Responding to the mouse

- Sliders, buttons, and other widgets, have built-in ways of responding to movements of the mouse, and to clicks of the mouse button when then cursor is positioned over them
- We may need to program an application to respond to such events occurring when the cursor is not positioned over any displayed widgets
  - For example: In a garden layout design program we may need to click at all the places where we want trees planted (that is, displayed)
- There are two groups of mouse events and event handlers:
  - The `MouseListener` mechanism (mainly click events)
  - The `MouseMotionListener` mechanism (moving and dragging)

The MouseListener mechanism

- The program `implements MouseListener`, and registers itself in `createGUI` as interested in mouse events within the drawing panel using:

```
panel.addMouseListener(this);
```

- Then it *must* contain definitions of *five* event handling methods
- Here are three obvious ones:
  - Called when the mouse button has been pressed and released without moving ("clicked"):

```
public void mouseClicked(MouseEvent e) { }
```

  - Called when a mouse button has been pressed:

```
public void mousePressed(MouseEvent e) { }
```

  - Called when a mouse button has been released:

```
public void mouseReleased(MouseEvent e) { }
```

- Note: The empty method bodies `{ }` must be filled in as needed

- There are two others, of less general use:
  - Called when the mouse moves over the window:

```
public void mouseEntered(MouseEvent e) { }
```

  - Called when the mouse moves off the window:

```
public void mouseExited(MouseEvent e) { }
```

- So, we can fine-tune the way our programs respond to mouse clicks in quite a detailed way
  - We can choose *which* events the program responds to by adding event handling code to the appropriate method body(ies)
  - But *all* method definitions *must* be present in at least the minimal forms above
- Within the methods we can obtain the coordinates of the cursor when the event occurred using library methods:

```
x = e.getX();      // Both ints
y = e.getY();
```


Example: Message jumps to mouse click location
(outline)

Note: So x, y are relative to the *panel*

```
private int x = 50, y = 50;

  // In createGUI
panel.addMouseListener(this);

public void mouseClicked(MouseEvent e) {
  x = e.getX();
  y = e.getY();
  Graphics g = ...;
  paintScreen(g);
}

private void paintScreen(Graphics g) {
  g.drawString("Hello World", x, y);
}
```

Demo

The MouseMotionListener mechanism

- The program `implements MouseMotionListener`, and registers itself in `createGUI` as interested in mouse events using:

```
addMouseMotionListener(this);
```

- It *must* contain definitions of *two* event handling methods:
  - Called when the mouse is moved a little with no buttons held down:

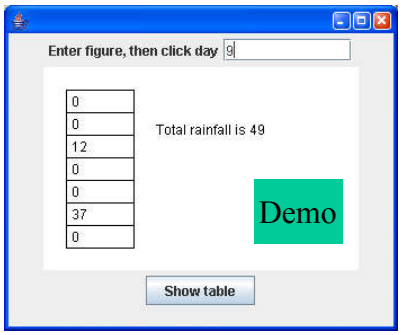
```
public void mouseMoved(MouseEvent e) { }
```

  - Called when the mouse is moved a little with a button held down:

```
public void mouseDragged(MouseEvent e) { }
```

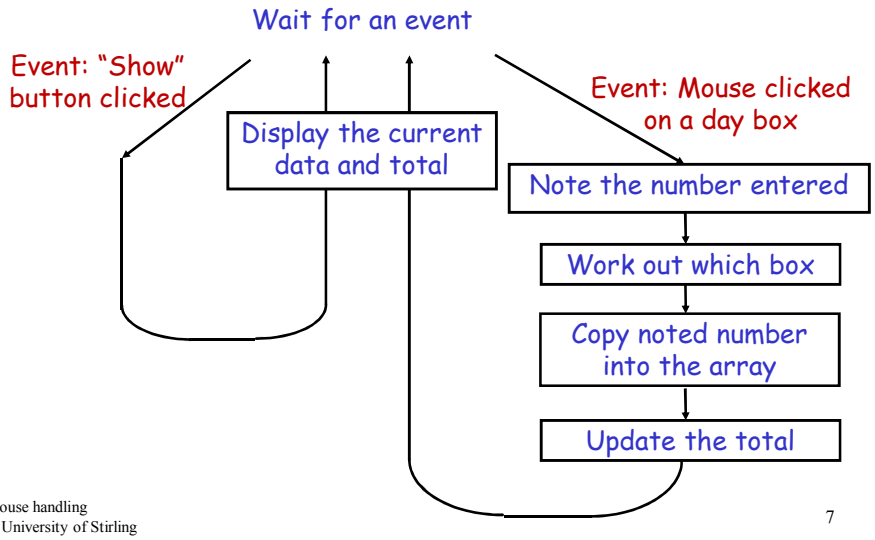
- If the mouse is moved/dragged continuously, then these events happen repeatedly (and frequently!)
- Other details are as for MouseListener

Example: The Rainfall program



- This program manages a list of seven integers representing the rainfall on the seven days in a week:
  - Data is entered by typing a number into the text field, then clicking on a day box
  - The total rainfall for the week is always on display
- The day boxes are *drawn* - they are not widgets
- The program makes use of a one-dimensional array, a text field, and a `mouseClicked` event handler
- (There is also a "Show table" button to force initial display of the table)

- Apart from the `main` launch, there are only *two* relevant events, so the behaviour design is like this:



The Rainfall program

(focusing on the array and mouse aspects)

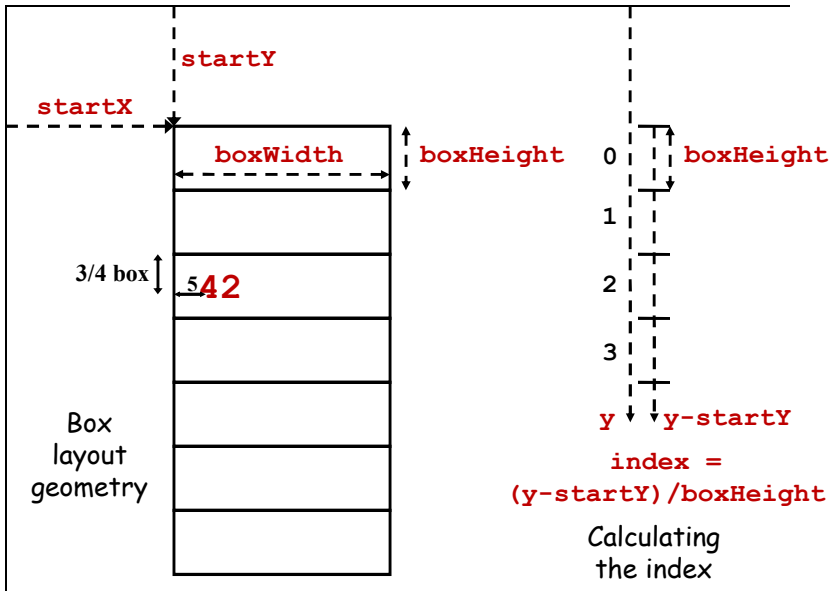
```
public class Rainfall extends JFrame
        implements MouseListener, ActionListener {

  // Size constant and array creation
  private final int tableSize = 7;
  private int[] rain = new int[tableSize];
  private int sum;          // For the total

  private void createGUI() {
    // Initialize array to 0s and set up GUI
    for (int day = 0; day < tableSize; day++)
      rain[day] = 0;
    sum = 0;

    ...(usual window set up)...
    panel.addMouseListener(this);
  } // createGUI
```

paintScreen draws the following box layout:




```
public void mouseClicked(MouseEvent e) {
  // Get data and y coord
  int newValue = Integer.parseInt(value.getText());
  int y = e.getY();
  if (startY <= y &&     // Check validity of y
      y <= startY+tableSize*boxHeight) {
    // OK, calculate index of box
    int index = (y - startY)/boxHeight;
    rain[index] = newValue;     // Insert into array
  }
  value.setText("");    // Clear the text field
  addValues();          // Recompute total
  ...(redraw the screen)...
} // mouseClicked
```

Ideally we should also check that the x coordinate of the mouse click is in the correct range from `startX` to `startX+boxWidth`

End of lecture