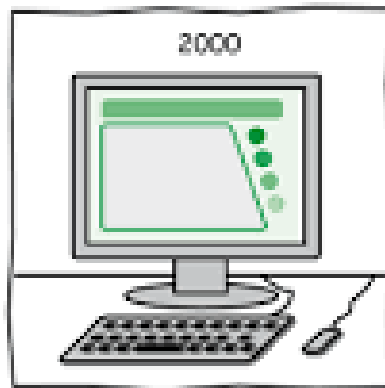
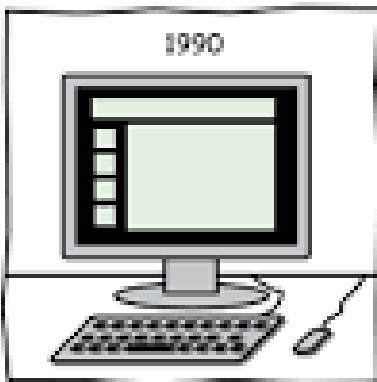


# CS 142

## Lecture 28: GUIs

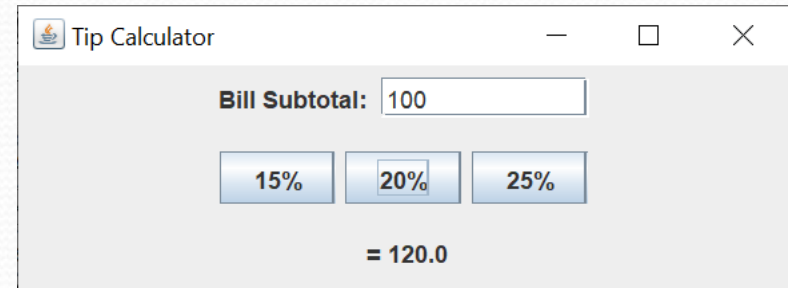
### THE EVOLUTION OF INTERFACE DESIGN



Thanks to Marty Stepp and Stuart Reges for parts of these slides

# Exercise: Tip Calculator

- Write a program that allows the user to calculate a 15%, 20% or 25% tip by pressing a button.
  - Make the window 400 by 150
  - Program should exit when the window is closed
  - Tip calculation only happens when a button is pressed



# Making our GUI an Object

- Usually we do not create a GUI in main, instead we make it in an object. Then in main we can write:

```
public static void main(String[] args) {  
    new MyGui();  
}
```

# Accessing ActionEvent

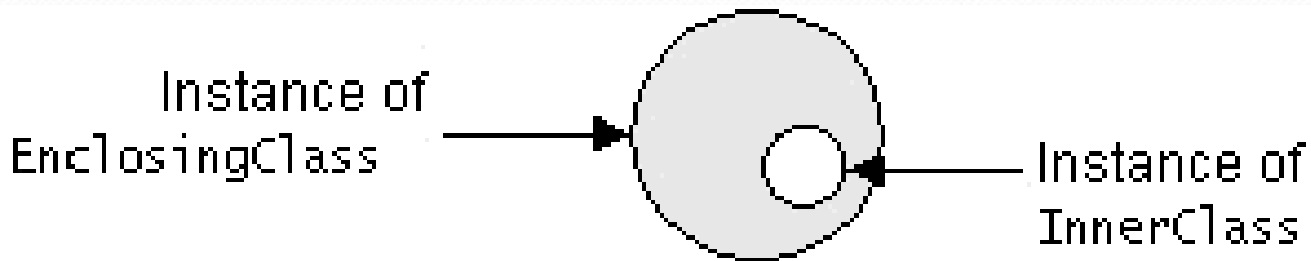
- `public Object getSource()`  
Returns the object on which the Event initially occurred.
- `public String getActionCommand()`  
Returns the command string associated with this action.
  - this will be the button text if a button triggered the event

# How can we access our TextField?

- We can't get it from the `ActionEvent` as no event has occurred on it
- We can't change the method header of `actionPerformed`
- It only exists in the constructor of our `TipCalculator` class
- Solution: make it a field?

# Nested classes

- **nested class:** A class defined inside of another class.
- Usefulness:
  - Nested classes are hidden from other classes (encapsulated).
  - Nested objects can access/modify the fields of their outer object.
- Event listeners are often defined as nested classes inside a GUI.



# Nested class syntax

```
// enclosing outer class
public class name {
    ...

    // nested inner class
    private class name {
        ...
    }
}
```

- Only the outer class can see the nested class or make objects of it.
- Each nested object is associated with the outer object that created it, so it can access/modify that outer object's methods/fields.
  - If necessary, can refer to outer object as **OuterClassName**.this

# Static inner classes

```
// enclosing outer class
public class name {
    ...

    // non-nested static inner class
    public static class name {
        ...
    }
}
```

- Static inner classes are *not* associated with a particular outer object.
- They cannot see the fields of the enclosing class.
- *Usefulness:* Clients can refer to and instantiate static inner classes:  
**Outer.Inner name = new Outer.Inner (params) ;**

# GUI event example

```
public class MyGUI {
    private JFrame frame;
    private JButton stutter;
    private JTextField textfield;

    public MyGUI() {
        ...
        stutter.addActionListener(new StutterListener());
    }
    ...

    // When button is clicked, doubles the field's text.
    private class StutterListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            String text = textfield.getText();
            textfield.setText(text + text);
        }
    }
}
```

# Making your own DrawingPanel

- To draw shapes on a window we need to add an image
  - `BufferedImage name = new BufferedImage(width, height, colorType)`
  - We will be using color type `BufferedImage.TYPE_INT_RGB`
- An image can't go directly in the frame. It must go inside an `ImageIcon` and the `ImageIcon` must go in a `Label` and then *finally* we can put the `Label` on the `Frame`

```
Jframe frame = new JFrame();  
BufferedImage img =  
    new BufferedImage(400, 500, BufferedImage.TYPE_INT_RGB);  
JLabel surround = new JLabel(new ImageIcon(img));  
frame.add(surround);
```

# Drawing

- Just like with the `DrawingPanel`, we need a pen to draw shapes.
  - You can get a pen from the image the same way you did from the `DrawingPanel`:

```
Graphics name = yourBufferedImage.getGraphics ()
```

Now you can call all the same methods you called on the `Graphics` object you got from `DrawingPanel`.

# Animation

- We need to make some code to run every  $n$  amount of seconds
  - Use the `Timer` class
  - Create a class that extends `TimerTask` and give it a `run()` method
    - Think of the timer as an alarm clock that only has a snooze button

```
Timer timer = new Timer();           // from java.util
TimerTask task = new MyTimer();
timer.schedule(task, 200, 100); // starts timer to run task
                                // after 200ms and then every 100ms
```

```
public class MyTimer extends TimerTask {
    private int count;
    public void run() {
        count++;
        pen.drawRect(10 * count, 20 * count, 40, 30);
        frame.repaint(); // redraws the picture on the screen
    }                    // so we can see the update
}
```

# JFrame Properties

- JFrames have the following unique properties that you can get or set in your graphical programs:

<b>name</b>	<b>type</b>	<b>description</b>	<b>methods</b>
default close operation	<code>int</code>	what should happen when frame is closed	<code>getDefaultCloseOperation,</code> <code>setDefaultCloseOperation</code>
icon image	<code>Image</code>	icon in the window's title bar	<code>getIconImage,</code> <code>setIconImage</code>
layout	<code>LayoutManager</code>	how the frame should position its components	<code>getLayout,</code> <code>setLayout</code>
resizable	<code>boolean</code>	whether the window can be resized	<code>isResizable,</code> <code>setResizable</code>
title	<code>String</code>	window's title bar text	<code>getTitle,</code> <code>setTitle</code>

# Component Properties

- All components also have the following properties:

<b>name</b>	<b>type</b>	<b>description</b>	<b>methods</b>
background	Color	background color	getBackground, setBackground
enabled	boolean	whether the component can be interacted with	isEnabled, setEnabled
font	Font	font used to display any text on the component	getFont, setFont
foreground	Color	foreground color	getForeground, setForeground
location	Point	(x, y) position of component on screen	getLocation, setLocation
size	Dimension	width, height of component	getSize, setSize
preferred size	Dimension	width, height that the component wants to be	getPreferredSize, setPreferredSize
visible	boolean	whether the component can be seen on screen	isVisible, setVisible

# JButton and JLabel

*The most common component—  
a button is a clickable onscreen  
region that the user interacts with  
to perform a single command*



*A text label is simply a string of text  
displayed on screen in a graphical  
program. Labels often give infor-  
mation or describe other components*

Look in:

- `public JButton(String text)`  
`public JLabel(String text)`  
Creates a new button / label with the given string as its text.
- `public String getText()`  
Returns the text showing on the button / label.
- `public void setText(String text)`  
Sets button / label's text to be the given string.

# JTextField and JTextArea

*A text field is like a label, except that the text in it can be edited and modified by the user. Text fields are commonly used for user input, where the user types information in the field and the program reads it*

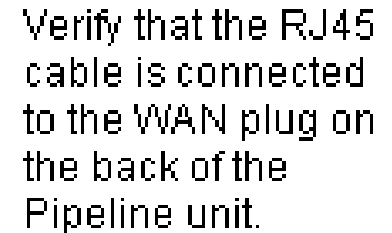


George Washington

Thomas Jefferson

*A text area is a multi-line text field*

- `public JTextField(int columns)`
- `public JTextArea(int lines, int columns)`  
Creates a new text field that is the given number of columns (letters) wide.
- `public String getText()`  
Returns the text currently in the field.
- `public void setText(String text)`  
Sets field's text to be the given string.



Verify that the RJ45  
cable is connected  
to the WAN plug on  
the back of the  
Pipeline unit.