Session 07:

GUI Programming

Dr Tomasz Kurowski
t.j.kurowski@cranfield.ac.uk

NetBeans

Introduction

- Until now, we have only used dialog boxes and command window for input and output.
  - e.g. System.out.println – to display results
    JOptionPane.showInputDialog – to get user input

- These approaches have their limitations (e.g. if you want to get 10 inputs from the user).

- Today we will learn about Java GUI programming

- First, the GUI components were bundled in a library called Abstract Windows Toolkit (AWT) (platform specific)

Swing components

- Platform independent user-interface components (Button, Textfield, TextArea, etc.)

- The AWT helper classes (Graphics, colour, Font, etc.) are used to give the “look and feel” specific to the operating system used.

Getting Started with GUI Programming
OUTLINE

- Frames
  - Creating frames, centering frames, adding components to frames
- Event-Driven Programming
  - Event Source, Listener, Listener Interface
- GUI (Graphical User Interface) Class Hierarchy
- JComponent
- Jbutton, JLabel
- JTextField, JTextLabel
- JRadioButton, JCheckBox, Borders and Menus

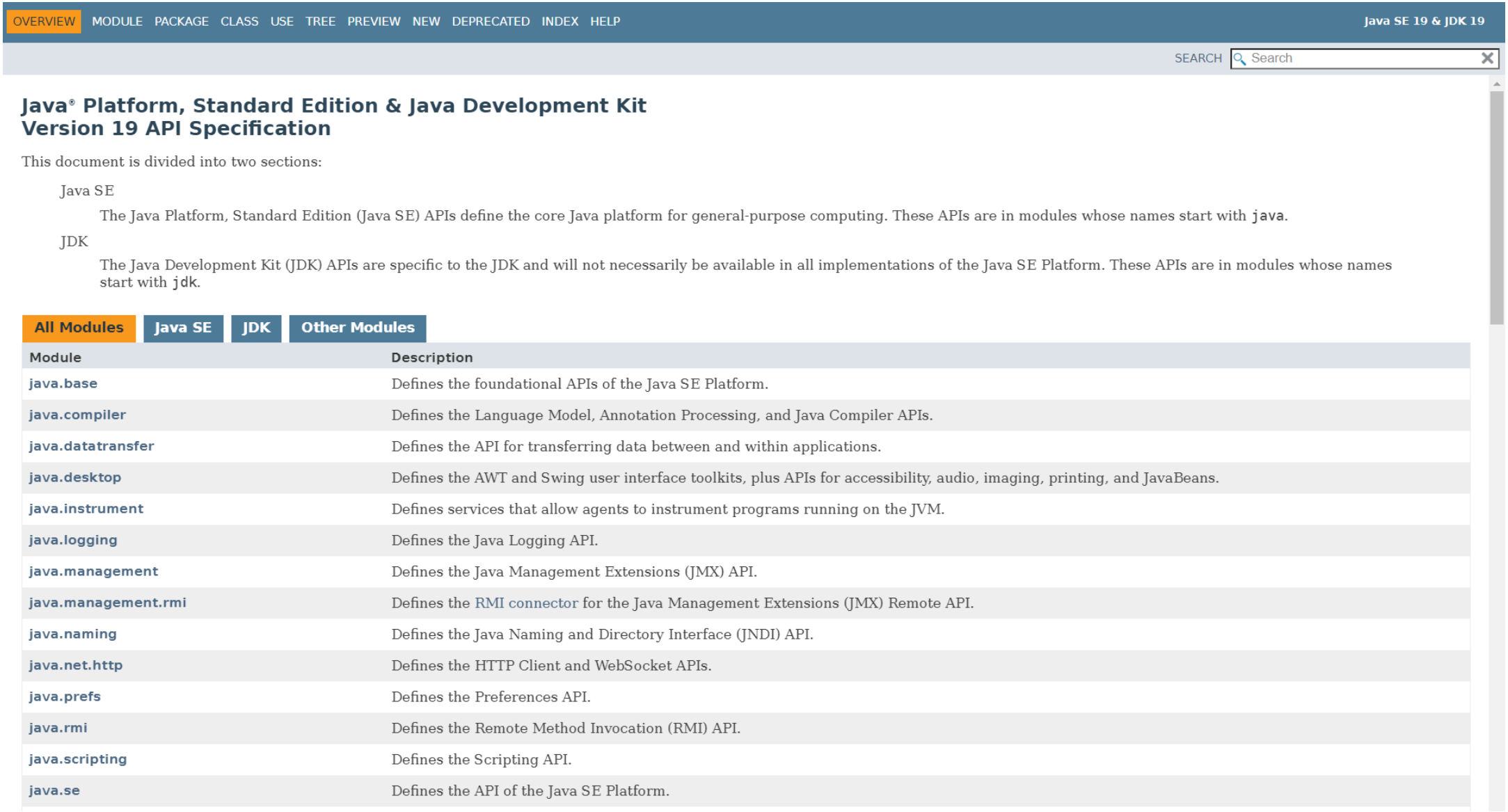
Javadoc

- Detailed documentation can be found online – we can't cover everything here!
- The official Java documentation can be accessed via the Oracle website on

  https://docs.oracle.com/en/java/javase/19/docs/api/index.html

  Java(TM) 19 Documentation

Javadoc



OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP Java SE 19 & JDK 19

SEARCH Search

Java® Platform, Standard Edition & Java Development Kit Version 19 API Specification

This document is divided into two sections:

Java SE

The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with `java`.

JDK

The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with `jdk`.

All Modules | Java SE | JDK | Other Modules

Module	Description
java.base	Defines the foundational APIs of the Java SE Platform.
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
java.datatransfer	Defines the API for transferring data between and within applications.
java.desktop	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.
java.instrument	Defines services that allow agents to instrument programs running on the JVM.
java.logging	Defines the Java Logging API.
java.management	Defines the Java Management Extensions (JMX) API.
java.management.rmi	Defines the RMI connector for the Java Management Extensions (JMX) Remote API.
java.naming	Defines the Java Naming and Directory Interface (JNDI) API.
java.net.http	Defines the HTTP Client and WebSocket APIs.
java.prefs	Defines the Preferences API.
java.rmi	Defines the Remote Method Invocation (RMI) API.
java.scripting	Defines the Scripting API.
java.se	Defines the API of the Java SE Platform.

Javadoc

OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP Java SE 19 & JDK 19

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH Search

Module java.base
Package java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>, Constable, ConstantDesc

---

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence, Constable, ConstantDesc
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

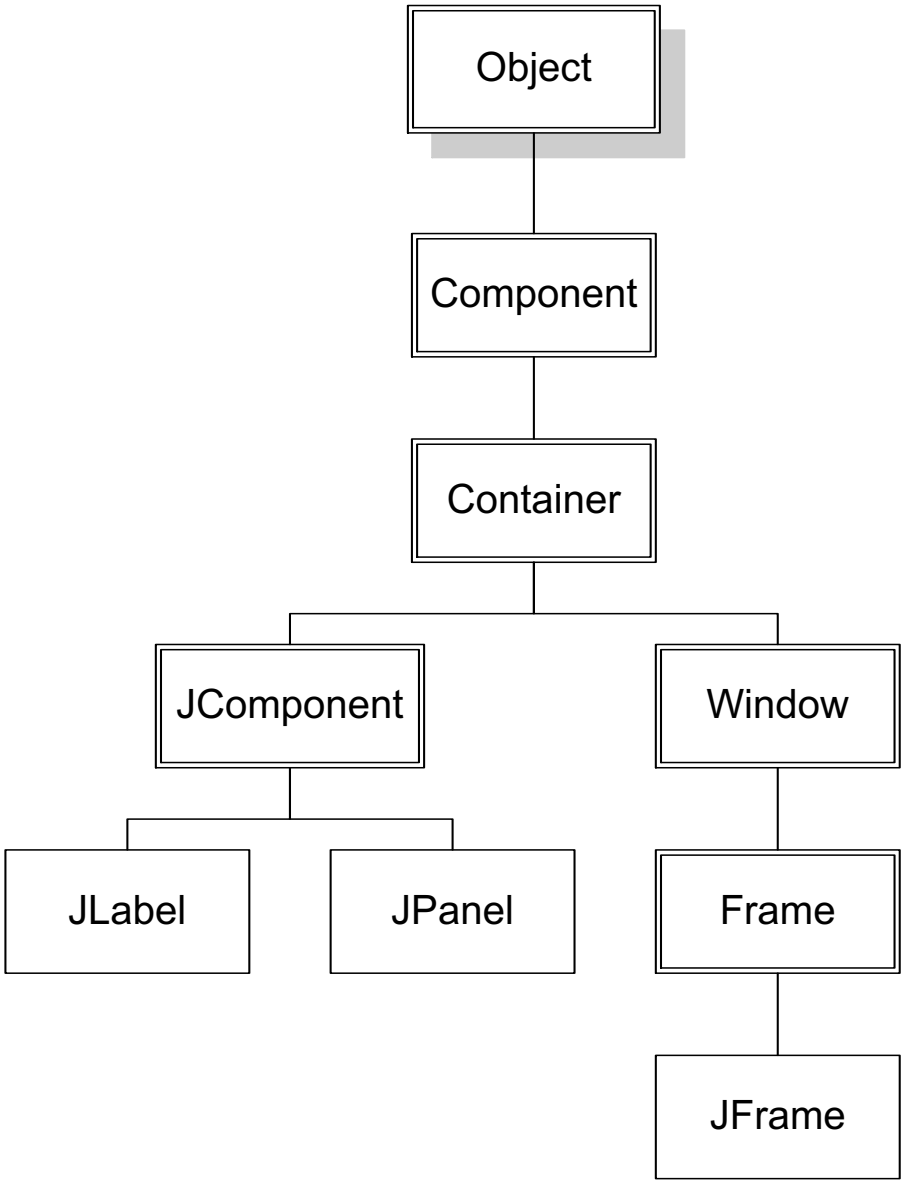
Here are some more examples of how strings can be used:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2, 3);
String d = cde.substring(1, 2);
```

The class `String` includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard version specified by the `Character` class.

The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. For additional information on string concatenation and conversion, see *The Java Language Specification*.

The Swing Class Hierarchy



Frames

- A Frame is a window that is not contained inside another window.
- A Frame is the base that contains other user interface components in Java graphical applications.
- The Frame class can be used to create windows.
- The ‘Swing’ GUI frame component is the JFrame class
- Java GUI classes inherit many methods from their superclasses, including the ability to draw themselves – the paintComponent() method.

Creating Frames

```
import javax.swing.*;
public class MyFrame {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Test Frame");
    frame.setSize(400, 300);
    frame.setVisible(true);
    frame.setDefaultCloseOperation(
          JFrame.EXIT_ON_CLOSE);
  }
}
```

Run (MyFrame)

Centering Frames

By default, a frame is displayed in the upper-left corner of the screen.

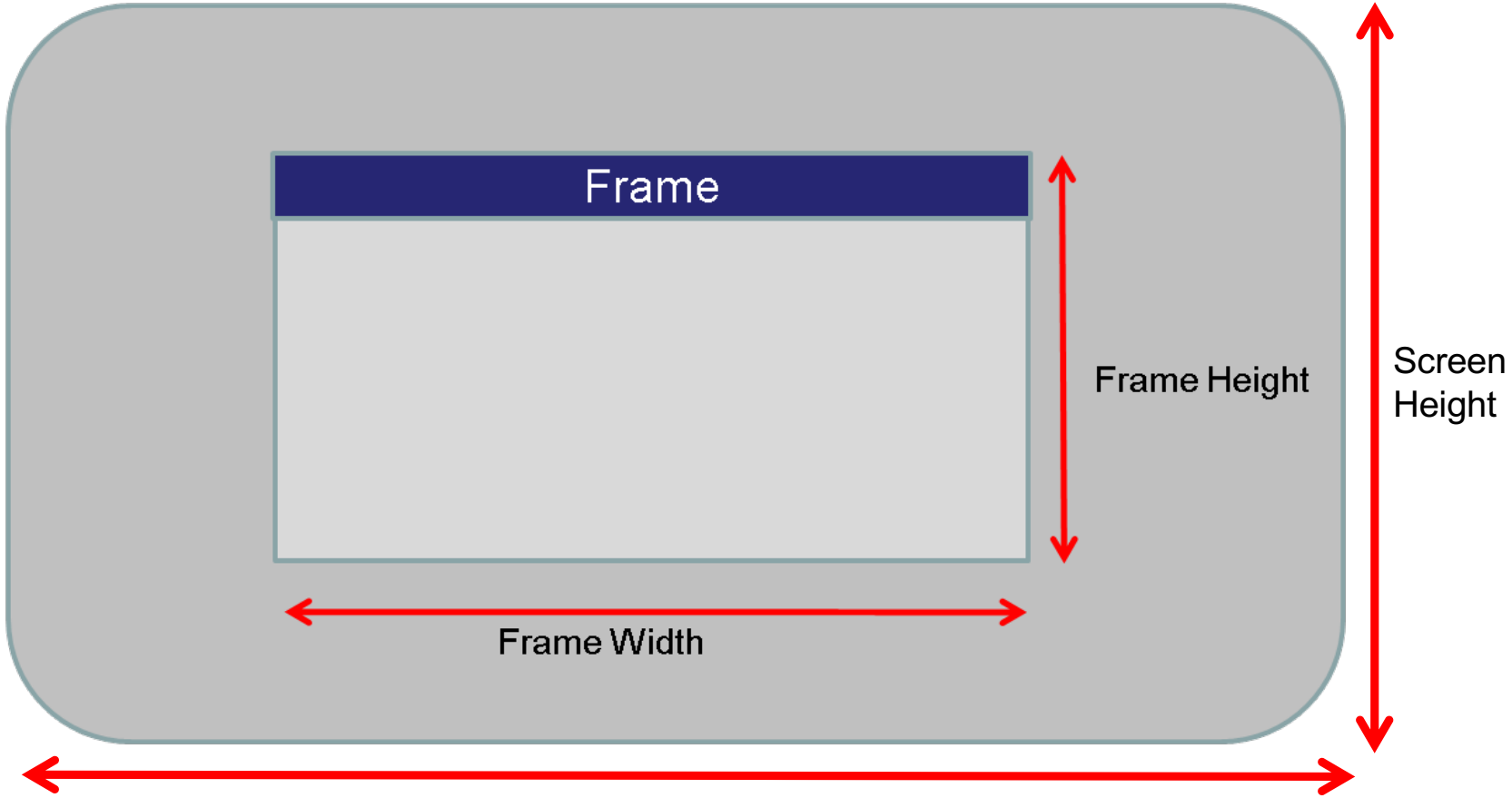
It is also displayed as small as possible – hence in example we call : *frame.setSize(400, 300);*

To display a frame at a specified location, you can use the *setLocation(x, y)* method in the JFrame class.

This method places the upper-left corner of a frame at location (x, y).

Example shows how to get screen size (Toolkit)

Centering Frames, cont.



CenterFrame

Run

Adding Components into a Frame

A Frame alone is not much use. We want to add GUI components such as Buttons, text boxes and labels

Components are always added to *Containers*. A Container is a Java class (object) that can contain other components.

A JFrame class (and all Swing components) has a Container associated with it. To get it, use the ***getContentPane ()*** method

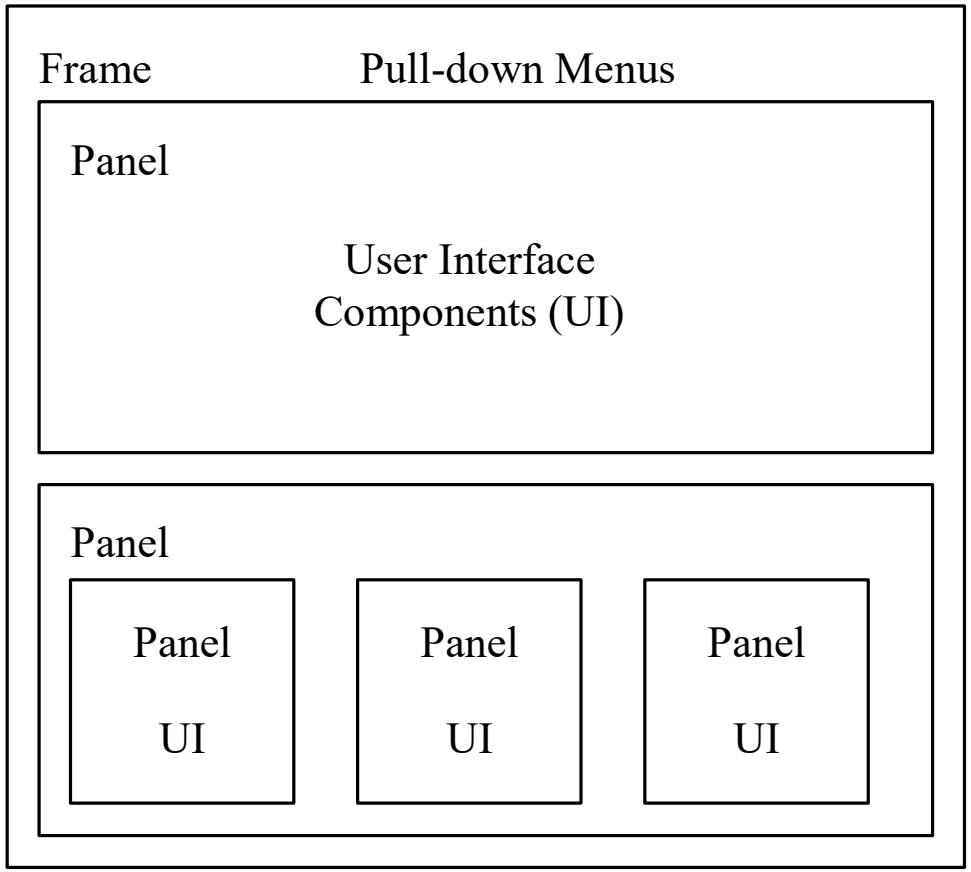
```
// Get the content pane of the frame
Container myContainer = frame.getContentPane();
myContainer.add (new JButton ("OK"))

// Alternative declaration and invocation :
frame.getContentPane().add(new JButton("OK"));
```

MyFrameWithComponents

Run

UI Components



Java GUI Structure

- Standard JFrame does not give us all required facilities
- Recommended method is to derive our own class by **extend**ing existing JFrame object, & adding custom data and methods :
  - **public class** myFrame **extends** JFrame {
- myFrame constructor – useful place to write code to add other components to frame
- Add additional methods – e.g JButton *actionPerformed ()* handling code
- Entry point : *main()* method can be included within our new class or placed in a separate class, – Netbeans prefers separate class & file

Java Gui Structure – Single Class

```
import javax.swing.JFrame;

public class myFrame extends JFrame {

   public myFrame () {                    /** Default constructor */
            setTitle("My frame Demo");
   }

…


   public static void main(String[] args) {  /** Main method */
       myFrame mf = new myFrame ();
       mf.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
       mf.setSize (250, 300);
       mf.setVisible (true);
   }
}
```

Java GUI Structure – Two Classes

```
import javax.swing.JFrame;

public class myFrame extends JFrame {
    public myFrame () {                          /** Default constructor */
              setTitle("My frame Demo");
      }
    …. other myFrame specific code
}

public class testFrame {
    public static void main(String[] args) { /** Main method */
           myFrame mf = new myFrame ();
           mf.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
           mf.setSize (250, 300);
           mf.setVisible (true);
      }
}
```

Using Panels as Containers

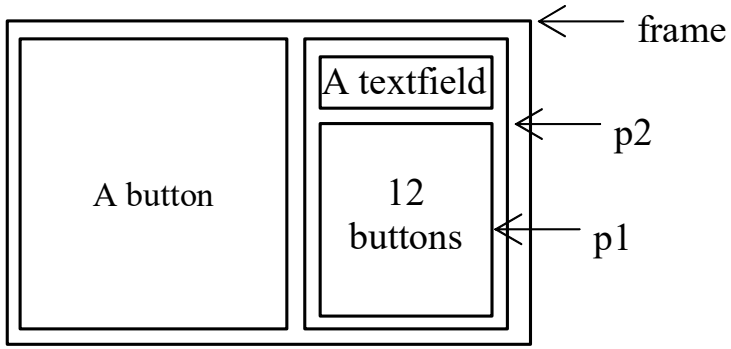
- Panels act as smaller containers for grouping user interface components.
- It is recommended that you place the user interface components in panels and place the panels in a frame.
- You can place panels inside a ‘holding‘ panel.
- Components can be directly added to a panel :
  - JPanel p1 = **new** JPanel();
  - p1.add (new JButton("Start"));

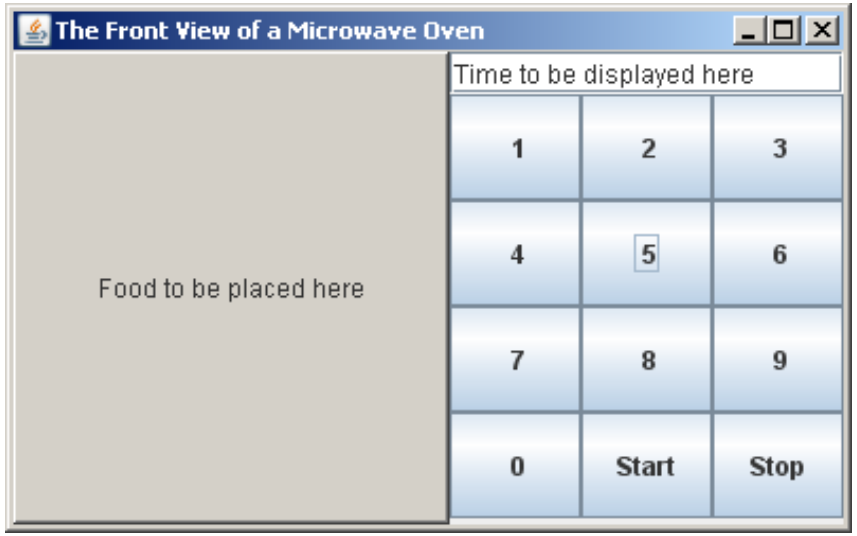
Example Testing Panel

This example uses panels to organize components.

The program creates a user interface for a Microwave oven.

p1 uses a grid layout, p2 uses a border layout





TestPanels

Run

Events

- An *event* is a type of signal to our program that something has happened.
- An event is generated by external actions such as mouse movements, mouse button clicks, keystrokes or by the operating system, such as a timer or a window closing
- Java implements a set of classes that encapsulates the behaviour of events - all derived from EventObject
- We must supply the code for the JVM to call when an event occurs. (Default – nothing happens !)
- Uses ‘Interface’ – Template code that specifies the Methods we must supply, but doesn’t give us the Base class to derive from.
- Java Classes can only extend (inherit) from one parent, but can implement a number of interfaces

Selected User Actions

There are numerous types of Events defined in Java.

These are some of the more commonly used :

User Action	Source Object	Event Type Generated
Clicked on a button	`JButton`	`ActionEvent`
Changed text	`JTextComponent`	`TextEvent`
Double-clicked on a list item	`JList`	`ActionEvent`
Selected or deselected an item with a single click	`JList`	`ItemEvent`
Selected or deselected an item	`JComboBox`	`ItemEvent`

Handling Events – HowTo

- Create a component that can generate an event E.g. JButton creates an ActionEvent when clicked.

```
private JButton btn1 = new JButton ("Click Here");
```

- Create a class that implements the listener interface for the event of interest :

```
public class Click extends JFrame implements
ActionListener
```

Handling Events – HowTo

- Provide the Method for the JVM to call when the event occurs. The ActionListener interface requires an actionPerformed Method

```
public void actionPerformed (ActionEvent e) {
    if (e.getSource() == btn1)
        System.out.println (“Button Cliked”);
}
```

- Register the listener with the source

```
btn1.addActionListener (this);
```

Example Handling Simple Action Events

- Objective: Display two buttons OK and Cancel in the window. A message is displayed on the console to indicate which button is clicked, when a button is clicked.

TestActionEvent

Run (NetBeans)

Netbeans GUI Editing

- Netbeans simplifies GUI design by handling layout and event management automatically.

- Event handlers automatically generated by Netbeans :
  - Add JButton object to JFrame form
  - Select (click on) JButton on form
  - Select ‘Events’ tab on properties window
  - Double click on ‘Action Performed’ (default button Event)
  - Netbeans add source code handler Method for that component. Called when user clicks on object
  - You add custom event handler code to Method body

The `Color` Class

```
Color c = new Color(r, g, b);
```

r, g, and b specify a color by its red, green, and blue components. r, g, b are ints in range 0 .. 255

The Color class defines some standard colour constants as public final static variables (constants) such as

```
Color.red, Color.blue, Color.green
```

Example:

```
Color c = new Color(128, 100, 100);
Color c1 = new Color (Color.red);
```

Setting Colors

You can use the following methods to set the component's background and foreground colors

These methods are defined in the basic Component object, and inherited by all its child classes

```
setBackground(Color c)

setForeground(Color c)
```

Example: **`setBackground(Color.yellow);`**

You can also use the `g.setColor(Color c)` method of the graphics class to set the 'pen' colour for all subsequent drawing operations

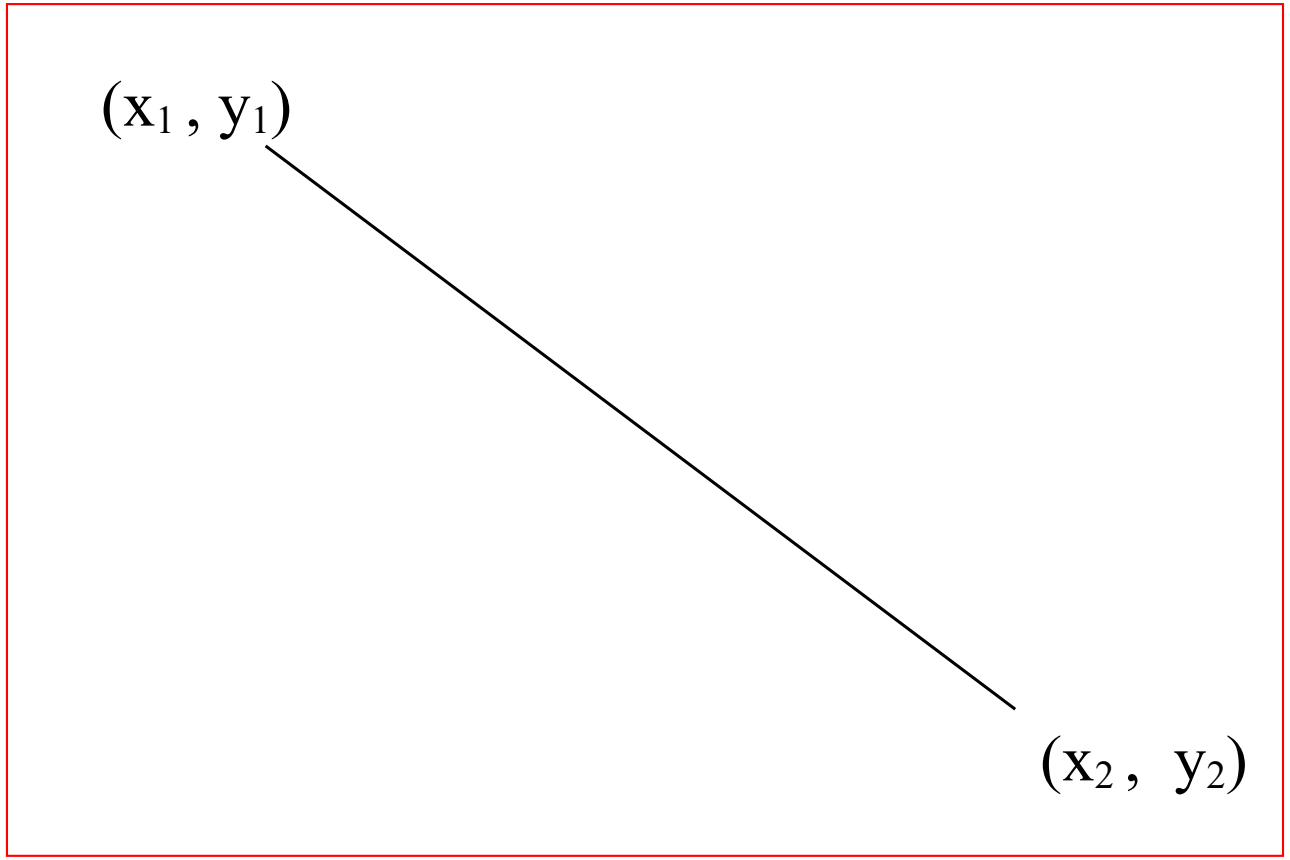
Drawing Geometric Figures

- These methods are all part of the Graphics class.
- You must have a Graphics object (provide by paintComponent() method) to use these.
- Drawing Lines
- Drawing Rectangles
- Drawing Ovals
- Drawing Arcs
- Drawing Polygons
- Most methods have two versions – one to draw an outline object – drawXXXX() , and one to draw a filled object - fillXXXX()
- Most methods are part of the *java.awt.Graphics* package.
  - Swing components still rely on the awt package

Drawing Lines

```
@Override
   public void paintComponent(Graphics g) {

       super.paintComponent(g);
       g.drawLine(x1, y1, x2, y2);
   }
```

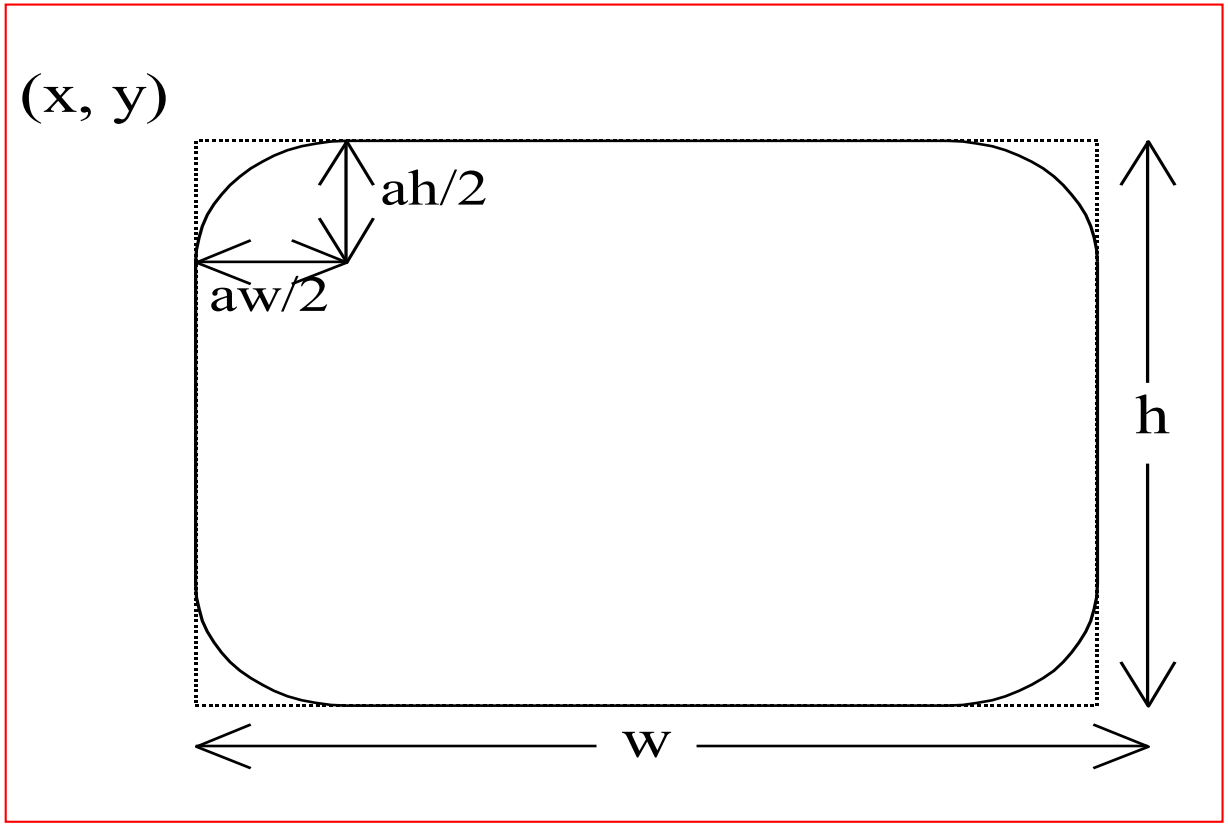


Drawing Rectangles

```
g.drawRect(x, y, w, h);
g.fillRect(x, y, w, h);
```

Drawing Rounded Rectangles

```
g.drawRoundRect(x, y, w, h, aw, ah);
g.fillRoundRect(x, y, w, h, aw, ah);
```

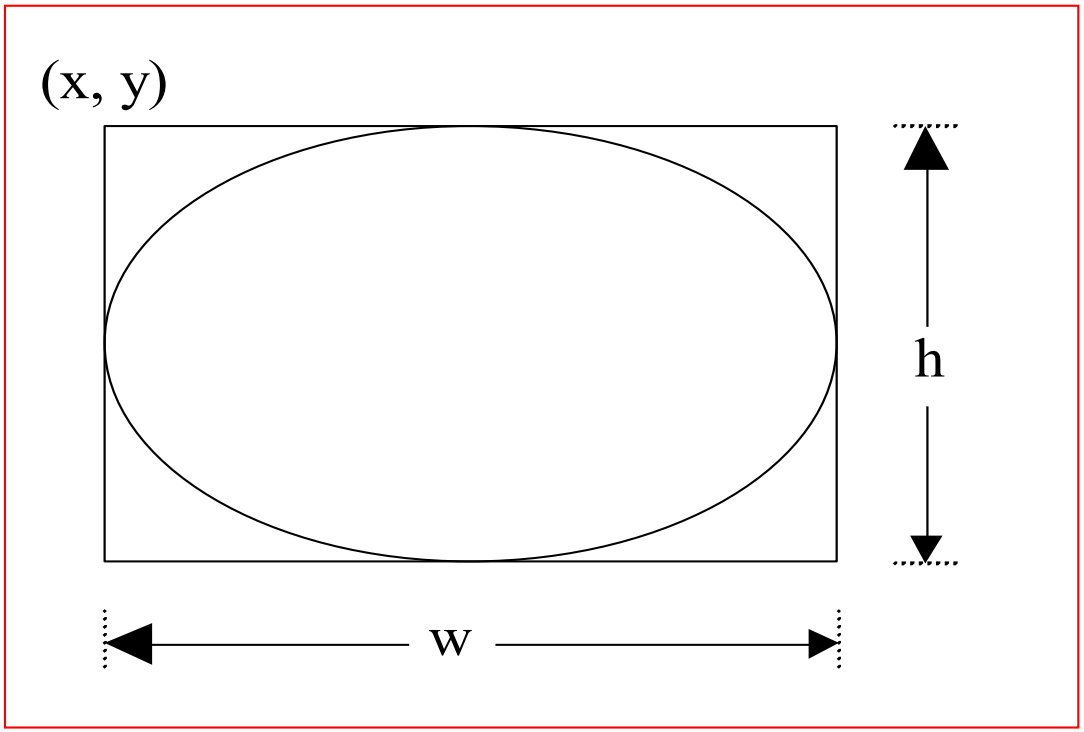


DrawRect

Run

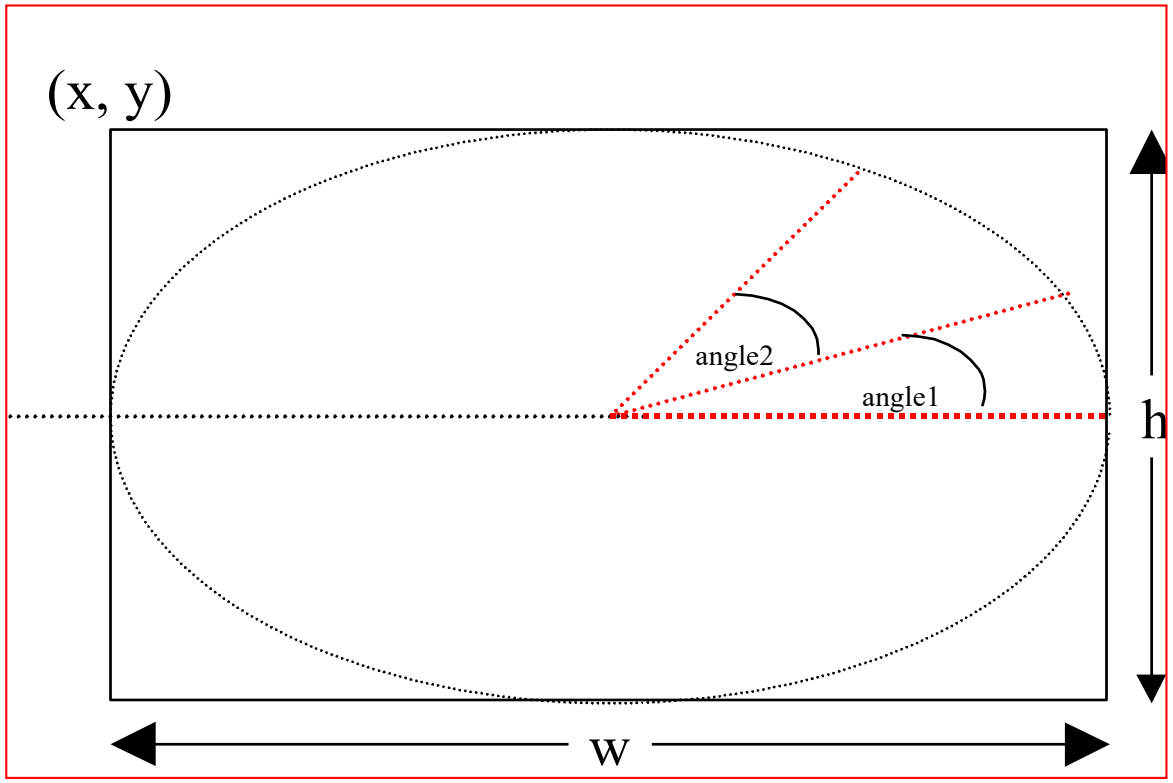
Drawing Ovals

```
g.drawOval(x, y, w, h);
g.fillOval(x, y, w, h);
```



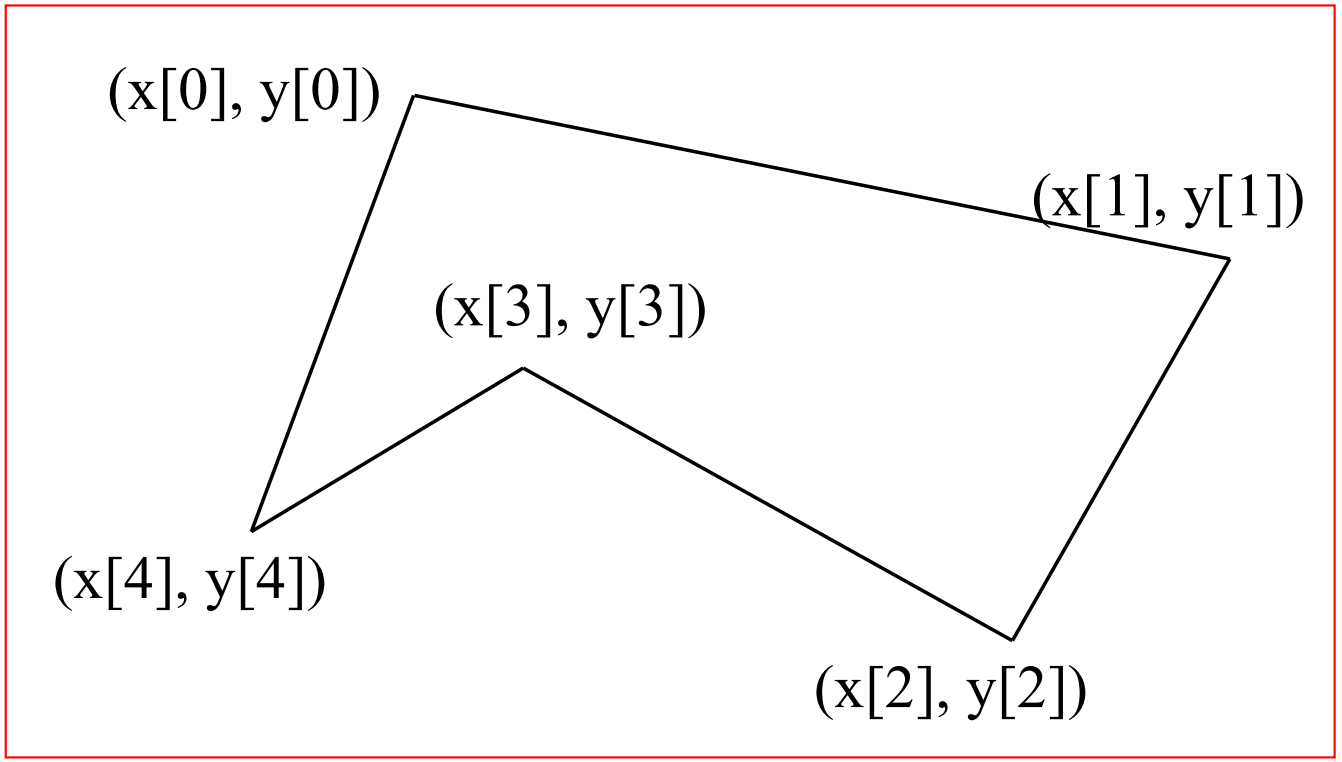
Drawing Arcs

```
g.drawArc(x, y, w, h, angle1, angle2);
g.fillArc(x, y, w, h, angle1, angle2);
```



Drawing Polygons

```
int[] x = {40, 70, 60, 45, 20};
int[] y = {20, 40, 80, 45, 60};
g.drawPolygon(x, y, x.length);
g.fillPolygon(x, y, x.length);
```



Example Drawing a Clock

- Objective: Use drawing and trigonometric methods to draw a clock showing the specified hour, minute, and second in a frame.