Software Development 2

Some GUI Examples

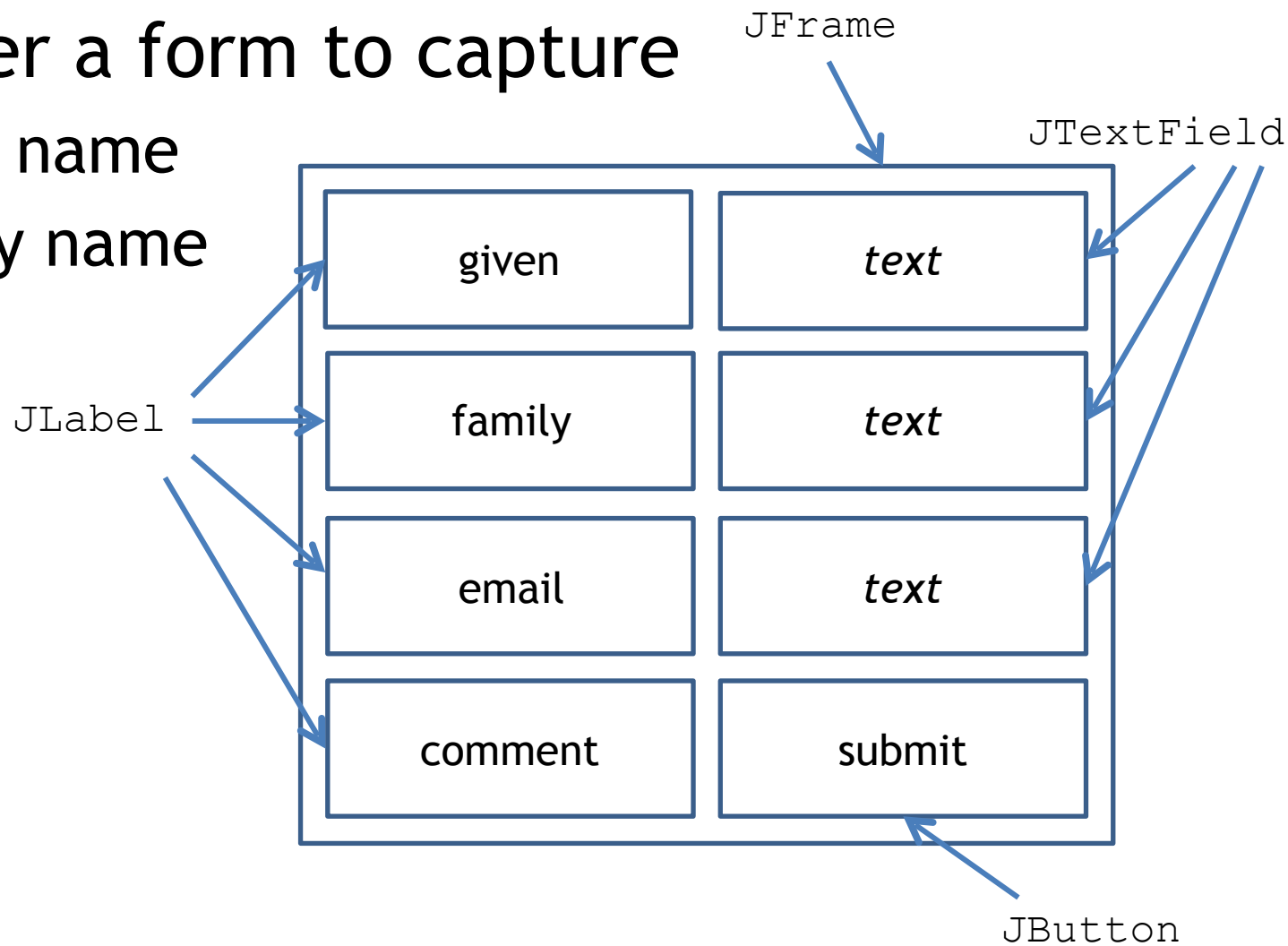
F27SB

Previous Lecture

- Program structure
- Editable text
- Dynamically changing the interface

Example: form

- consider a form to capture
  - given name
  - family name
  - …



Example: form

When **submit** button is pressed:

- check all `JTextField`s have entries
- place any error message in *comment*
- write all entries to file
- swap *comment/submit* panel with *more/stop*

Example: form

Then when **more** pressed:

- swap panels back again
- clear all `JTextField`s

And when **stop** pressed:

- close the file
- exit the program

Example: form

```
class Form extends JFrame implements ActionListener
{
   JLabel given,family,email,comment; //fields
   JTextField gt,ft,et; //for user input
   JButton submit,more,stop; //buttons

   PrintWriter file; //for writing to a file
```

Example: form

New methods to set up `JLabel/JButton/JTextField`

- also adds new object to specified `Container` *c*

```
JLabel setupLabel(String s, int style, Container c)
  {  JLabel l = new JLabel(s,JLabel.CENTER);
     l.setFont(new Font("serif",style,18));
     c.add(l);
     return l;
  }
```

Example: form

- Button method also adds an action listener:

```
JButton setupButton(String s, Container c)
{   JButton b = new JButton(s);
    b.setFont(new Font("serif",Font.ITALIC,18));
    c.add(b);
    b.addActionListener(this);
    return b;
}
```

Example: form

- Action events from text fields are ignored
  - Text will be read from all of them when submit is pressed

```
JTextField setupTextField(Container c)
{  JTextField t = new JTextField();
   t.setFont(new Font("sanserif",Font.PLAIN,18));
   c.add(t);
   return t;
}
```

Example: form

```
public Form()
{   setLayout(new GridLayout(4,2)); // form grid

    // set up and add the field labels and text fields
    given = setupLabel("Given name",Font.PLAIN,this);
    gt = setupTextField(this);
    family = setupLabel("Family name",Font.PLAIN,this);
    ft = setupTextField(this);
    email = setupLabel("Email address",Font.PLAIN,this);
    et = setupTextField(this);
    comment = setupLabel("",Font.ITALIC,this);

    // set up, add, and register listener of the button
    submit = setupButton("SUBMIT",this);
```

Example: form

```
    // also setup more and stop, but don't yet add them!
    more = new JButton("MORE");
    more.setFont(new Font("serif",Font.ITALIC,18));
    more.addActionListener(this);

    stop = new JButton("STOP");
    stop.setFont(new Font("serif",Font.ITALIC,18));
    stop.addActionListener(this);
}
```

Example: form

```
// handle events from buttons
public void actionPerformed(ActionEvent e)
{  if(e.getSource()==submit)
    doSubmit();
   else
   if(e.getSource()==more)
    doMore();
   else
   if(e.getSource()==stop)
    doStop();
}
```

Example: form

```
// will be called when submit button is pressed
void doSubmit() {
   // make sure valid input has been entered
   if(gt.getText().equals(""))
      comment.setText("Enter given name");
   else
   if(ft.getText().equals(""))
    comment.setText("Enter family name");
   else
   if(et.getText().equals(""))
    comment.setText("Enter email address");
   else
   // if valid, then save input to disk file
   {  file.println(gt.getText());
      file.println(ft.getText());
      file.println(et.getText());
```

Example: form

```
        // and swap comment/submit with more/stop buttons
        remove(comment);
        comment.setVisible(false);
        remove(submit);
        submit.setVisible(false);
        add(more);
        more.setVisible(true);
        add(stop);
        stop.setVisible(true);
        setVisible(true);
    }
}
```

Example: form

```
// will be called when more button is pressed
void doMore() {
   // reset text input fields
   comment.setText(""); gt.setText("");
   ft.setText(""); et.setText("");
   // replace more/stop with comment/submit
   remove(more);
   more.setVisible(false);
   remove(stop);
   stop.setVisible(false);
   add(comment);
   comment.setVisible(true);
   add(submit);
   submit.setVisible(true);
}
```

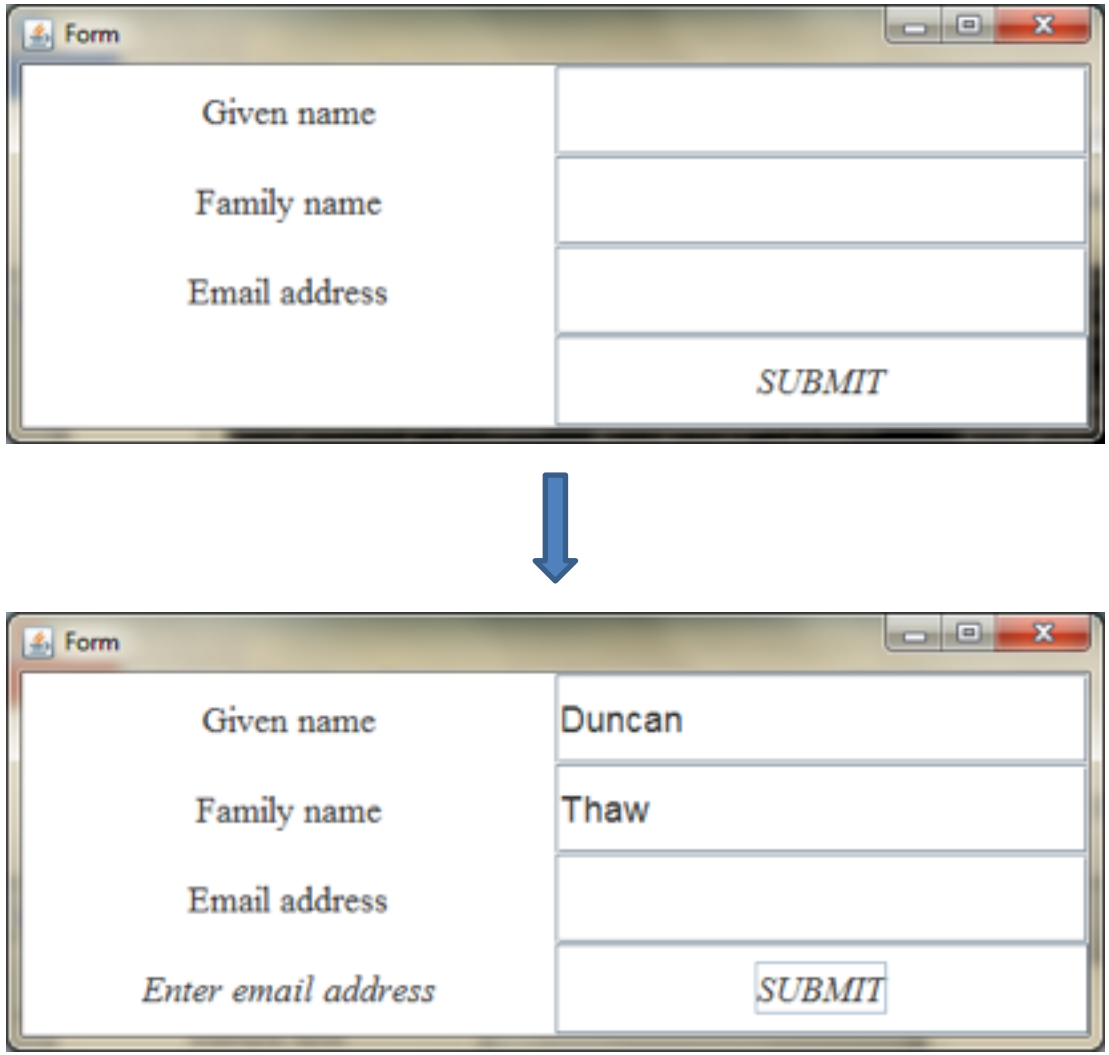
Example: form

```
// will be called when stop button is pressed
void doStop()
{   file.close();
    System.exit(0);
}
```

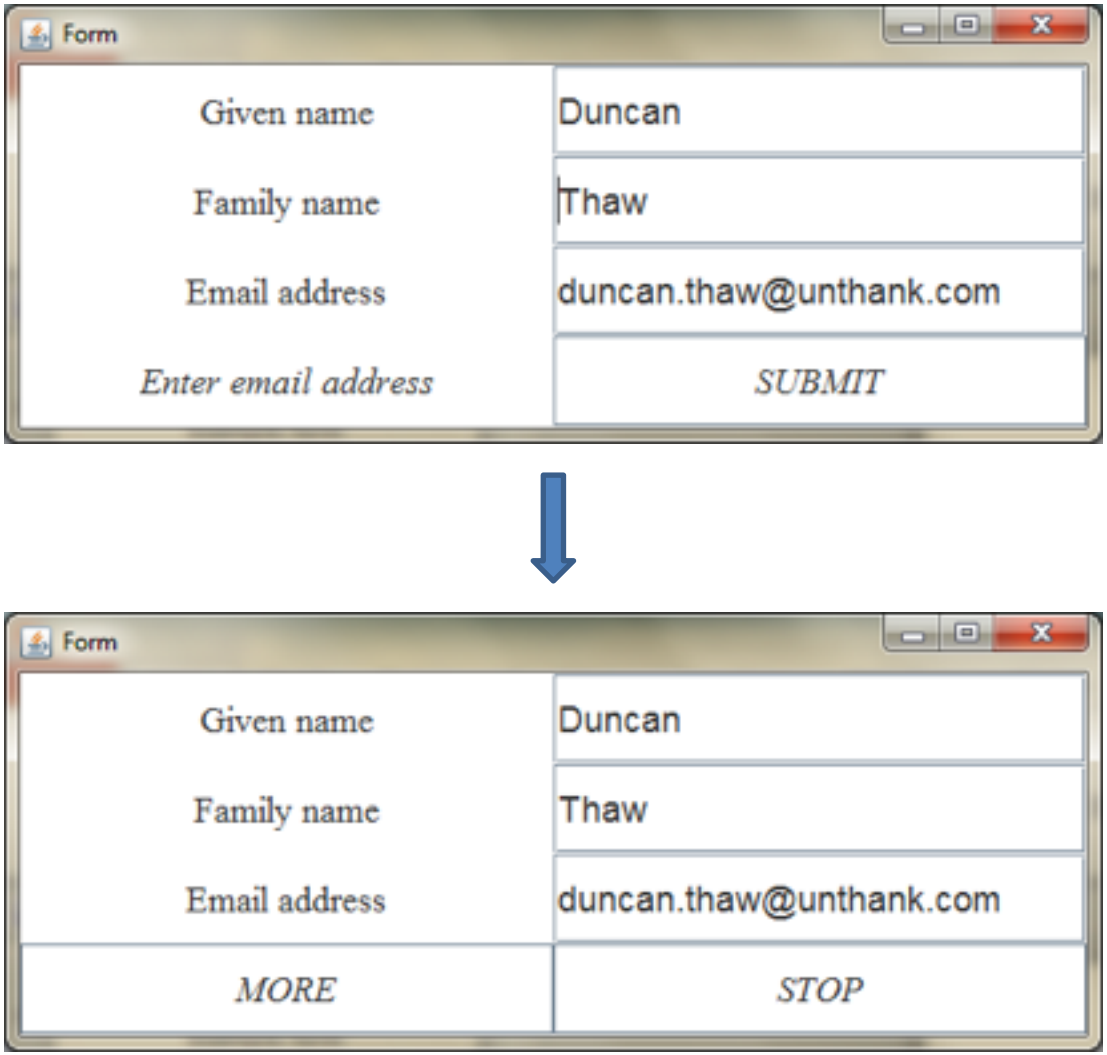
Example: form

```
   // setup the printwriter to write to a specified file
   public void setup() throws IOException
   {  file = new PrintWriter
                  (new FileWriter("register.log"),true);
   }
}

class TestForm
{  public static void main(String [] args)
    throws IOException
   {  Form f;
      f = new Form(); // create form
      ...
      f.setup(); // and setup file output
   }
}
```

Example: form



Example: form



Today's Lecture

- Some simple application examples
  - Numeric keypad
  - Sliding blocks puzzle
  - Noughts and crosses

- But first
  - Some thoughts on code structure
  - Some more info about listeners

Code Structure

- So, we want to display a button in a frame…

Code Structure

- This code displays a button in a frame:
  - But the style is poor, because it's all in main()

```
public class ButtonExample {
      public static void main(String[] args) {
            JButton b = new JButton("Press Me");
            JFrame f = new JFrame("Frame");
            f.setLayout(new FlowLayout());
            f.add(b);
            f.setSize(200, 200);
            f.setVisible(true);
      }
}
```

Code Structure

- This code is better, since main() is shorter:
  - But it still isn't very object oriented

```
public class ButtonExample2 {
      public ButtonExample2() {
            JButton b = new JButton("Press Me");
            JFrame f = new JFrame("Frame");
            f.setLayout(new FlowLayout());
            f.add(b);
            f.setSize(200, 200);
            f.setVisible(true);
      }
      public static void main(String[] args) {
            ButtonExample2 be = new ButtonExample2();
}}
```

Code Structure

- This code is more object-oriented:
  - Since it uses inheritance to extend JFrame

```
public class ButtonExample3 extends JFrame {
      public ButtonExample3() {
            JButton b = new JButton("Press Me");
            setLayout(new FlowLayout());
            add(b);
            setSize(200, 200);
            setVisible(true);
      }
      public static void main(String[] args) {
            ButtonExample3 be = new ButtonExample3();
      }
}
```

Code Structure

- These three pieces of code are all equivalent
  - They do the same thing when run
- But they are not equal
  - ButtonExample3 is arguably better than the others
- Code structure is important
  - Code which follows OOP principles tends to be more maintainable, readable and reusable

Code Structure

- Next, we want to respond to button clicks…

Code Structure

- We can add a button listener:
  - We could use an anonymous class for this

```
public class ButtonExample4 extends JFrame {
      public ButtonExample4() {
            JButton b = new JButton("Press Me");
            setLayout(new FlowLayout());
            b.addActionListener(new ActionListener() {
               public void actionPerformed(ActionEvent e){
                   System.out.println("Pressed!");
               }
            });
            add(b);
            setSize(200, 200);
            setVisible(true);
      } ...
```

Code Structure

- But it would be neater to add a new class
  - Less clutter in the constructor

```
public class ButtonExample5 extends JFrame {
      public ButtonExample5() {
            JButton b = new JButton("Press Me");
            setLayout(new FlowLayout());
            b.addActionListener(new MyButtonListener());
            ...
}

class MyButtonListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
            System.out.println("Pressed!");
      }
}
```

Code Structure

- Even better to let the frame handle it
  - Avoids creating an extra class and better cohesion

```
public class ButtonExample6 extends JFrame
implements ActionListener {
      public ButtonExample6() {
            JButton b = new JButton("Press Me");
            setLayout(new FlowLayout());
            b.addActionListener(this);
            add(b);
            setSize(200, 200);
            setVisible(true);
      }
      public void actionPerformed(ActionEvent e) {
            System.out.println("Pressed!");
      } ...
```

Code Structure

- There are often multiple ways of implementing the same behaviour
  - But they are not always equivalent in terms of style

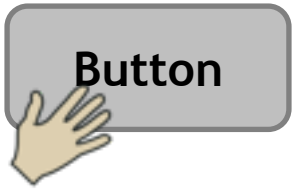
- Some useful rules of thumb
  - Avoid clutter
  - Try to keep your code structure simple
  - Make use of inheritance

Listeners (behind the scenes)

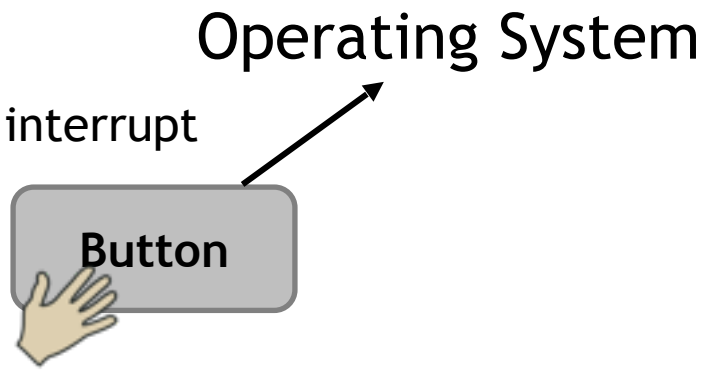
- You don’t need to know this bit for the exam
  - But it might help you understand what’s going on
- This is a simplification
  - There is a lot going on inside Java!
  - But you don’t need to know about most of it

Listeners (behind the scenes)

- User clicks on button with mouse

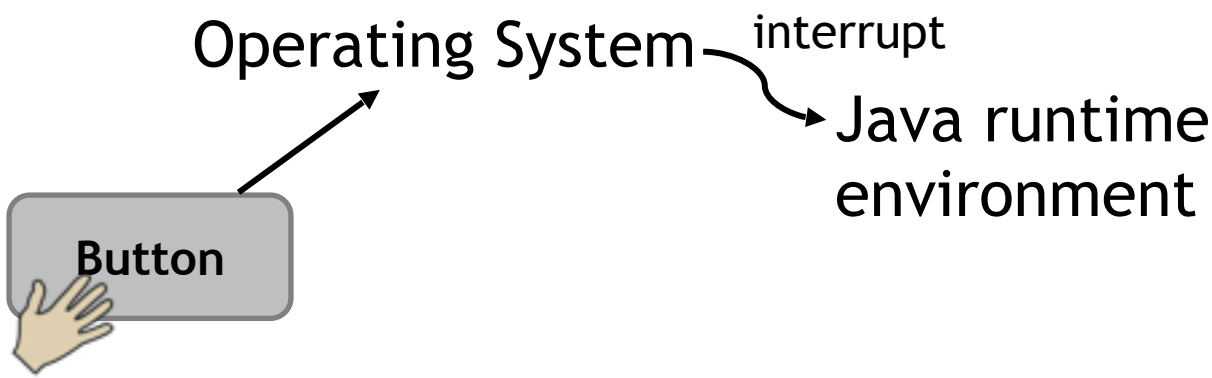


Listeners (behind the scenes)



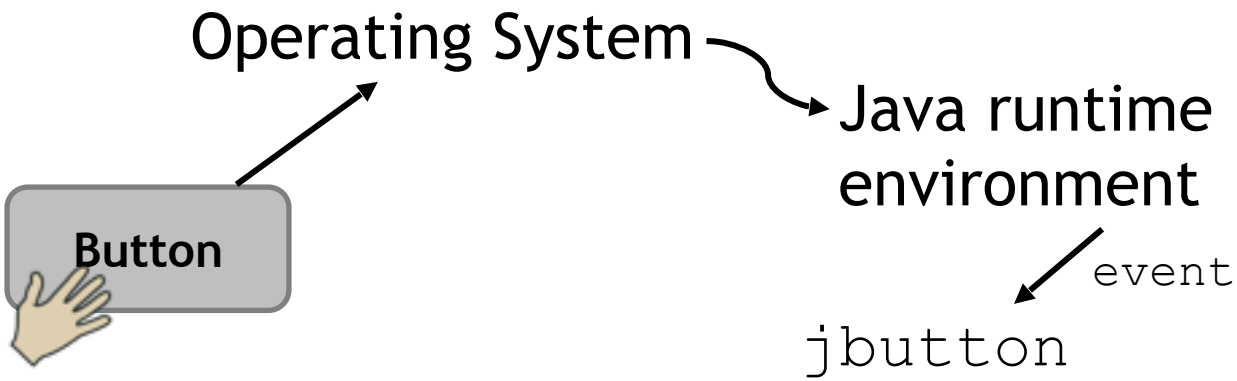
- When the user clicks the mouse button, an interrupt is raised, which is passed to the OS

Listeners (behind the scenes)



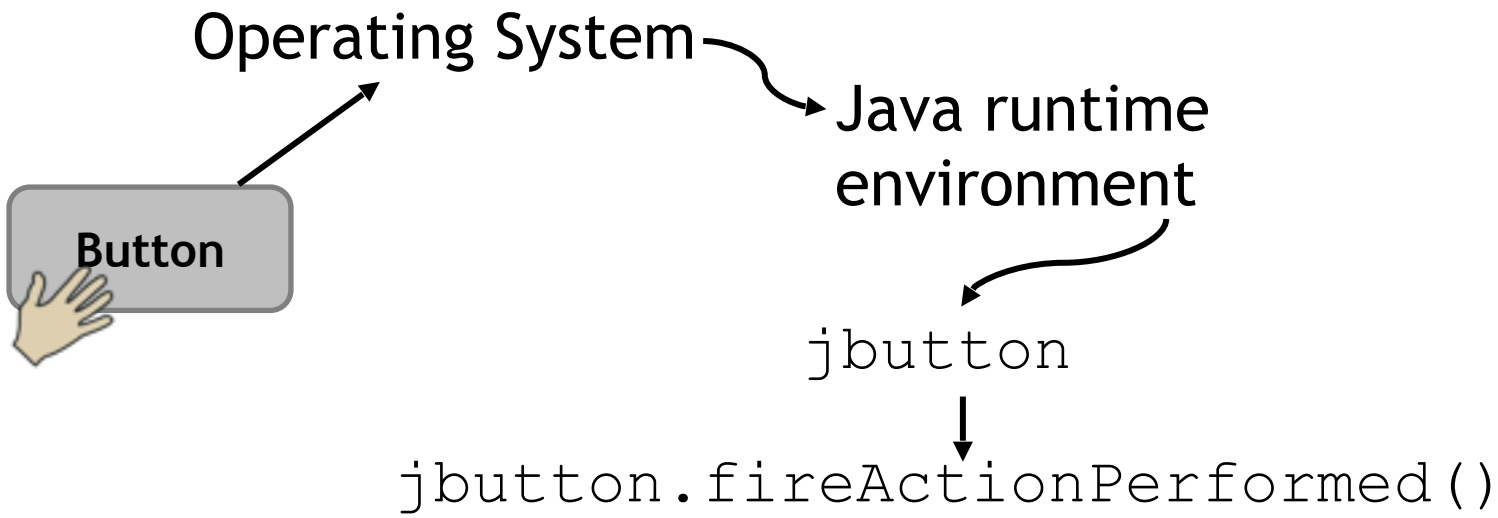
- The OS notices that the mouse pointer was over a Java program's window, and passes the interrupt to Java to handle

Listeners (behind the scenes)



- Java identifies the object corresponding to the on-screen element that was clicked

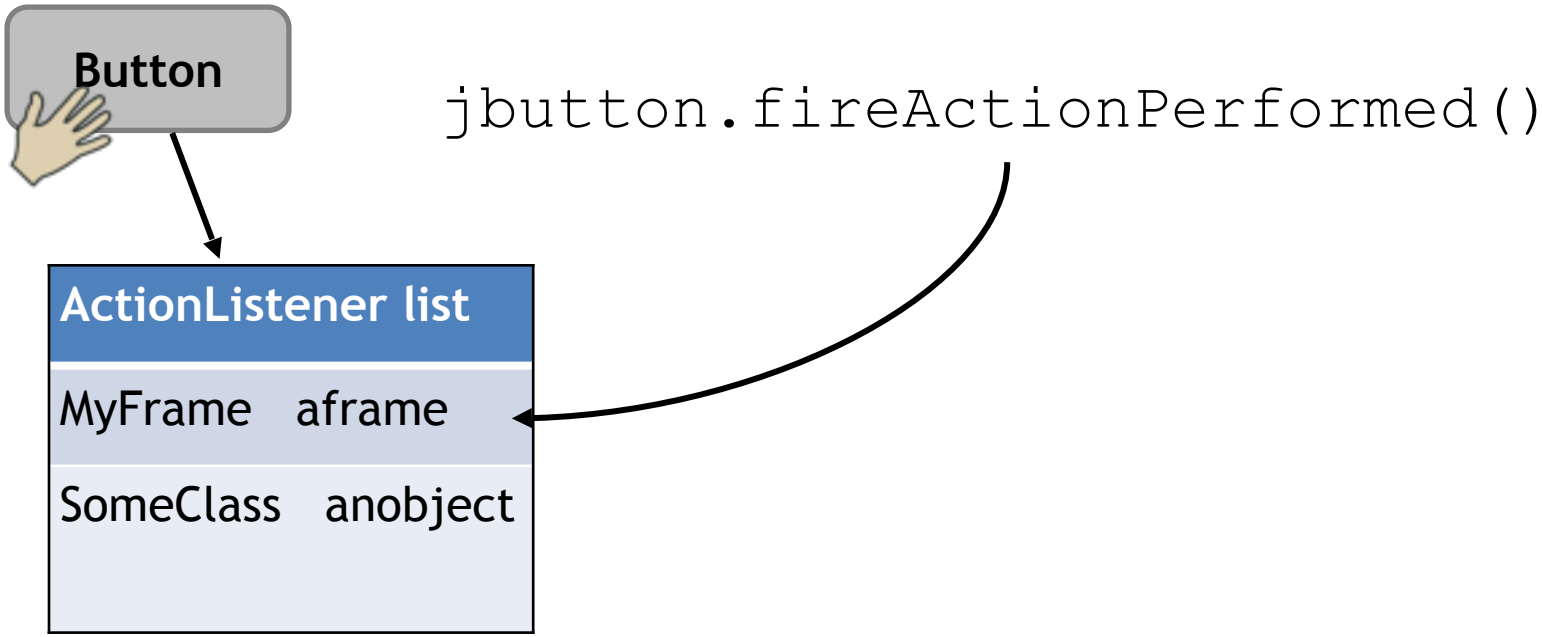
Listeners (behind the scenes)



- The button's `fireActionPerformed` method gets called

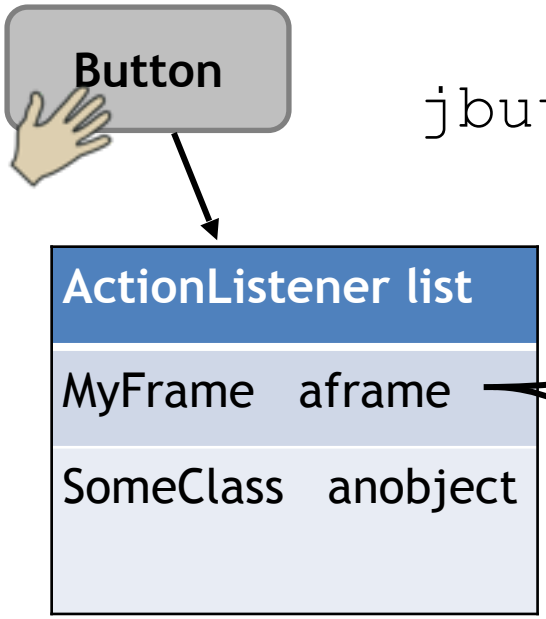
Listeners (behind the scenes)

- This method iterates through the JButton's registered `ActionListener`s



Listeners (behind the scenes)

- And calls the `actionPerformed` method belonging to each of these

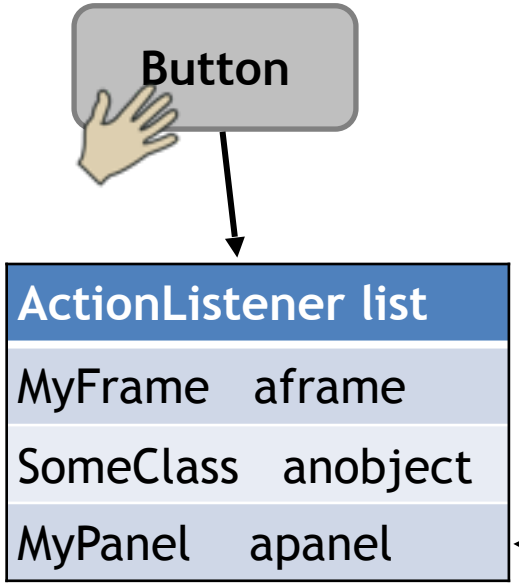



```
jbutton.fireActionPerformed()
```

```
public class MyFrame
      extends JFrame implements
      ActionListener {
   ...
   public void
   actionPerformed(ActionEvent e){
       // does something
   }
}
```

Listeners (behind the scenes)

- When the button's `addActionListener` method is called, an object is added to this list




```
public class MyPanel
      extends JPanel implements
      ActionListener {
   ...
   public MyPanel() ...
      jbutton.
         addActionListener(this);
   ...
}
```

Listeners (behind the scenes)

```
// A simplified version of JButton
public class JButton extends various stuff {

        // a list of ActionListeners
        List<ActionListener> listenerList = new
                                ArrayList<ActionListener>();

        // add a new ActionListener to the list
        public void addActionListener(ActionListener l) {
                listenerList.add(l);
        }

        // called by Java when someone clicks on the button
        public void fireActionPerformed(ActionEvent event) {
                for(ActionListener listener : listenerList)
                        listener.actionPerformed(event);
        }
…}
```

Listeners (behind the scenes)

- This is an example of the **Observer** design pattern
  - You don't need to know about design patterns
  - These are covered in Year 3 CS
  - But if you're interested:
    - https://sourcemaking.com/design_patterns

Room: **F27SB**

QUICK QUIZ

Question 1: Will this work?

```
import javax.swing.*; import java.awt.event.*;
public class MyPanel extends JPanel
                     implements ActionListener {
   public MyPanel() {
      JButton b = new JButton("Press Me");
      add(b);
      b.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e) {
      System.out.println("Boo!");
   }
   static public void main(String[] args) {
      JFrame frame = new JFrame();
      frame.add(new MyPanel());
      frame.setSize(200,200); frame.setVisible(true);
   }
}
```

Question 1: Will this work?

```
import javax.swing.*; import java.awt.event.*;
public class MyPanel extends JPanel
                     implements ActionListener {
   public MyPanel() {
      JButton b = new JButton("Press Me");
      add(b);
      b.addActionListener(this);
   }
   public void actionPerformed(A
      System.out.println("Boo!")
   }
   static public void main(Strin
      JFrame frame = new JFrame(
      frame.add(new MyPanel());
      frame.setSize(200,200); fr
   }
}
```

Yes! It's not just subclasses of JFrame that can implement ActionListener. It might make sense, for example, for a JPanel to handle events from the buttons it contains.

Question 2: Will this work?

```
import javax.swing.*; import java.awt.event.*;
public class MyButton extends JButton
                      implements ActionListener {
   public MyButton(String title) {
      super(title);
      addActionListener(this);
   }
   public void actionPerformed(ActionEvent e) {
      System.out.println("Boo!");
   }
   static public void main(String[] args) {
      JFrame frame = new JFrame();
      frame.add(new MyButton("Press Me!"));
      frame.setSize(200,200); frame.setVisible(true);
   }
}
```

Question 2: Will this work?

```
import javax.swing.*; import java.awt.event.*;
public class MyButton extends JButton
                      implements ActionListener {
   public MyButton(String title) {
      super(title);
      addActionListener(this);
   }
   public void actionPerformed(Ac
      System.out.println("Boo!")
   }
   static public void main(Strin
      JFrame frame = new JFrame(
      frame.add(new MyButton("Pr
      frame.setSize(200,200); fr
   }
}
```

Yes! It's certainly a bit unusual for a button to handle its own events, but syntactically there's no reason why it can't do this.

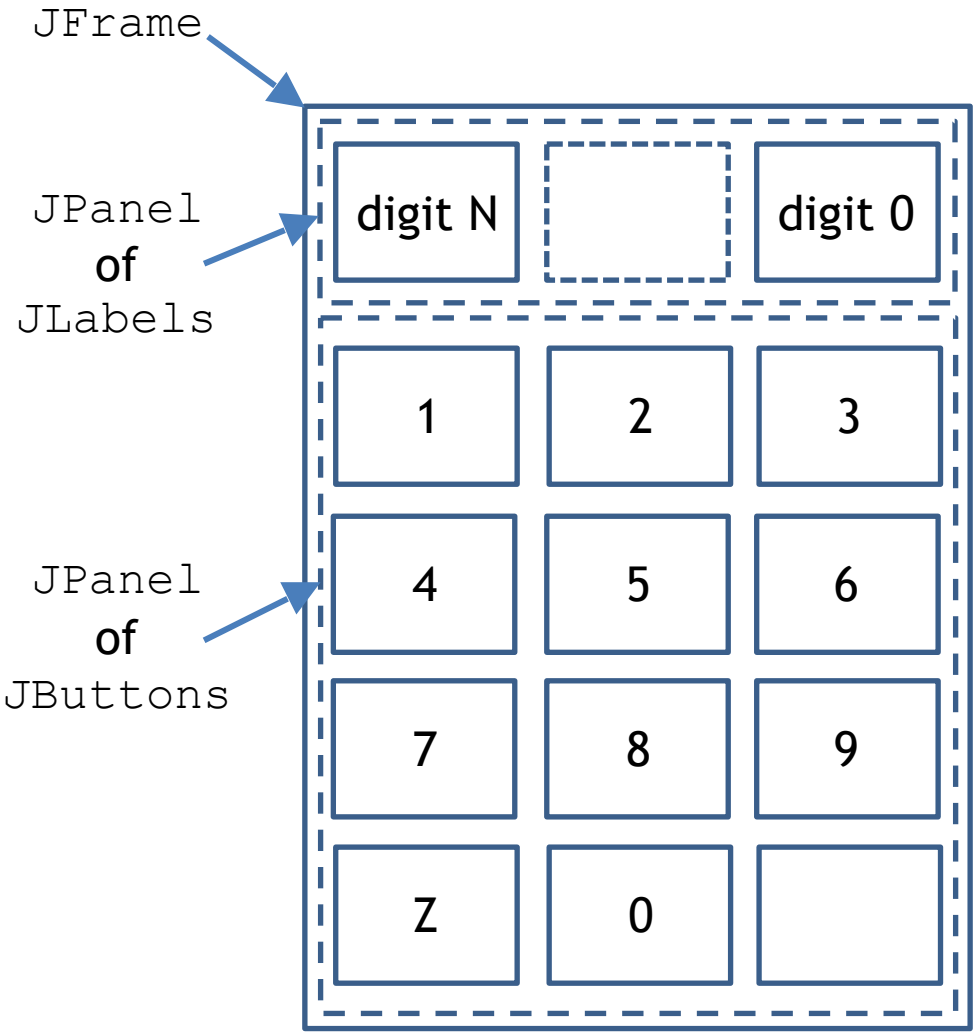
APPLICATION EXAMPLES VOTE

Numeric key pad

Lets user enter a number using a keypad:

- 10 decimal digit keys
  - 1-9 and 0
- clear key
  - Shown as “Z”
- multiple digit display

Numeric key pad



Numeric key pad

- It has a state variable: the current value
- Keys 0-9
  - shifts content of all display labels left
  - puts new key value in right-most display label
  - update state variable: multiply current value by 10 and add new digit

Numeric key pad

- e.g. current value is 14

- user presses *5*

- current value: 14 * 10 + 5 = 145

Numeric key pad

- Also a Z key
  - set all displays to 0 and sets current value to 0

- use `BorderLayout` for `JFrame`
- `JPanels` for display and keys
- add display to `North`
- add keys to `Center`
- use `GridLayout` for `JPanels`

Numeric key pad

```
class Keypad extends JFrame implements ActionListener {

   JLabel [] display;        // digit display labels
   final int DISPLAYS = 10; // number of digits

   JButton[] keys;           // button array for 0-9 keys
   JButton CLR;              // Z key
   final int KEYS = 10;      // number of keys

   JPanel d, k; // panels for display and keys

   long value; // state variable: current value in display
```

Numeric key pad

```
// creates a label with specified string
JLabel setupLabel(String s)
{  JLabel l = new JLabel(s,JLabel.CENTER);
   l.setFont(new Font("Sansserif",Font.PLAIN,18));
   l.setBackground(Color.white);
   l.setOpaque(true);
   return l;
}
// creates a button, adds to container, and adds listener
public JButton setupButton(String s,Container c)
{  JButton b = new JButton(s);
   b.setFont(new Font("Sansserif",Font.PLAIN,18));
   b.setBackground(Color.white);
   b.setOpaque(true);
   b.addActionListener(this);
   c.add(b);  return b;
}
```

Numeric key pad

```
public Keypad() {
   int i;
   // create array of JLabels for display
   d = new JPanel(new GridLayout(1,DISPLAYS));
   display = new JLabel[DISPLAYS];
   for(i=0;i<DISPLAYS;i++)
      display[i]=setupLabel("0");

   // and add them to the GUI
   for(i=DISPLAYS-1;i>=0;i--)
      d.add(display[i]);
   add(BorderLayout.NORTH,d);
```

Numeric key pad

```
    // create array of JButtons for keys
    // add them in order: 1-9, Z, 0
    k = new JPanel(new GridLayout(4,3));
    keys = new JButton[KEYS];
    for(i=1;i<KEYS;i++)
       keys[i] = setupButton(i+"",k); // 1-9
    CLR = setupButton("Z",k); // Z
    keys[0] = setupButton("0",k); // 0
    add(BorderLayout.CENTER,k);

    value = 0; // initialise state variable
}
```

Numeric key pad

```
// handle button presses
public void actionPerformed(ActionEvent e) {

   // if Z key pressed
   if (e.getSource() == CLR) {
       // reset state variable
       value = 0;
       // reset labels
       for (int i = 0; i < DISPLAYS; i++)
          display[i].setText("0");
          System.out.println(value);
```

Numeric key pad

```
// if a numeric key is pressed
} else {
   // loop through keys to see which one pressed
   for (int i = 0; i < KEYS; i++)
       if (e.getSource() == keys[i]) {
          // update state variable
          value = 10 * value + i;
          // update display
          for (int j = DISPLAYS - 2; j >= 0; j--)
             // shift along existing numbers
             display[j + 1].setText(display[j].getText());
          // set right-most digit to new number
          display[0].setText(i + "");
          return; // no need to continue checking keys
       }
}
```

Numeric key pad

Sliding blocks puzzle

Arrange blocks in some specified order from some initial configuration

- cannot pick blocks up
- can only slide block into free space

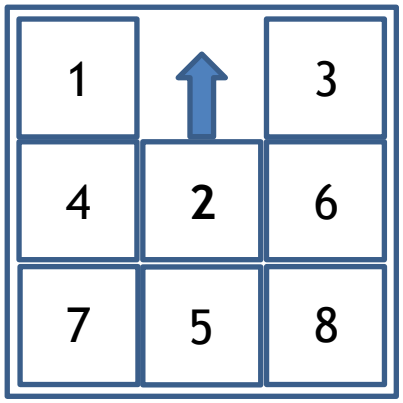
Our implementation:

- 8 square numbered blocks
- a square tray with 3*3 block positions

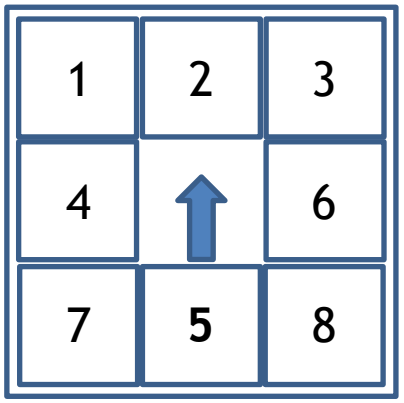
Sliding blocks puzzle

- If there's space, a block moves when clicked:

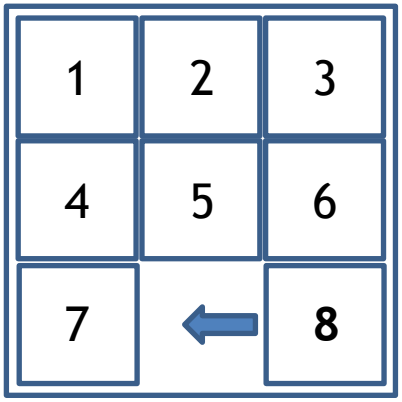
a)



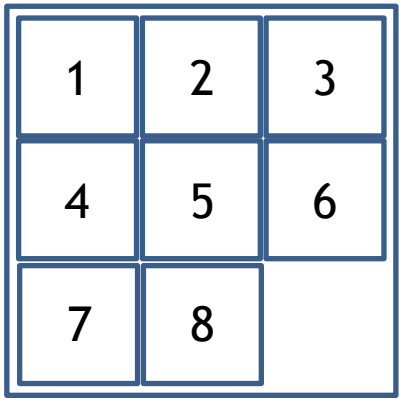
b)



c)



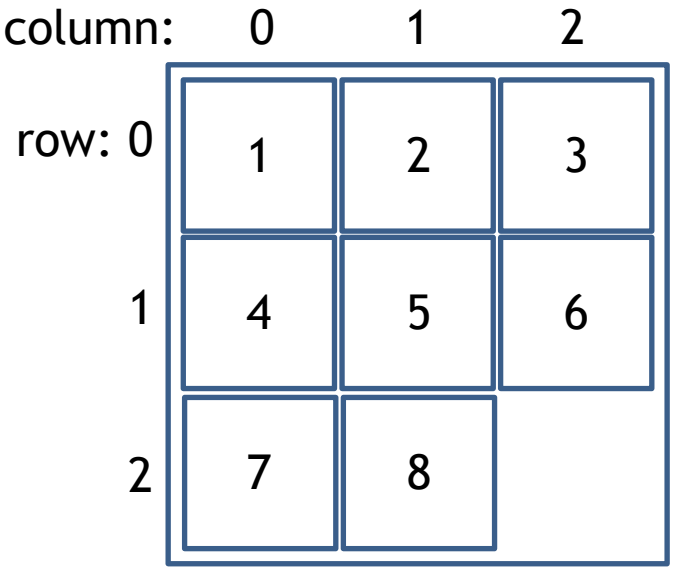
d)



Sliding blocks puzzle

Represent tray as 3*3 array of `JButtons`

- numbered blocks and a space
- block in row i/column j has number: i*3+j+1



- block in row 0 column 0 = 0*3+0+1 = 1
- block in row 0 column 1 = 0*3+1+1 = 2
- block in row 0 column 2 = 0*3+2+1 = 3
- block in row 1 column 0 = 1*3+0+1 = 4
- block in row 1 column 1 = 1*3+1+1 = 5
- block in row 1 column 2 = 1*3+2+1 = 6
- block in row 2 column 0 = 2*3+0+1 = 7
- block in row 2 column 1 = 2*3+1+1 = 8

Sliding blocks puzzle

User clicks `JButton` to move block into space

- program then swaps text on selected `JButton` and space `JButton`
- But only if the clicked block is next to the space
  - i.e. in same row and block is to left/right of space
  - i.e. in same column and block is above/below space

Sliding blocks puzzle

Suppose:

- block is in row `i` and column `j`
- space is in row `spaceI` and column `spaceJ`

Then block can move into space if:

```
i==spaceI && (j==spaceJ-1 ||
              j==spaceJ+1)
||
j==spaceJ && (i==spaceI-1 ||
              i==spaceI+1)
```

Sliding blocks puzzle

```
class Blocks extends JFrame implements ActionListener
{  JButton [][] blocks; // the blocks (and space)
   int spaceI,spaceJ;   // the current location of the space

   // creates button, adds to container, adds action listener
   public JButton setupButton(String s,Container c)
   {  JButton b = new JButton(s);
      b.setFont(new Font("Sansserif",Font.PLAIN,18));
      b.setBackground(Color.white);
      c.add(b);
      b.addActionListener(this);
      return b;
   }
```

Sliding blocks puzzle

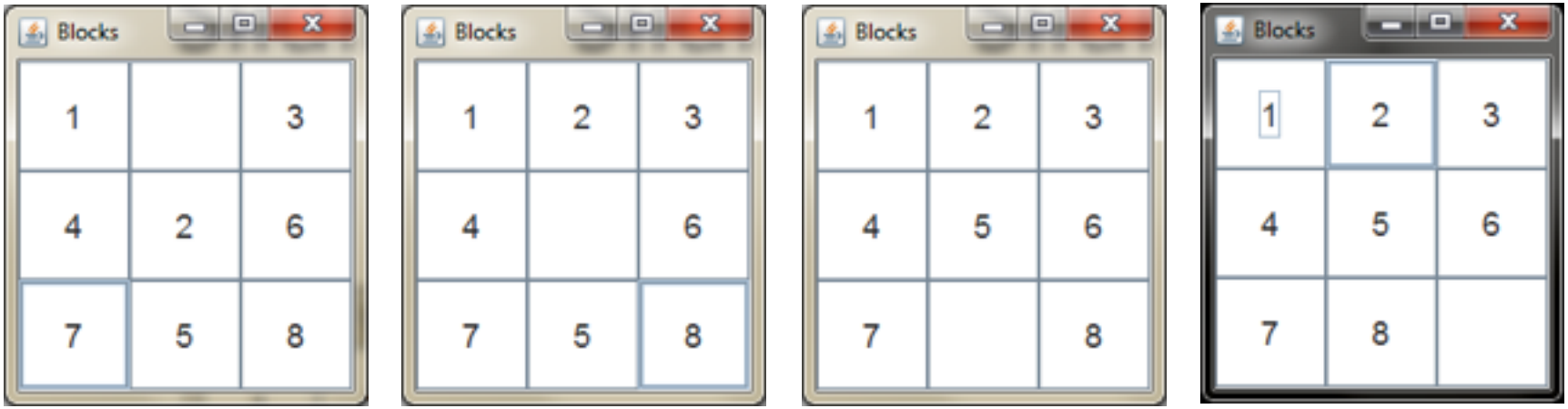
```
 public Blocks()
 {  setLayout(new GridLayout(3,3)); // 3x3 game grid

    // create and add the buttons
    blocks = new JButton[3][3];
    for(int i=0;i<3;i++)
      for(int j=0;j<3;j++)
        blocks[i][j]= setupButton((i*3+j+1)+"",this);

    // make one of these buttons into a space
    blocks[2][2].setText("");

    // remember where the space is
    spaceI=2;
    spaceJ=2;
}
```

Sliding blocks puzzle

```
public void actionPerformed(ActionEvent e) {
   // loop through buttons and identify source of event
   for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++)
          if (e.getSource() == blocks[i][j])

             // check whether this block is able to move
             if ((i==spaceI && (j==spaceJ-1 || j==spaceJ+1))
                  || (j==spaceJ && (i==spaceI-1 || i==spaceI+1))) {

                // if so, swap its label with the space
                // and remember the new location of the space
                blocks[spaceI][spaceJ].setText(blocks[i][j].getText());
                spaceI = i; spaceJ = j;
                blocks[spaceI][spaceJ].setText("");
                return;
}}
```

Sliding blocks puzzle

```
class TestBlocks
{   ... }
```



Noughts and crosses

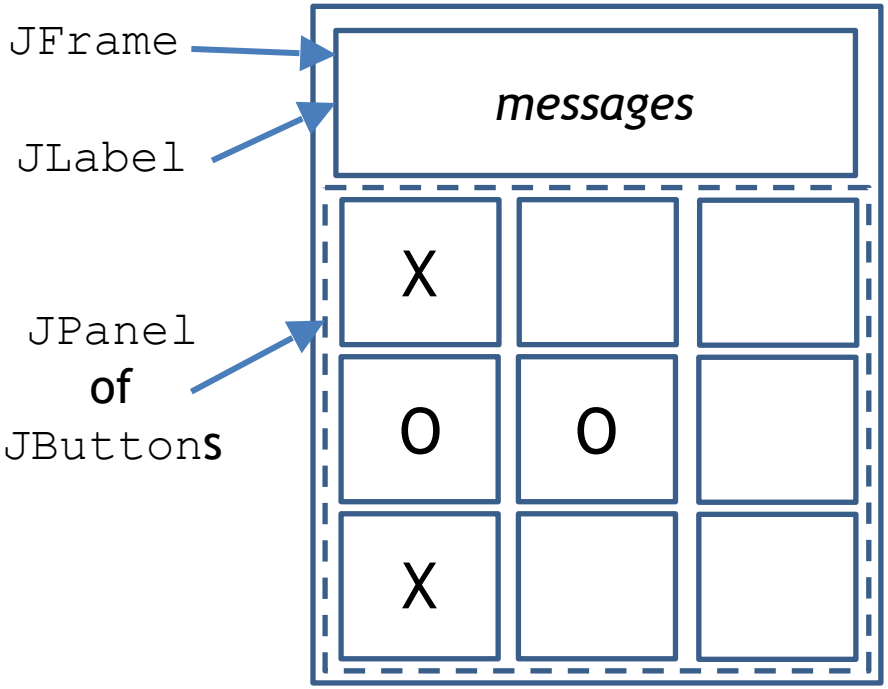
- game played on 3*3 board
- each player makes either “O” or “X” mark
- players take it in turns to place mark on free grid square of their choice
- first player to place 3 marks in horizontal/ vertical/diagonal straight line wins

X		
O	O	
X		

Noughts and crosses

Represent board as 3*3 grid of `JButton`s

- Initially with no text, marks are added during play



Noughts and crosses

Computer and user take turns:

- user clicks on their preferred `JButton`
- computer just chooses the first empty one

After user plays:

- check if user clicked on legal square i.e. no mark
- if so, set the selected button's text to the user's mark
- check if user has won, i.e. whether three marks in a line
- computer then plays in next free square
- check if computer has won

Noughts and crosses

```
class Noughts extends JFrame implements ActionListener {
        JButton[][] squares; // each position has a JButton
        JLabel messages;     // displays messages to the user
        JPanel board;        // contains the JButtons

        // create button, add to container, add listener
        public JButton setupButton(String s, Container c) {
                JButton b = new JButton(s);
                b.setFont(new Font("Sansserif", Font.PLAIN, 24));
                b.setBackground(Color.white);
                c.add(b);
                b.addActionListener(this);
                return b;
        }
```

Noughts and crosses

```
public Noughts() {
        board = new JPanel(new GridLayout(3, 3)); // 3x3 grid

        // create grid of JButtons and add to GUI
        squares = new JButton[3][3];
        for (int i = 0; i < 3; i++)
                for (int j = 0; j < 3; j++)
                        squares[i][j] = setupButton("", board);
        add(BorderLayout.CENTER, board);

        // create message label and add at top of GUI
        messages = new JLabel("Select a square to play X.",
                                JLabel.CENTER);
        messages.setFont(new Font("Serif", Font.ITALIC, 18));
        messages.setBackground(Color.white);
        add(BorderLayout.NORTH, messages);
}
```

Noughts and crosses

```
// returns a boolean value indicating whether the game has been won
boolean checkwin(String mark) {
        boolean rwin, cwin;

        // iterate through rows and columns
        for (int i = 0; i < 3; i++) {
             rwin = true;
             cwin = true;
             for (int j = 0; j < 3; j++) {
                  // is row complete?
                  rwin = rwin && squares[i][j].getText().equals(mark);
                  // is column complete?
                  cwin = cwin && squares[j][i].getText().equals(mark);
             }
             if (rwin || cwin)
                  return true;
        }
```

Noughts and crosses

```
        // then check the two diagonals
        return (squares[0][0].getText().equals(mark) &&
            squares[1][1].getText().equals(mark) &&
            squares[2][2].getText().equals(mark))
           ||
           (squares[0][2].getText().equals(mark) &&
            squares[1][1].getText().equals(mark) &&
            squares[2][0].getText().equals(mark));
}
```

Noughts and crosses

```
// carry out computer's turn
// this simply places a mark in the first empty space
boolean computerplay() {

//iterate through buttons top-left to bottom-right
       for (int i = 0; i < 3; i++)
           for (int j = 0; j < 3; j++)
               if (squares[i][j].getText().equals("")) {
                      squares[i][j].setText("O");
                   return true;
               }

       // returns false if no empty spaces are remaining
       return false;
}
```

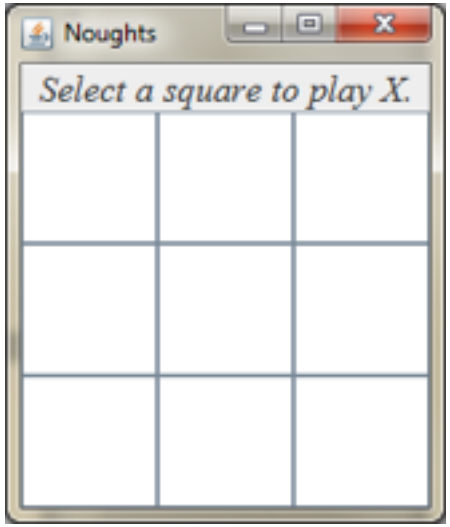
Noughts and crosses

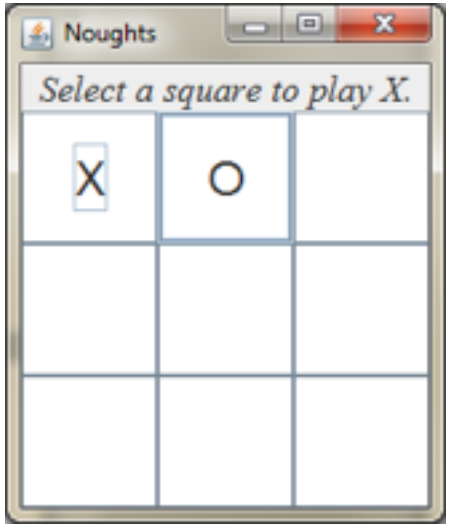
```
public void actionPerformed(ActionEvent e) {

      // loop through the buttons and identify source of event
      for (int i = 0; i < 3; i++)
          for (int j = 0; j < 3; j++)
              if (e.getSource() == squares[i][j])

              // if empty, mark the user's selected space
              if (squares[i][j].getText().equals("")) {
                  squares[i][j].setText("X");

              // check whether user has won
              if (checkwin("X"))
                  messages.setText("You win.");
```

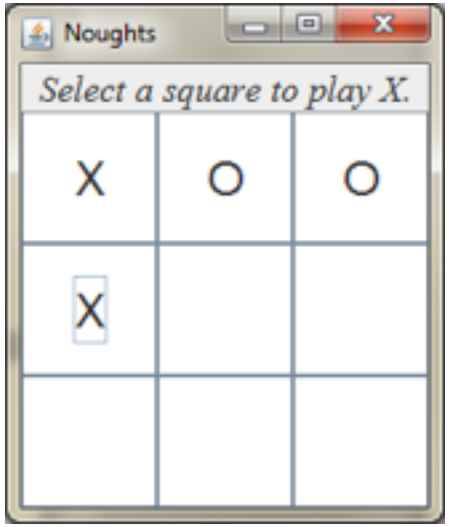
Noughts and crosses

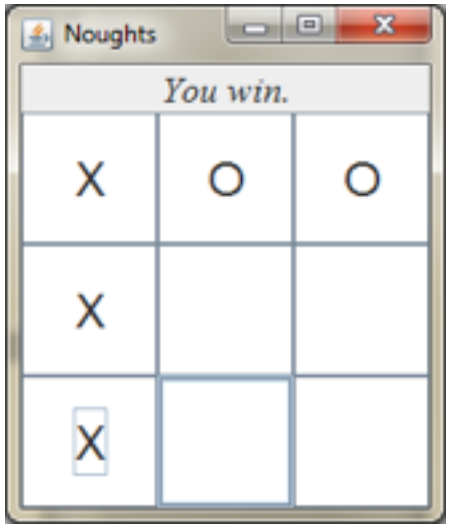
```
                // computer's turn
                else if (!computerplay())
                    // if no space remaining
                    messages.setText("Draw.");
                else if (checkwin("O"))
                    messages.setText("I win.");
                else
                    messages.setText("Select a square to play X.");
                return;

        // if not empty, tell the user
        } else {
                messages.setText("Can't play in that square.");
                return;
        }
}
class TestNoughts {  ... }
```

Noughts and crosses







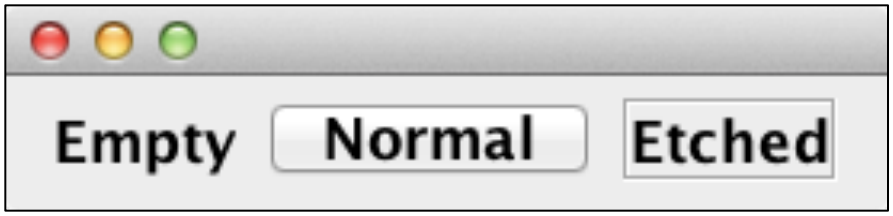


Not examinable

ADVANCED BUTTONS

Advanced [not examinable]

- `BorderFactory`
  - A factory class containing static methods, e.g.
    - ```
      Border border =
          BorderFactory.createEtchedBorder();
      ```
  - Can be applied to `JButtons` and other `JComponent`s, e.g.
    - ```
      JButton button = new JButton();
      button.setBorder(border);
      ```



Advanced [not examinable]

- HTML
  - Useful for more complex formatting, e.g.
  -
```
JButton button = new
  JButton("<html><center>Press<br>
  <font size=\"7\" color=\"#FF0080\">
  Me!</font></center></html>");
```

Advanced [not examinable]

- A `JComponent`'s appearance is determined by its `paintComponent` method
- You can override this to draw whatever you want
- i.e. implement this method in a sub-class:

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    // your drawing code here
}
```

THAT’S IT!

Next Lecture

- Interactive systems design
- State diagrams