JAV
Spring 2014

Lecture 9 & 10

GUI programming (3)

Advanced issues

Lecture outline

- GUI application development
- Multi-tasking GUI
- Dialog
- Scroll bar: JScrollBar

Lecture 9

- Tabular display: JTable
- GUI tool kit: Font, Color
- Custom GUI using drawing

Lecture 10

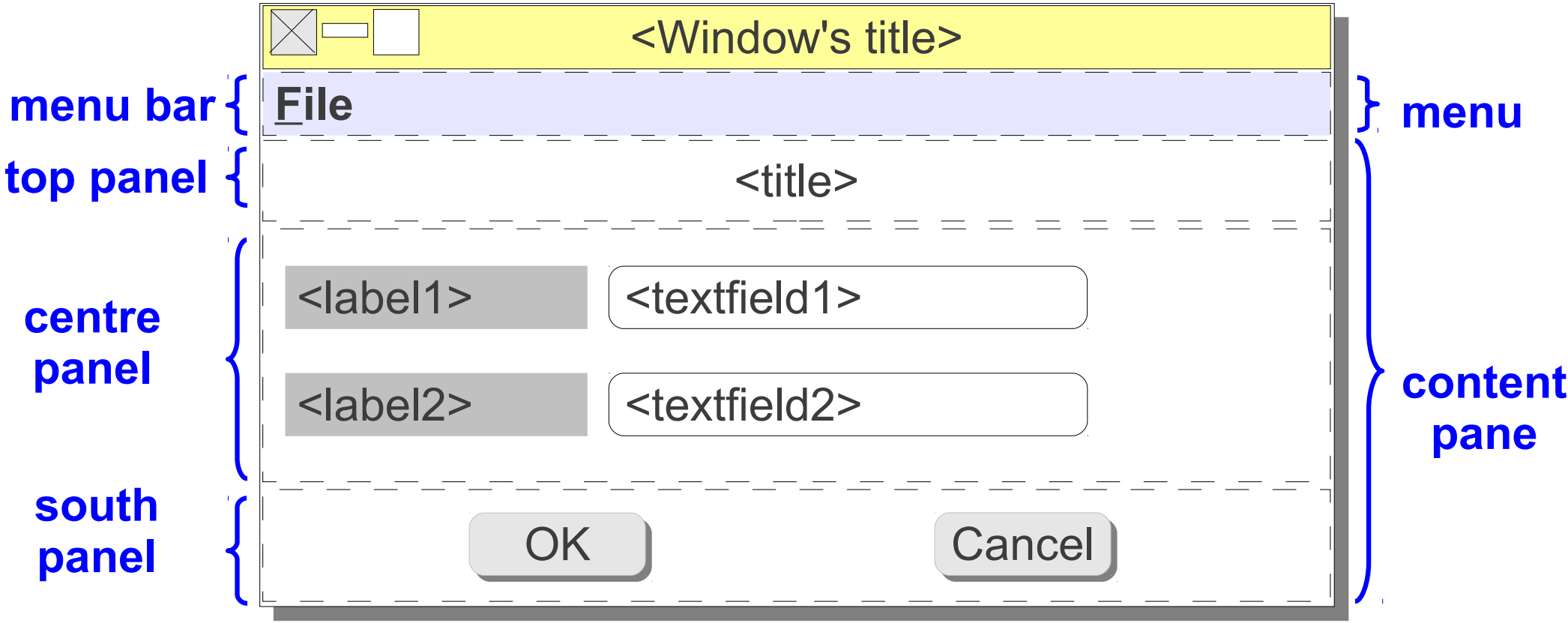
GUI application development

- Design
- Implementation

Design

- **Model**:
  - Create domain-specific classes (e.g. Customer)
- **View**:
  - Create window and display components
- **Controller**:
  - Define event handlers (user interaction)
  - Start up: initialise view & model
  - Display the view

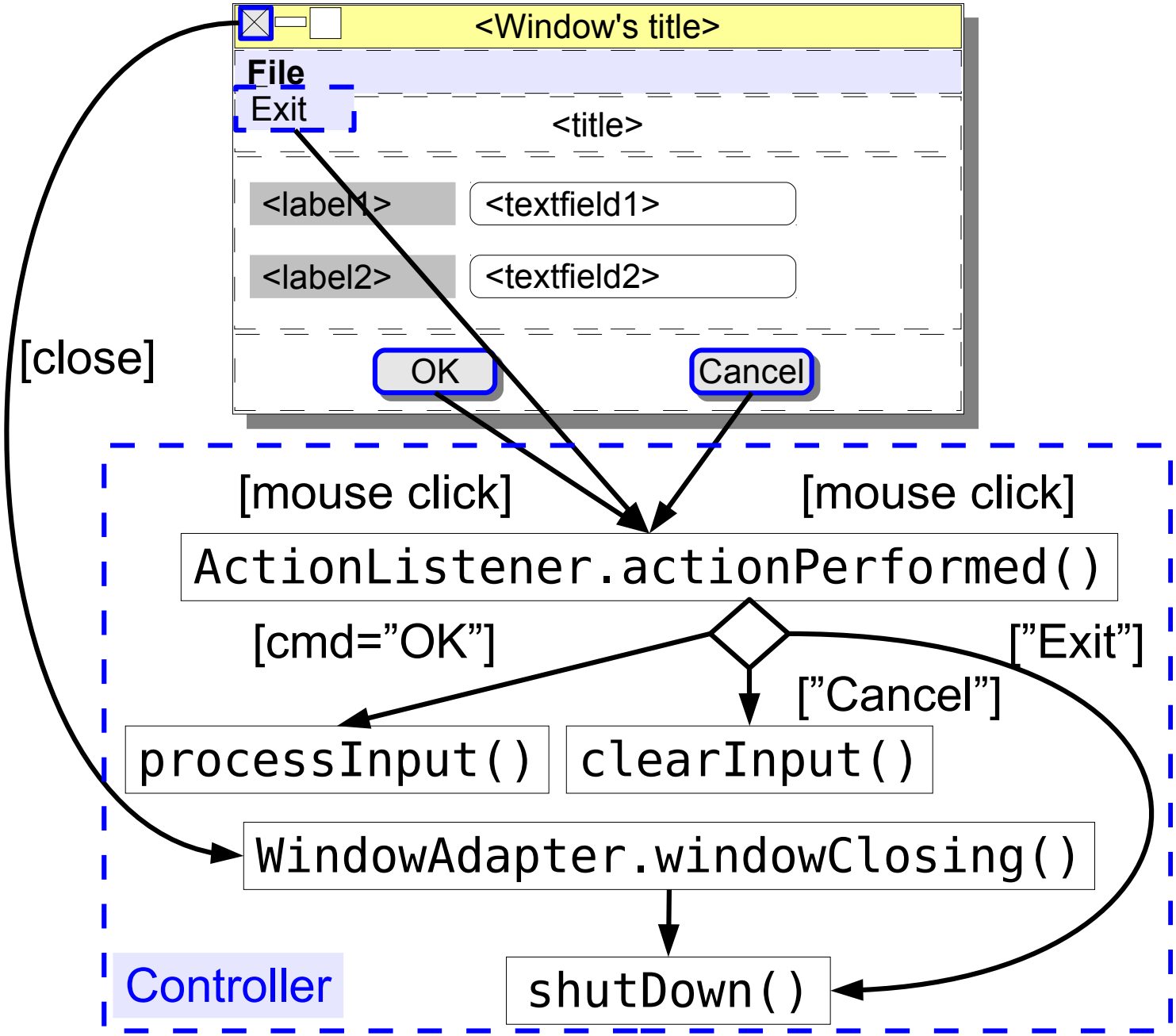
View design


<Window's title>
menu bar
File
menu
top panel
<title>
centre panel
<label1>
<textfield1>
<label2>
<textfield2>
content pane
south panel
OK
Cancel

Window and components

- Window (top-level container):
  - `JFrame`: border layout and 3 child panels
  - Menu bar: a File menu with an Exit item
- Child containers and components:
  - `JPanel` (3): north, south, centre
  - north: a title, flow layout (align: centre)
  - south: 2 buttons, flow layout (align: centre)
  - centre: labels and text fields, grid layout

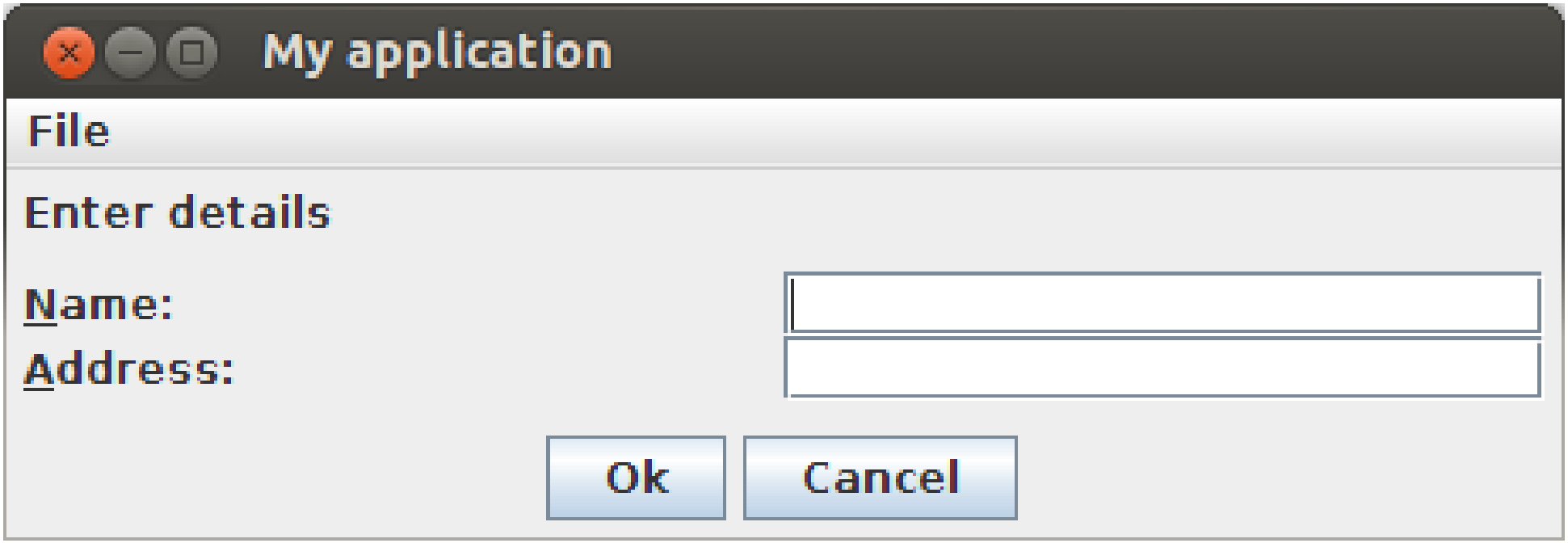
Controller design



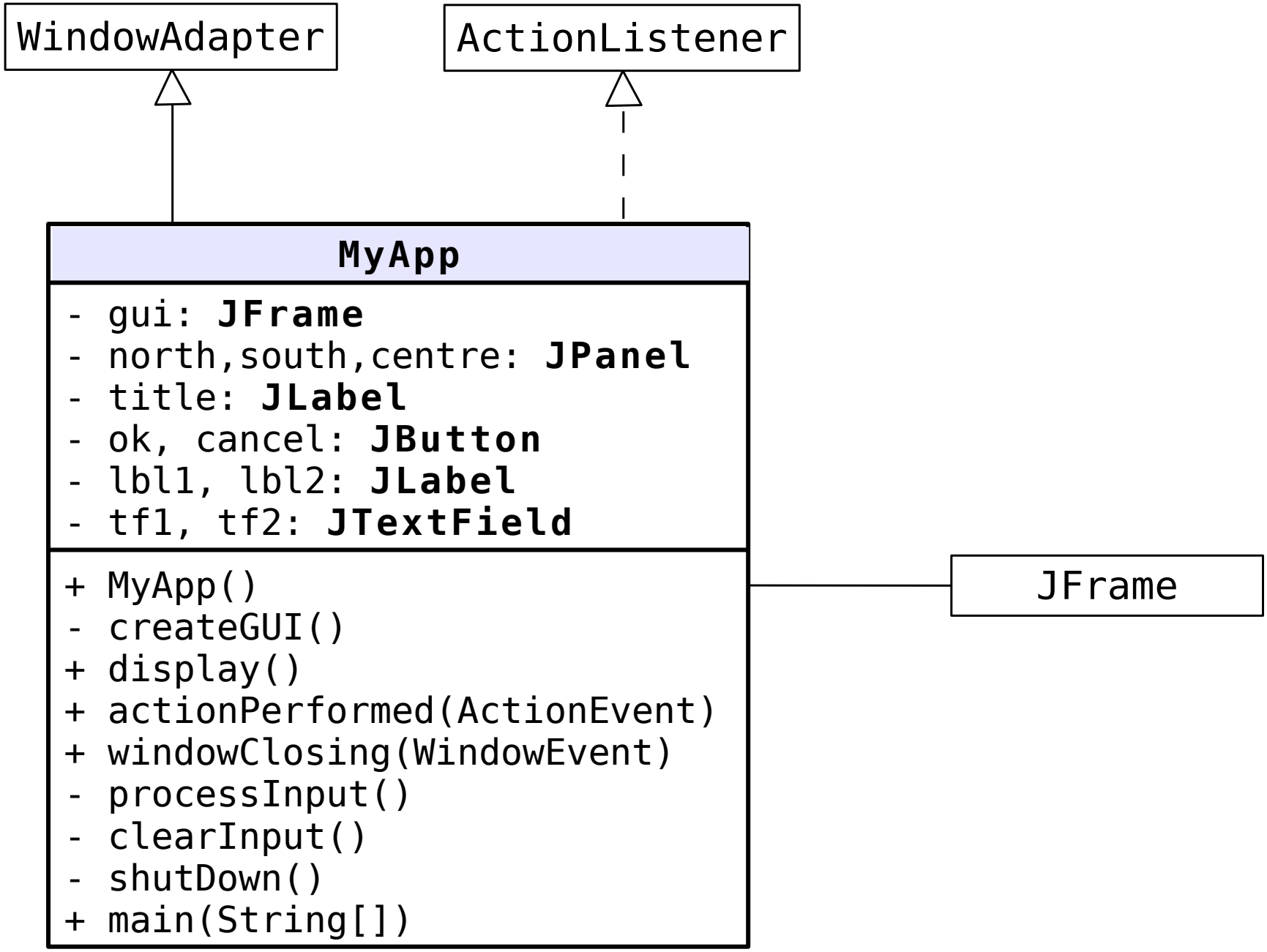
Design #1: all-in-one

- Model, View and Controller are combined into one class
- Used for small applications:
  - **model**: primitive data values
  - **view**: simple interface
  - **controller**: simple user actions
- **Pros**: less code to write
- **Cons**: tightly coupled → more difficult to maintain (e.g. when view specification is changed)

Example: MyApp



MyApp design #1



Implementation

- Basic GUI development tasks, and
- Container-related tasks:
  - set up the window: layout, menu
  - create & set up the container objects
  - add display components to the containers
  - add the containers to the window

DEMO

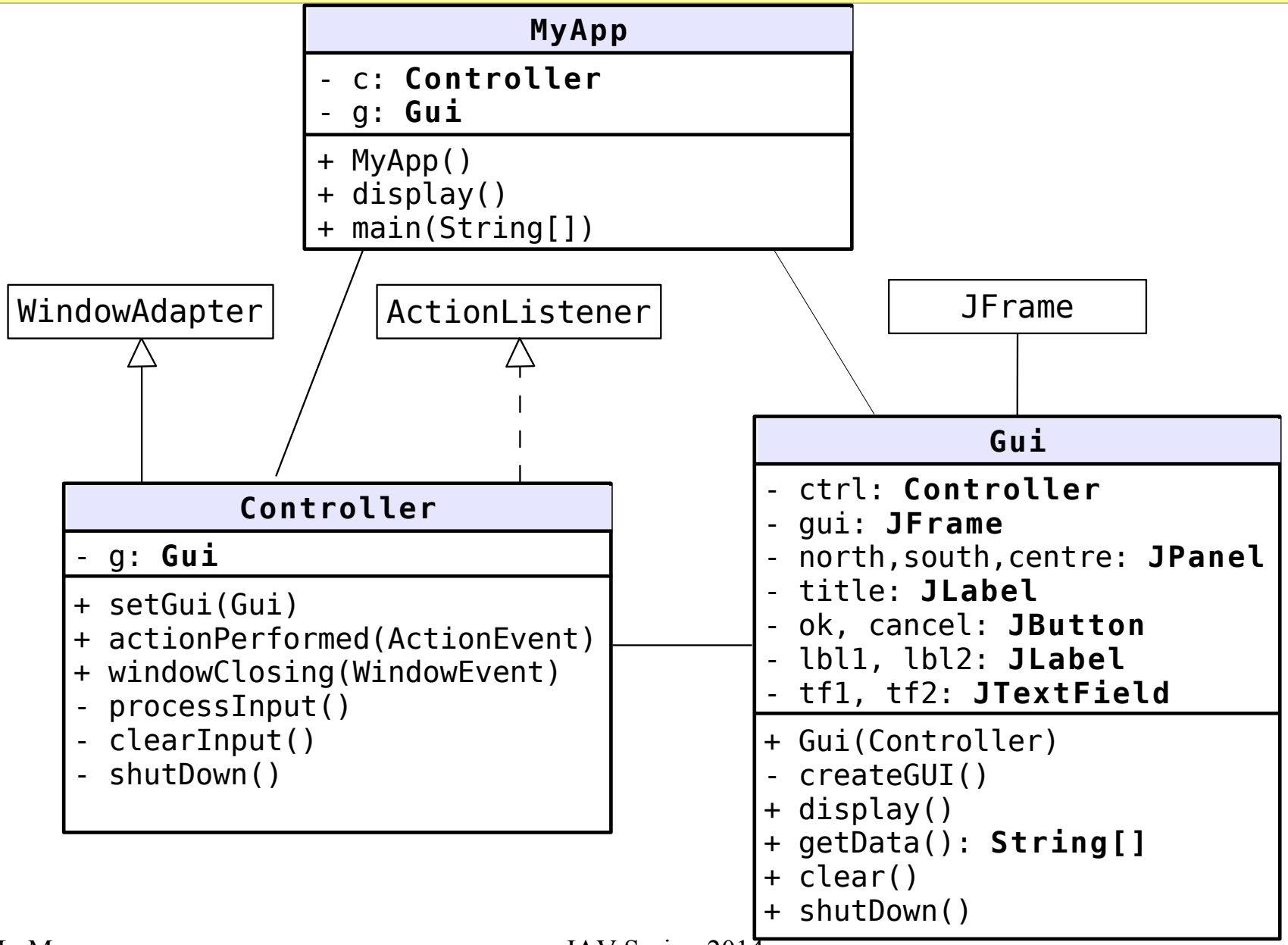
Example: MyApp #1

```
gui.app.allinone.MyApp
```

Design #2: independent controller

- Model, View may be combined into one class
- Controller is a separate class
- Used for medium-large applications:
  - **model**: domain-specific classes (e.g. Customer, Order, etc) that <u>may not</u> require separate classes
  - **view**: simple view, specific to each domain class
  - **controller**: data handling is likely to change
- **Pros**: easier to maintain (e.g. when data handling logics or view specifications are changed)
- **Cons**: more complex to design and code

Example: MyApp design #2



DEMO

MyApp #2

```
gui.app.independent.MyApp
```

Multi-tasking GUI

- A multi-tasking GUI application can handle multiple events at the same time
- Examples:
  - store program data to a database
  - view a report
  - print data

Multi-tasking in Swing

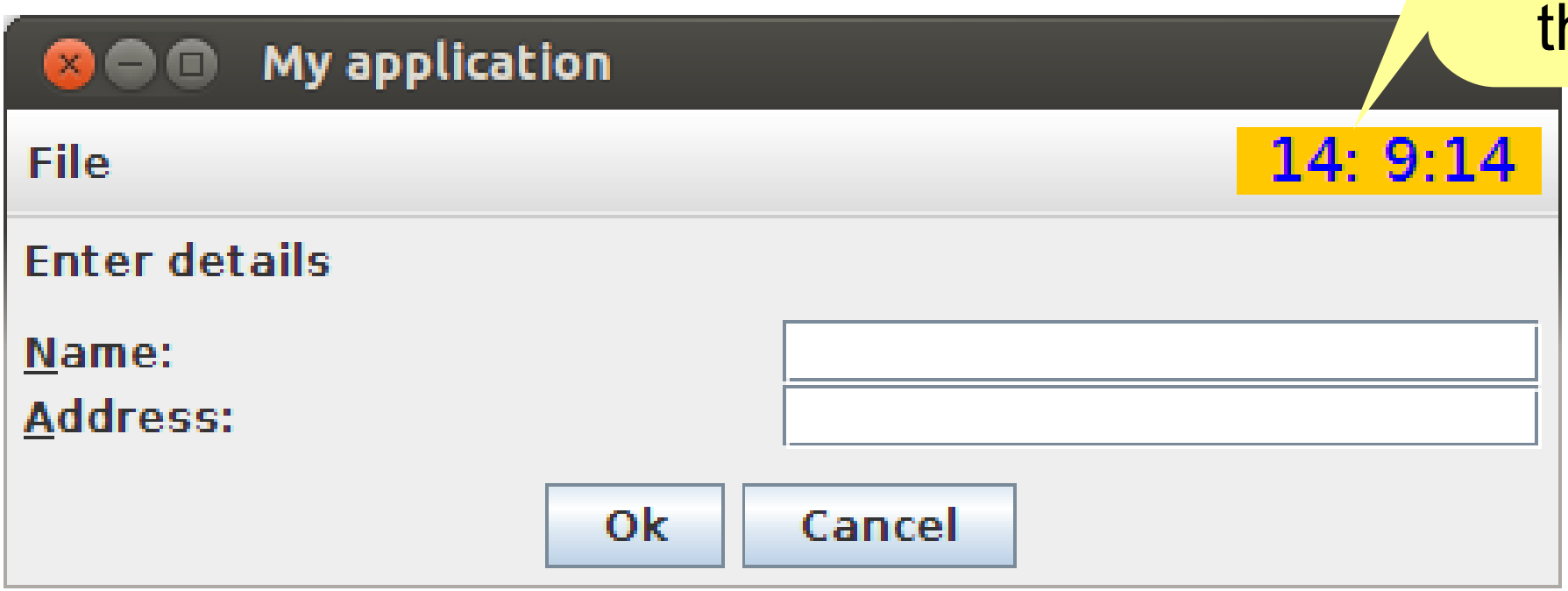
- Wrap the task in a `Runnable` object
- Start the task object using a `Thread` object
- Task thread is run concurrently with the GUI's thread:
  - user interaction is not blocked

DEMO

Multi tasking

```
gui.app.multitask.MyApp
```

Timer task running on a separate thread



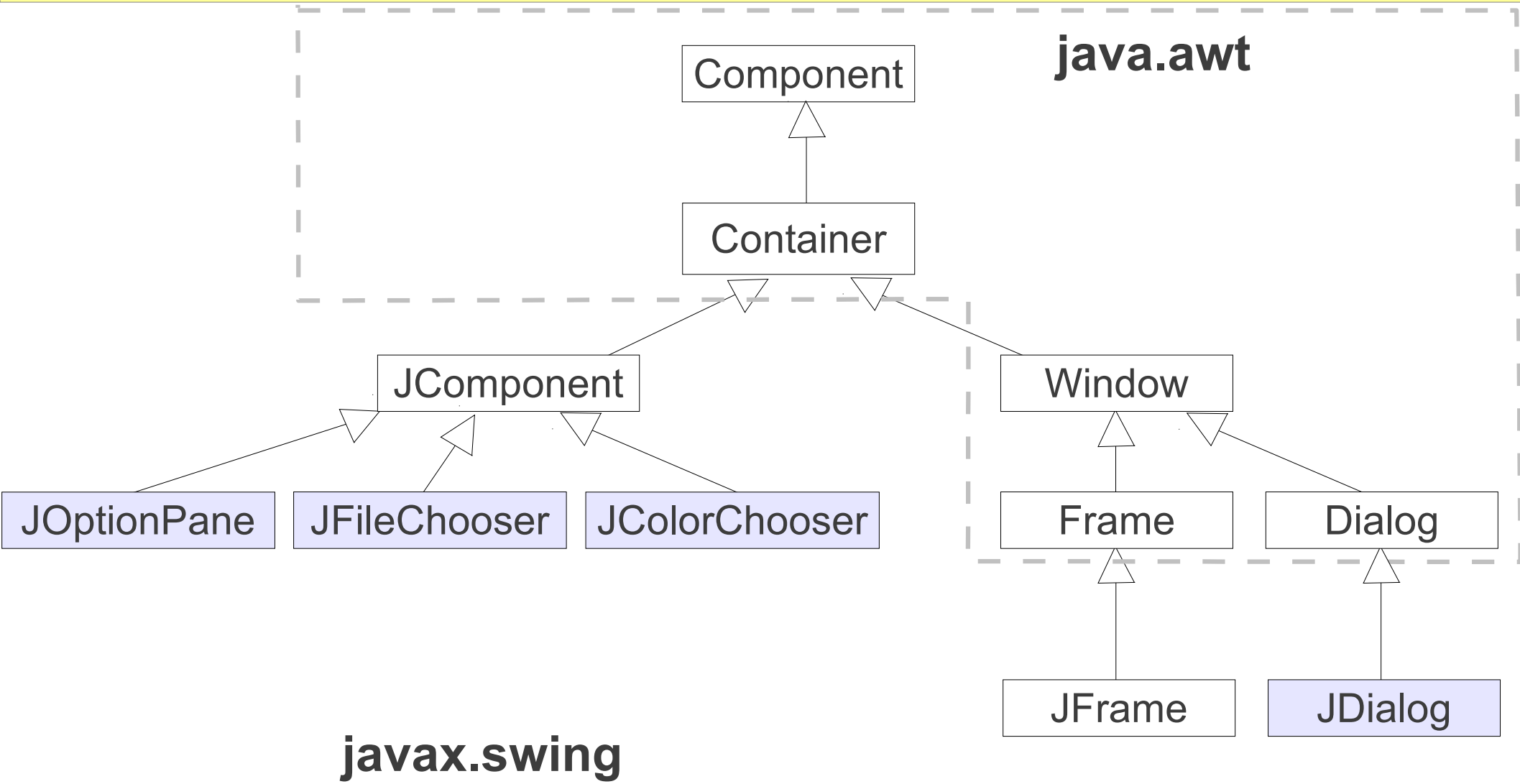
Dialog

- Separate sub-window that:
  - displays temporary notice or
  - obtains basic, context-dependent input
- Examples:
  - program message (informational, error)
  - progress status
  - browse a file or choose a colour
- Attached to a window (its parent)
- Can be modal or non-modal

Swing dialogs

- `JOptionPane`: simple, standard dialog
- [0] `JFileChooser`: browse a file
- [0] `JColorChooser`: choose a color
- [0!] `JDialog`: custom dialog

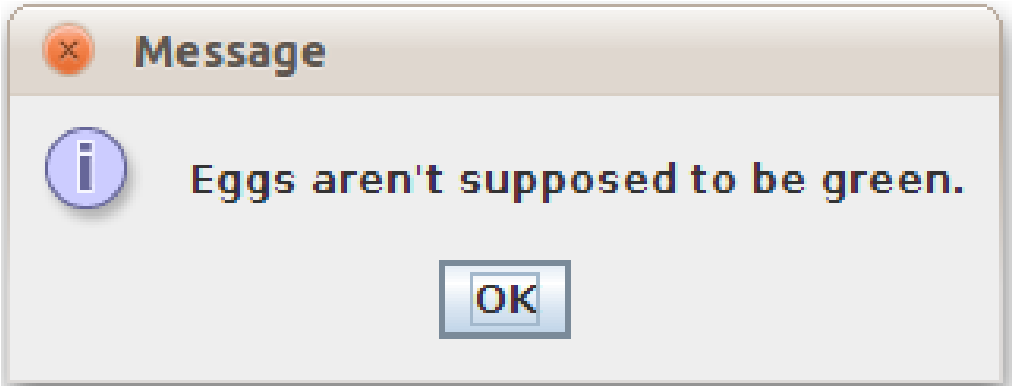
Dialog component hierarchy

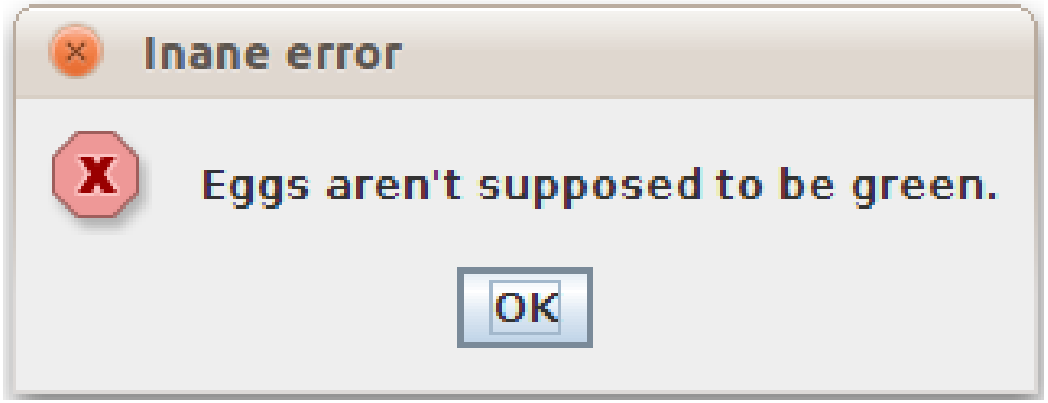


Class `JOptionPane`

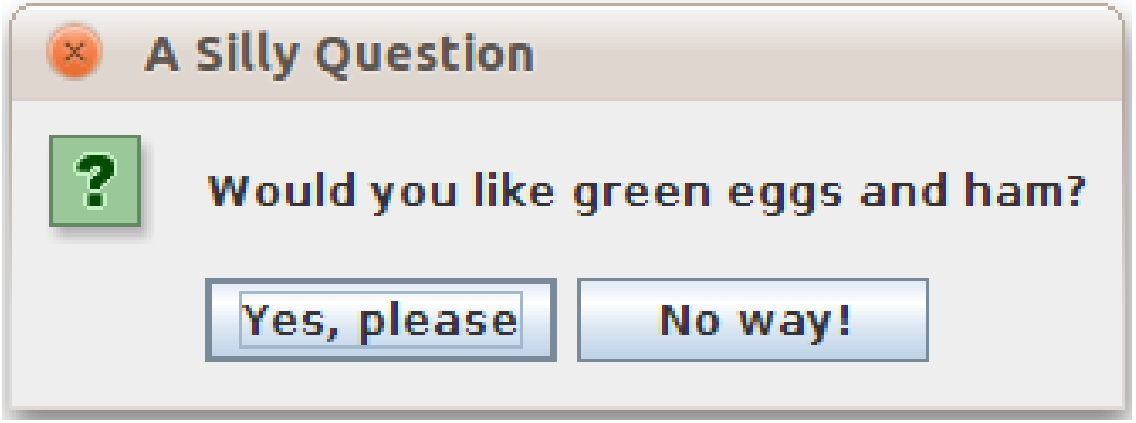
- A container that uses `JDialog` as the window
- Creates modal dialogs
- Customisable features:
  - title
  - message or a collection of components
  - icons
  - buttons
  - button texts

Example (1)



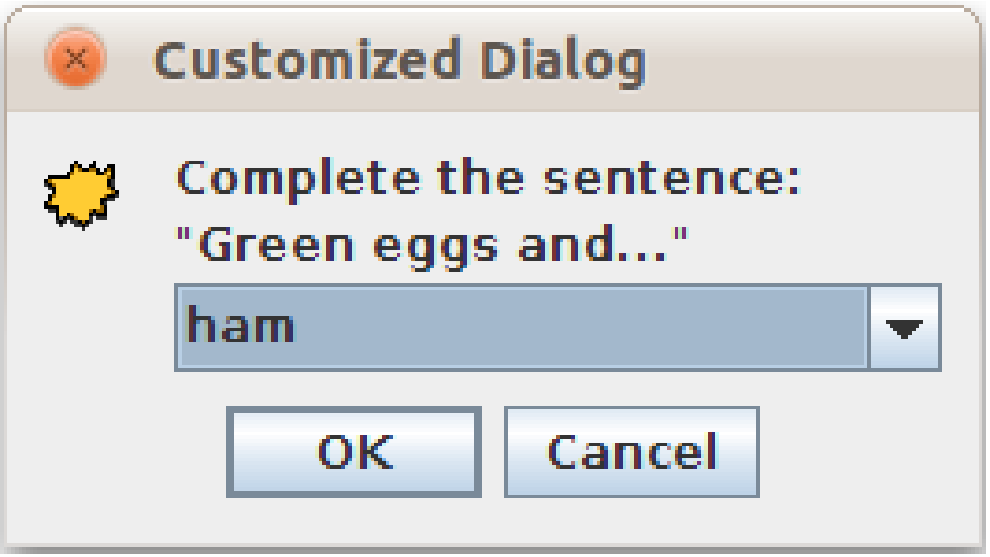


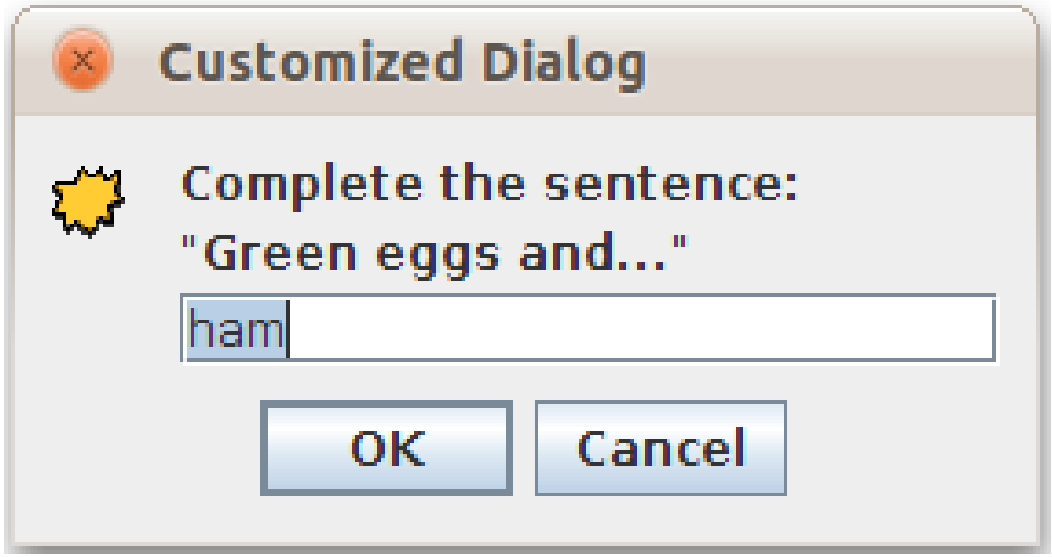
message dialogs



option dialog

Example (2)





input dialogs

Types of dialog

- Message dialog:
  - one-button dialog
- Option dialog:
  - like a message but has a variety of buttons
- Input dialog:
  - to obtain a text input

Methods to create dialogs

- `showMessageDialog`
- `showOptionDialog`
- `showInputDialog`

showMessageDialog

- `parentComponent`: the parent window (frame)
- `mesg`: the message to show
- `title`: the dialog title
- `messageType`:
  - INFORMATION_MESSAGE
  - ERROR_MESSAGE
  - WARNING_MESSAGE
  - PLAIN_MESSAGE

showOptionDialog

- `parentComponent`
- `mesg`
- `title`
- `optionType`: a combination of Yes/No/Cancel
- `messageType`
- `icon`: an Icon object
- `options` (optional): list of button texts (matches with optionType)
- `initialValue`: initial (selected) button

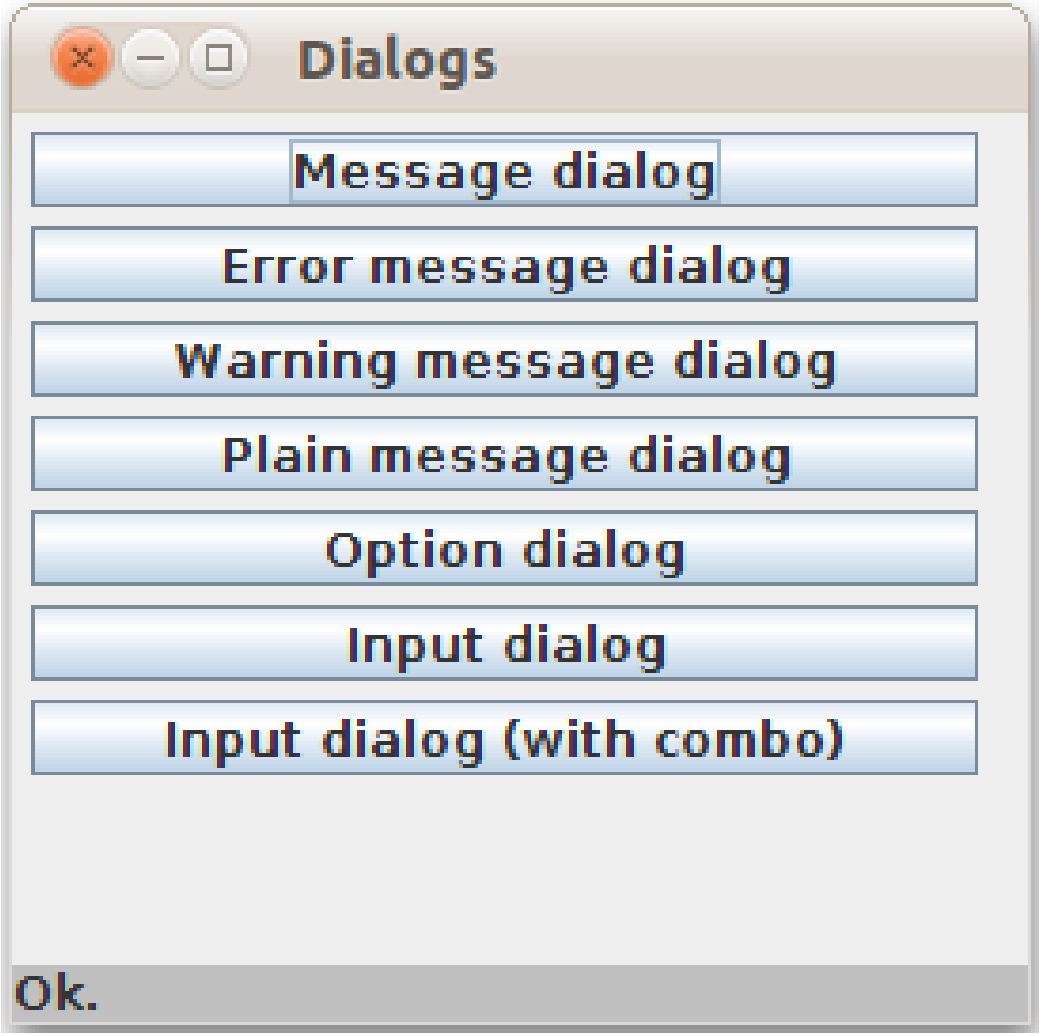
showInputDialog

- `parentComponent`
- `mesg`
- `title`
- `messageType`
- `icon`: an Icon object
- `options` (optional): list of allowed values to select
- `initialValue`: initially (selected) value

DEMO

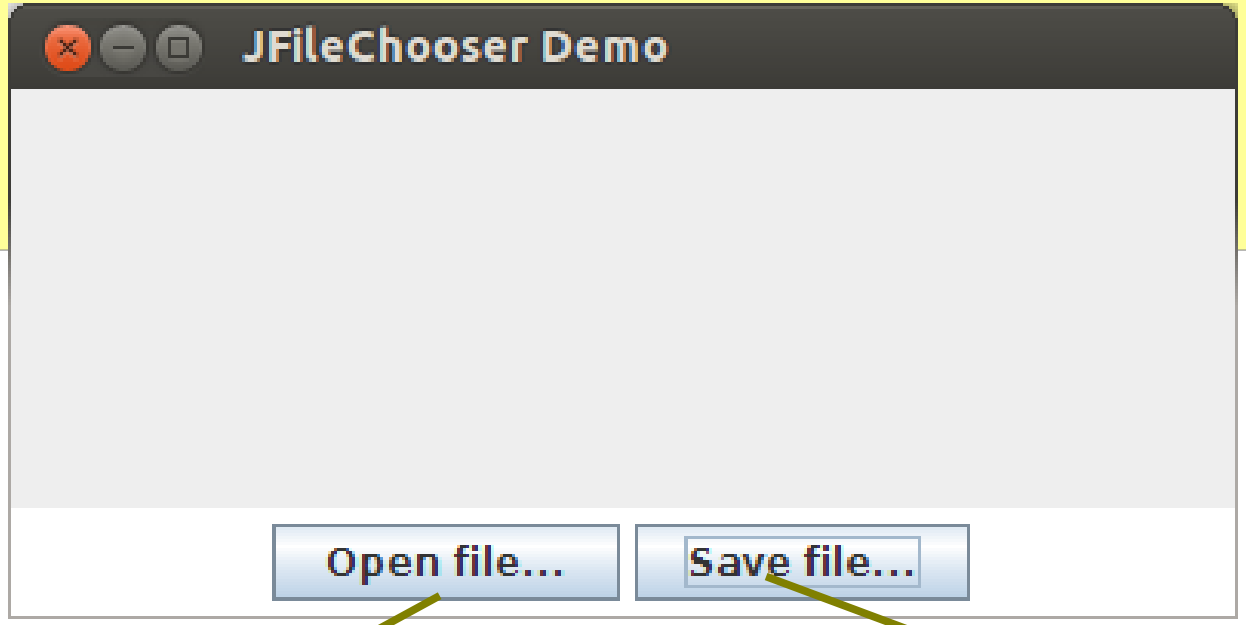
JOptionPane

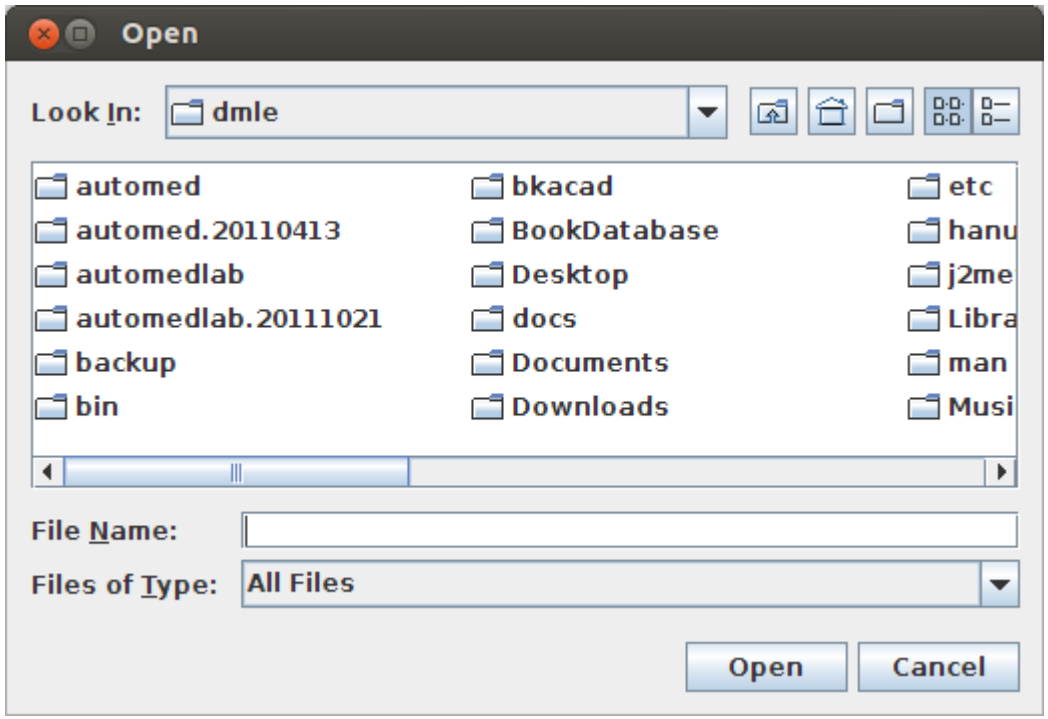
gui.dialogs.SimpleDialogDemo

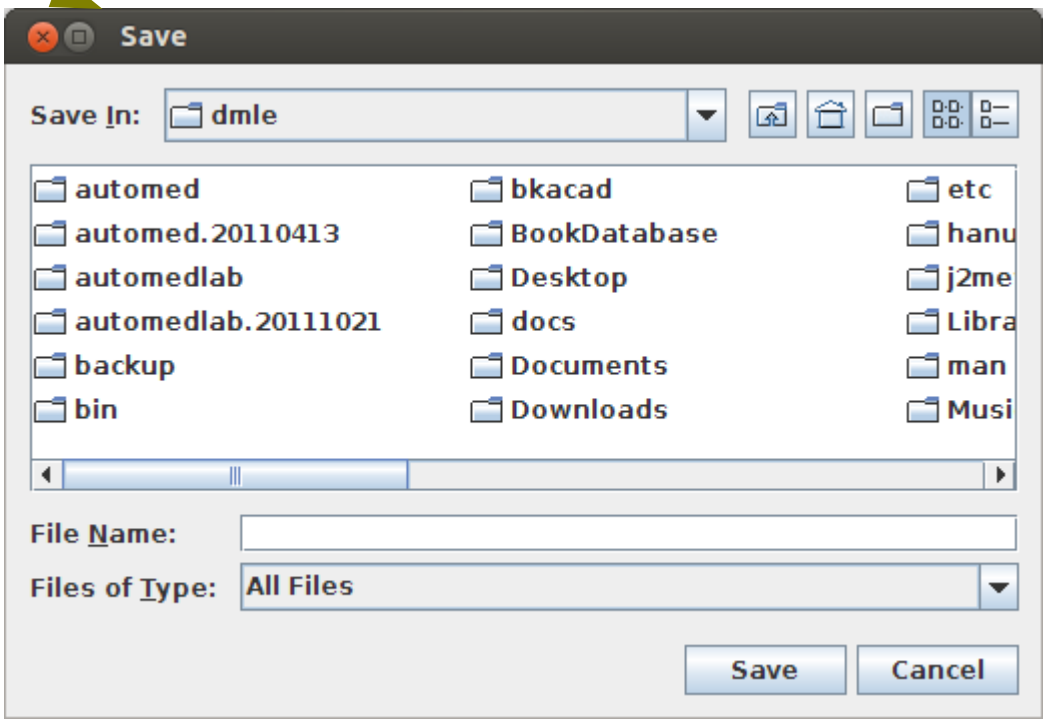


[0]JFileChooser

gui.dialogs.
JFileChooserDemo



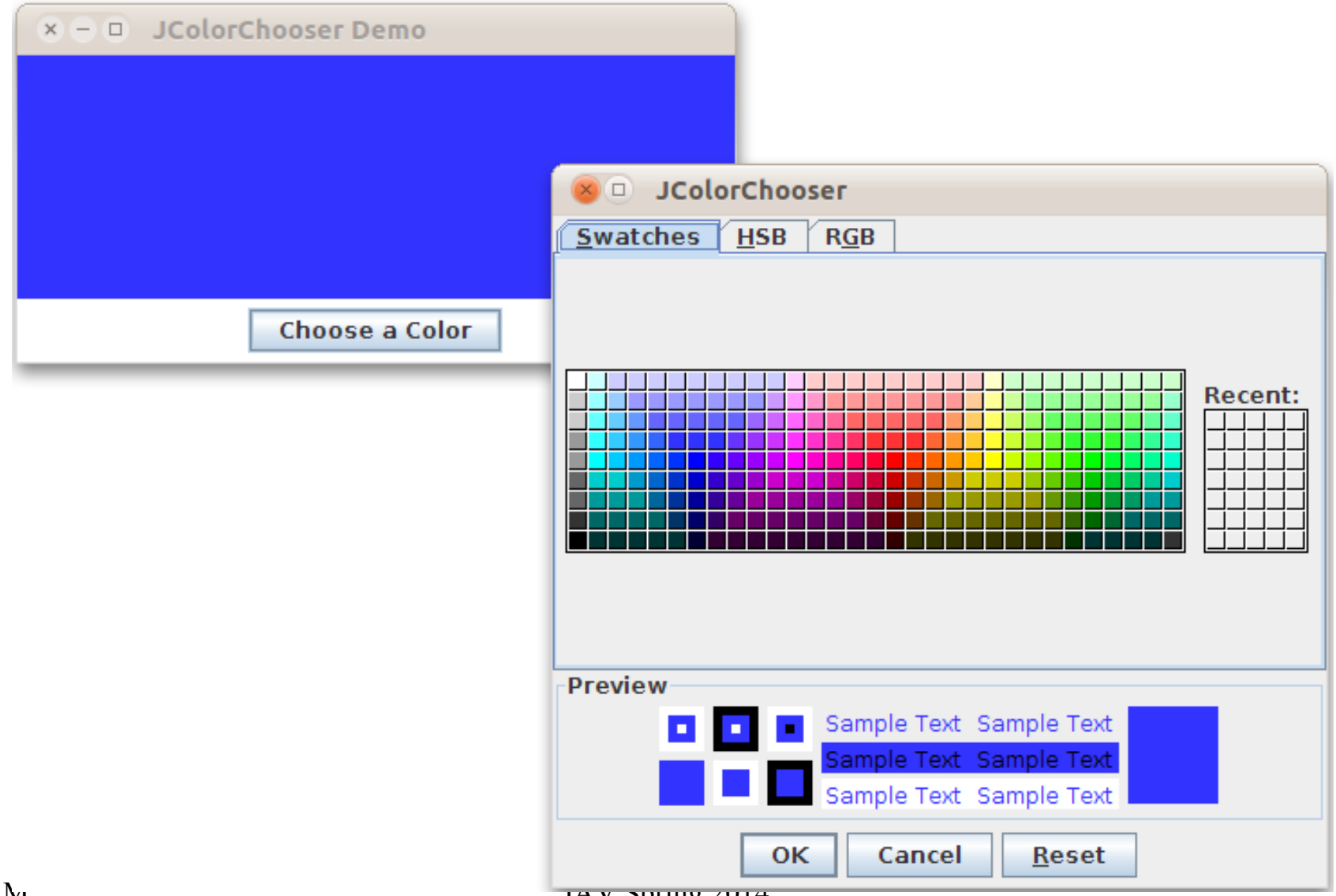




DEMO

[0] JColorChooser

gui.dialogs.JColorChooserDemo



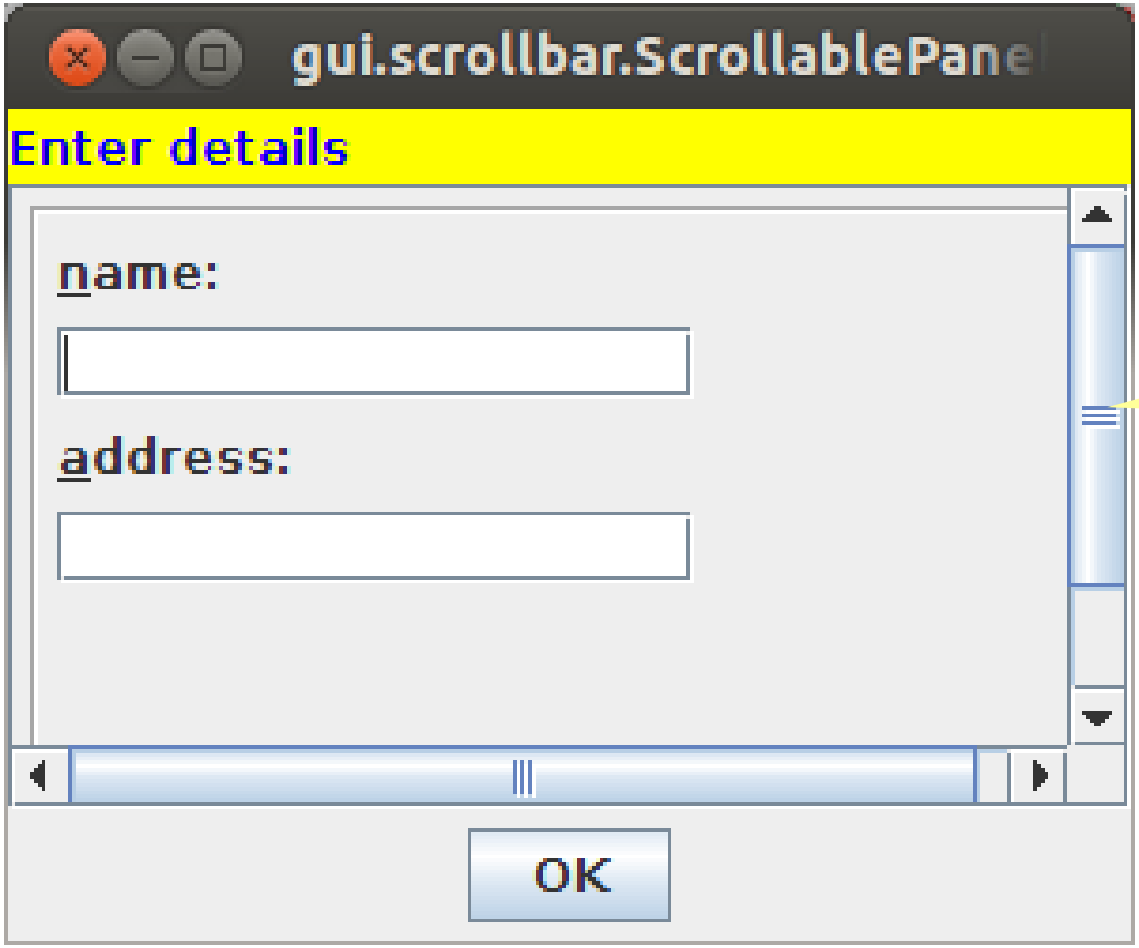
Scroll bar

- Class: `JScrollPane`
- Represents a fixed, sliding view of a display component
- Create a `JScrollPane` object using the component as input
- Add the `JScrollPane` object to the window
- Examples:
  - scrollable panel
  - scrollable text field
  - scrollable table (later)

DEMO

[0] Scrollable panel

`gui.scrollbar.ScrollablePanelDemo`

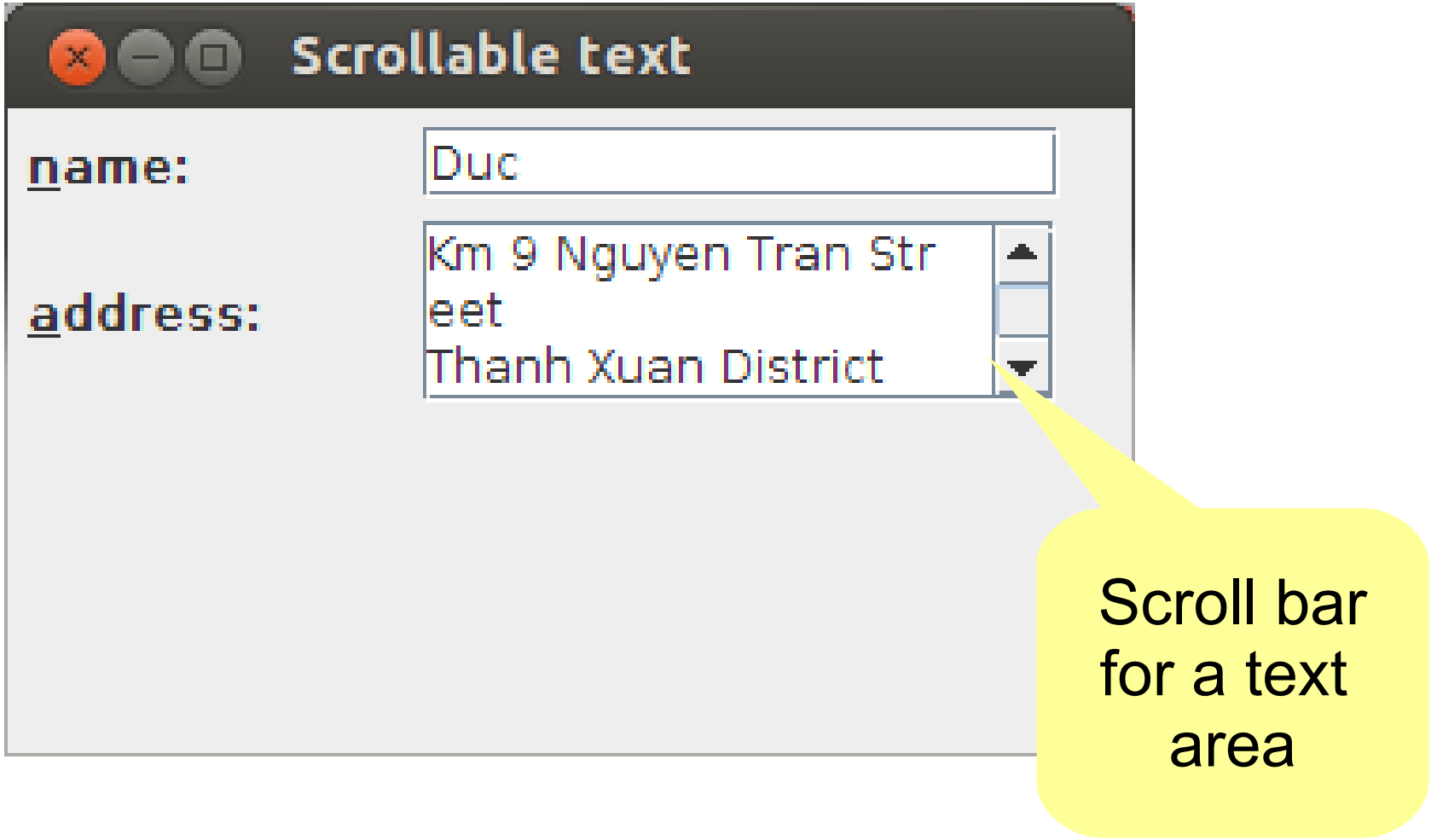


Scroll bar for a panel

DEMO

[0] Scrollable text field

`gui.text.ScrollableTextArea`



Exer
cise

Class exercise

- Extend `MyApp` application to:
  - validate data entered by user
  - display an info. message for successful data entry
  - display an error message for erroneous data entry
  - ?..?

Tabular display: `JTable`

- Swing provides `JTable` to display data in a tabular form
- A table contains a header row and one or more rows of data
- The header row is an array of column names
- A data row is an array of values (possibly of different types)
- A column is an array of values of the same type
- Objects of different types can be displayed in a table

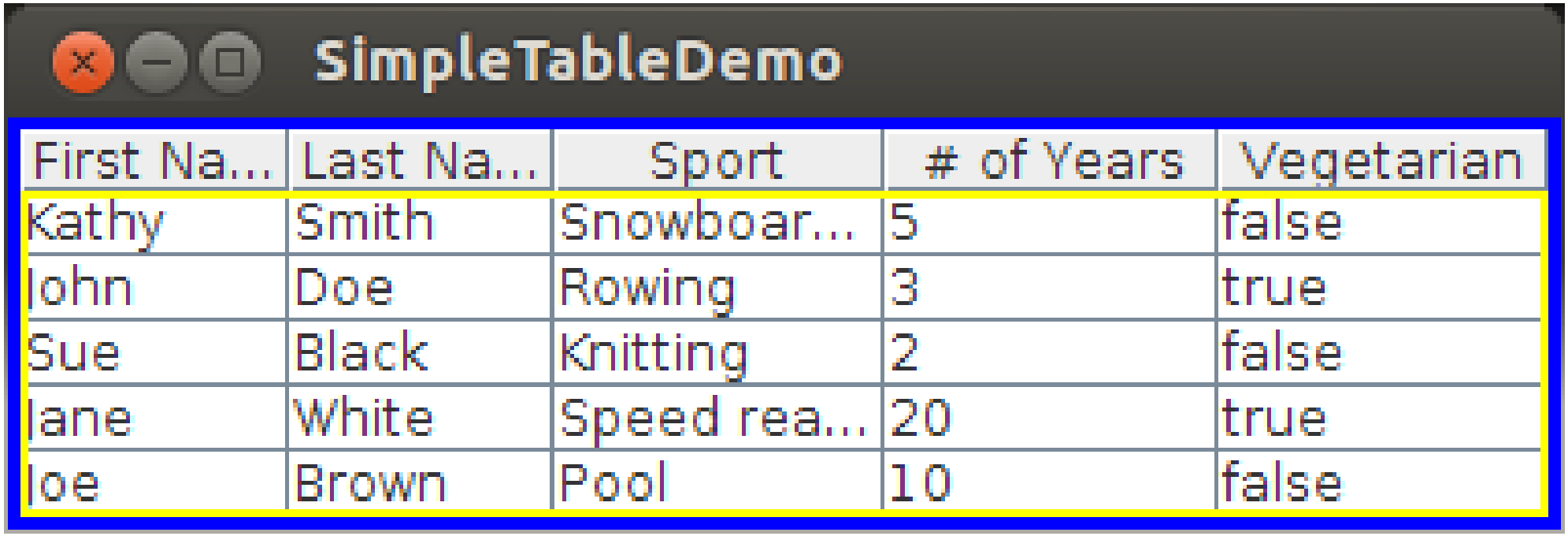
A simple `JTable`

- Create headers
- Create data rows
- Create a `JTable` object
- Put table object into a scroll bar object
- Add scroll bar object to window

DEMO

A simple JTable

gui.tables.SimpleTableDemo



First Na...	Last Na...	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboar...	5	false
John	Doe	Rowing	3	true
Sue	Black	Knitting	2	false
Jane	White	Speed rea...	20	true
Joe	Brown	Pool	10	false

Create headers

```
Object[] head = {
        "First Name",
        "Last Name",
        "Sport",
        "# of Years",
        "Vegetarian" };
```

Create data rows

```
Object[][] data = {
  {"Kathy","Smith","Snowboarding",5,false},
  {"John","Doe","Rowing",3,true },
  {"Sue","Black","Knitting",2,false },
  {"Jane","White","Speed reading",20,true},
  {"Joe","Brown","Pool",10,false}
};
```

Create a JTable object

```
JTable table = new JTable(data, head);
```

Put table object into a scroll bar object

```
// put table in a scroll bar
JScrollPane scroll = new JScrollPane(table);
```

Add scroll bar object to window

```
// add scroll bar to a window
w.add(scroll);
```

Table model

- Class: `DefaultTableModel`,
  `AbstractTableModel`,
  `TableModel`
- Manages the table data
- To get the table model:

`getModel(): TableModel`

- To change the table model:

`setModel(TableModel)`

Column model

- Class: `DefaultTableColumn`,

  `TableColumnModel`
- Manages all the table columns
- To get the column model:

```
getColumnModel(): TableColumnModel
```

- To change the column model:

```
setColumnModel(TableColumnModel)
```

Table header

- Class: `JTableHeader`
- Manages the table header
- To get the table header:

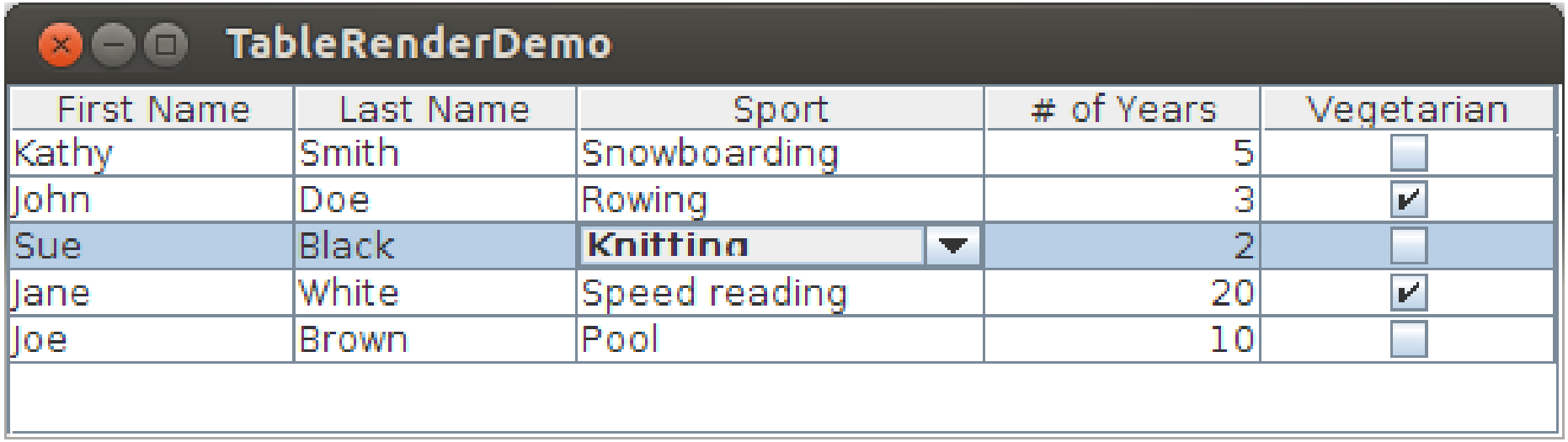
```
getTableHeader(): JTableHeader
```

- To change the table header:

```
setTableHeader(JTableHeader)
```

Another JTable example

gui.tables.TableRenderDemo



GUI tool kit

- Font
- Custom color
- Display tool kit

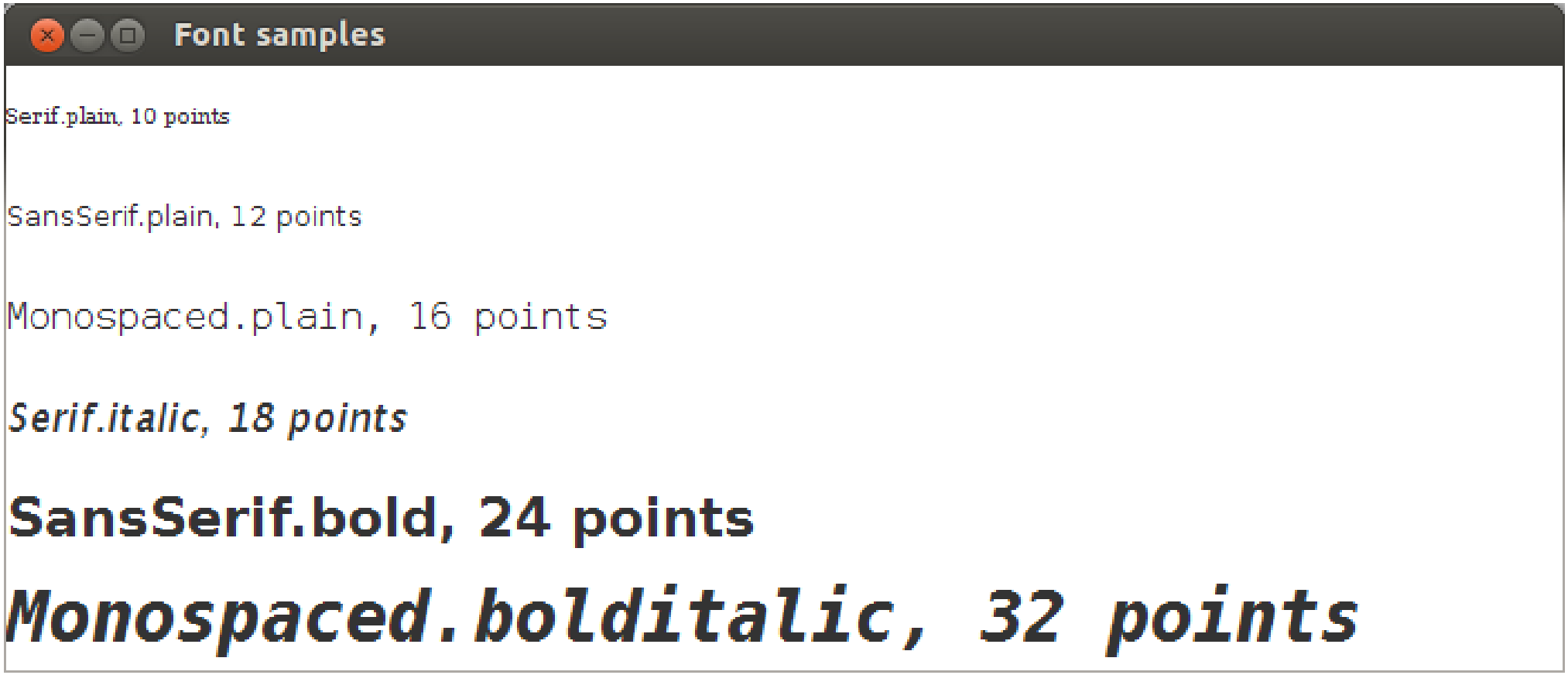
Font

- Class: `java.awt.Font`
- Constructor arguments:
  - family name: e.g. Times, SansSerif, Monospace,
  - style: a bit-wise of Font.PLAIN, Font.BOLD, Font.ITALIC
  - size: number of points (point = 1/72 inch)

DEMO

Font

gui.font.FontDemo



Custom colour

- Class: `java.awt.Color`
- Create a new Color object with arguments red, green, and blue
- R,G,B values are either in [0,255] or [0,1]:

```
// using integral
Color brown = new Color(200,150,0);
// using float: (float) 200/255, ...
```

Useful Color methods

- `getRed(): int`
  - returns the red value in the range [0,255]
- `getGreen(): int`
  - returns the green value
- `getBlue(): int`
  - returns the blue value
- `brighter(): Color`
  - returns a brighter color of the current one
- `darker(): Color`
  - returns a darker color of the current one

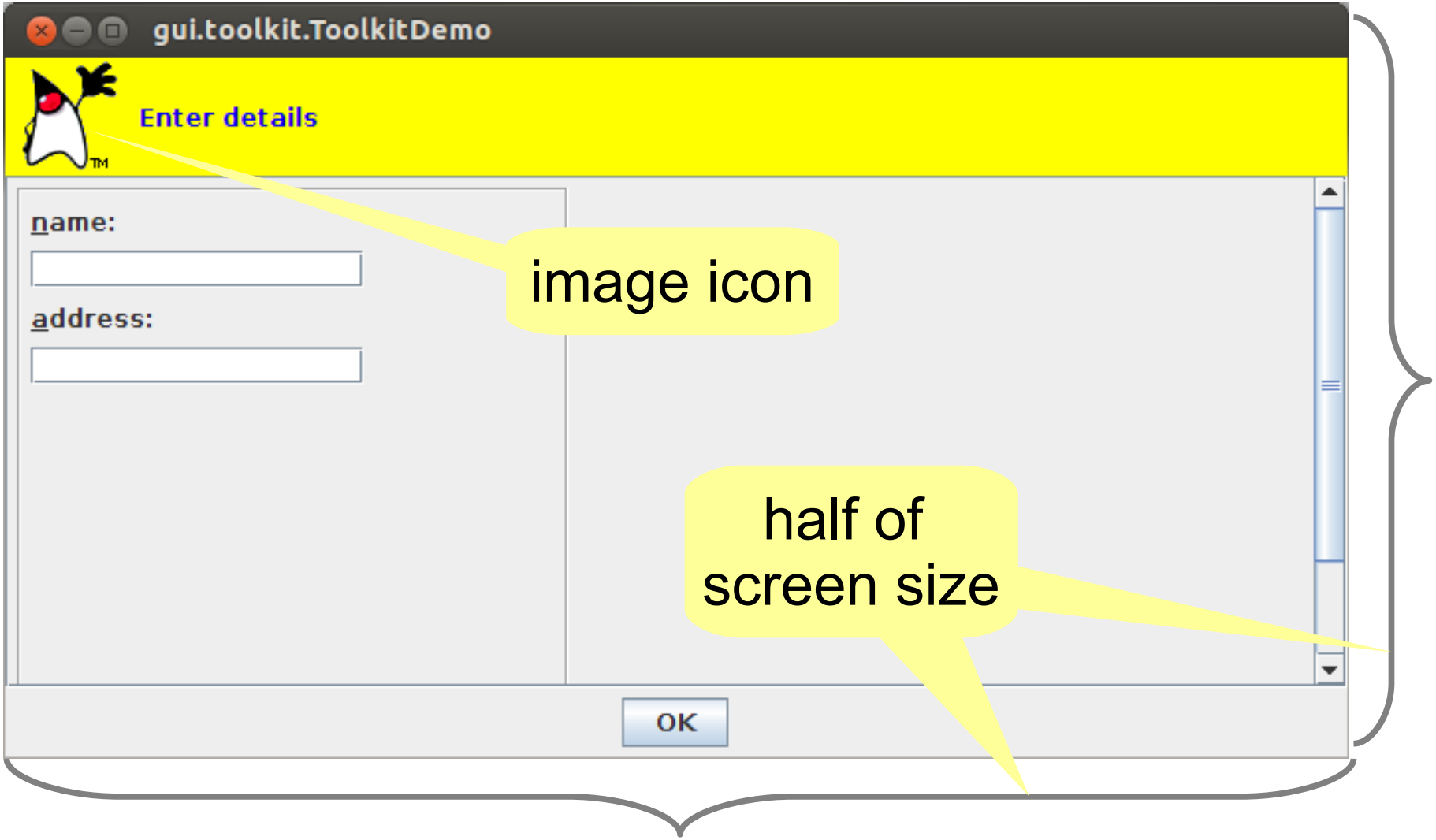
Display tool kit

- Class: `java.awt.Toolkit`
- Provides utility methods for:
  - getting screen size
  - creating an image from a file

DEMO

Tool kit

gui.toolkit.ToolkitDemo

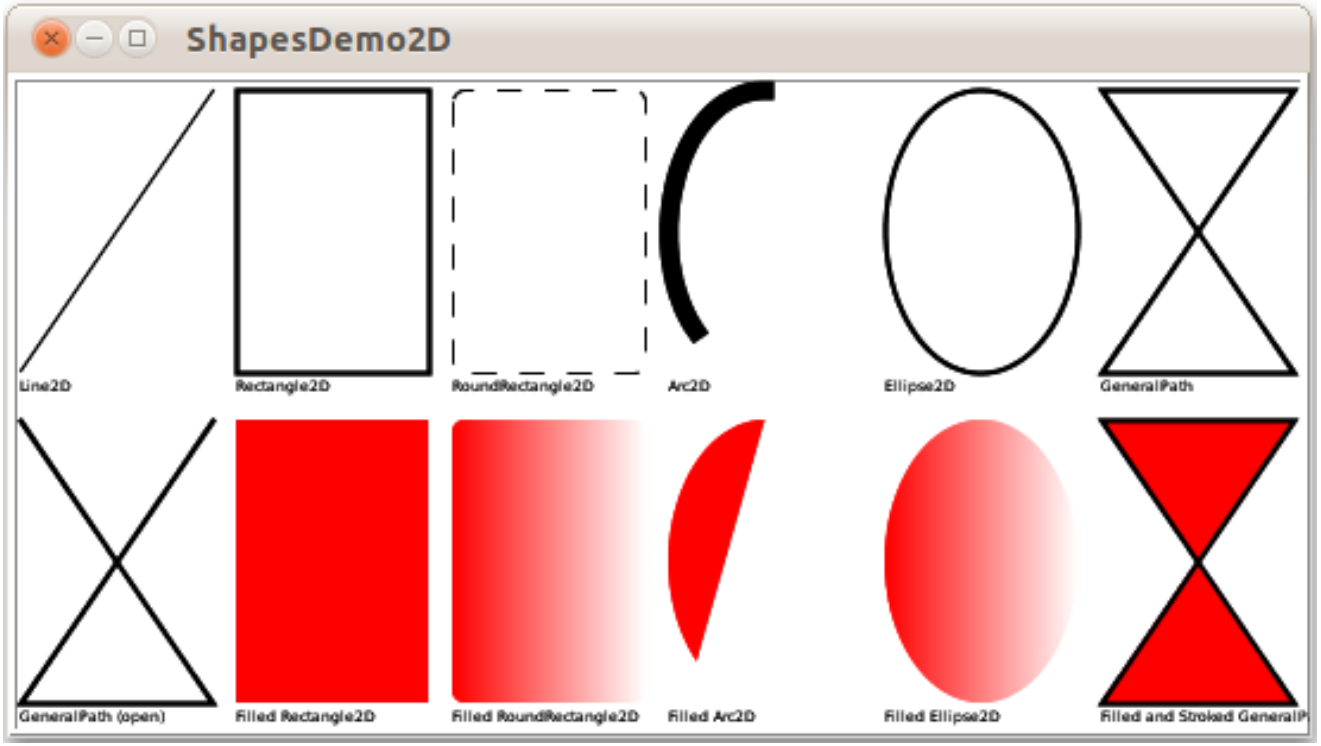


Custom GUI using drawing

- GUI drawing is used for special graphical requirements
- Every Swing component has an associated graphics object
- Class: `java.awt.Graphics`,

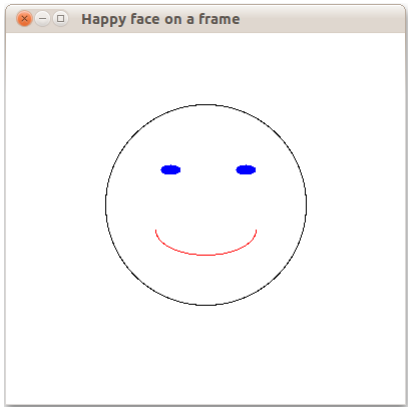
  `java.awt.Graphics2D`

Basic examples



basic shapes

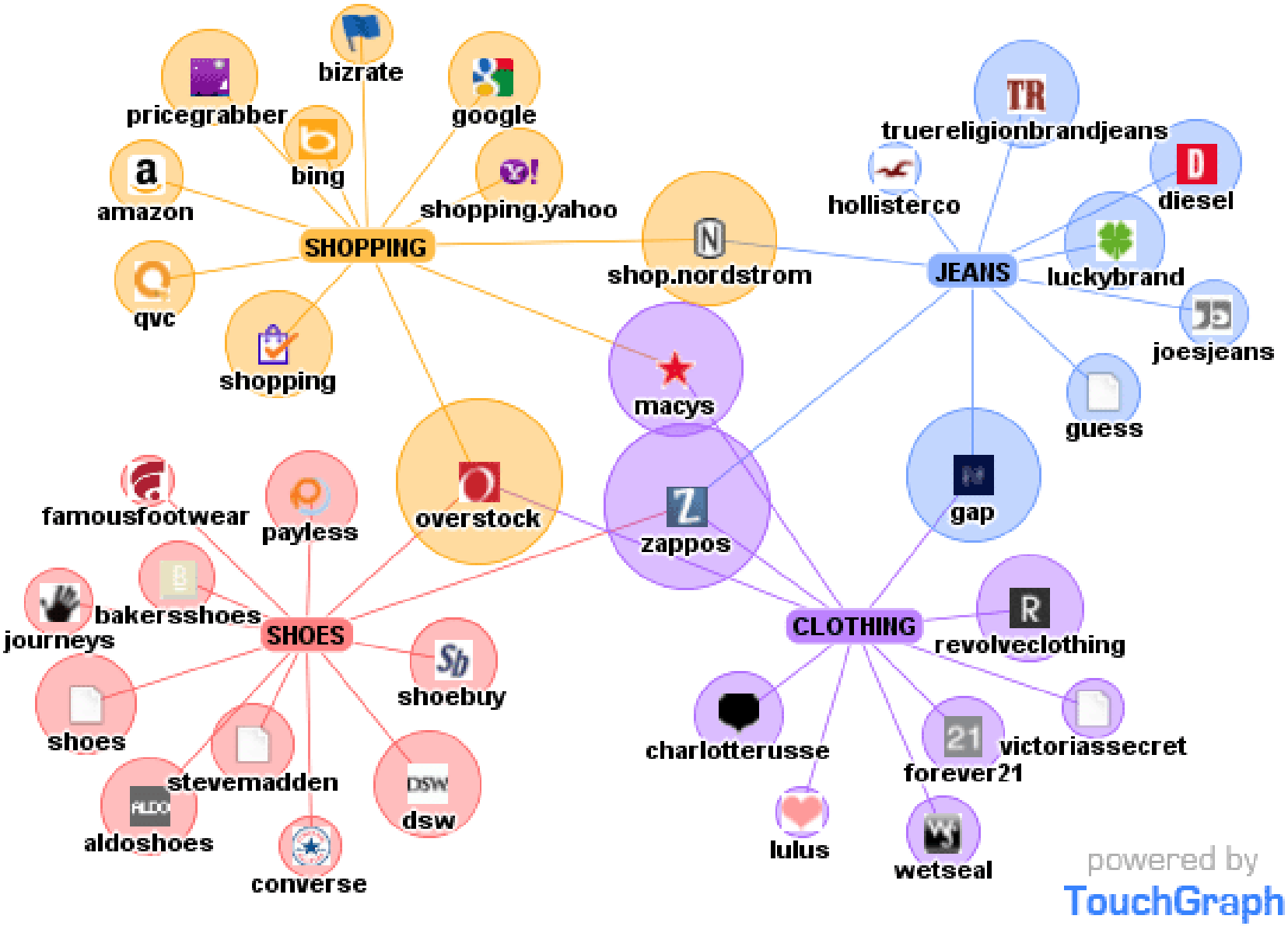
a pear!



a happy face

Example: graph drawing

http://www.touchgraph.com



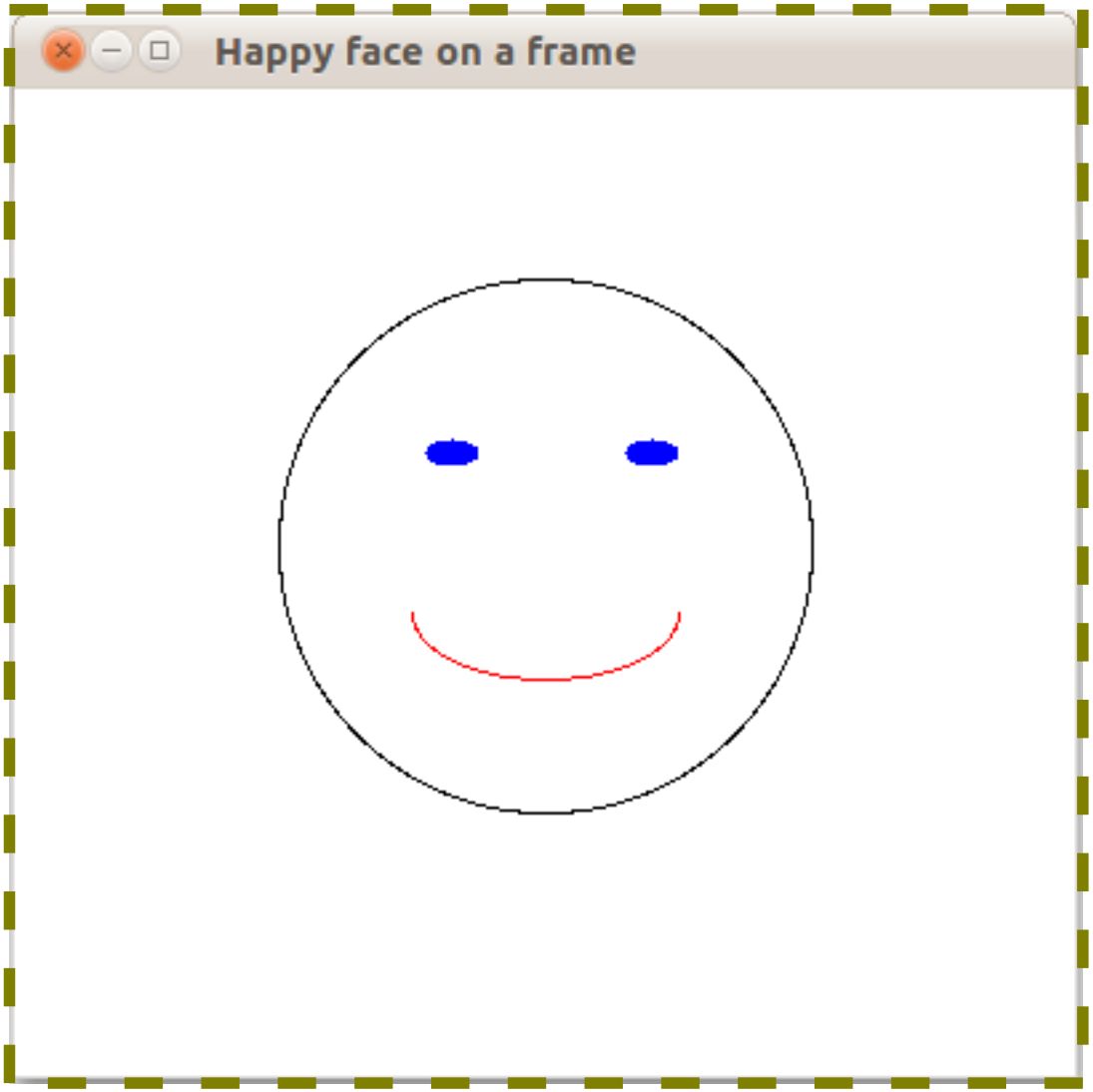
Class `Graphics`

- Encapsulates the GUI-related state of a display component
- Specifies the display area of the component:
  - all drawings on the component will appear within this area
- Provides methods for drawing primitive shapes (lines, circles, rectangles, etc.):
  - `drawX()`: draws an outline of shape X
  - `fillX()`: fills the area defined by `drawX()`
- Font and color of each drawing can be changed

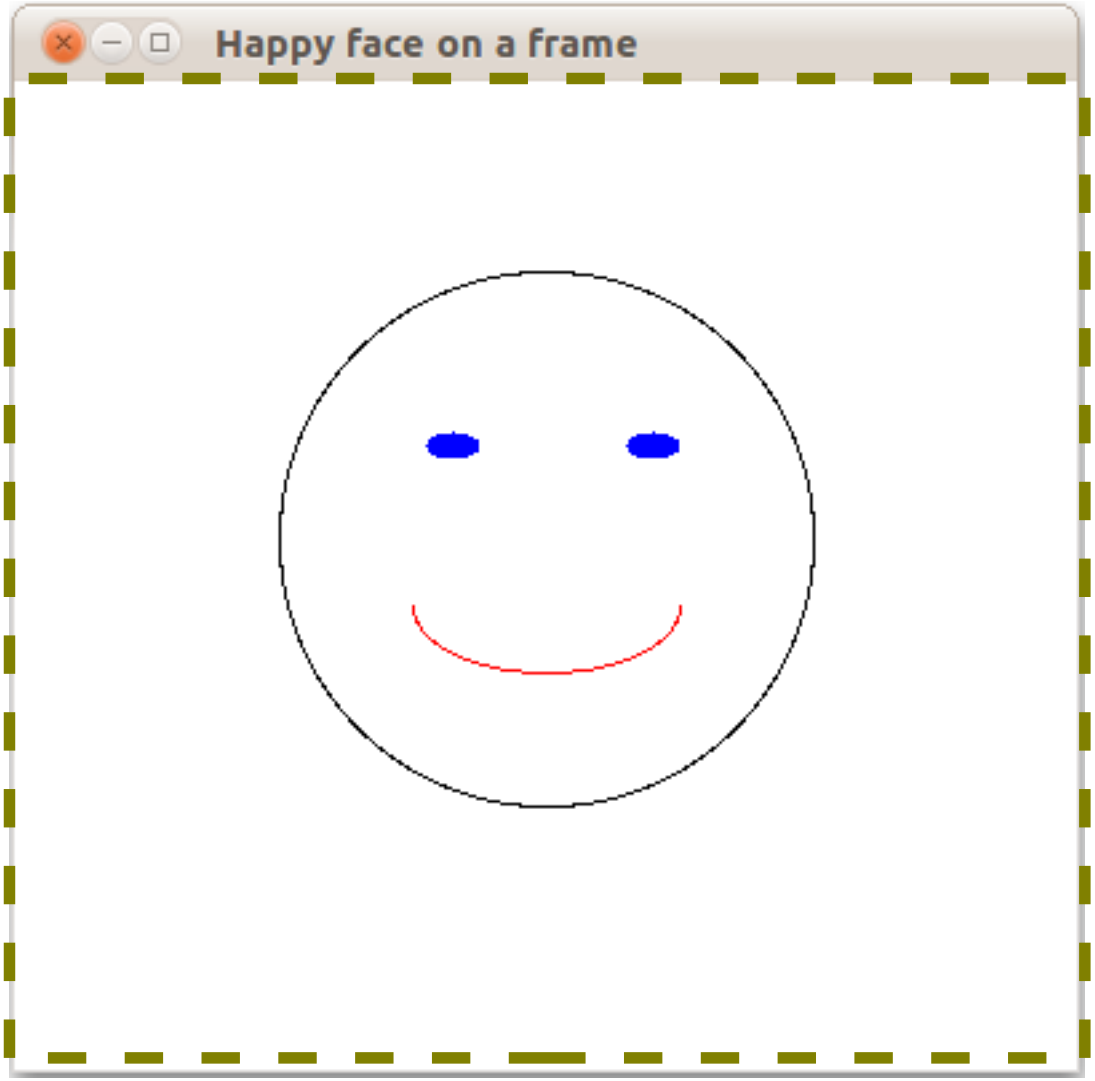
Display area

- Top-level container (e.g. JFrame):
  - the entire window
- Nontop-level container:
  - display area of the container of the component

Top-level display area



Component display area



GUI drawing basics

- Sub-class a `JComponent` object:
  - *commonly* `JFrame` or `JPanel`
- Override the `paint` or `paintComponent` method:
  - must invoke the super-class method first!
  - use `Graphics` object to draw the desired basic shapes
- Display the object:
  - use a `JFrame` if object is not a top-level container

Using JFrame

```
public class HappyFaceColor extends JFrame {
   public HappyFaceColor() {
     // set up this frame
   }
   @Override
   public void paint(Graphics g) {
     // invoke super class method first!
     super.paint(g);
     // custom drawing using g
   }
}
```

Using JPanel (1)

```
class Drawing extends JComponent {
  @Override
  public void paintComponent(Graphics g) {
    // invokes super class method first!
    super.paintComponent(g);
    // custom drawing using g
  }
}
```

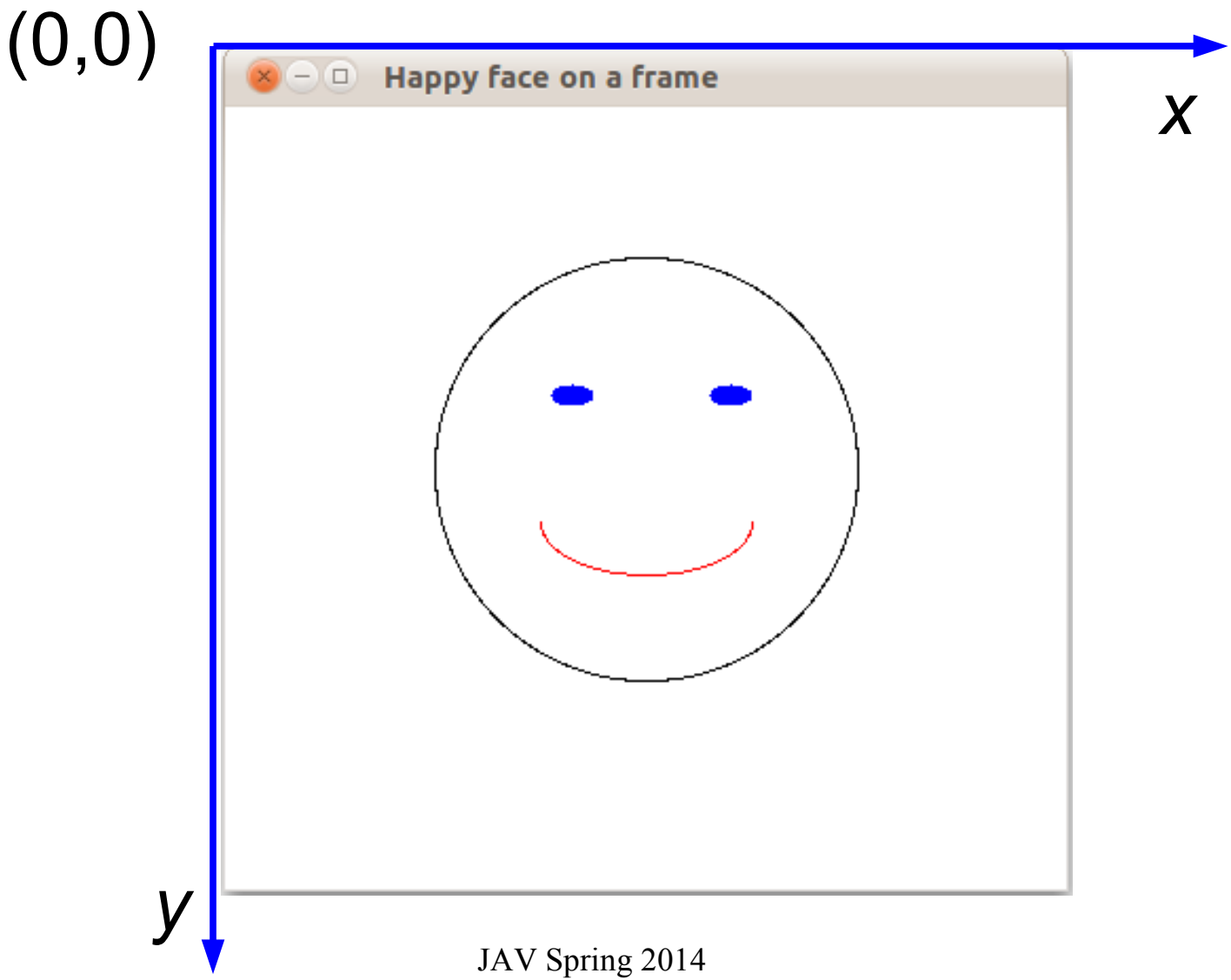
Using JPanel (2)

```
public class DrawingApp {
    private JFrame frame;
    public void createAndShowGUI() {
      // ...
      // setup drawing
      Drawing draw = new Drawing();
      // setup drawing panel
      JPanel drawPanel = new JPanel();
      drawPanel.add(draw);
      ...
      frame.add(drawPanel);
      // ...
    }
    ...
}
```

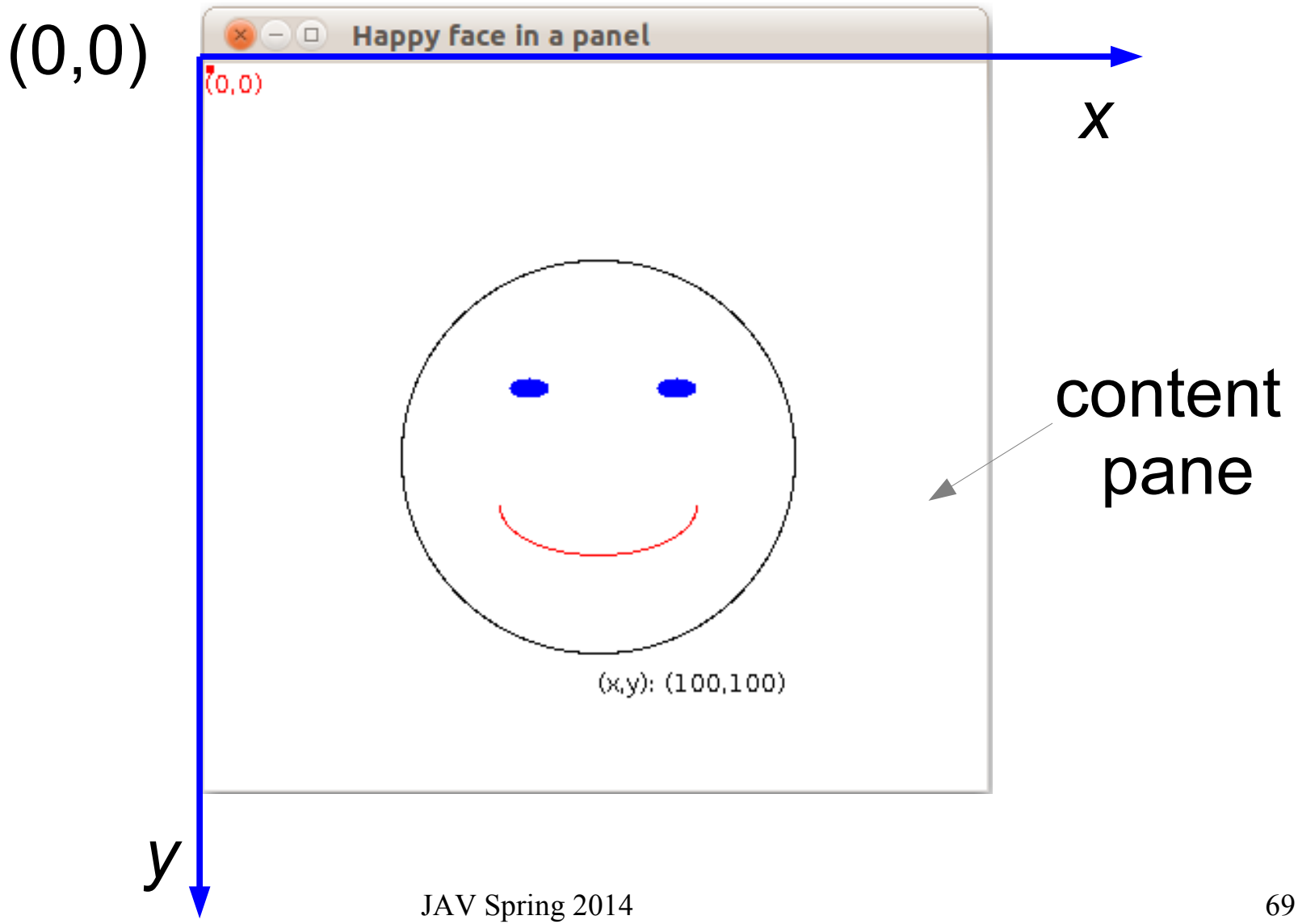
Drawing coordinate space

- The coordinate space of the component display area:
  - origin (0,0) is the top-left corner
  - x-axis extends rightward
  - y-axis extends downward

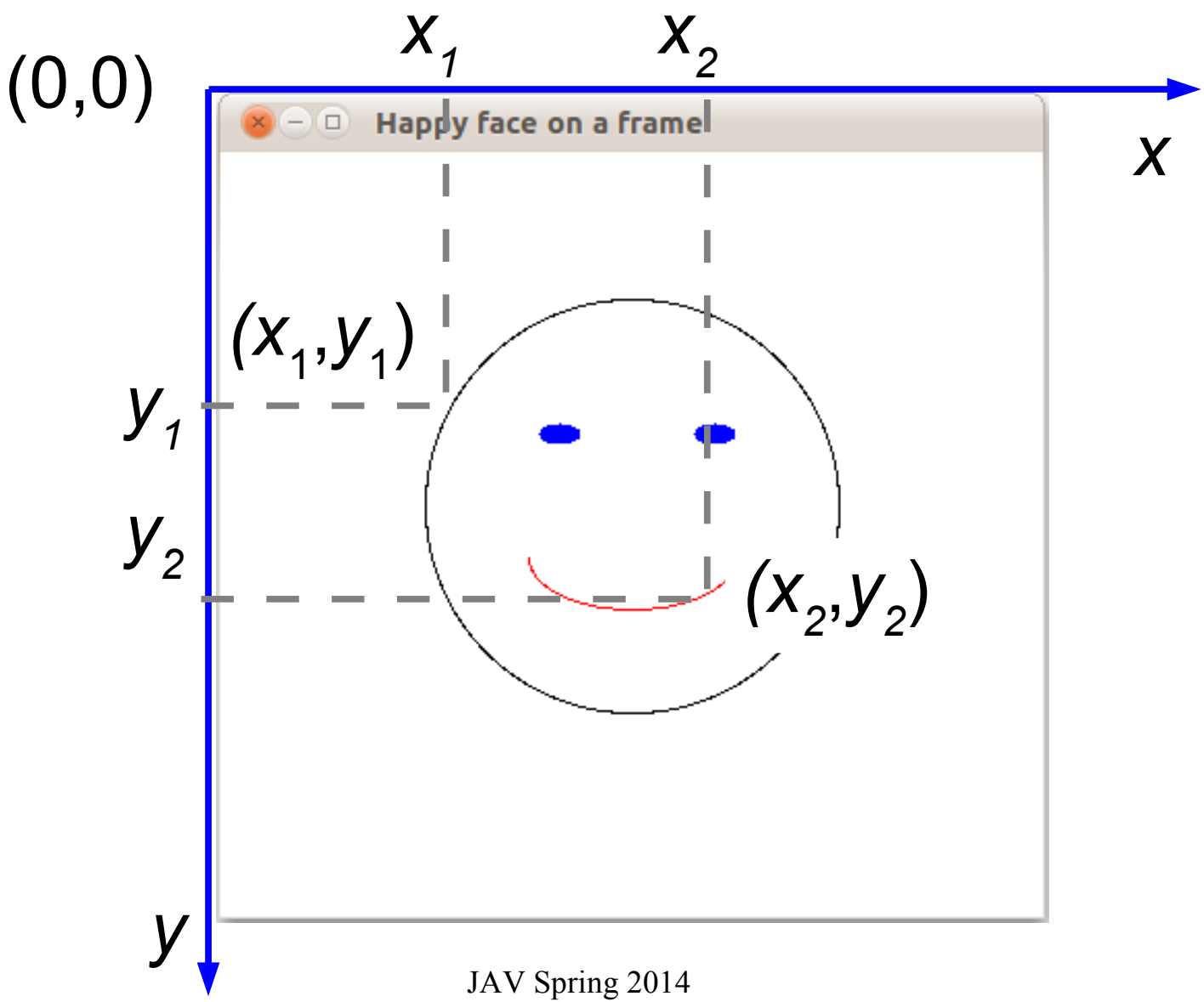
Coordinate space: JFrame



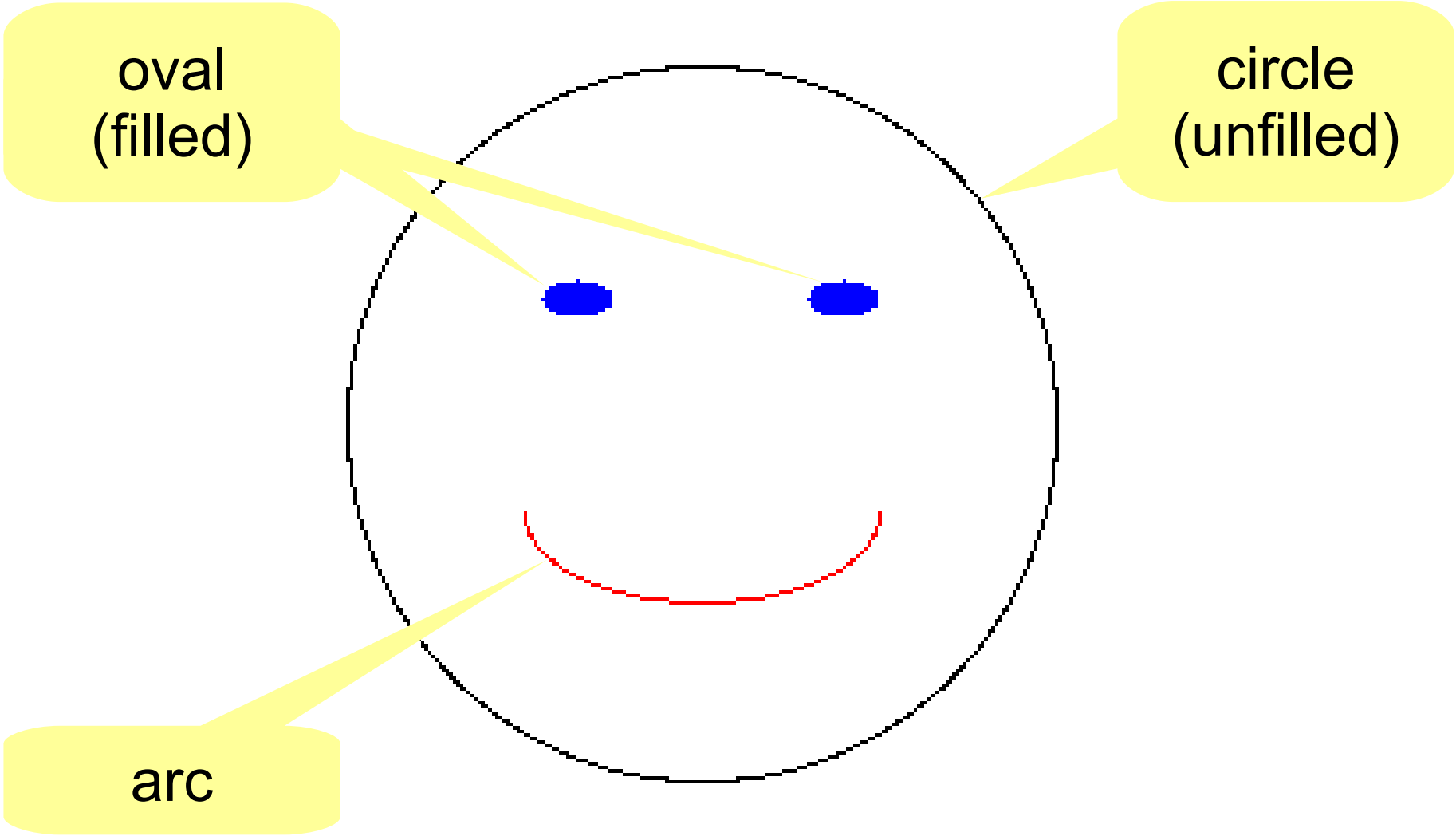
Coordinate space: content pane



Drawing points



Basic drawing shapes

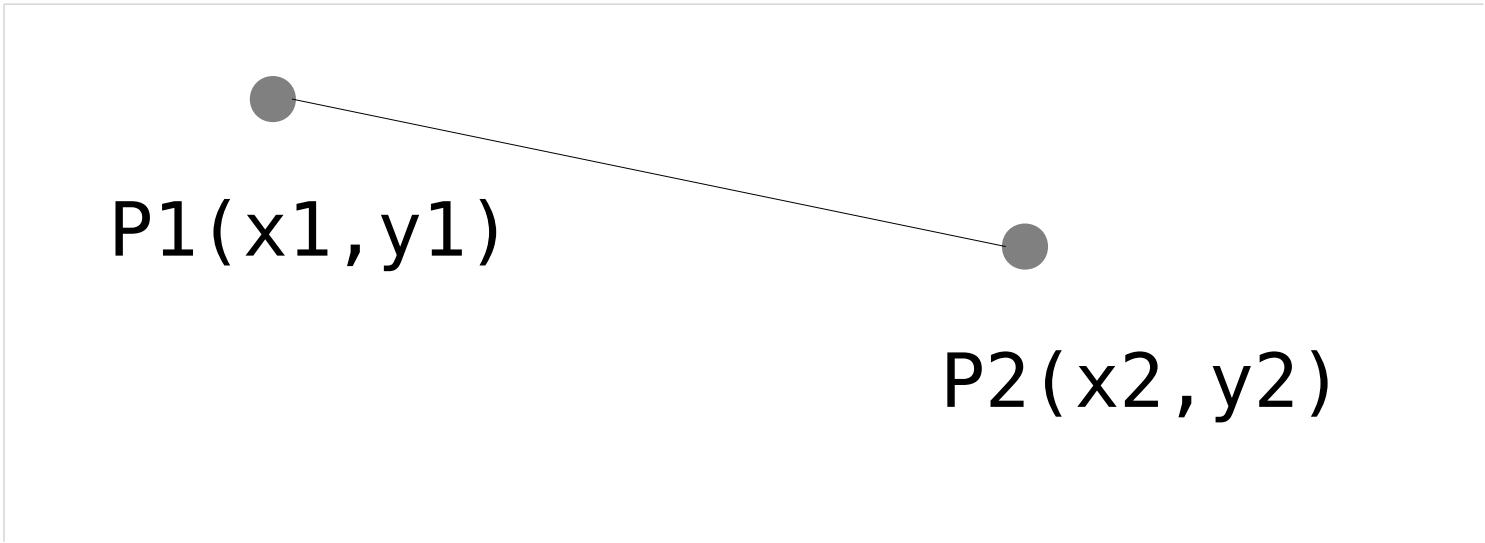


drawLine()

```
public void drawLine(int x1, int y1,
                     int x2, int y2)
```

- Draw a line *segment* between two points `P1(x1,y1)`, `P2(x2,y2)`

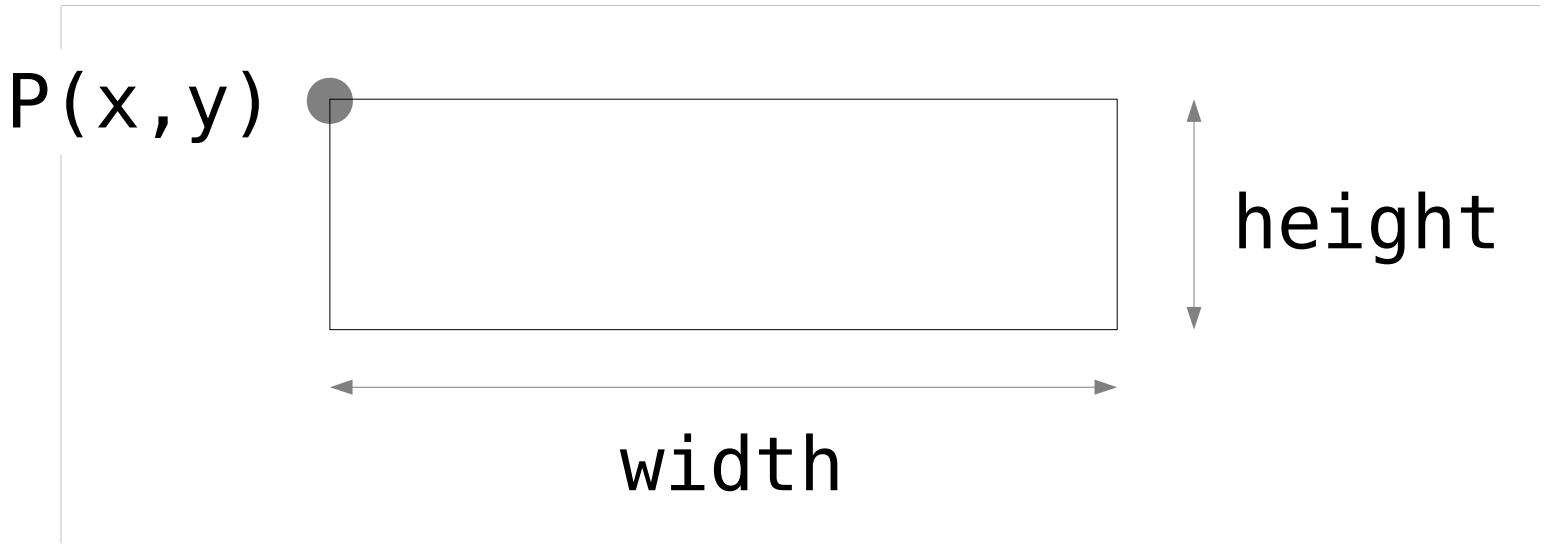
(0,0)



drawRect()

```
public void drawRect(int x, int y,
                     int width, int height)
```

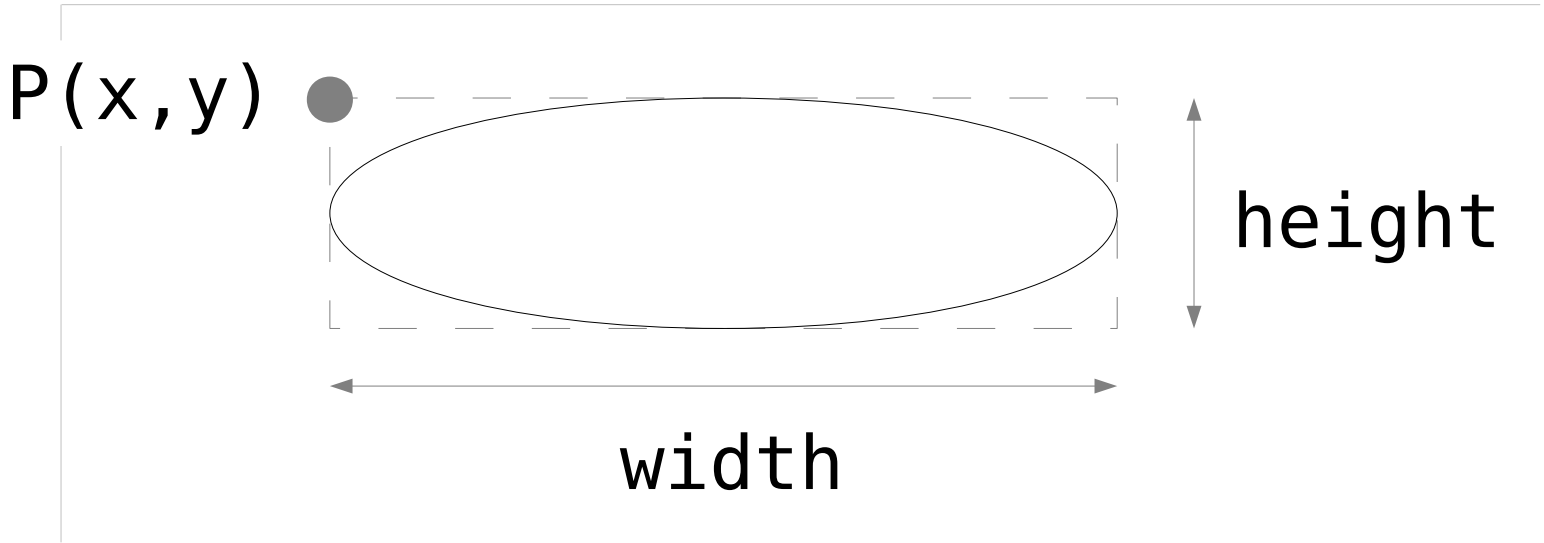
- Draw an (unfilled) rectangle whose top-left corner is `P(x,y)` and whose dimension is `width`, `height`



drawOval()

```
public void drawOval(int x, int y,
                     int width, int height)
```

- Draw an (unfilled) oval bounded by an (invisible) rectangle: top-left corner = `P(x,y)` & dimension =(`width`, `height`)



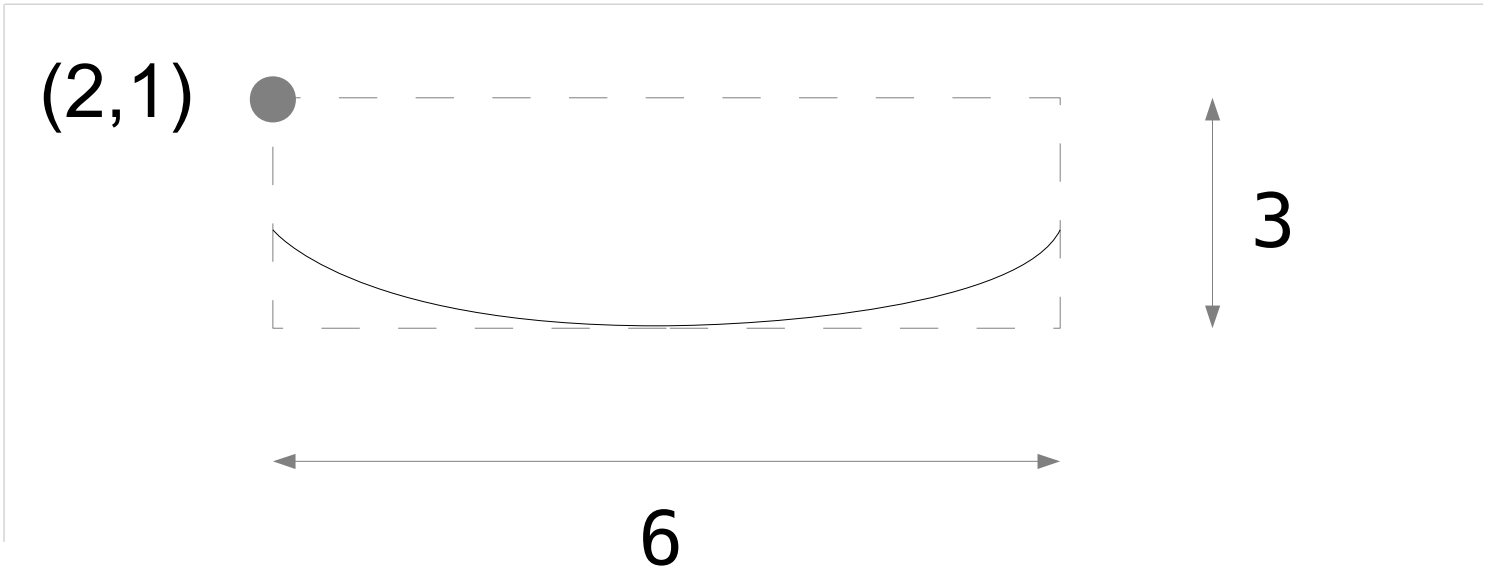
drawArc()

```
public void drawArc(int x, int y,
            int width, int height,
            int startAngle, int sweepAngle)
```

- Draw an (unfilled) arc of an oval
  - bounded by an (invisible) rectangle <`P(x,y), width, height`>,
  - start position is at `startAngle` and
  - end position is at the angle `startAngle + sweepAngle`

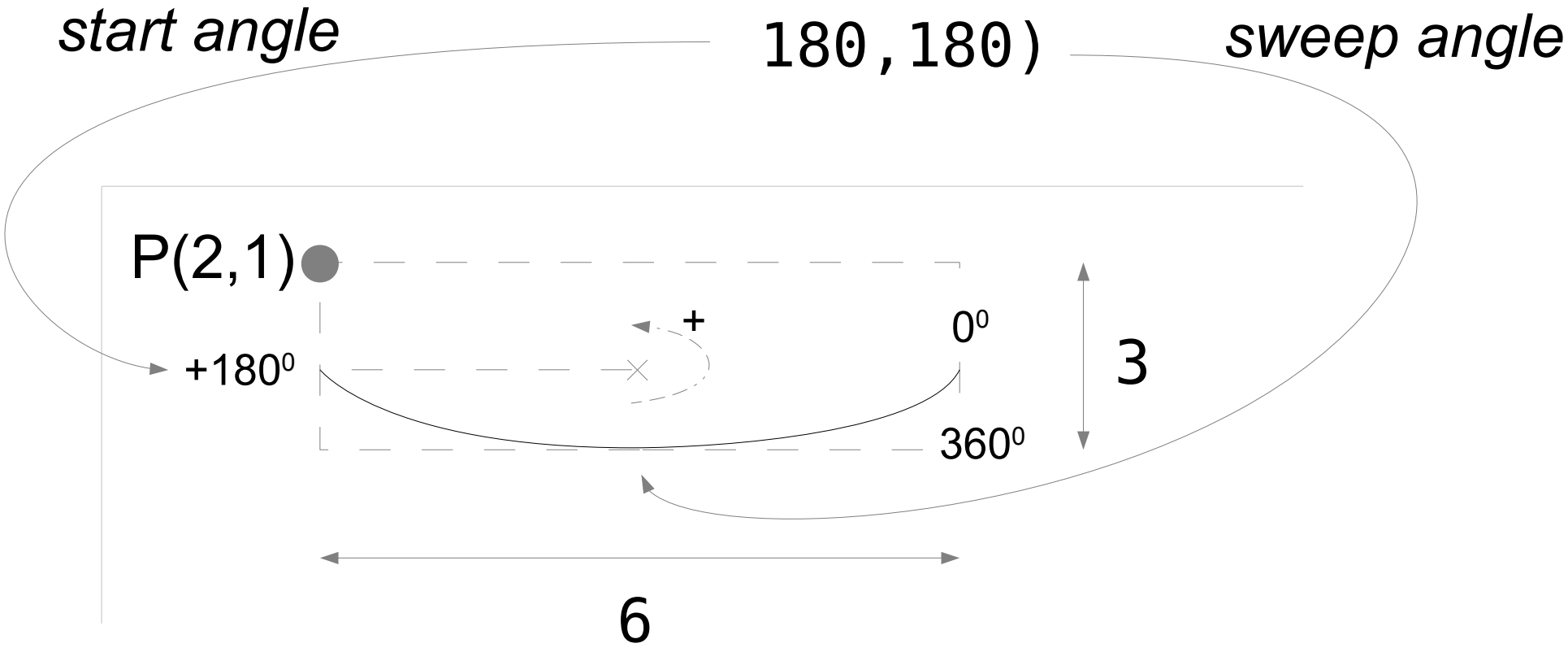
drawArc() method example

```
public void drawArc(2, 1,
                    6, 3
                    180,180)
```



drawArc() method example (2)

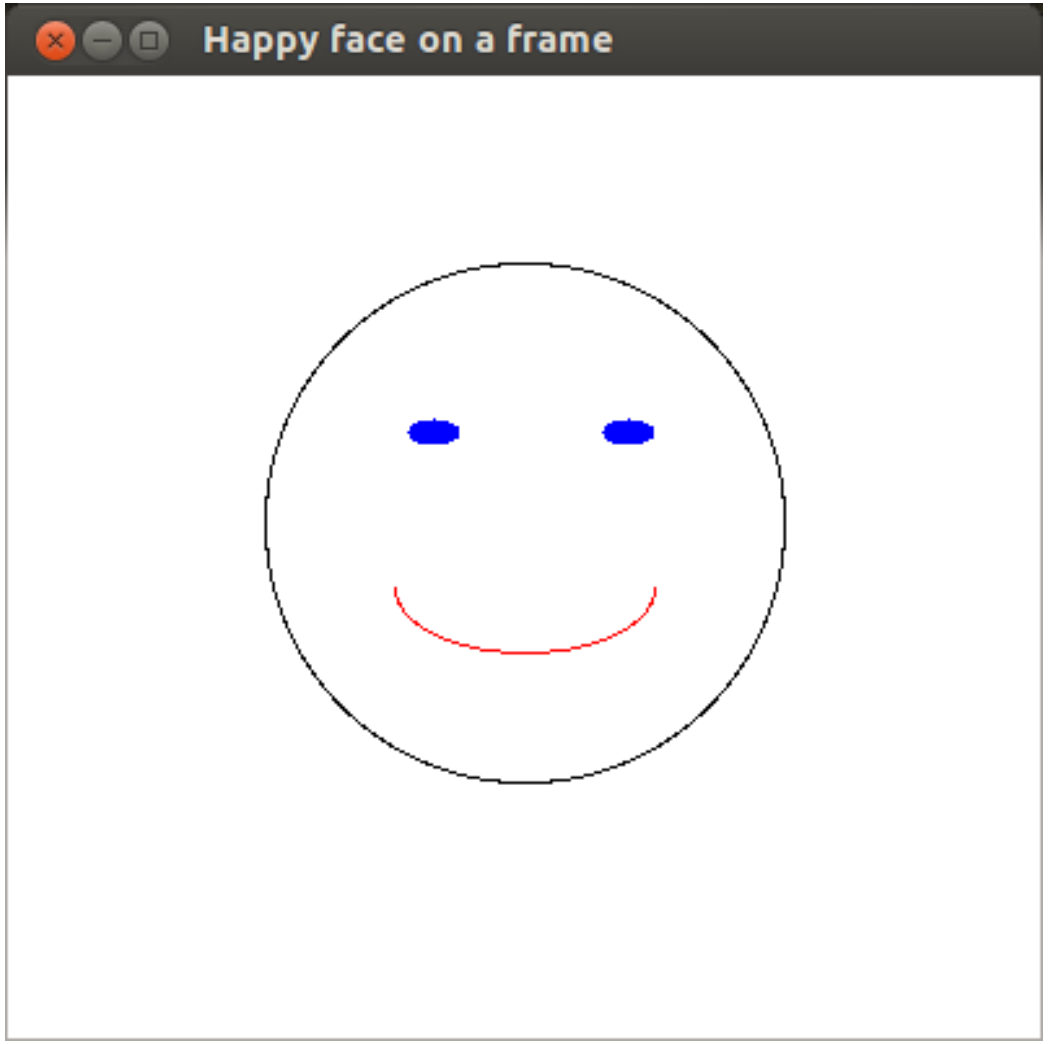
```
public void drawArc(2, 1,
                    6, 3
                    180,180)
```



DEMO

Happy face

```
gui.drawing.HappyFaceColorFrame
```



DEMO

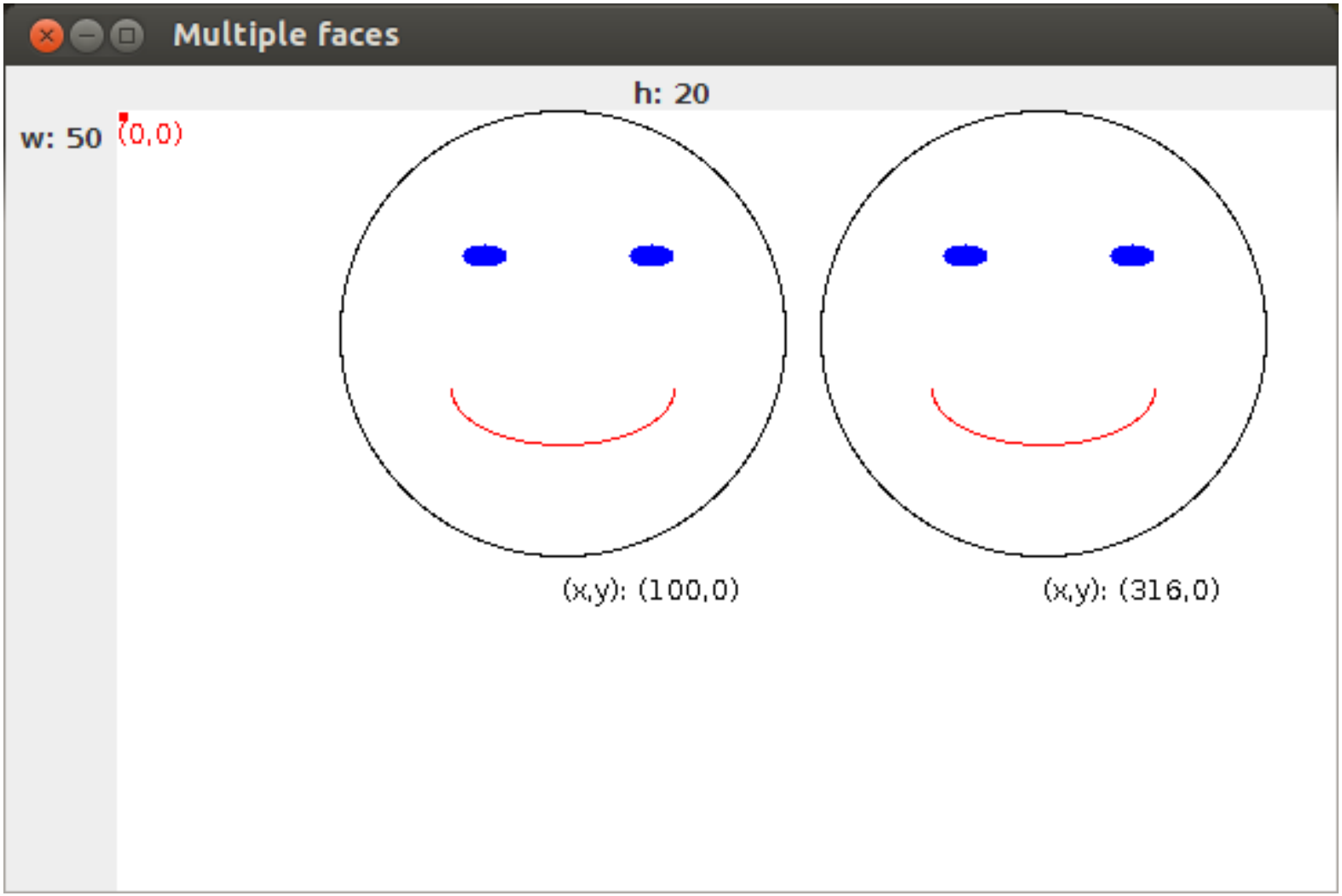
Happy face panel

```
gui.drawing.HappyFaceColorPanel
```

DEMO

Multiple happy faces

gui.drawing.MultipleHappyFaces



Summary

- GUI application development follows the MVC pattern
- Multi-tasking GUI programming using thread
- A variety of short message dialogs can be created using 4 dialog-typed classes
- Tabular view of data are created using `JTable`
- Custom GUI drawing is supported using the `Graphics` class and the utility classes `Font` and `Color`

References

Savitch W., Absolute Java, 4th, Addision-Wesley, 2009

- Chapter 17,18

Oracle, The Java Tutorial, Oracle, 2011, http://docs.oracle.com/javase/tutorial

- Lesson: Creating a GUI With JFC/Swing, Using Swing Components

- Trail: 2D Graphics