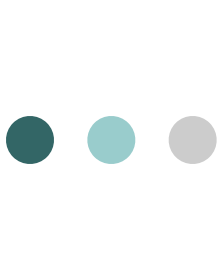


# CS 260

- Event-driven Programming and Graphical User Interfaces (GUIs) with Swing/AWT

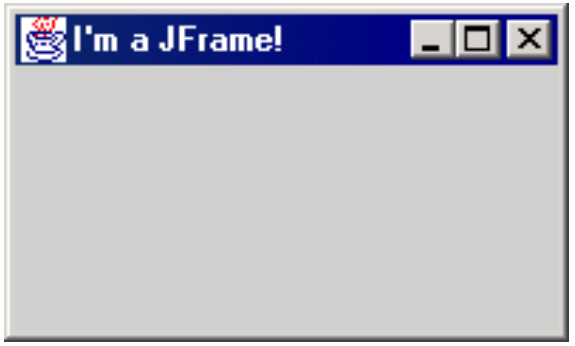
- Reference:
  materials by M. Ernst, S. Reges, D. Notkin, R. Mercer, Wikipedia

# JFrame

- a graphical window to hold other components

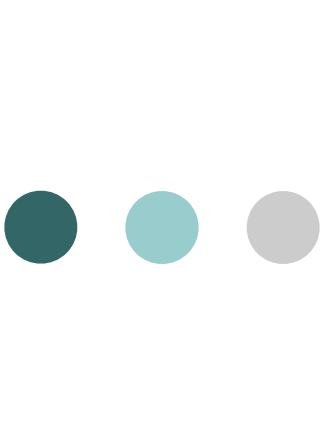
- public JFrame()
  public JFrame(String title)
  Creates a frame with an optional title.
  - Call setVisible(true) to make a frame appear on the screen after creating it.

- public void add(Component comp)
  - Places the given component or container inside the frame.



# More JFrame

- public void setDefaultCloseOperation(int op)
  - Makes the frame perform the given action when it closes.
  - Common value passed: JFrame.EXIT_ON_CLOSE
  - If not set, the program will never exit even if the frame is closed.
- public void setSize(int width, int height)
  Gives the frame a fixed size in pixels.
- public void pack()
  Resizes the frame to fit the components inside it snugly.



# Event Listener

Java GUI Programming

# JButton

- a clickable region for causing actions to occur
- public JButton(String text)
  - Creates a new button with the given string as its text.
- public String getText()
  - Returns the text showing on the button.
- public void setText(String text)
  - Sets button's text to be the given string.

# GUI example

```
import java.awt.*;
import javax.swing.*;

public class GuiExample1 {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(new Dimension(300, 100));
        frame.setTitle("A frame");

        JButton button = new JButton();
        button.setText("Click me!");
        button.setBackground(Color.RED);
        frame.add(button);

        frame.setVisible(true);
    }
}
```

# Sizing and positioning

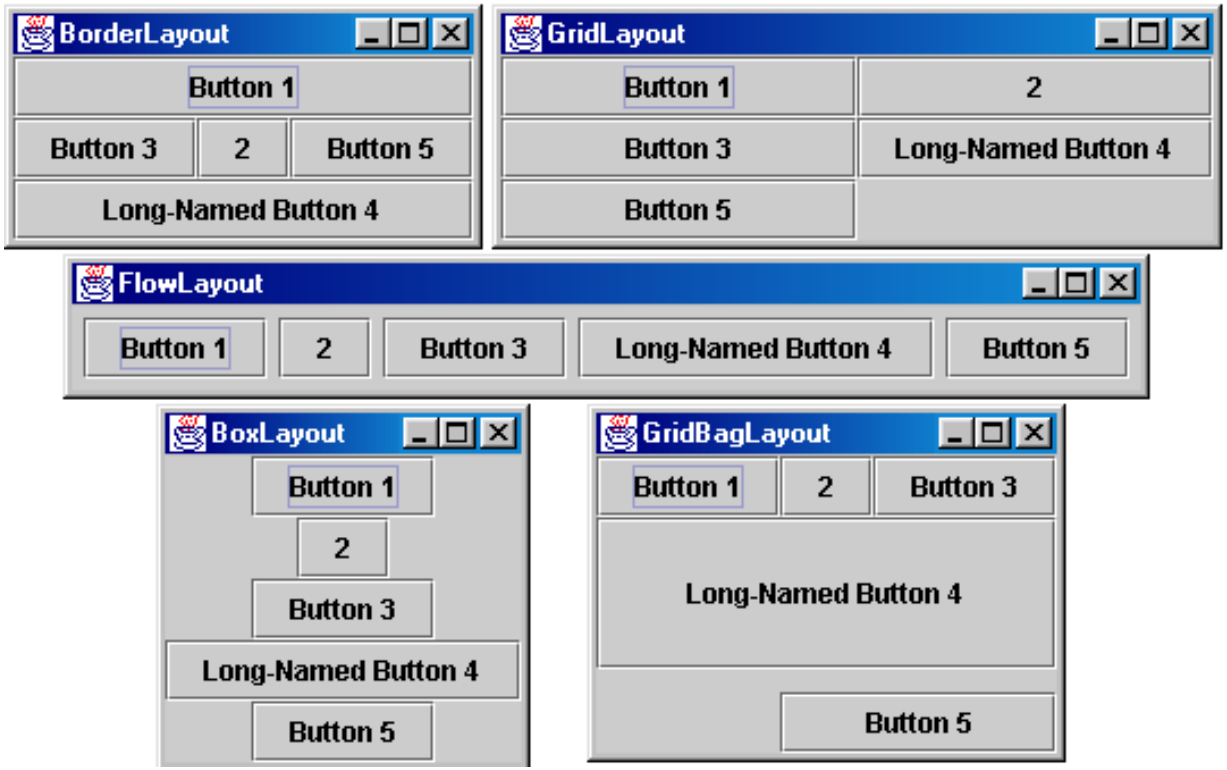
- How does the programmer specify where each component appears, how big each component should be, and what the component should do if the window is resized / moved / maximized / etc.?

- Absolute positioning (C++, C#, others):
  Programmer specifies exact pixel coordinates of every component.
  - "Put this button at (x=15, y=75) and make it 70x31 px in size."

- Layout managers (Java):
  Objects that decide where to position each component based on some general rules or criteria.
  - "Put these four buttons into a 2x2 grid and put these text boxes in a horizontal flow in the south part of the frame."

# Containers and layout

- Place components in a container; add the container to a frame.
  - container: An object that stores components and governs their positions, sizes, and resizing behavior.



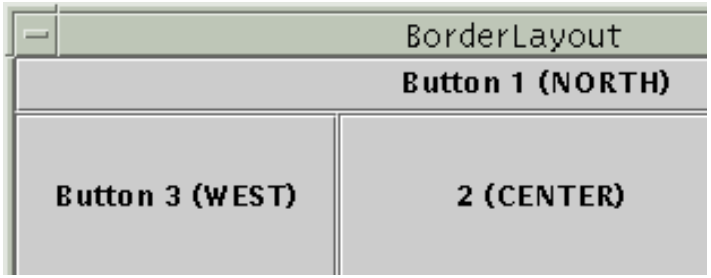
# JFrame as container

- A JFrame is a container. Containers have these methods:
- public void add(Component comp)
  public void add(Component comp, Object info)
  Adds a component to the container, possibly giving extra information about where to place it.
- public void remove(Component comp)
- public void setLayout(LayoutManager mgr)
  Uses the given layout manager to position components.
- public void validate()
  Refreshes the layout (if it changes after the container is onscreen).

# Preferred sizes

- Swing component objects each have a certain size they would "like" to be: Just large enough to fit their contents (text, icons, etc.).
  - This is called the preferred size of the component.
  - Some types of layout managers (e.g. FlowLayout) choose to size the components inside them to the preferred size.
  - Others (e.g. BorderLayout, GridLayout) disregard the preferred size and use scheme defined in the layout to size the components.
- Buttons at preferred size


Not preferred size:



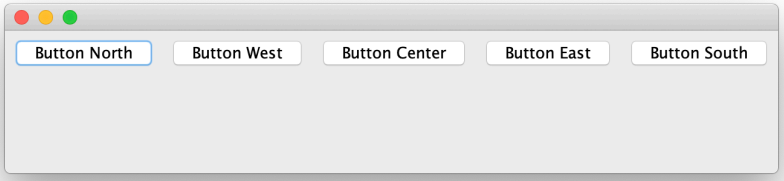
# FlowLayout

- public FlowLayout()
- treats container as a left-to-right, top-to-bottom "paragraph".
  - Components are given preferred size, horizontally and vertically.
  - Components are positioned in the order added.
  - If too long, components wrap around to the next line.

- myFrame.setLayout(new FlowLayout());
- myFrame.add(new JButton("Button 1"));
  - The default layout for containers other than JFrame (seen later).

# FlowLayout

```
JFrame f = new JFrame();

JButton b1 = new JButton("Button North");
JButton b2 = new JButton("Button West");
JButton b3 = new JButton("Button Center");
JButton b4 = new JButton("Button East");
JButton b5 = new JButton("Button South");
// adding buttons to the frame
// setting grid layout of 3 rows and 3 columns
f.setLayout(new FlowLayout());


f.add(b1);
f.add(b2);
f.add(b3);
f.add(b4);
f.add(b5);

f.setSize(640, 140);
f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```



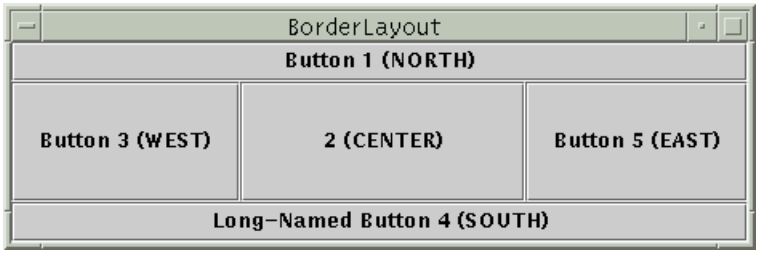
# BorderLayout

- public BorderLayout()

- Divides container into five regions:
  - NORTH and SOUTH regions expand to fill region horizontally, and use the component's preferred size vertically.
  - WEST and EAST regions expand to fill region vertically, and use the component's preferred size horizontally.
  - CENTER uses all space not occupied by others.

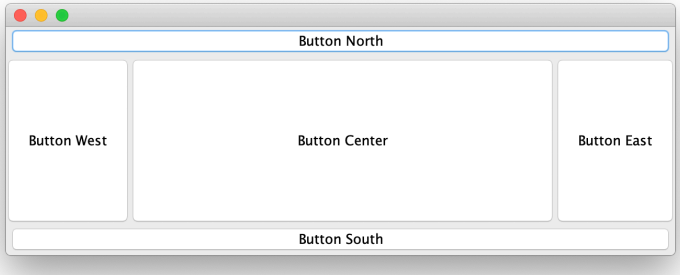
  - myFrame.setLayout(new BorderLayout());
  - myFrame.add(new JButton("Button 1"), BorderLayout.NORTH);

  - This is the default layout for a JFrame.



# BorderLayout

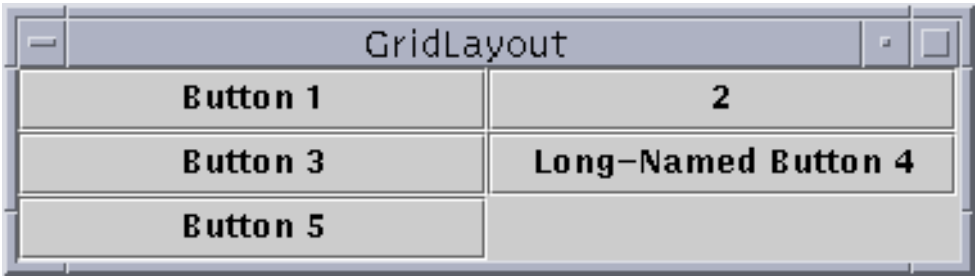
```
JFrame f = new JFrame();

JButton b1 = new JButton("Button North");
JButton b2 = new JButton("Button West");
JButton b3 = new JButton("Button
Center");
JButton b4 = new JButton("Button East");
JButton b5 = new JButton("Button South");
// adding buttons to the frame
// setting grid layout of 3 rows and 3
columns
f.setLayout(new BorderLayout());


f.add(b1, BorderLayout.NORTH);
f.add(b2, BorderLayout.WEST);
f.add(b3, BorderLayout.CENTER);
f.add(b4, BorderLayout.EAST);
f.add(b5, BorderLayout.SOUTH);

f.setSize(640, 240);
f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON
_CLOSE);
```



# GridLayout

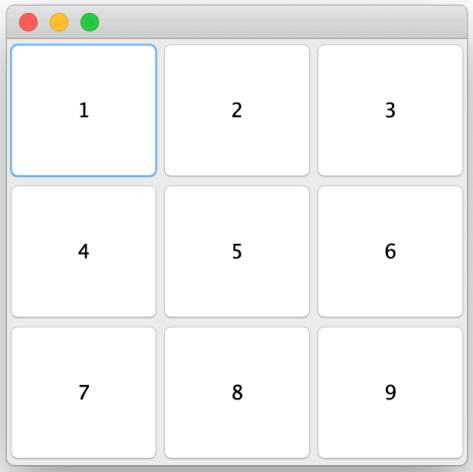
- public GridLayout(int rows, int columns)

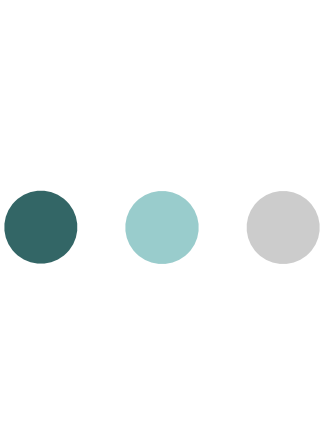
- Treats container as a grid of equally-sized rows and columns.
- Components are given equal horizontal / vertical size, disregarding preferred size.
- Can specify 0 rows or columns to indicate expansion in that direction as needed.



# GridLayout

```
JFrame f = new JFrame();

JButton b1 = new JButton("1");
JButton b2 = new JButton("2");
JButton b3 = new JButton("3");
JButton b4 = new JButton("4");
JButton b5 = new JButton("5");
JButton b6 = new JButton("6");
JButton b7 = new JButton("7");
JButton b8 = new JButton("8");
JButton b9 = new JButton("9");
// adding buttons to the frame
f.add(b1);
f.add(b2);
f.add(b3);
f.add(b4);
f.add(b5);
f.add(b6);
f.add(b7);
f.add(b8);
f.add(b9);

// setting grid layout of 3 rows and 3 columns
//f.setLayout(new GridLayout(3, 3, 20, 25));
f.setLayout(new GridLayout(3, 3));
f.setSize(300, 300);
f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```





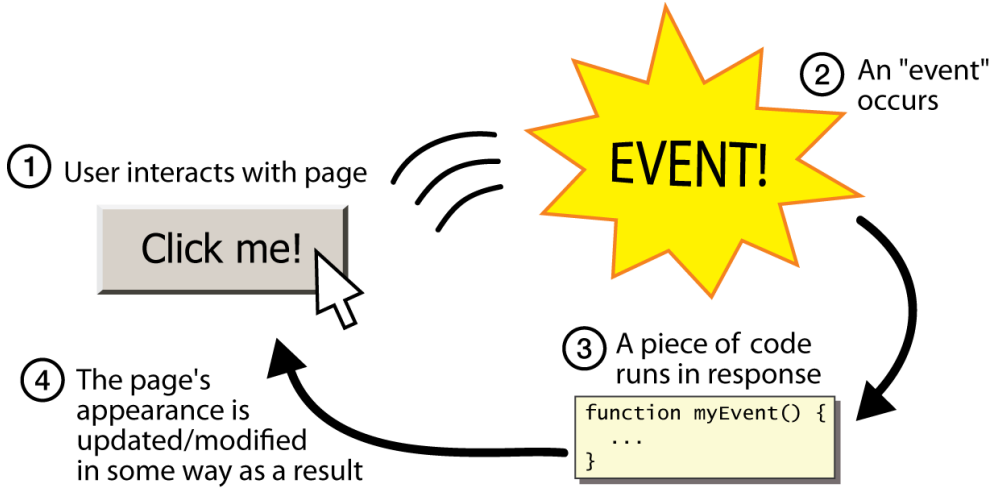
# Event Listeners

responding to events

# Graphical events

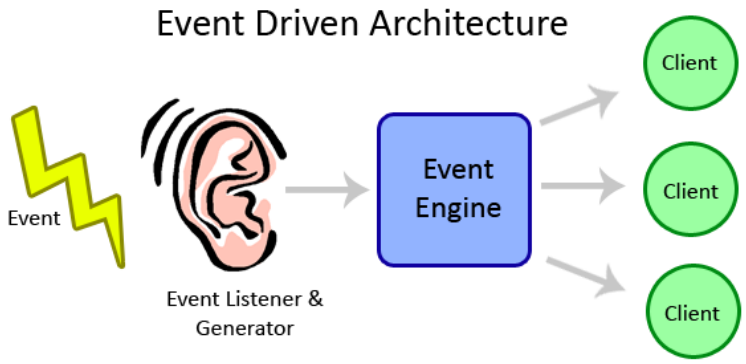
- event: An object that represents a user's interaction with a GUI component; can be "handled" to create interactive components.

- listener: An object that waits for events and responds to them.
  - To handle an event, attach a listener to a component.
  - The listener will be notified when the event occurs (e.g. button click).



# Event-driven programming

- event-driven programming: A style of coding where a program's overall flow of execution is dictated by events.
  - Rather than a central "main" method that drives execution, the program loads and waits for user input events.
  - As each event occurs, the program runs particular code to respond.
  - The overall flow of what code is executed is determined by the series of events that occur, not a pre-determined order.



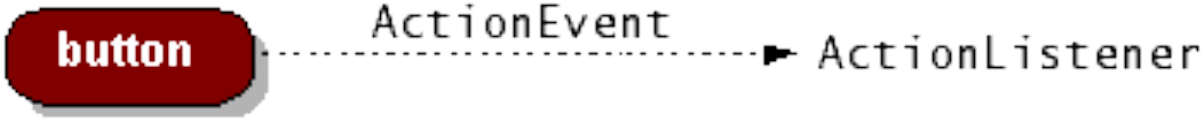
# Event hierarchy

- EventObject
  - AWTEvent (AWT)
    - ActionEvent
    - TextEvent
    - ComponentEvent
      - FocusEvent
      - WindowEvent
      - InputEvent
        - KeyEvent
        - MouseEvent

```
import java.awt.event.*;

 EventListener
AWTEventListener
ActionListener
TextListener
ComponentListener
FocusListener
WindowListener

KeyListener
MouseListener
```

# Action events

- action event: An action that has occurred on a GUI component.
  - The most common, general event type in Swing. Caused by:
    - button or menu clicks,
    - check box checking / unchecking,
    - pressing Enter in a text field, ...
  - Represented by a class named ActionEvent
  - Handled by objects that implement interface ActionListener



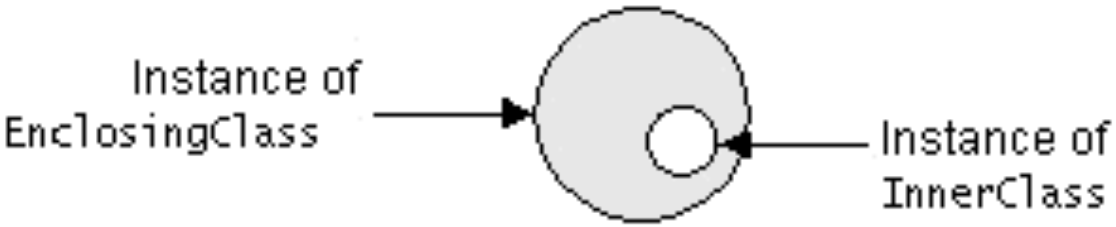
# Implementing a listener

```
public class name implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        code to handle the event;
    }
}
```

- JButton and other graphical components have this method:
  - public void addActionListener(ActionListener al)
    Attaches the given listener to be notified of clicks and events that occur on this component.

# 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

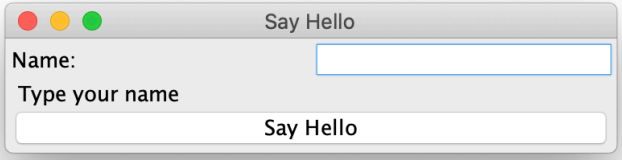
# Static inner classes

```
// 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: HelloGUIEvent

```
public HelloGUIEvent(){
	// set up components
	helloField = new JTextField(15);

	messageLabel = new JLabel( "  Type your name  ");
	computeButton = new JButton("Say Hello");
	computeButton.addActionListener(this);

	// layout
	JPanel north = new JPanel(new GridLayout(1, 1));
	north.add(new JLabel(" Name: "));
	north.add(helloField);
	// overall frame
	frame = new JFrame("Say Hello");

	frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	frame.setLayout(new BorderLayout());
	frame.add(north, BorderLayout.NORTH);
	frame.add(messageLabel, BorderLayout.CENTER);
	frame.add(computeButton, BorderLayout.SOUTH);
	frame.pack();
	frame.setVisible(true);

}
```



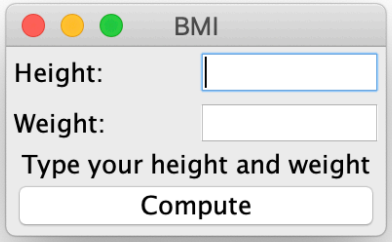
# GUI event example: HelloGUIEvent

```
public class HelloGUIEvent implements
ActionListener{

      JTextField helloField;
      JButton computeButton;
      JFrame frame;
      JLabel messageLabel;
```


```
      // Handles clicks on say hello button
      public void actionPerformed(ActionEvent event) {
      // read the name
      String nameText = helloField.getText();
      messageLabel.setText("  Hello: " + nameText);
      }
```

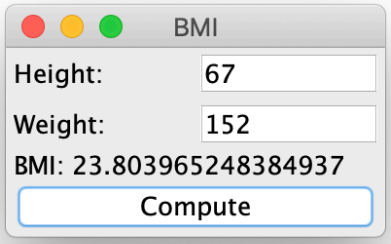
# GUI event example

```
public BMIGUI(){
        // set up components
        heightField = new JTextField(5);
        weightField = new JTextField(5);
        JLabel bmiLabel = new JLabel( "  Type your height and weight  ");
        computeButton = new JButton("Compute");

        // layout
        JPanel north = new JPanel(new GridLayout(2, 2));
        north.add(new JLabel(" Height: "));
        north.add(heightField);
        north.add(new JLabel(" Weight: "));
        north.add(weightField);

        // overall frame
        frame = new JFrame("BMI");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout());
        frame.add(north, BorderLayout.NORTH);
        frame.add(bmiLabel, BorderLayout.CENTER);
        frame.add(computeButton, BorderLayout.SOUTH);
        frame.pack();
        frame.setVisible(true);
        }
```



# GUI event example

```
public class BMIGUI implements
ActionListener{

	JTextField heightField, weightField;
	JButton computeButton;
	JFrame frame;
	JLabel bmiLabel;

	// Handles clicks on Compute button by computing the BMI.
	public void actionPerformed(ActionEvent event) {

		// read height and weight info from text fields
		String heightText = heightField.getText();
		double height = Double.parseDouble(heightText);
		String weightText = weightField.getText();
		double weight = Double.parseDouble(weightText);

		// compute BMI and display it onscreen
		double bmi = weight / (height * height) * 703;
		bmiLabel.setText("BMI: " + bmi);

	}
```

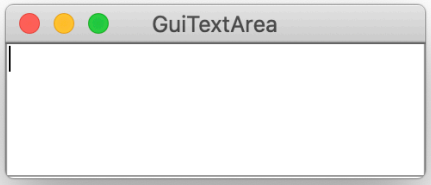


# GUI Text Areas, Scrollbars, and Fonts

- Text fields are useful for single-line text input, but they don't work well when the user wants to type a larger or more complex message.
- Fortunately, there is another kind of component called a text area that represents a text input box that can accept multiple lines.
- Text areas are represented by JTextArea objects.

# GUI Text Areas, Scrollbars, and Fonts

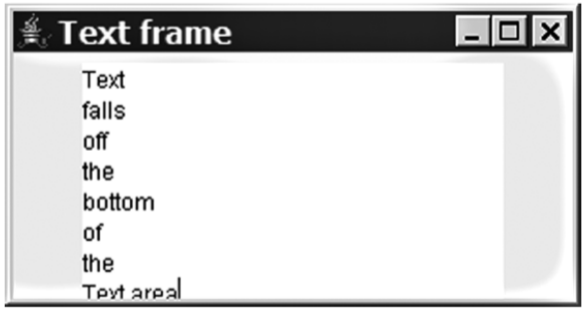
```
JTextArea linesOfText;
public GuiTextArea(){

      linesOfText = new JTextArea(5, 20);
      // overall frame
      frame = new JFrame("GuiTextArea");
      frame.setLayout(new GridLayout(2, 2));

      frame.add(linesOfText);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.pack();
      frame.setVisible(true);

}
```



# GUI Text Areas, Scrollbars, and Fonts

- In a larger GUI example, an event listener might examine the text that is written in a text area by calling its getText method or set new text in the text area by calling its setText method.

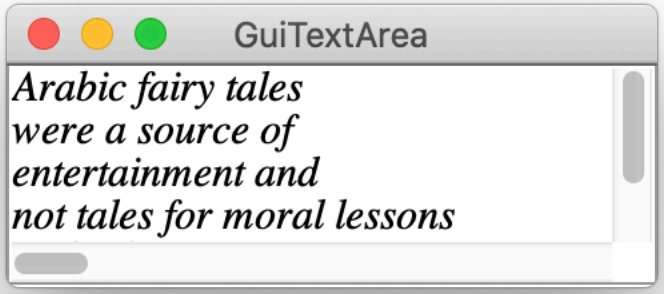
- Currently, when the user types too much text to fit in the text area, the text disappears off the bottom of the text box:



# GUI Text Areas, Scrollbars, and Fonts

- To fix this problem, we can make the text area scrollable by adding familiar navigation components called scrollbars to it.
- Scrollbars are represented by instances of a special container component called a JScrollPane.
- To make a component scrollable, create a JScrollPane, add the component to the scroll pane, and then add the scroll pane to the overall frame.
- You construct a JScrollPane object by passing the relevant component as a parameter:

- `// use scrollbars on this text area`
  `frame.add(new JScrollPane(area));`



# GUI Text Areas, Scrollbars, and Fonts

```
linesOfText = new JTextArea(5, 20);
linesOfText.setFont(new Font("Serif", Font.ITALIC, 16));
// overall frame
frame = new JFrame("GuiTextArea");
frame.setLayout(new GridLayout(1, 1));


frame.add(new JScrollPane(linesOfText));
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.pack();
frame.setVisible(true);
```