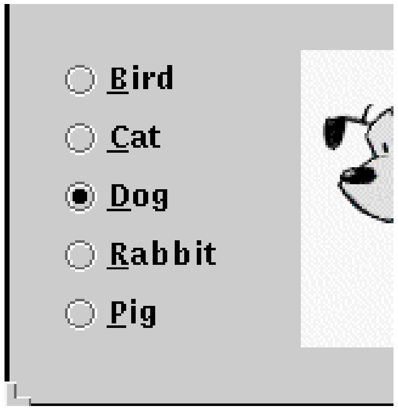
`new JCheckBox("`***`text`***`")`
`new JCheckBox("`***`text`***`", `***`checked`***`)`
`new JRadioButton("`***`text`***`")`
	Create new check box or radio button

`jcb``.isSelected()`
`jcb``.setSelected(`***`boolean`***`);`
	set/return whether box is checked

`jcb``.getText()`
`jcb``.setText("`***`text`***`");`
	set/return text in the check box

ButtonGroup

a logical collection to ensure that exactly one radio button from a group is checked at a time



- `public` **`ButtonGroup`**`()`
- `public void` **`add`**`(JRadioButton button)`

  – The `ButtonGroup` is not a graphical component, just a logical group; the `RadioButtons` themselves also need to be added to an onscreen container to be seen.

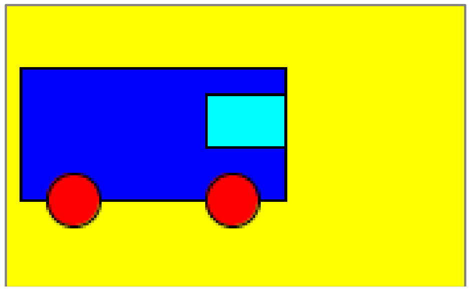
2D Graphics Canvas with GUIs

GCanvas

- `GCanvas`: A component for 2D graphics, shapes, colors, etc.
  - Graphical objects like `GLine`, `GOval`, etc., all work on a `GCanvas`.
  - A `GraphicsProgram` has a `GCanvas` in its CENTER.

- To mix 2D drawing/animation with an overall program GUI:
  - Write your overall program as one that extends `Program` .
  - Write a separate class that extends `GCanvas` for the drawing part.
  - Add your canvas to your Program's window (probably in the CENTER).

```
public class ClassName extends GCanvas {
    ...
}
```



Canvas class design

- Your canvas class is not the Program, so typically you want methods in it that the Program class can call to do the following:
  - set any relevant properties of the canvas
  - tell the canvas to redraw / "update" itself

```
public class CarCanvas extends GCanvas {
    public CarCanvas() {
        // initialize/add any graphical shapes ...
    }

    public void setCarColor(Color c) { ... }
    public void setCarSpeed(int speed) { ... }

    public void updateCar() {
        // called in an animation loop; update position ...
    }
}
```

Animated canvas

- Your Program class can create your canvas object and update it in an animation loop.

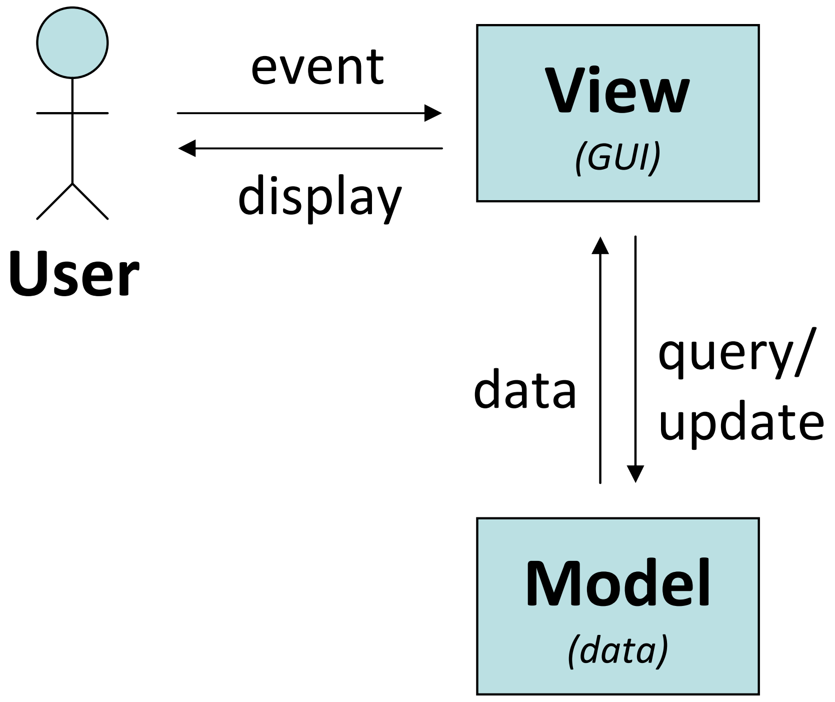
```
public class CarGUI extends Program {
    private CarCanvas canvas;

    public void init() {
        canvas = new CarCanvas();
    }

    public void run() {
        while (true) {
             canvas.updateCar();
             pause(50);
        }
    }
}
```

Model/View Separation

Model and View

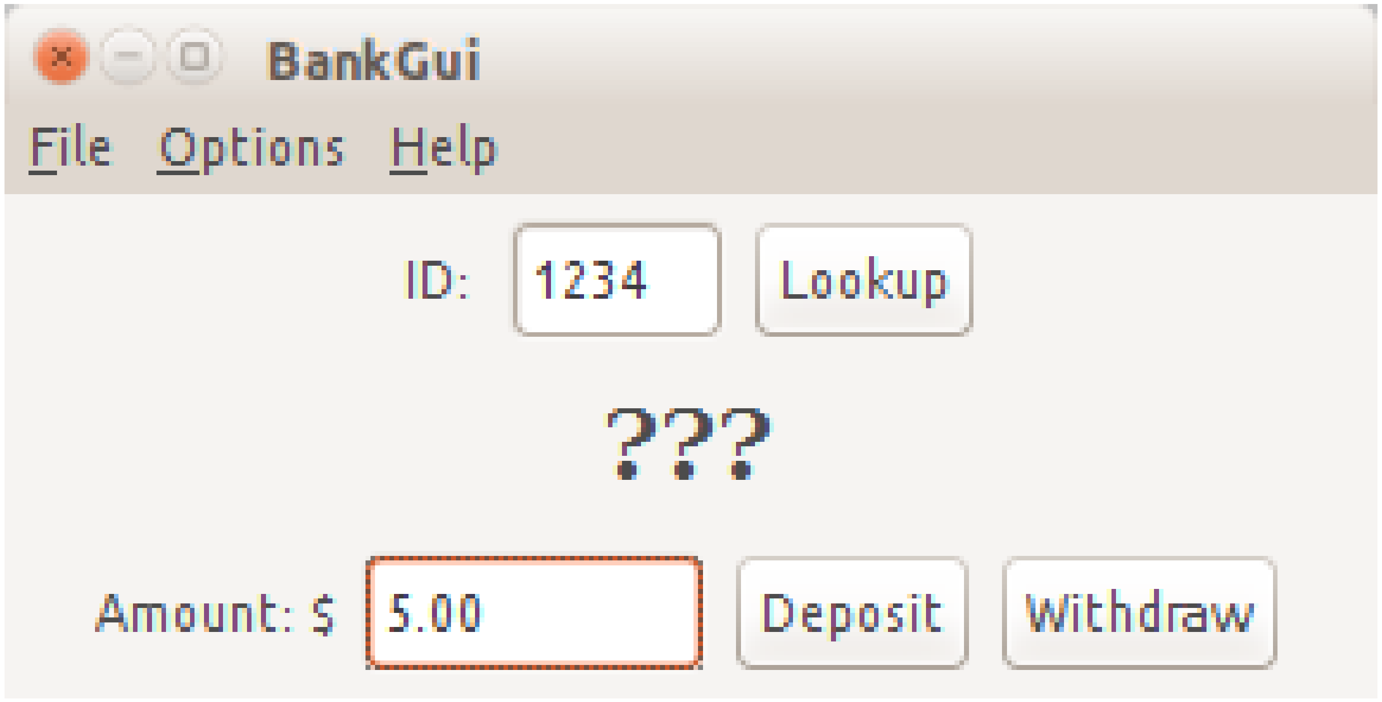
- **model**: Classes/objects that represent core data of an app.
  - *responsibilities:* store, load, save, search through data
- **view**: Classes/objects used to display the model to the user.
  - *responsibilities:* read user input; display data to user; handle events

- Typical code design pattern:
  - **View** listens for user events.
  - User clicks/types information.
  - **View** asks **model** for relevant data, or tells model to modify/update data.
  - **Model** gives relevant data to the **view**.
  - **View** displays this data to the user.



Bank GUI

- Example: Bank Account Management GUI program
  - *model classes:* `BankAccount`; `BankDatabase`; etc.
  - *view classes:* `BankGui`; `BankCanvas`; etc.


BankGui
File Options Help
ID: 1234
Lookup
???
Amount: $ 5.00
Deposit
Withdraw