EVENTS AND GRAPHICAL USER INTERFACES

GUIs

- A graphical user interface (GUI) in Java is created with at least three kinds of objects
  - components, events, listeners
- A *component* is a graphical screen element
  - label, button, text field, check box, etc.
- Some components are also containers, which hold other components
  - frame, panel, applet, dialog box

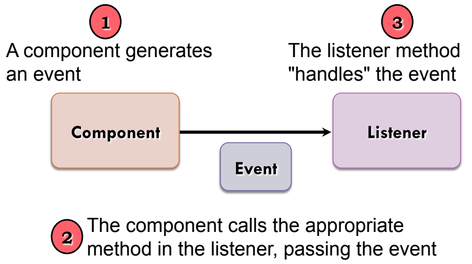
Events

- An *event* is an object that represents some activity to which we want to respond
  - a graphical button is pressed
  - a check box is toggled
  - the mouse is moved
  - the mouse is dragged
  - the mouse button is clicked
  - a keyboard key is pressed
  - a timer expires
- A component generates or "fires" an event

Listeners

- A *listener* object "waits" for an event to occur and responds accordingly
- Listeners are created by implementing a listener interface or deriving it from a listener adapter class
- We generally make use of component and event classes from the Java API library, and write listeners as event handlers
- We must establish the relationship between the components that fire events and the listeners that respond to them

GUI Processing



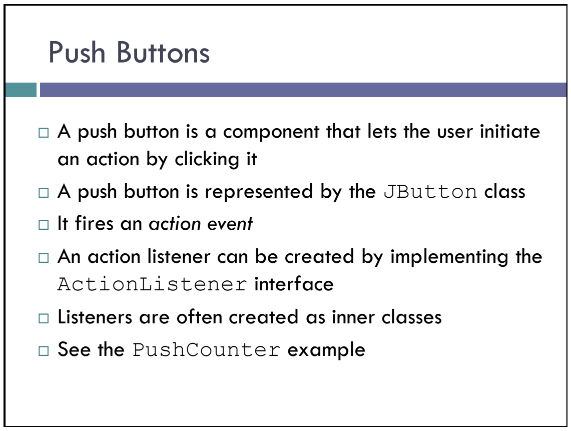
1 A component generates an event

3 The listener method "handles" the event

2 The component calls the appropriate method in the listener, passing the event

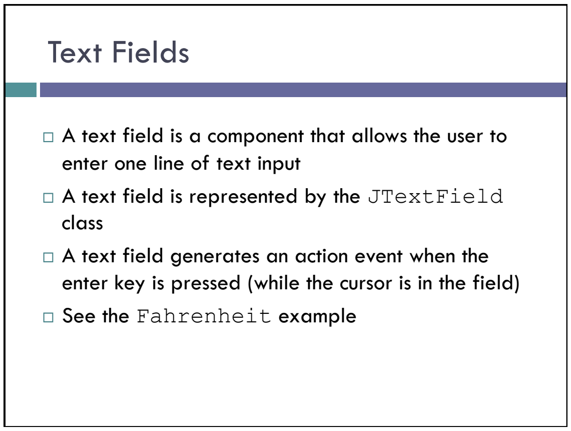
Model-View-Controller

- A software design should reflect three key roles and keep them separated:
  - Model - manages domain-specific data
  - View - presents the model in a user interface
  - Controller - manages the interaction
- The examples we'll look at focus on the event processing in Java
- The model and view are merged



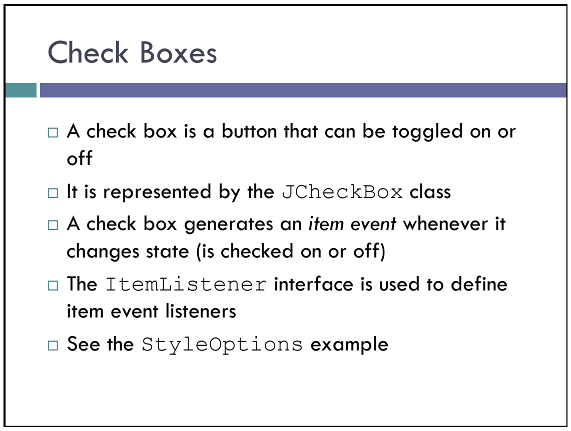
Push Buttons

- A push button is a component that lets the user initiate an action by clicking it
- A push button is represented by the `JButton` class
- It fires an *action event*
- An action listener can be created by implementing the `ActionListener` interface
- Listeners are often created as inner classes
- See the `PushCounter` example



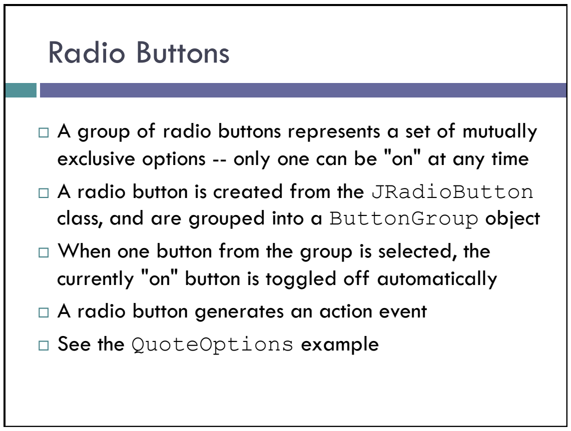
Text Fields

- A text field is a component that allows the user to enter one line of text input
- A text field is represented by the `JTextField` class
- A text field generates an action event when the enter key is pressed (while the cursor is in the field)
- See the `Fahrenheit` example



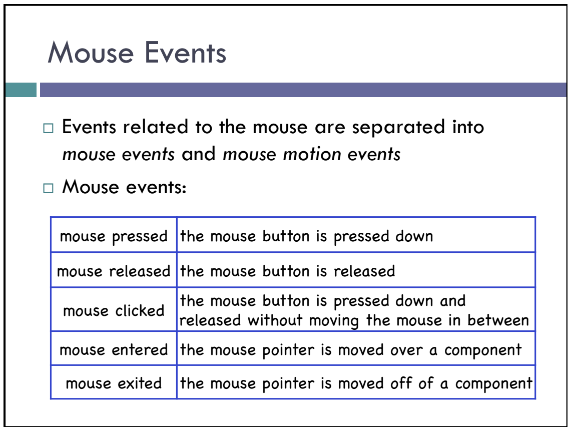
Check Boxes

- A check box is a button that can be toggled on or off
- It is represented by the `JCheckBox` class
- A check box generates an *item event* whenever it changes state (is checked on or off)
- The `ItemListener` interface is used to define item event listeners
- See the `StyleOptions` example



Radio Buttons

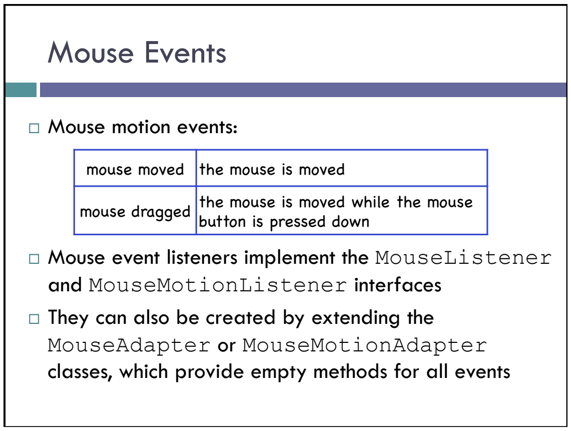
- A group of radio buttons represents a set of mutually exclusive options -- only one can be "on" at any time
- A radio button is created from the `JRadioButton` class, and are grouped into a `ButtonGroup` object
- When one button from the group is selected, the currently "on" button is toggled off automatically
- A radio button generates an action event
- See the `QuoteOptions` example



Mouse Events

- Events related to the mouse are separated into *mouse events* and *mouse motion events*
- Mouse events:

mouse pressed	the mouse button is pressed down
mouse released	the mouse button is released
mouse clicked	the mouse button is pressed down and released without moving the mouse in between
mouse entered	the mouse pointer is moved over a component
mouse exited	the mouse pointer is moved off of a component



Mouse Events

- Mouse motion events:

mouse moved	the mouse is moved
mouse dragged	the mouse is moved while the mouse button is pressed down

- Mouse event listeners implement the `MouseListener` and `MouseMotionListener` interfaces
- They can also be created by extending the `MouseAdapter` or `MouseMotionAdapter` classes, which provide empty methods for all events

Mouse Events

- For a given program, we may only care about one or two mouse events
- Empty methods can be used to satisfy the listener interface
- See the `Dots` example
- *Rubberbanding* is the visual effect in which a shape is stretched as it is drawn with the mouse
- See the `RubberLines` example

Key Events

- A *key event* is generated when a keyboard key is used

key pressed	a key is pressed down
key released	a key is released
key typed	a key is pressed down and released

- Listeners implement the `KeyListener` interface or extend the `KeyAdapter` class
- Constants in the event object can be used to determine which key was pressed
- See the `Direction` example

Sliders

- A *slider* is a component that allows the user to specify a value within a numeric range
- It is represented by the `JSlider` class
- The minimum and maximum values are set by values passed to the constructor
- A slider can be oriented vertically or horizontally and can have optional tick marks and labels
- A slider produces a *change event* when it is moved
- See the `SlideColor` example

The Timer Class

- A timer generates an action event at specified intervals
- The `Timer` class contains methods to start and stop the timer, and to set the interval delay
- It's defined in the `javax.swing` package and considered to be a GUI component though it has no visual representation
- Timers can be used to create animations
- See the `Rebound` example

Other Components

- *Dialog boxes* of various types can be created using the `JOptionPane` class
- There are two specialized dialog boxes:
  - color choosers (`JColorChooser`)
  - file choosers (`JFileChooser`)
- *Combo boxes* combine a text box and a drop-down menu (JComboBox)
- Other containers include scroll panes (`JScrollPane`) and split panes (`JSplitPane`)

Extras

- *Borders* of various styles can be put around any component to enhance the look or create distinct visual spaces
- Components can be disabled (grayed out) when they shouldn't be used
- *Tool tips* can be defined to appear when the mouse cursor rests momentarily on a component
- Mnemonics can be set so that components can be triggered using the keyboard

Layout Managers

- A layout manager is an object that determines the way components are arranged in a container

Layout Manager	Defined In
Flow Layout	AWT
Border Layout	AWT
Card Layout	AWT
Grid Layout	AWT
Box Layout	Swing
Overlay Layout	Swing

Layout Managers

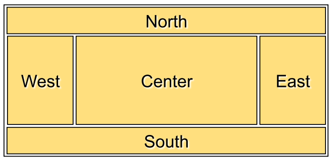
- Every container has a default layout manager
- The `setLayout` method is used to explicitly set the layout manager
- Each layout manager has its own particular rules governing how components are arranged
- Some layout managers pay attention to a component's preferred size and others do not
- The layout manager is consulted as components are added and as the container is resized
- See the `LayoutDemo` example

Flow Layout

- Flow layout puts as many components as possible on a row
- Rows are created as needed to accommodate all of the components
- Components are displayed in the order they are added
- Horizontal alignment and horizontal and vertical gaps can be explicitly set
- Flow layout is the default for a panel

Border Layout

- A border layout defines five areas:



North

West

Center

East

South

- A single component can be added to each area
- The areas expand or contract as needed to accommodate components or fill space

Grid Layout

- A grid layout displays components in a rectangular grid of rows and columns
- One component per cell
- All cells have the same size
- As components are added, they fill the grid from left-to-right and top-to-bottom (by default)
- The size of each cell is determined by the overall size of the container

Box Layout

- A box layout organizes components in one row horizontally or in one column vertically
- Components are placed top-to-bottom or left-to-right in the order they are added
- Many different configurations can be created using multiple containers with box layout
- Invisible components can be added to take up space between components
  - Rigid areas have a fixed size
  - Glue determines where excess space goes