COMP1681 / SE15
Introduction to Programming

Lecture 29

GUIs and Graphics

Learning Objectives

Today's objectives are:

- to understand the basic principles of creating graphics and GUIs in Java.

Learning Objectives

Today's objectives are:

- to understand the basic principles of creating graphics and GUIs in Java.
- to know how to create a JFrame and use some of its basic methods.

Learning Objectives

Today's objectives are:

- to understand the basic principles of creating graphics and GUIs in Java.
- to know how to create a JFrame and use some of its basic methods.
- to be able to add simple components to a JFrame.

Learning Objectives

Today's objectives are:

- to understand the basic principles of creating graphics and GUIs in Java.
- to know how to create a JFrame and use some of its basic methods.
- to be able to add simple components to a JFrame.
- to use simple graphics drawing methods.

Java's Graphics

Java's API provides a rich toolkit for constructing graphical and GUI-based programs.

Mastering Java's graphics requires *a lot* of experimentation.

However, it is much easier if you start with code examples and then modify them to do what you want.

Java's Graphics

Java's API provides a rich toolkit for constructing graphical and GUI-based programs.

Mastering Java's graphics requires *a lot* of experimentation.

However, it is much easier if you start with code examples and then modify them to do what you want.

Use of Java's GUI/graphics API classes requires the following package imports:

- `import java.awt.*;`
- `import javax.swing.*;`

Basics GUI/Graphics API Classes

The Java API includes a large number of GUI/graphics-related classes.

Some of the most useful are:

- `JFrame` — implements basic window objects.
- `Container` — used to group components of a complex GUI window.
- `Canvas` — a type of component that can display graphics.
- `JButton` — a button widget component.
- `Applet` — implements a web-viewable graphic object.

Creating a JFrame

Like any object, a JFrame object is created as follows:

```
JFrame myJFrame = new JFrame();
```

There is also a constructor that takes a `String` argument, which specifies a title for the `JFrame`:

```
JFrame myJFrame = new JFrame("My JFrame");
```

Creating a JFrame

Like any object, a JFrame object is created as follows:

```
JFrame myJFrame = new JFrame();
```

There is also a constructor that takes a `String` argument, which specifies a title for the `JFrame`:

```
JFrame myJFrame = new JFrame("My JFrame");
```

`JFrame` objects also have various configuration methods, to set their size, position and visibility. These are illustrated on the next slide.

Note that you need to call the method `setVisible(true)`, otherwise the `JFrame` will not be displayed.

JFrame Creation Example

The following example illustrates basic creation and configuration of a `JFrame`:

```
import java.awt.*;
import javax.swing.*;

class BasicJFrameExample {

    public static void main (String[] args) {

        JFrame jf = new JFrame("Hello");
        jf.setSize( 400, 400 );
        jf.setLocation( 200, 200 );
        jf.setVisible(true);
    }
}
```

The ContentPane

The graphical components displayed by a `JFrame` are embedded within an object of type `Container`, which is called the *Content Pane*.

This example shows how to get the 'content pane' of a `JFrame` and add a `JButton` widget:

```
class ContentPaneExample {
    public static void main (String[] args) {
        JFrame jf = new JFrame("JFrame with a JButton");
        jf.setSize( 400, 400 );
        jf.setLocation( 200, 200 );
        Container jfContentPane = jf.getContentPane();
        jfContentPane.add( new JButton("A Button") );
        jf.setVisible(true);
    }
}
```

Defining a JFrame Extension

While the `JFrame` class provides basic window functionality, it is too generic to be useful for any real application.

When using `JFrame`s one typically `extend`s the `JFrame` class in order to add specific functionality.

Typically, an extended `JFrame` will incorporate additional components displaying graphics or widgets (such as buttons).

The arrangement of components within a `JFrame` is typically controlled by a 'layout manager'. Use of the `FlowLayout` manager is illustrated on the next slide.

Example JFrame Extension


```
public class JFrameButtons extends JFrame {

   public JFrameButtons( ) {
      super(); // call constructor of parent (JFrame)
      Container pane = getContentPane();
      pane.setLayout(new FlowLayout());
      JButton but1 = new JButton("This is a button" );
      pane.add( but1 );
      JButton but2 = new JButton("Another button" );
      pane.add( but2 );
   }

   public static void main (String[] args) {
      JFrameButtons jfg = new JFrameButtons();
      jfg.setSize( 400, 200 );
      jfg.setVisible(true);
} }
```

Graphics in a JFrame

To display graphics in a `JFrame` one typically `extend`s a component such as `Canvas` onto which graphics can be painted.

The graphics are coded by *over-riding* the `paint(Graphics g)` method of the `Canvas` widget.

Graphics in a JFrame

To display graphics in a `JFrame` one typically `extend`s a component such as `Canvas` onto which graphics can be painted.

The graphics are coded by *over-riding* the `paint(Graphics g)` method of the `Canvas` widget.

This method is not called explicitly by the programmer, but will be automatically called by the `JFrame` object. Whenever it needs to display or refresh the contents of its window, the `JFrame` will call the `paint(Graphics g)` method of each component in its content pane (it will also supply the parameter `g`, which determines where and how the drawing commands are executed.)

Graphics in a JFrame

To display graphics in a `JFrame` one typically `extend`s a component such as `Canvas` onto which graphics can be painted.

The graphics are coded by *over-riding* the `paint(Graphics g)` method of the `Canvas` widget.

This method is not called explicitly by the programmer, but will be automatically called by the `JFrame` object. Whenever it needs to display or refresh the contents of its window, the `JFrame` will call the `paint(Graphics g)` method of each component in its content pane (it will also supply the parameter `g`, which determines where and how the drawing commands are executed.)

Thus the execution of `paint(Graphics g)` is triggered in a similar way to how an object's `toString()` method is called by API printing methods such as `System.out.println(...)`.

Example Extension of Canvas


```
import java.awt.*;
import javax.swing.*;

class MyCanvas extends Canvas {

  public void paint(Graphics g) {
     g.setColor(Color.magenta);
     g.drawRect(50, 50, 100, 60);
     g.fillOval(250, 90, 50, 100);
     g.setColor(Color.cyan);
     g.fillRect(155, 155, 40, 40);
  }
}
```

JFrame Incoporating a Canvas


```
import java.awt.*;  import javax.swing.*;

public class JFrameGraphic extends JFrame {

   public JFrameGraphic( String title ) {
      super(title);
      Container pane = this.getContentPane();
      MyCanvas  canvas = new MyCanvas();
      pane.add( canvas );
   }

   public static void main (String[] args) {
       JFrame.setDefaultLookAndFeelDecorated(true);
       JFrameGraphic jfg = new JFrameGraphic("Hello");
       jfg.setSize( 400, 400 );
       jfg.setLocation( 200, 200 );
       jfg.setVisible(true);
} }
```