CSC1016S GUI Notes

* **Graphical User Interface (GUI):** Windowing system that interacts with the user
* In Java, the **AWT (Abstract Window Toolkit)** package was the original Java package for creating GUIs
* The **Swing** package is an improved version of AWT
* Some AWT classes are replaced by Swing classes, while others are still needed
* Swing is a simple package, ideal for learning about GUI
* The concepts of GUI programming are widely applicable:
  + Web development
  + App development
  + Used in popular programming environments

import javax.swing.[ClassName];

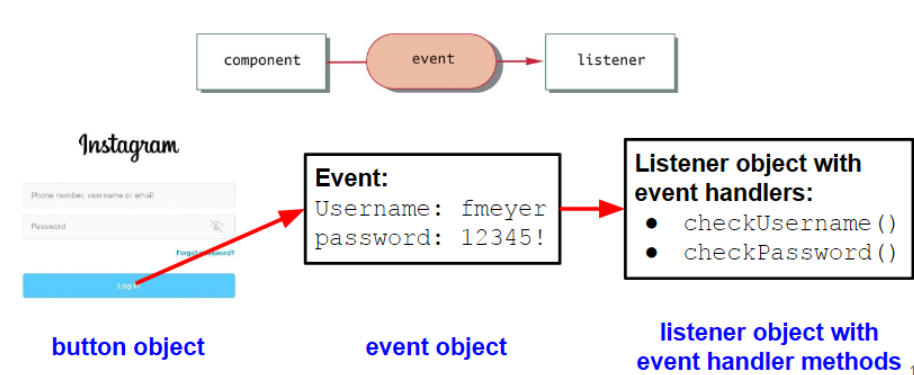
import java.awt.[ClassName];

# Event-Driven Programming

* **Event-driven programming**: programming style based on a signal-and-response approach
* Flow of control is determined by events such as user action (mouse clicks, key presses)
* Dominant paradigm used in GUIs and by web applications

## Events and Listeners

* How does it work?
  + Windows contain GUI components
  + Components send **events** to other objects called **listeners**
  + An **event** is an object that acts as a signal to a listener
  + A **listener** **object** performs some action in response to the event
    - Invokes an **event handler** with the event as an argument
  + The sending of an event is called **firing the event**
* A component may have several listeners
  + Each listener may respond to a different kind of event, or multiple listeners might respond to the same events
* What is an **event handler**?
  + Methods of the listener object that specify what happens when events are received
  + The programmer will define these event handlers methods
  + Example: checkUsername()



# JFrame

import **javax.swing.JFrame**;

JFrame firstWindow = new JFrame(“Demo program for JFrame”);

* Includes a border and the usual buttons for minimizing, changing the size of and closing the window
* The argument represents the heading of the window
* Can add additional components to the frame:

JButton button1 = new JButton("Click");

firstWindow.add(button1);

JLabel label1 = new JLabel("Hello");

firstWindow.add(label1)

* Can set properties of the frame:
  + firstWindow.**setSize**(100, 50);
    - Size of components is given in pixels
  + firstWindow.**setVisible**(true);
    - Must set the frame to visible as it is invisible by default
  + firstWindow.setTitle(“Updated Title”);

## Closing GUI Programs

* GUI components are based on a kind of infinite loop
* The windowing system normally stays on screen until the user indicates that it should go away
* In order to end a GUI program, System.exit(0) must be used when the user asks to end the program
* By default, clicking close would make the window invisible and inaccessible, but would not end the program
* Can use the following method:
  + firstWindow.**setDefaultCloseOperation**(JFrame.**EXIT\_ON\_CLOSE**);
  + There are other options along with EXIT\_ON\_CLOSE for other behaviour

# More Swing Components

## JTextField

JTextField in1 = new JTextField("Enter name", 20);

String inputString = in1.getText();

In1.setText("");

**JTextArea** allows multiple lines

## JMenu

A menu is an object of the class **JMenu**. A choice on a menu is called a menu item and is an object of the class **JMenuItem**.

JMenu menu = new JMenu("Menu");

JMenuItem item1 = new JMenuItem("MenuItem1");

item1.addActionListener(new ItemListener());

menu.add(item1);

**JMenuBar** is a container for menus, usually placed at the top of the windowing interface

JMenuBar bar = new JMenuBar();

bar.add(menu);

# Layout Managers

## Different Kinds of Swing Objects

Every Swing GUI should have all 3:

* **Components** refer to the basic elements of GUI. Each component represents a specific way for the user to interact with the GUI.
  + e.g. JButton, JLabel, JTextField, JTextArea, JMenu
* **Containers** serve as a holding space for GUI components. They can have components added to them.
  + e.g. JFrame, JPanel
  + Can have nested containers
* **Layout manager** is used to position the components inside the container
  + The add method adds components to a container, but does not specify arrangement
  + Layout manager can be used to describe how multiple containers are arranged
  + There are a number of layout manager classes:
    - BorderLayout (default)
    - FlowLayout
    - GridLayout
  + To add a layout manager to a JFrame named window:
    - window.setLayout(new BorderLayout());
  + Each container can have its own layout manager

## BorderLayout Manager

* Places components that are added into a JFrame object into 5 reagons
* One component per region
* Centre region expands to take up any unused space



## FlowLayout Manager

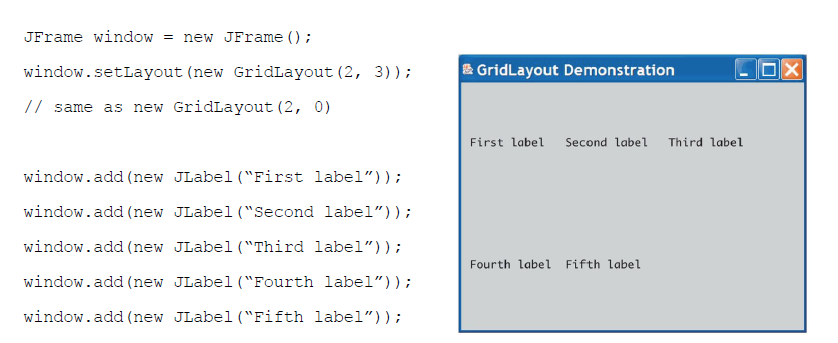
* Simplest
* Arranges components one after the other, going from left to right, in the order they are added
* Wraps to the next line if it runs out of space



## GridLayout Manager

* Arranges components in a 2D grid with some number of rows and columns
* Each entry is the same size
* Each component stretches so that it fills its grid position
* Items are placed in the grid from left to right, top to bottom

window.setLayout(new GridLayout(rows, columns));



### How GridLayout Handles Rows/Columns

setLayout(new GridLayout(rows, 0));

* Grid with specified number of rows and as many columns as required

setLayout(new GridLayout(rows, columns));

* Same - the column argument will essentially be ignored and it will use as many columns as required.

setLayout(new GridLayout(0, columns));

* Grid with specified number of columns and as many rows as required.

#### Examples:

A white background with black text

Description automatically generated

# Action Listeners

* In order to let a button do something, you need to:
  + Specify what objects are its listeners (**register listeners**)
  + Definite the methods that will be invoked automatically when the event is sent to the listener
* Buttons fire events known as action events, which are held by listeners called action listeners
* Can have multiple components linked to a single action listener
* A listener object buttonEar is created and registered as a listener for the button named button1:

EndingListener buttonEar = new EndingListener();

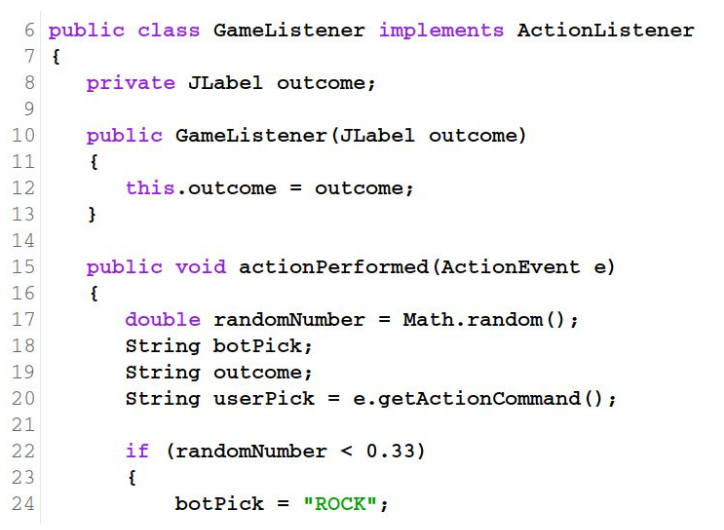
button1.addActionListener(buttonEar);

* An action listener class is defined as follows:

A screen shot of a computer

Description automatically generated

* The action listener receives the action event as the parameter **e** to its actionPerformed method (automatically invoked)
  + This is the only way we allow the action listener class to receive information about the event
* You can create a constructor in the action listener class that takes in a component and sets that to an instance variable



## Action Commands

* When a user clicks a button, an event is fired that normally goes to one or more action listeners
* This action even includes a String instance variable called the action command for the button or menu item
* String can be retrieved with e.**getActionCommand**()
* The default value of the action command string is the button text
* setActionCommand can be used to change the action command for a component
  + This is useful when multiple components have the same text
* Way to identify which button has been activated

A screen shot of a computer code

Description automatically generated

## Combining Action Listeners with the Main Class

A white box with black text

Description automatically generated

* This is useful when you need to change lots of components in the actionPerformed method
* Do not have to pass them all as parameters to the Listener class

public class SwingDemo extends JFrame implements ActionListener {

public SwingDemo {} // Constructor

public void actionPerformed(ActionEvent e) {} // Listener that can access instance variables

public static void man(String[] args) {} // Main method that calls the constructor

}

# Colours and Fonts

* A colour is an object of the class **Color** from the **java.awt** package
* The Color class has constants that represent basic colours
* Can set colour of GUI component with **setBackground**
  + button.setBackground(Colour**.PINK**);
* JFrame cannot be coloured directly
  + Must edit the **content pane**
  + frame.**getContentPane()**.setBackground(Color.Blue);

## Defining Custom Colours

* Done by specifying RGB values
* Integers or floats may be used to specify the amount of R/G/B
* Integers must be between 0-255 (inclusive)

Color brown = new Color(200, 150, 0)

* Float values must be between 0.0/1.0 (inclusive)

Color brown = new Color((float)(200.0/255),(float)(150.0/255),(float)0.0)

* Remember that floats are smaller versions of doubles (32 instead of 64 bits)

float num3 = 2.5; // won’t compile – Java assumes decimals are doubles

float num3 = (float) 2.5; // will compile – double is cast to float

* For this reason, you must use 0.5**f** or casting to specify float type
* There are get methods for the RGB values
* Also brighter/darker versions:

brown.brighter() // returns a brighter version

brown.darker() // returns a darker version

## JColorChooser

Part of **javax.swing** package

* showDialog method provides a colour choosing window
* Will return a colour:

Color sampleColor =JColorChooser.showDialog(frame, "window title", initialColor)

NOTE: returns null if no colour is selected

## Fonts

* A colour is an object of the class **Font** from the **java.awt** package

Font font1 = new Font("SansSerif", Font.PLAIN, SIZE) // specify font name, style modifier and size

* Any font available on a system can be specified as the font name
* Java guarantees "Monospaced", "SansSerif", and "Serif"
* Style modifiers:
  + Font.PLAIN
  + Font.BOLD
  + Font.ITALIC
* Multiple styles can be combined with | (or symbol)
* Character height specified in units known as points (1/72 of an inch)

### Setting Fonts

* **setFont()** method is used

Font f1 = new Font("Courier", Font.BOLD, 18);

JButton closeButton = new JButton("Close");

closeButton.setFont(f1);

# Window Listeners

* Window listeners handle events fire by window-related events
* Includes: opening, minimising, deactivating
* Defined as:

public class ClassName implements WindowListener

* Have to define seven methods, but you can define empty bodies for methods you don’t need
* Window event is an object of the class **WindowEvent** which is passed as an argument to all seven methods



* **addWindowListener** method can register a window listender to a JFrame window:

JFrame frame = new JFrame();

ExitListener listener = new ExitListener();

frame.addWindowListener(listener);

// OR

frame.addWindowListener(new ExitListener());

* **dispose()** method of the JFrame class eliminates the JFrame without ending the program
  + Resources consumed by the window and its components can be reused
  + Unless all elements are eliminated (i.e. this is the only window), the program continues
* **setDefaultCloseOperation** method of the JFrame class must still be set
  + Default is that clicking close would make the window invisible and inaccessible, but not end the program
  + ****Can set this to DO\_NOTHING\_ON\_CLOSE if you want a custom window listener

# Icons

* Labels, buttons and menu items can have icons (**in addition to or instead of a string**)
* An icon is usually just a small picture
* Icon is an object of the **ImageIcon** class
* Based on the digital picture files (.gif, .jpg)

ImageIcon wavingIcon = new ImageIcon(“waving.gif”);

* File name is given as a string and the class converts the picture file to a Swing icon
* Picture must be in the same directory unless a complete/relative path is given

JLabel label = new JLabel("Mood check");

label.setIcon(icon);

* Can be added to JButton and JMenuItem in the same way
* If the button/menu item has no text, you should use the setActionCommand method to explicitly set the action

## Insets

* Objects of the class Insets are used to specify the size of the margin in a button or menu item
* Arguments are given in pixels



# Scroll Bars

* When a text area is created, the number of lines that are visible and the number of characters per line are specified as follows:

JTextArea area = new JTextArea(15, 30);

* Often we do not want to set a firm limit – can be done with scroll bars
* Text is then viewed through a view port that shows only part of the text at a time
* When creating a JScrollPane, the text area **to be viewed** is given as an argument

JScrollPane scrolled = new JScrollPane(area);

panel.add(scrolled);

## Scroll Bar Policies

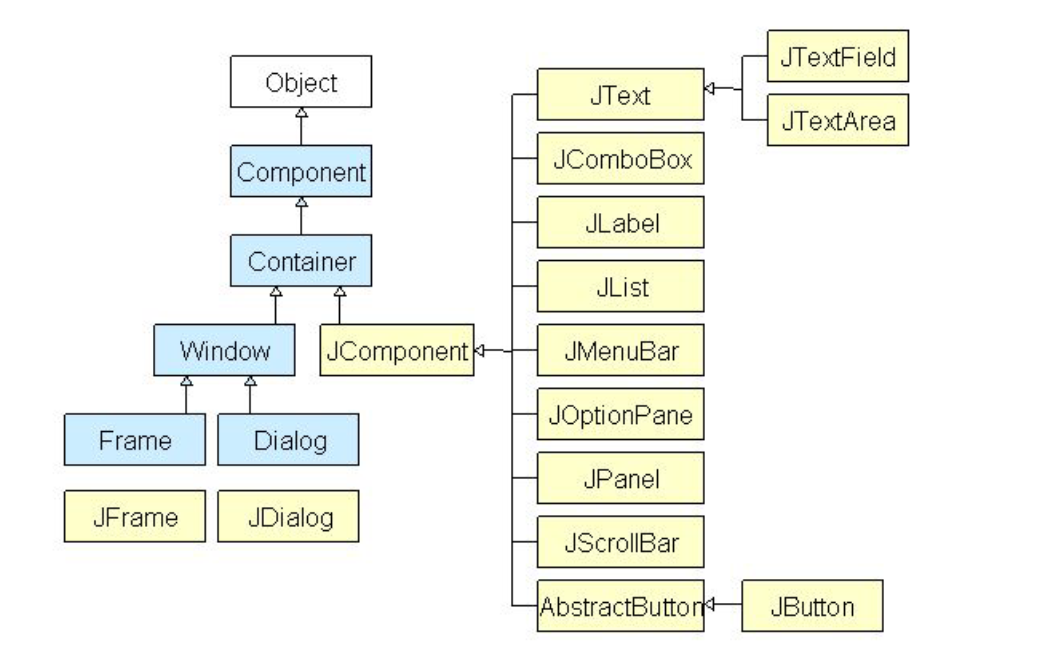
scrolled.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);

scrolled.setVerticalScrollBarPolicy(JscrollPane.VERTICAL\_SCROLLBAR\_ALWAYS);

* If these are omitted, scroll bars will only be visible when needed

# Swing Inheritance

* GUI components make use of inheritance
  + Base class: component with attributes like x, y, width, height
  + Derived classes: button, label, text field



* A better was to define a Swing window is to make it a derived class of the class JFrame
* This is the standard way to define a windowing interface, instead of creating all windows from scratch in the main class
* Examples:

public class LoginWindow extends JFrame

public class HomePage extends JFrame

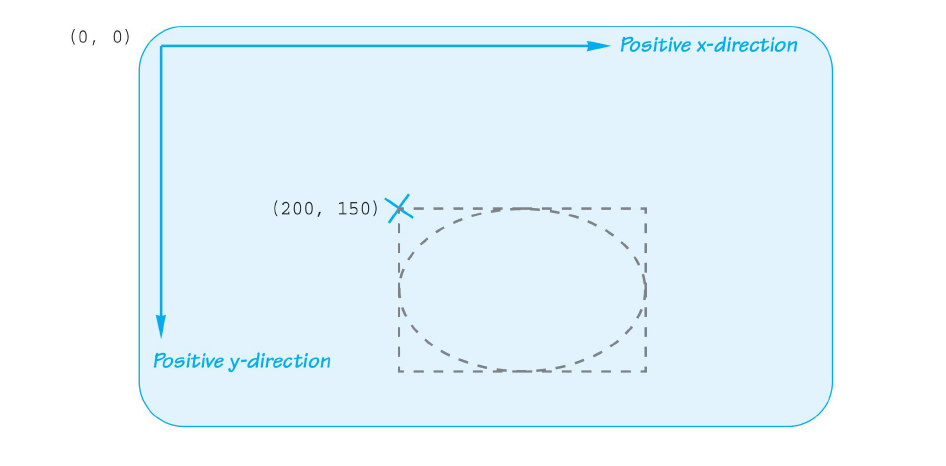
A screenshot of a computer program

Description automatically generated

* Almost all initialisation (adding components, registering listeners) for the window is placed in the constructor of the derived class

# Graphics

* Graphics class is found in the java.awt package
* Each component/container we have been working with has an associated Graphics object
* The object has data specifying the area of the screen covered by the component/container
* Java uses a drawing coordinate system where the origin (0,0) is the UPPER-LEFT corner
* Units are measured in pixels
* Area used for drawing is usually a JFrame or JPanel



* When drawing a rectangle, the upper left corner is specified
* For other figures, the upper left corner of a **bounding box** is specified

## Paint Method

* Swing components and containers have a method **paint**
* Draws the component or container on the screen
* Called automatically when the figure is displayed on the screen
  + When the user interface is created (e.g. with setVisible(true)), paint method is called with the appropriate Graphics instance
* Must be overwritten to draw geometric figures (circles, rectangles)

// How to redefine the paint method

public void paint (Graphics g)

{

super.paint(g); // STEP 1: Call parent constructor

// STEP 2: Call drawing methods

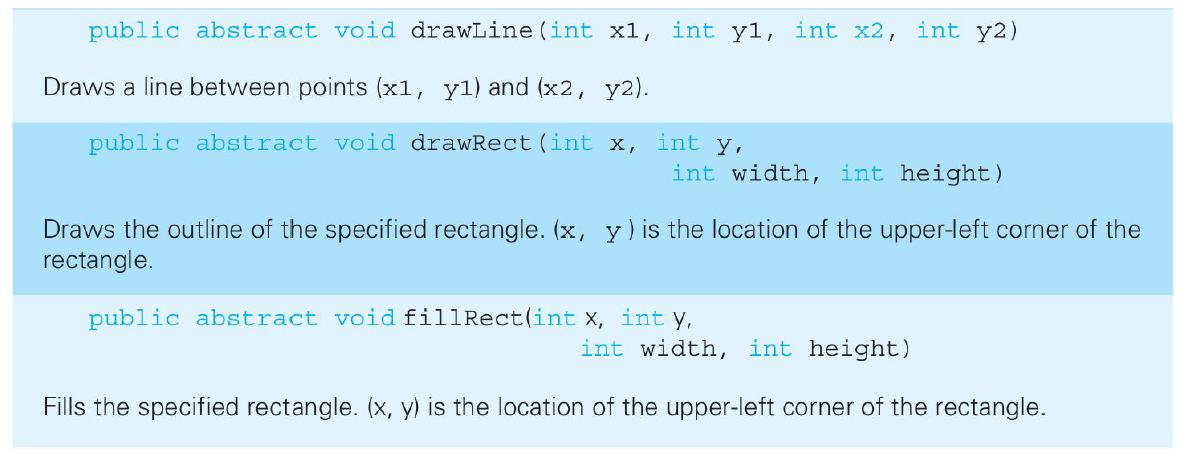
g.drawOval(...);

g.drawRect(...);

g.drawLine(...);

}

* g is the Graphics object associated with the JFrame
* Object g can be used as the calling object for drawing methods
* Drawing will take place instead the area of g



A blue screen with black text

Description automatically generated

* Arguments of an oval specify the smallest rectangle that can enclose the oval

A blue text with black text

Description automatically generated

Examples of Arc:

A diagram of a graph

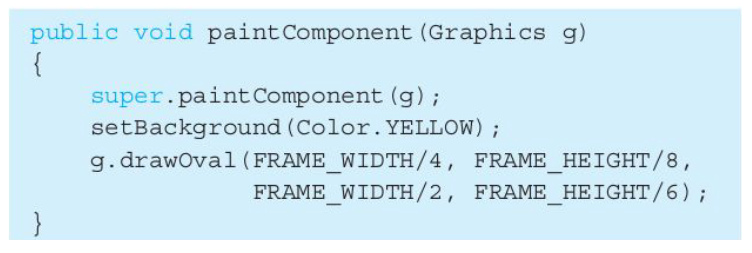
Description automatically generated with medium confidence

A blue and white text

Description automatically generated

## paintComponent Method

* This is used instead of paint when working with the JPanel



## repaint Method

* Used to change the content of a window
* Does things in the background, then invokes the method paint
* Must be explicitly invoked, unlike paint

## setColor Method

* Changes the colour of the pen
* Can change multiple times during the paint method, allowing you to draw with different colours

g.setColor(Color.BLUE)

# All Imports Used in These Notes

javax.swing.[ComponentName]

java.awt.Color

java.awt.Font

java.awt.Graphics

java.awt.event.ActionListener

java.awt.event.ActionEvent

java.awt.event.WindowEvent

java.awt.event.WindowListener

NOTE: can summarise to:

java.awt.\*

javax.swing.\*

java.awt.event.\*