David M. Keil, Framingham State University

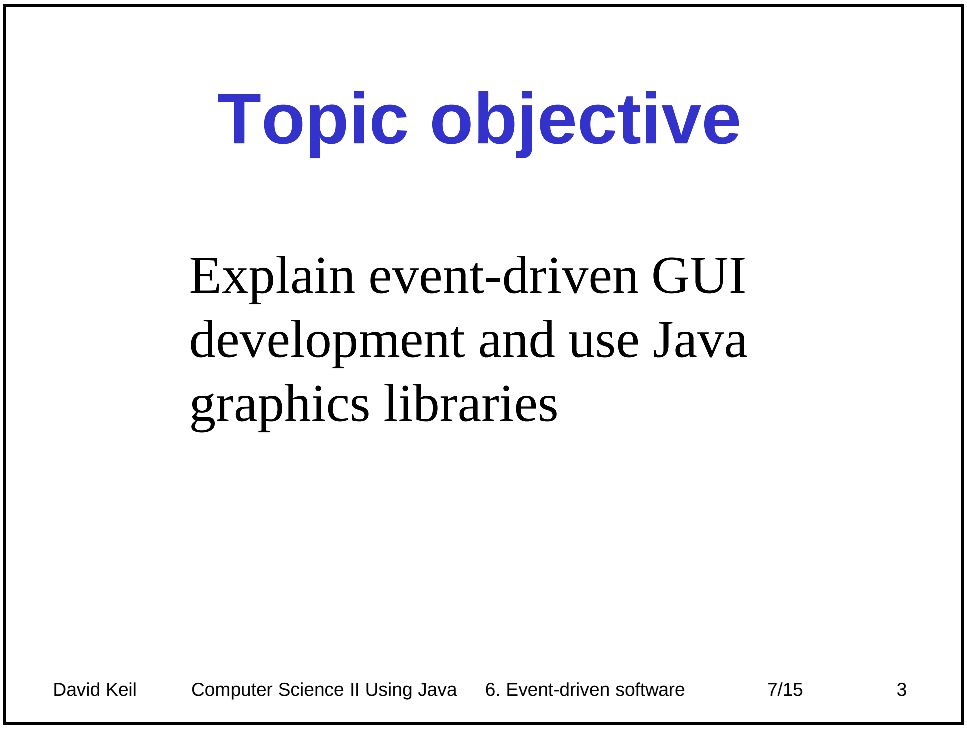
CSCI 252 Computer Science II Using Java

6. Event-driven GUIs and Java graphics

1. Developing event-driven software
2. Graphical user interface construction
3. Java graphical tools

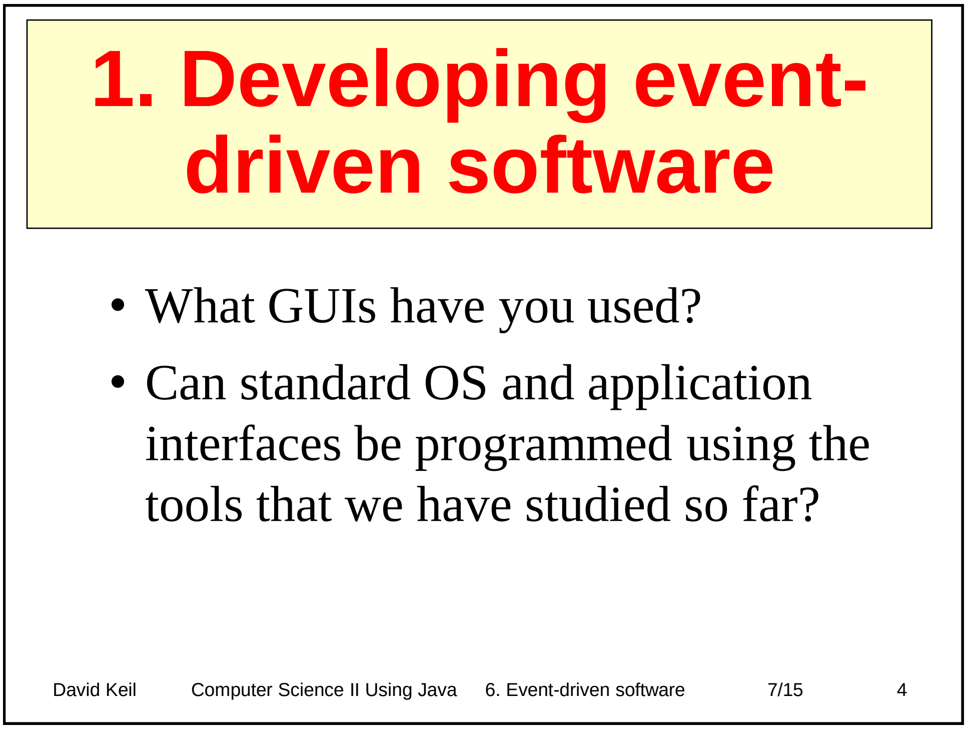
Inquiry

- What is a GUI?
- An *event*?
- How are *windows*, *buttons*, and *shapes* drawn in Java?
- How can you draw *lines* and *shapes* on the screen?



Topic objective

Explain event-driven GUI development and use Java graphics libraries



1. Developing event-driven software

- What GUIs have you used?
- Can standard OS and application interfaces be programmed using the tools that we have studied so far?

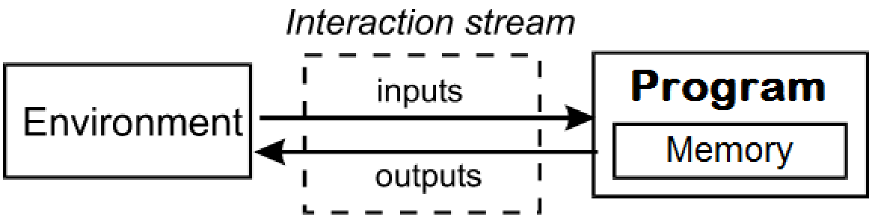
Subtopic objectives

6.1a Explain event-driven programming*

6.1b Write event-driven code†

Streams and interaction

- In an interaction stream, inputs and outputs alternate
- A user-controlled I/O loop may never end
- The diagram below shows an interactive computation



Interaction stream

Environment

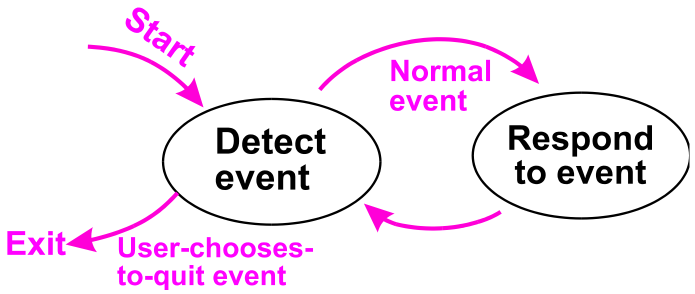
inputs

outputs

Program

Memory

Event-driven programming



- An event is normally the user’s input
- Examples of events: keypress, menu choice, mouse click

Event-driven design

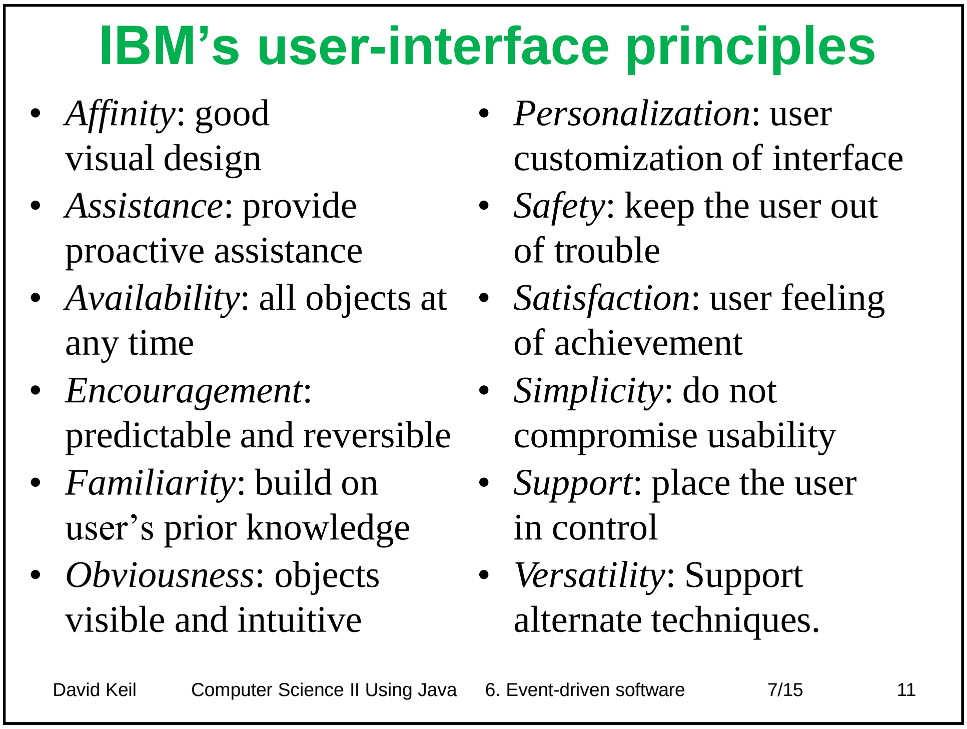
- *Browsers* and most other apps are *interactive*, alternating input and output
- *Command-line environments:* URL line in browser, Google prompt, DOS or UNIX prompt
- *Features of graphical user interfaces:* windows, icons, menus, dialog boxes, buttons
- *Common paradigm:* User generates *events*, e.g., clicks, drags, keystrokes, timeouts
- Browser interacts via *hyperlinks* and via embedding of event-handling JavaScript in HTML files

Events in a GUI

- Event-driven programs use *event handler* code to process events such as keypresses, mouse clicks, menu choices, or passage of time
- Events are represented by objects
- *Event-listener* objects select relevant events out of all events that are generated
- *Event-source* objects generate events selected by event listeners
- HTML and JavaScript are used together to respond to some events

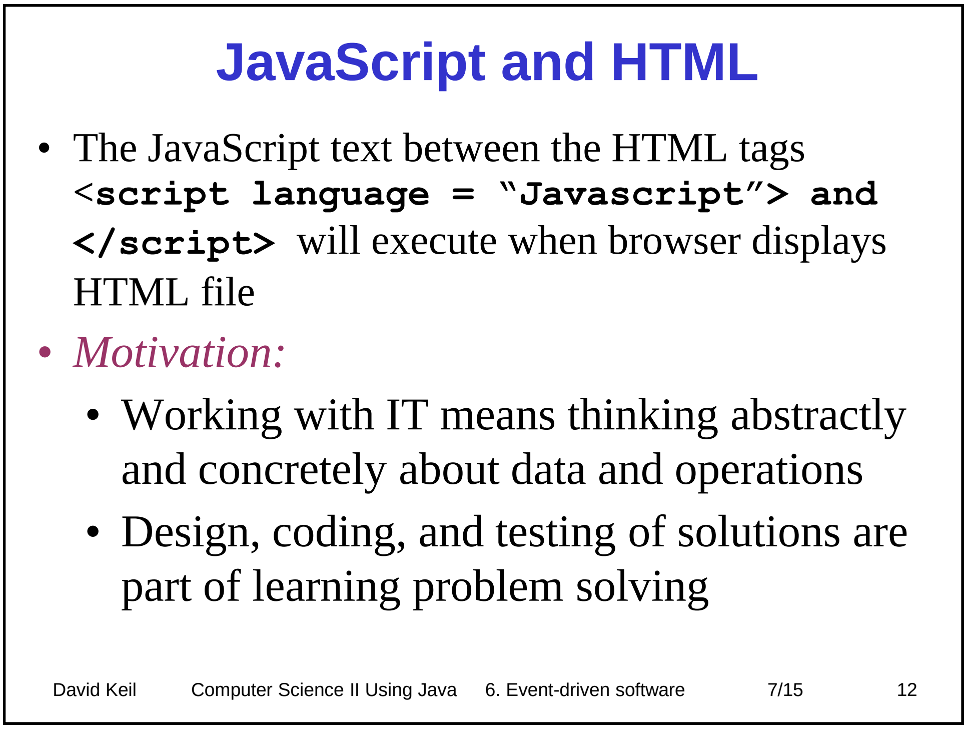
Problem specifications and user interfaces

- Designer must consider *assumptions* about
  - Problem domain (e.g., business, education, personal, healthcare)
  - User needs and expectations
- *Interface* refers to how application (e.g., at web site) appears and responds to user
- Most user interfaces today are *graphical*
- Implementation (coding) is partly independent of interface



IBM’s user-interface principles

- *Affinity*: good visual design
- *Assistance*: provide proactive assistance
- *Availability*: all objects at any time
- *Encouragement*: predictable and reversible
- *Familiarity*: build on user’s prior knowledge
- *Obviousness*: objects visible and intuitive
- *Personalization*: user customization of interface
- *Safety*: keep the user out of trouble
- *Satisfaction*: user feeling of achievement
- *Simplicity*: do not compromise usability
- *Support*: place the user in control
- *Versatility*: Support alternate techniques.



JavaScript and HTML

- The JavaScript text between the HTML tags **`<script language = “Javascript”> and </script>`** will execute when browser displays HTML file
- *Motivation:*
  - Working with IT means thinking abstractly and concretely about data and operations
  - Design, coding, and testing of solutions are part of learning problem solving

JavaScript example

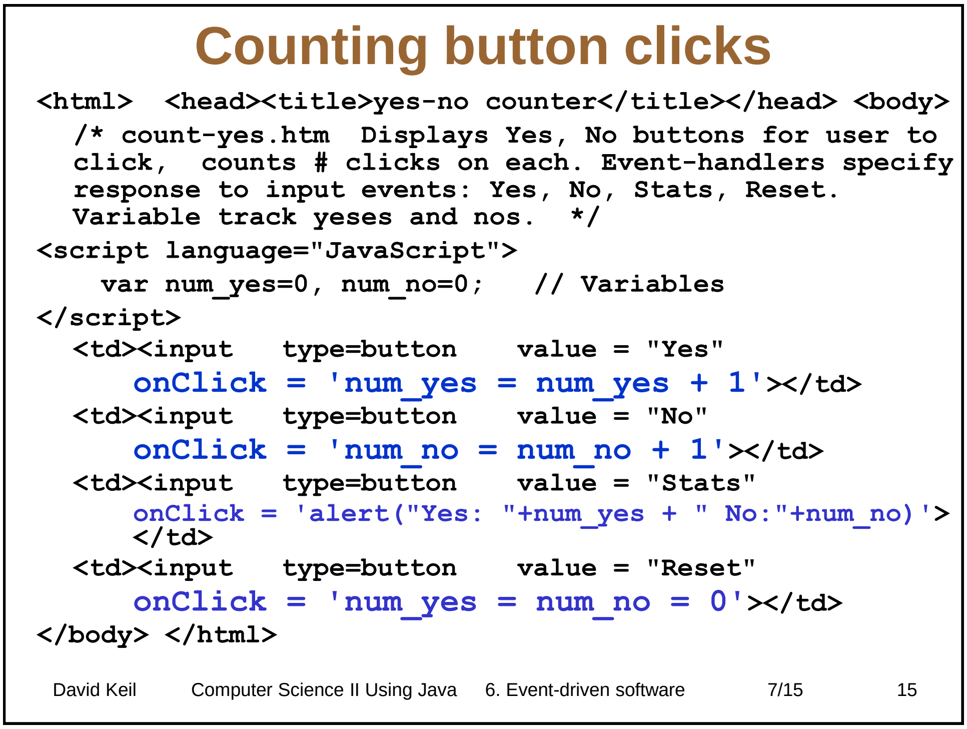
```
<html>   <!-- hello.htm -->
  <head><title>DK-Hello</title></head>
  <body>63.120 says Hello!
    <script language="JavaScript">
    alert("hello");
    </script>
  </body>
</html>
```

- Displays “hello” in an alert box (a kind of dialog)
- `alert` is a JavaScript function (a kind of procedure)
- JavaScript may be used after `script` tag

Button

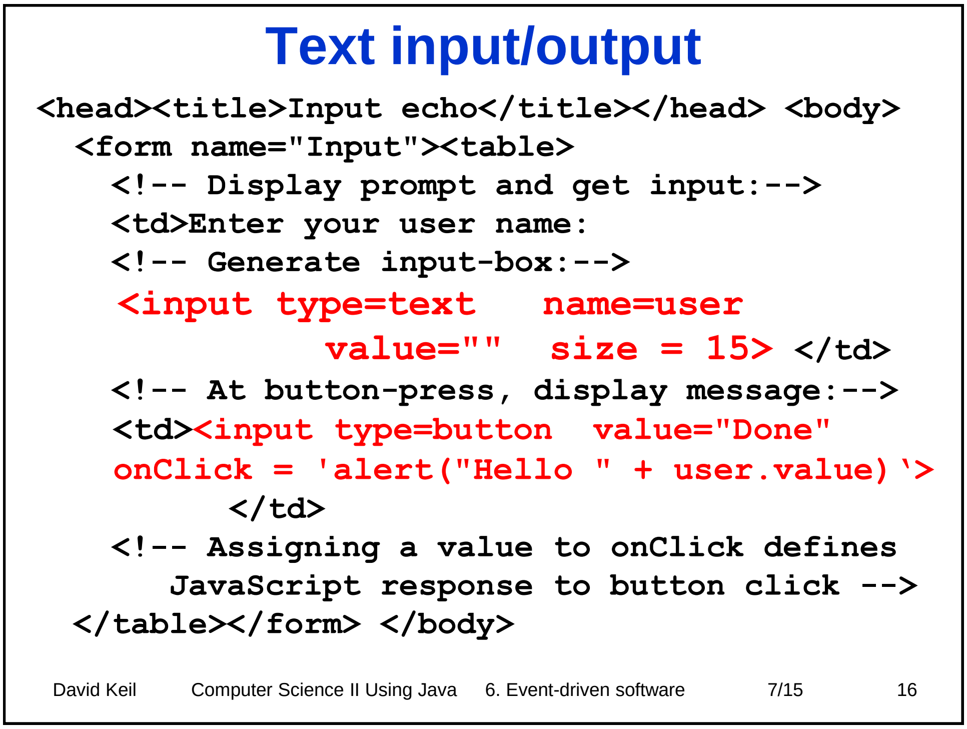
```
<html> <!--button.htm-->
<head><title>63120 Hello</title></head>
<body>
  <input type=button  value = "Hello"
         onClick = 'alert("Hi")'>
</body></html>
```

- This code displays “Hi” when “Hello” button is pressed
- `<input>` tag defines an input button object
- *Event handler:* code that specifies application’s response to a particular event, such as user click on a button



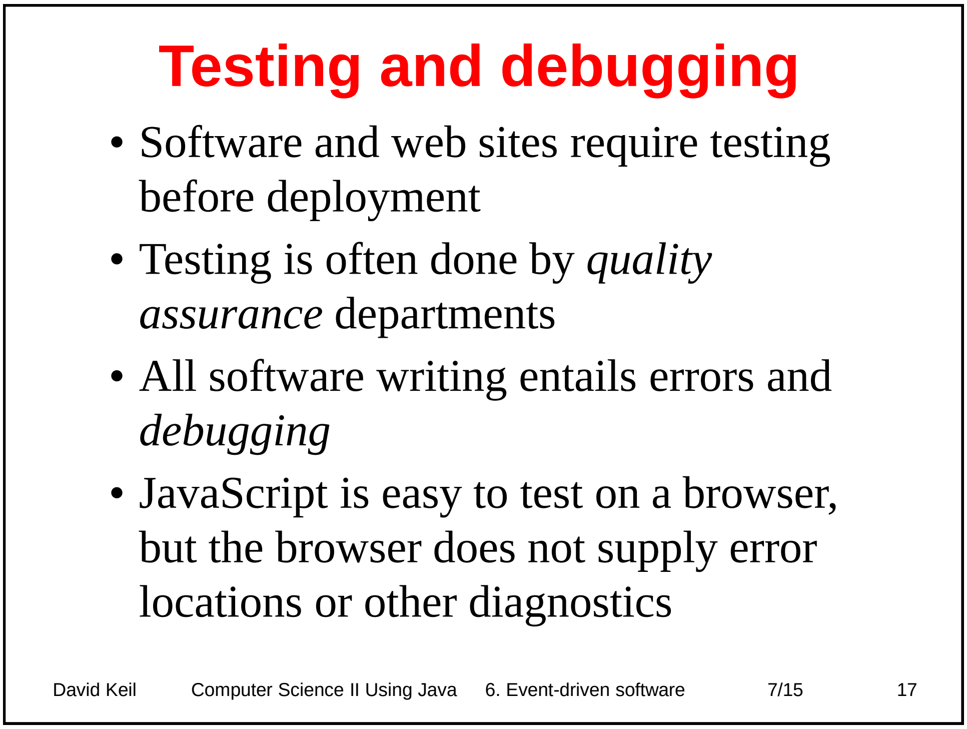
Counting button clicks

```
<html>  <head><title>yes-no counter</title></head> <body>
  /* count-yes.htm  Displays Yes, No buttons for user to
  click,  counts # clicks on each. Event-handlers specify
  response to input events: Yes, No, Stats, Reset.
  Variable track yeses and nos.  */
<script language="JavaScript">
    var num_yes=0, num_no=0;   // Variables
</script>
  <td><input   type=button    value = "Yes"
      onClick = 'num_yes = num_yes + 1'></td>
  <td><input   type=button    value = "No"
      onClick = 'num_no = num_no + 1'></td>
  <td><input   type=button    value = "Stats"
      onClick = 'alert("Yes: "+num_yes + " No:"+num_no)'>
      </td>
  <td><input   type=button    value = "Reset"
      onClick = 'num_yes = num_no = 0'></td>
</body> </html>
```



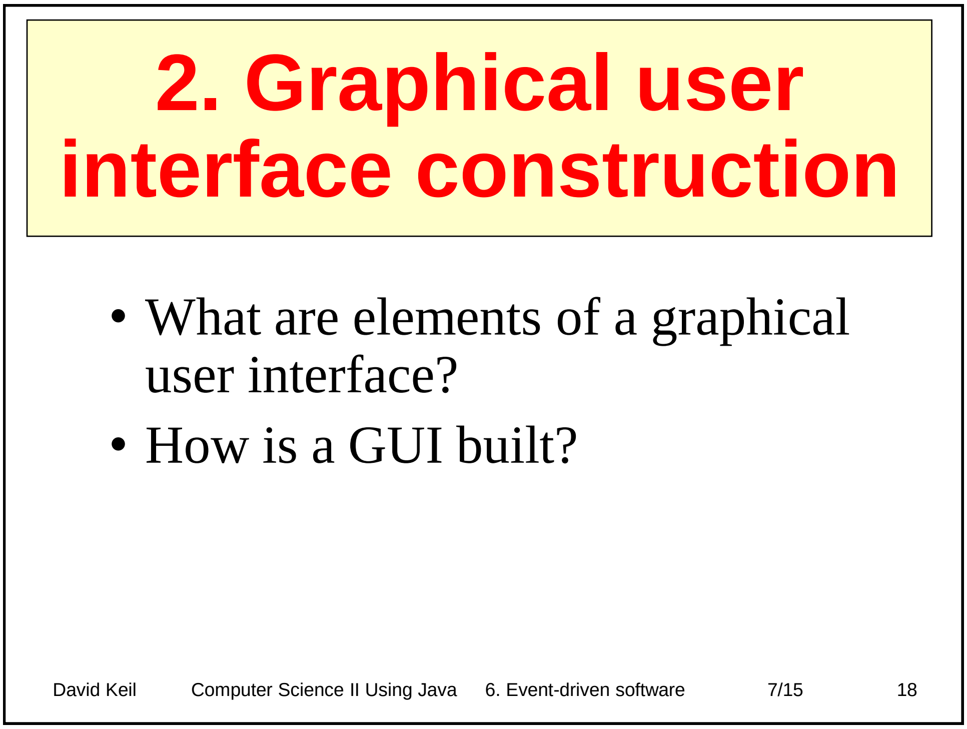
Text input/output

```
<head><title>Input echo</title></head> <body>
  <form name="Input"><table>
    <!-- Display prompt and get input:-->
    <td>Enter your user name:
    <!-- Generate input-box:-->
    <input type=text   name=user
               value=""  size = 15> </td>
    <!-- At button-press, display message:-->
    <td><input type=button  value="Done"
    onClick = 'alert("Hello " + user.value)'>
          </td>
    <!-- Assigning a value to onClick defines
       JavaScript response to button click -->
  </table></form> </body>
```



Testing and debugging

- Software and web sites require testing before deployment
- Testing is often done by *quality assurance* departments
- All software writing entails errors and *debugging*
- JavaScript is easy to test on a browser, but the browser does not supply error locations or other diagnostics



2. Graphical user interface construction

- What are elements of a graphical user interface?
- How is a GUI built?

Subtopic objectives

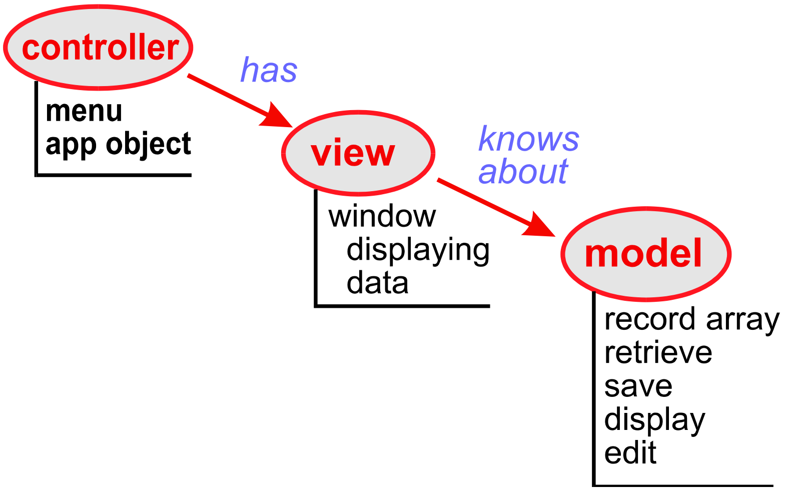
6.2a Describe elements of a graphical user interface*

6.2b Implement a Java-based GUI†

Model-view-controller architecture

Kind of class	**Example**
Model	Array of database records Spreadsheet cells in linked-list grid
View	Window Button
Controller	Menu Instance of application class

Model, view, and controller classes



controller

has

menu
app object

view

knows
about

window
displaying
data

model

record array
retrieve
save
display
edit

Commercial software interfaces

- Use collections for storage
- Store data in disk files and support updating of disk data
- Provide graphical user interfaces (GUIs) with menus for user control
- Extensive design process with large software development teams
- Extensive testing and validation
- Entail professional responsibilities

Application classes

- Used in Windows and Java programming
- A user-interface library defines a general-purpose application class
- Application programmer defines a class that *inherits* from library class, extends its features
- Application programmer may focus on special purpose of application rather than on user-interface details

Buttons and labels

- First declare button objects and labels (to identify the buttons):
  ```
  JButton button = new Jbutton(”Hi”);
  JLabel label = new JLabel(”Hi ”+x++);
  ```
- Then create panel containing buttons
- Then define class implementing *ActiveListener*
- Listener responds to button press
- See Background slides for how JavaScript supports GUI objects in web pages

String I/O using GUI

- To show output string *x* in a message dialog:

```
JOptionPane.showMessageDialog
   (null, x);
```

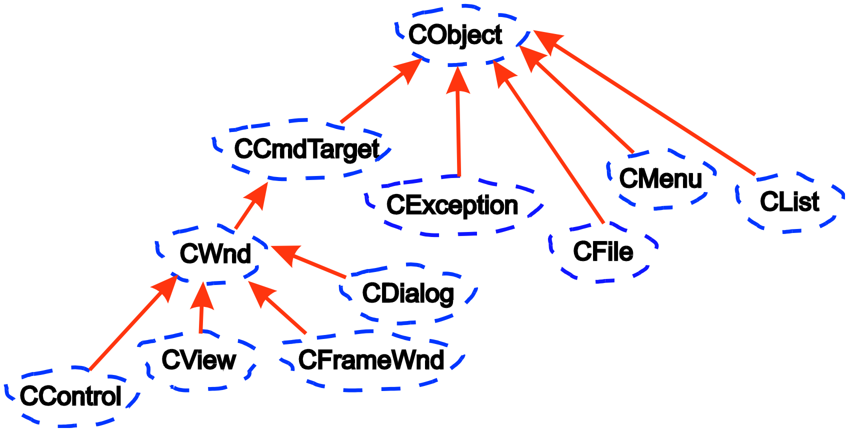
- To get input string *s* using dialog:

```
String s =
  JOptionPane.showInputDialog
      (”Enter name:”);
```

Window classes

- A window is a rectangular *view* that is displayed graphically
- Windows are used to display data, including part of a document, graphic, or file
- To enable user to interactively manage what is displayed, controls such as scrollers are part of window objects

Microsoft Foundation Classes inheritance hierarchy (partial)



CObject

CCmdTarget

CException

CMenu

CList

CFile

CWnd

CDialog

CView

CFrameWnd

CControl

3. Java graphical tools

- What shapes can you draw with the MS Office drawing tools?
- With *Paint*?
- What is required to specify a line segment on a coordinate axis? A circle? A rectangle?
- Have you programmed with graphics?

Subtopic objectives

6.3a Describe Java graphics tools

6.3b Write a graphics application†

Bitmap and vector graphics

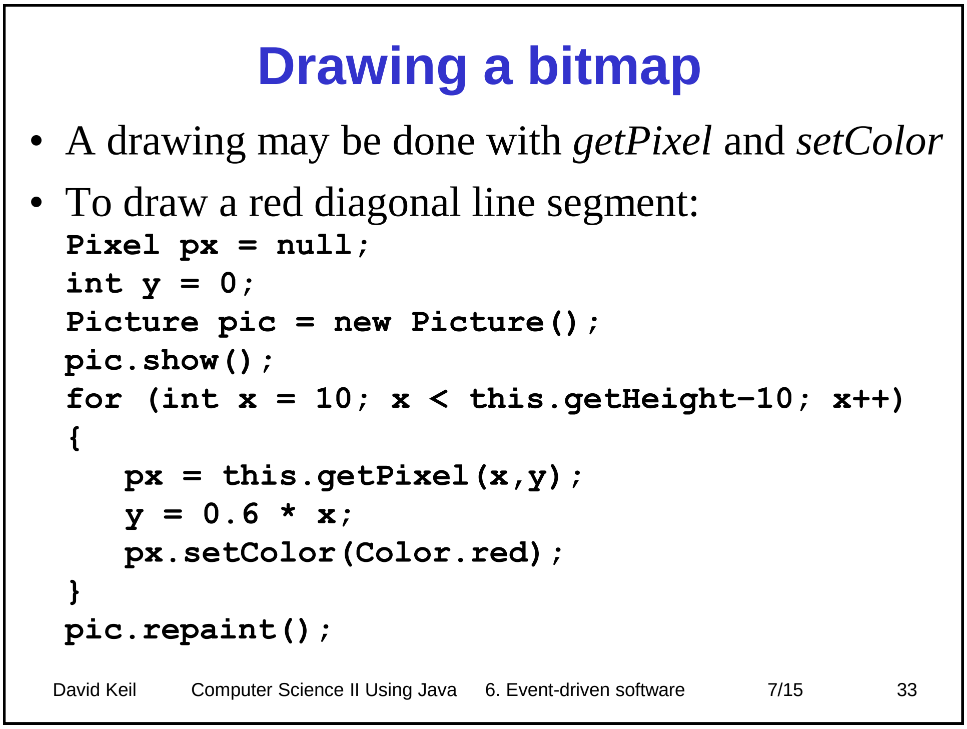
- A drawing may be rendered as a *bitmap*, pixel by pixel, or as instructions (*vector graphics*)
- Bitmap file formats: *TIFF, BMP, PNG*
- Java graphics packages:
  - *java.awt*: Active Windowing Toolkit
  - *java.io*: input/output classes
- Graphics classes: *Font, Graphics, Picture, graphicsEnvironment, Color, Graphics2D, Line2D, FontMetrics, Pixel*

Java graphics tools

- Packages: *javax.swing*, *java.awt*
- To create graphics window, declare *JFrame* object (example: *Emptyframeviewer.java*)
- To display an object, declare an instance (example: *Rectangle*) and call *draw* method
- *Alternative:* declare class to *inherit* from *JComponent*, add to frame, and call *paint* method

Frames in Java

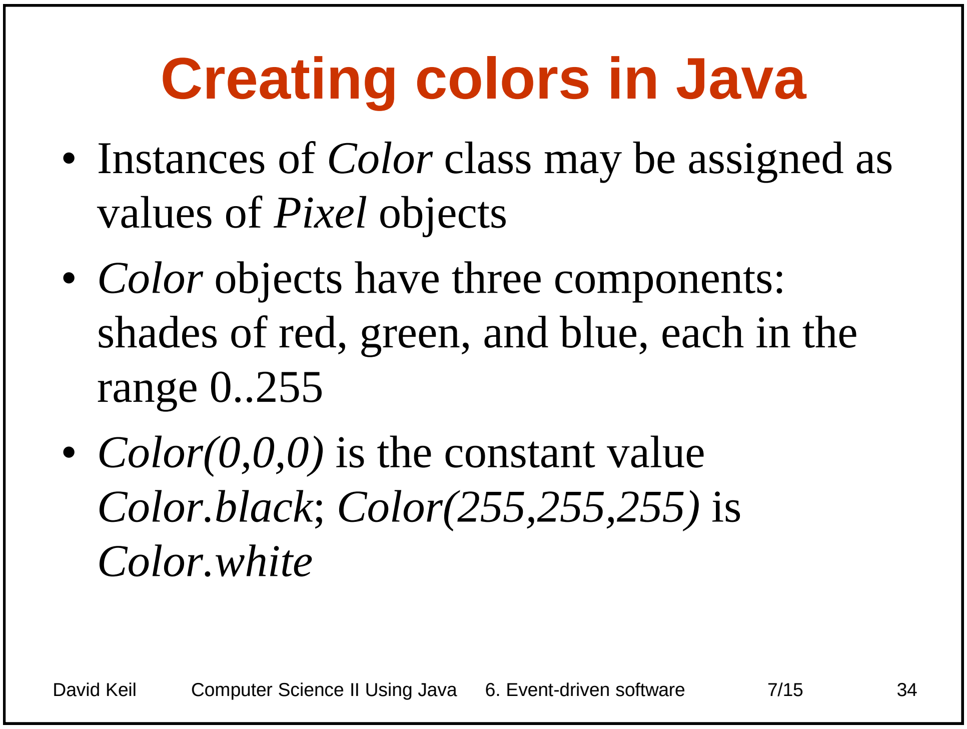
- Specialized frame classes extend predefined *JFrame* (from *Swing* package)
- Specialized event listeners implement *ActionListener* interface
- *JTextField* components support user text input, labeled by *JLabel* objects
- *JScrollpane* objects may contain multi-line *JTextArea* objects



Drawing a bitmap

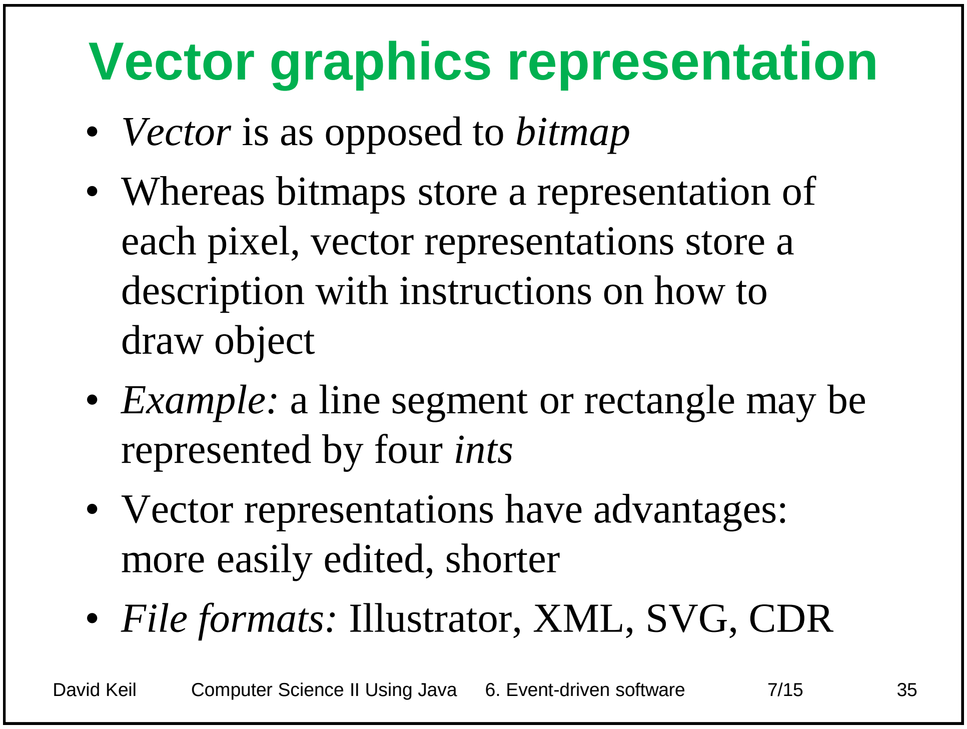
- A drawing may be done with *getPixel* and *setColor*
- To draw a red diagonal line segment:

```
Pixel px = null;
int y = 0;
Picture pic = new Picture();
pic.show();
for (int x = 10; x < this.getHeight-10; x++)
{
   px = this.getPixel(x,y);
   y = 0.6 * x;
   px.setColor(Color.red);
}
pic.repaint();
```



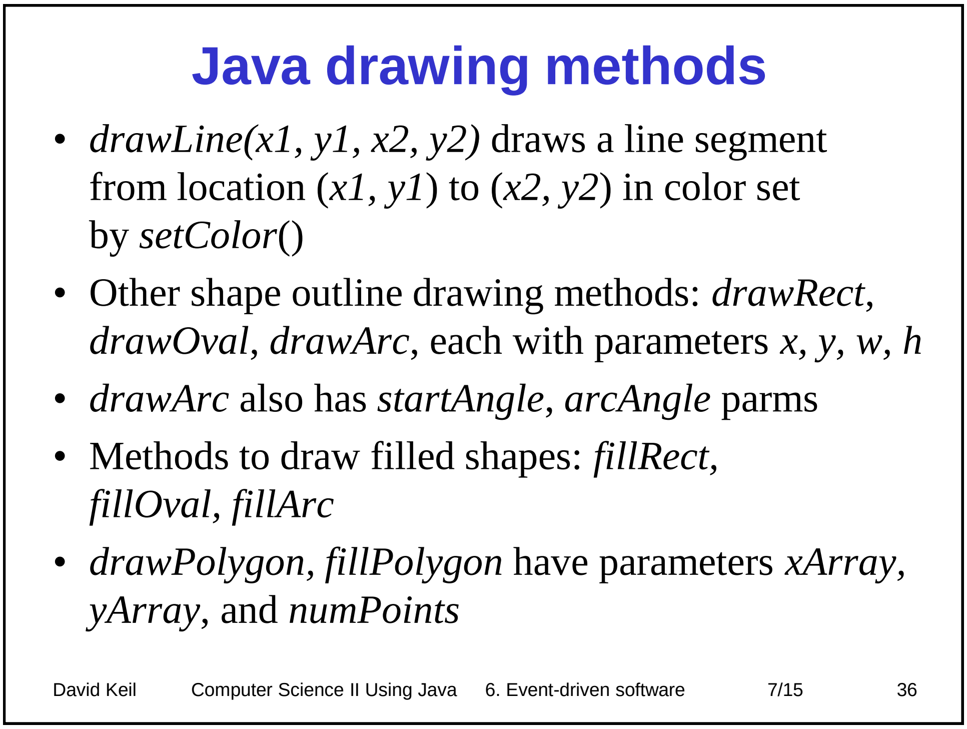
Creating colors in Java

- Instances of *Color* class may be assigned as values of *Pixel* objects
- *Color* objects have three components: shades of red, green, and blue, each in the range 0..255
- *Color(0,0,0)* is the constant value *Color.black*; *Color(255,255,255)* is *Color.white*



Vector graphics representation

- *Vector* is as opposed to *bitmap*
- Whereas bitmaps store a representation of each pixel, vector representations store a description with instructions on how to draw object
- *Example:* a line segment or rectangle may be represented by four *ints*
- Vector representations have advantages: more easily edited, shorter
- *File formats:* Illustrator, XML, SVG, CDR



Java drawing methods

- *drawLine(x1, y1, x2, y2)* draws a line segment from location (*x1*, *y1*) to (*x2*, *y2*) in color set by *setColor*()
- Other shape outline drawing methods: *drawRect*, *drawOval*, *drawArc*, each with parameters *x*, *y*, *w*, *h*
- *drawArc* also has *startAngle*, *arcAngle* parms
- Methods to draw filled shapes: *fillRect*, *fillOval*, *fillArc*
- *drawPolygon*, *fillPolygon* have parameters *xArray*, *yArray*, and *numPoints*

The *java.awt.Graphics2D* class

- A class derived from *Graphics*
- Features not possessed by *Graphics* methods:
  - Each shape is an object
  - Set brush width
  - Enable broken lines
  - Rotate, translate, scale, shear
  - Gradient or textured fill
  - Control of effect of overlapping
  - Clipping
  - Curve smoothing

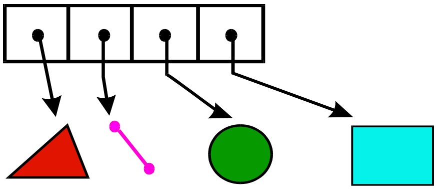
Drawing text

- *Method* (from *java.awt.Graphics*): *drawString(String s, int x, int y)*, where *x*, *y* specify leftmost point and vertical baseline of string
- Font and color are as previously set by *setFont*, *setColor*
- *Font* class has *name*, *style*, *size* attributes

A collection of graphical objects

- *Concept:* an expandable collection of descriptions of shapes or other objects, of different classes, each of which calls a *draw* method
- *Java features used:* inheritance, polymorphism
- *Implementation:*
  - Linked list of references to objects of base class, e.g., *Shape*
  - Each object is of a derived class, e.g. *Rectangle, Triangle, Arrow, Oval*
  - Draw is a *virtual method* defined only in the base-class definitions

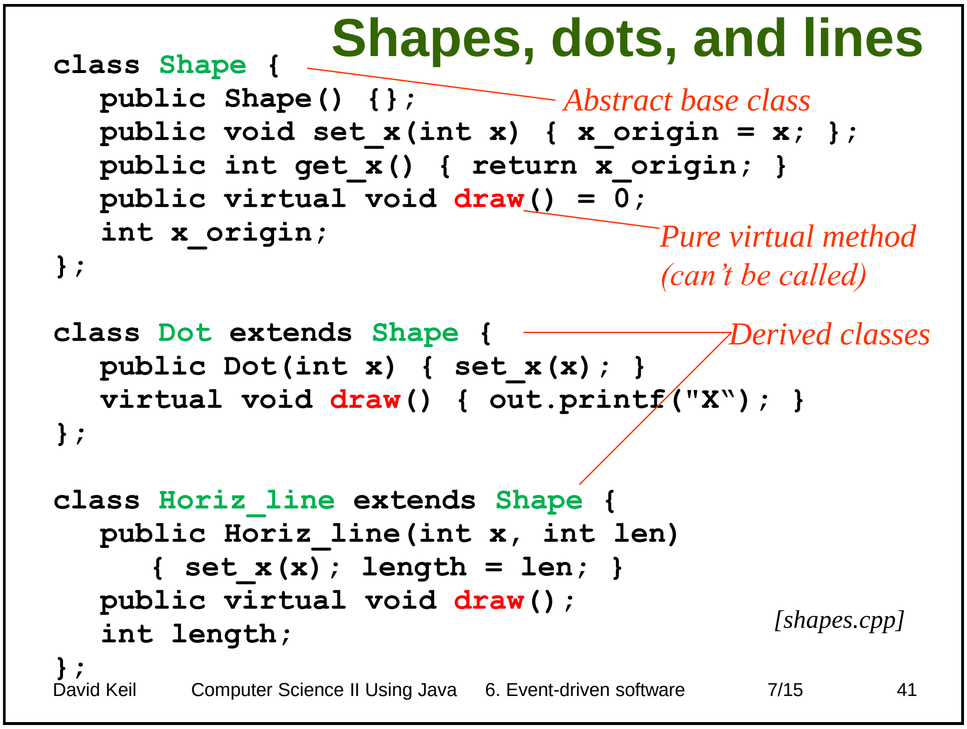
A collection of *Shape* objects of mixed classes



- Array of pointers to base class:

```
Shape[] item = new Shape[];
```

- Each pointer points to a dynamically allocated object of a derived class
- All derived classes are derived from *Shape*



Shapes, dots, and lines

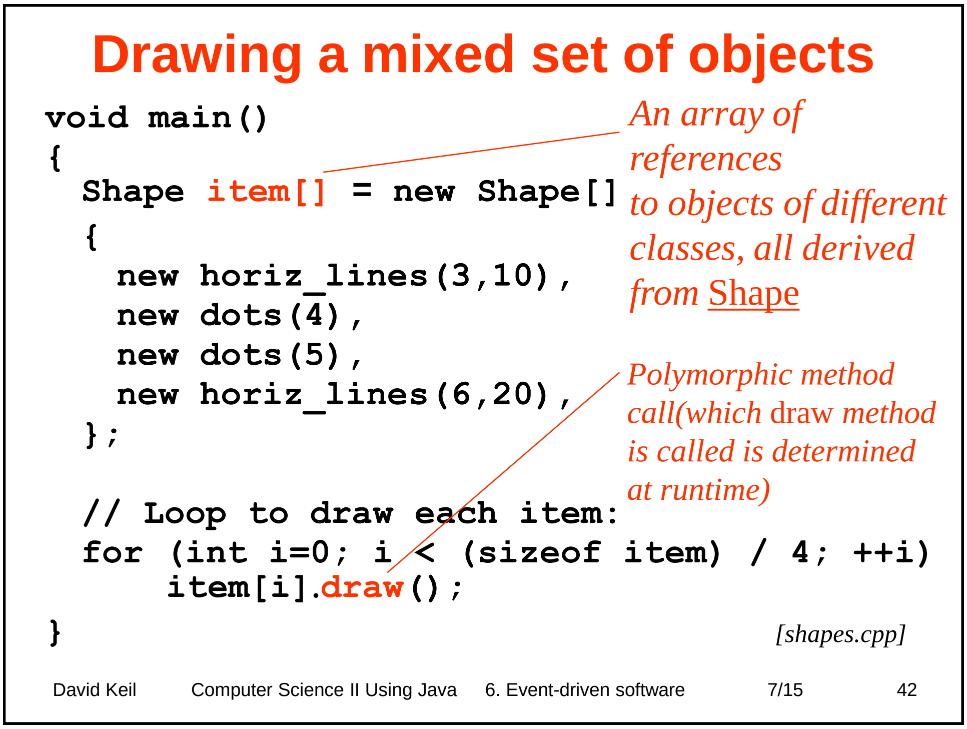
```
class Shape {
   public Shape() {};
   public void set_x(int x) { x_origin = x; };
   public int get_x() { return x_origin; }
   public virtual void draw() = 0;
   int x_origin;
};

class Dot extends Shape {
   public Dot(int x) { set_x(x); }
   virtual void draw() { out.printf("X“); }
};

class Horiz_line extends Shape {
   public Horiz_line(int x, int len)
      { set_x(x); length = len; }
   public virtual void draw();
   int length;
};
```

Abstract base class

Pure virtual method (can't be called)

Derived classes

[shapes.cpp]



Drawing a mixed set of objects

```
void main()
{
  Shape item[] = new Shape[]
  {
    new horiz_lines(3,10),
    new dots(4),
    new dots(5),
    new horiz_lines(6,20),
  };

  // Loop to draw each item:
  for (int i=0; i < (sizeof item) / 4; ++i)
       item[i].draw();
}
```

An array of references to objects of different classes, all derived from Shape

Polymorphic method call(which draw method is called is determined at runtime)

[shapes.cpp]

Abstract base classes (C++)

- An *abstract base class* is intended for use with polymorphism
- It cannot be instantiated
- A virtual member function with a null body makes a class abstract:

```
virtual void draw() = 0;
```

- Such a function is called a *pure virtual function*
- No-instantiation rule safety-protects abstract base classes (e.g., how is an undefined shape drawn?)

A C++ base-class *run* function

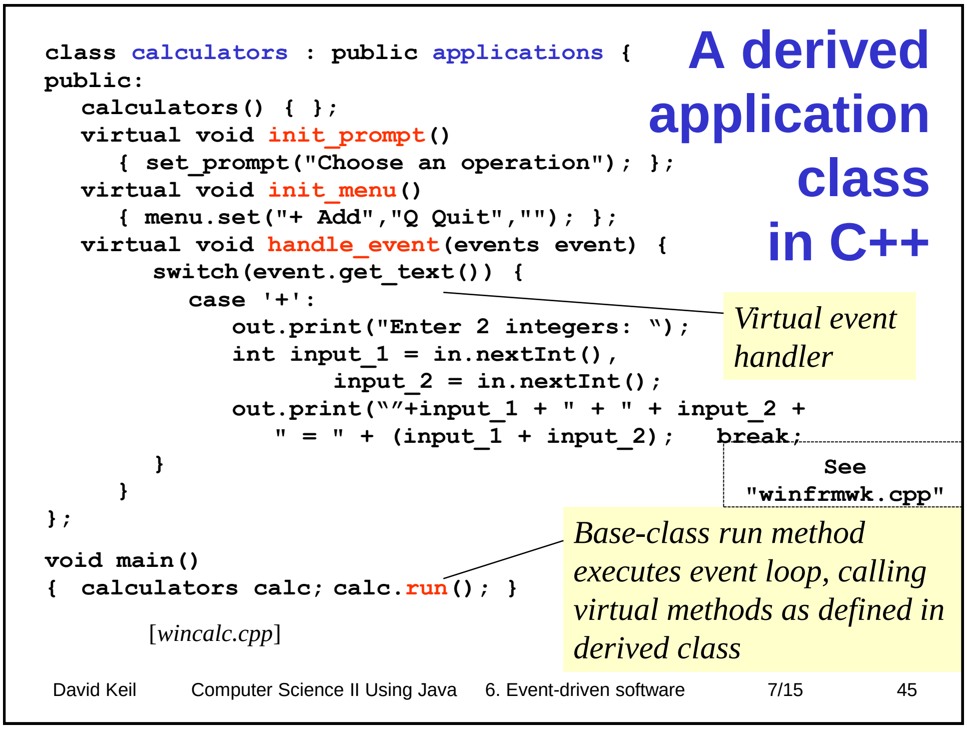
[*winfrmwk.cpp*]

```
void applications::run()
// Executes event loop, terminates on "Q" for Quit.
{
  init_prompt();
  init_menu();
  events event;
  char event_text;
  do
  {
     display_prompt();
     menu.draw();
     event.fetch();
     event_text = toupper(event.get_text());
     if (event_text != 'Q')
            handle_event(event);
  }
  while (event_text != 'Q');
}
```

Virtual functions defined in derived class

Virtual event handler gives control of response to derived class in app

Base class defines response to Quit input or other event defined by interface



A derived application class in C++

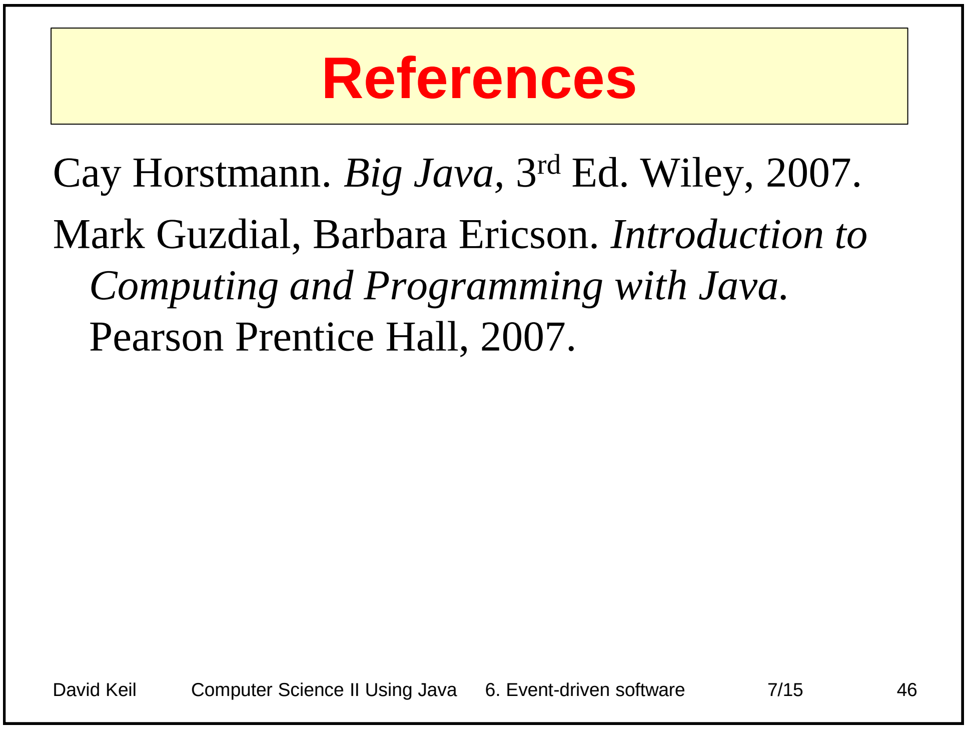
```
class calculators : public applications {
public:
   calculators() { };
   virtual void init_prompt()
     { set_prompt("Choose an operation"); };
   virtual void init_menu()
     { menu.set("+ Add","Q Quit",""); };
   virtual void handle_event(events event) {
        switch(event.get_text()) {
           case '+':
              out.print("Enter 2 integers: “);
              int input_1 = in.nextInt(),
                     input_2 = in.nextInt();
              out.print(“”+input_1 + " + " + input_2 +
                 " = " + (input_1 + input_2);   break;
        }
     }
};

void main()
{  calculators calc; calc.run(); }
```

Virtual event handler

See "winfrmwk.cpp"

Base-class run method executes event loop, calling virtual methods as defined in derived class

[*wincalc.cpp*]



References

Cay Horstmann. *Big Java*, 3rd Ed. Wiley, 2007.

Mark Guzdial, Barbara Ericson. *Introduction to Computing and Programming with Java.* Pearson Prentice Hall, 2007.