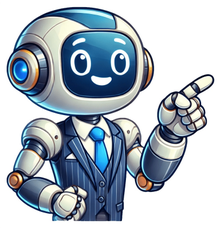
I'm not a bot



So far, we have covered the basic programming constructs (such as variables, data types, decision, loop, array and method) and introduced the important concept of Object-Oriented Programming (OOP). As discussed, OOP permits higher level of abstraction than traditional Procedural-Oriented Languages (such as C). You can create high-level abstract data types called classes to mimic real-life things. These classes are self-contained and are reusable. In this article, I shall show you how you can reuse the graphics classes provided in JDK for constructing your own Graphical User Interface (GUI) applications. Writing your own graphics classes (and re-inventing the wheels) is mission impossible! These graphics classes, developed by expert programmers, are highly complex and involve many advanced design patterns. However, re-using them are not so difficult, if you follow the API documentation, samples and templates provided. I shall assume that you have a good grasp of OOP, including composition, inheritance, polymorphism, abstract class and interface; otherwise, read the earlier articles. I will describe another important OO concept called nested class (or inner class) in this article. There are current three sets of Java APIs for graphics programming: AWT (Abstract Windowing Toolkit), Swing and JavaFX. AWT API was introduced in JDK 1.0. Most of the AWT UI components have become obsolete and should be replaced by newer Swing UI components. Swing API, a much more comprehensive set of graphics libraries that enhances the AWT, was introduced as part of Java Foundation Classes (JFC) after the release of JDK 1.1. JFC consists of Swing, Java2D, Accessibility, Internationalization, and Pluggable Look-and-Feel Support APIs. JFC has been integrated into core Java since JDK 1.2. The latest JavaFX, which was integrated into JDK 8, was meant to replace Swing. JavaFX was moved out from the JDK in JDK 11, but still available as a separate module. Other than AWT/Swing/JavaFX graphics APIs provided in JDK, other organizations/vendors have also provided graphics APIs that work with Java, such as Eclipse's Standard Widget Toolkit (SWT) (used in Eclipse), Google Web Toolkit (GWT) (used in Android), 3D Graphics API such as Java bindings for OpenGL (JOGL), Java3D, and etc. Furthermore, developers have moved to use technologies such as HTML5 as the basis of webapps. You need to refer to the "JDK API documentation" for the AWT/Swing APIs (under module java.desktop) while reading this chapter. The best online reference for Graphics programming is the "Swing Tutorial" @ . For advanced 2D graphics programming, read "Java 2D Tutorial" @ . For 3D graphics, read my 3D articles. Programming GUI with AWT I shall start with the AWT before moving into Swing to give you a complete picture of Java Graphics. AWT Packages AWT is huge! It consists of 12 packages of 370 classes (Swing is even bigger, with 18 packages of 737 classes as of JDK 8). Fortunately, only 2 packages - java.awt and java.awt.event - are commonly-used. The java.awt package contains the core AWT graphics classes: GUI Component classes, such as Button, TextField, and Label. GUI Container classes, such as Frame and Panel. Layout managers, such as FlowLayout, BorderLayout and GridLayout. Custom graphics classes, such as Graphics, Color and Font. The java.awt.event package supports event handling: Event classes, such as ActionEvent, MouseEvent, KeyEvent and WindowEvent, Event Listener Interfaces, such as ActionListener, MouseListener, MouseMotionListener, KeyListener and WindowListener, Event Listener Adapter classes, such as MouseAdapter, KeyAdapter, and WindowAdapter. AWT provides a platform-independent and device-independent interface to develop graphic programs that runs on all platforms, including Windows, macOS, and Unixes. AWT Containers and Components There are two groups of GUI elements: Component (Widget, Control): Components are elementary GUI entities, such as Button, Label, and TextField. They are also called widgets, controls in other graphics systems. Container: Containers, such as Frame and Panel, are used to hold components in a specific layout (such as FlowLayout or GridLayout). A container can also hold sub-containers. In the above figure, there are three containers: a Frame and two Panels. A Frame is the top-level container of an AWT program. A Frame has a title bar (containing an icon, a title, and the minimize/maximize/close buttons), an optional menu bar and the content display area. A Panel is a rectangular area used to group related GUI components in a certain layout. In the above figure, the top-level Frame contains two Panels. There are five components: a Label (providing description), a TextField (for users to enter text), and three Buttons (for user to trigger certain programmed actions). In a GUI program, a component must be kept (or added) in a container. You need to identify a container to hold the components. Every container has a method called add(Component c). A container (say aContainer) can invoke aContainer.add(aComponent) to add aComponent into itself. For example, Panel pnl = new Panel(); Button btn = new Button("Press"); pnl.add(btn); GUI components are also called controls (e.g., Microsoft ActiveX Control), widgets (e.g., Eclipse's Standard Widget Toolkit, Google Web Toolkit), which allow users to interact with (or control) the application. AWT Container Classes Top-Level Containers: Frame, Dialog and Applet Each GUI program has a top-level container. The commonly-used top-level containers in AWT are Frame, Dialog and Applet: A Frame provides the "main window" for your GUI application. It has a title bar (containing an icon, a title, the minimize, maximize/restore-down and close buttons), an optional menu bar, and the content display area. To write a GUI program, we typically start with a subclass extending from java.awt.Frame to inherit the main window as follows: import java.awt.Frame; public class MyGUIProgram extends Frame { ...... public MyGUIProgram() { ...... } public static void main(String[] args) { new MyGUIProgram(); }} An AWT Dialog is a "pop-up window" used for interacting with the users. A Dialog has a title-bar (containing an icon, a title and a close button) and a content display area, as illustrated. An AWT Applet (in package java.applet) is the top-level container for an applet, which is a Java program running inside a browser. Applet is no longer supported in most of the browsers. Secondary Containers: Panel and ScrollPane Secondary containers are placed inside a top-level container or another secondary container. AWT provides these secondary containers: Panel: a rectangular box used to layout a set of related GUI components in pattern such as grid or flow. ScrollPane: provides automatic horizontal and/or vertical scrolling for a single child component. others. Hierarchy of the AWT Container Classes The hierarchy of the AWT Container classes is as follows: As illustrated, a Container has a LayoutManager to layout the components in a certain pattern, e.g., flow, grid. AWT Component Classes AWT provides many ready-made and reusable GUI components in package java.awt. The frequently-used are: Button, TextField, Label, Checkbox, CheckboxGroup (radio buttons), List, and Choice, as illustrated below. AWT GUI Component: java.awt.Label A java.awt.Label provides a descriptive text string. Take note that System.out.println() prints to the system console, NOT to the graphics screen. You could use a Label to label another component (such as text field) to provide a text description. Check the JDK API specification for java.awt.Label. Constructors public Label(String strLabel, int alignment); public Label(String strLabel); public Label(); The Label class has three constructors: The first constructor constructs a Label object with the given text string in the given alignment. Note that three static constants Label.LEFT, Label.RIGHT, and Label.CENTER are defined in the class for you to specify the alignment (rather than asking you to memorize arbitrary integer values). The second constructor constructs a Label object with the given text string in default of left-aligned. The third constructor constructs a Label object with an initially empty string. You could set the label text via the setText() method later. Constants (final static fields) public static final LEFT; public static final RIGHT; public static final CENTER; These three constants are defined for specifying the alignment of the Label's text, as used in the above constructor. Public Methods public String getText(); public void setText(String strLabel); public int getAlignment(); public void setAlignment(int alignment); The getText() and setText() methods can be used to read and modify the Label's text. Similarly, the getAlignment() and setAlignment() methods can be used to retrieve and modify the alignment of the text. Constructing a Component and Adding the Component into a Container Three steps are necessary to create and place a GUI component: Declare the component with an identifier (name); Construct the component by invoking an appropriate constructor via the new operator; Identify the container (such as Frame or Panel) designed to hold this component. The container can then add this component onto itself via aContainer.add(aComponent) method. Every container has a add(Component) method. Take note that it is the container that actively and explicitly adds a component onto itself, NOT the other way. Example Label lblInput; lblInput = new Label("Enter ID"); add(lblInput); lblInput.setText("Enter password"); lblInput.getText(); An Anonymous Label Instance You can create a Label without specifying an identifier, called anonymous instance. In the case, the Java compiler will assign an anonymous identifier for the allocated object. You will not be able to reference an anonymous instance in your program after it is created. This is usually alright for a Label instance as there is often no need to reference a Label after it is constructed. Example add(new Label("Enter Name: ", Label.RIGHT)); Label xxx = new Label("Enter Name: ", Label.RIGHT)); add(xxx); AWT GUI Component: java.awt.Button A java.awt.Button is a GUI component that triggers a certain programmed action upon clicking. Constructors public Button(String btnLabel); public Button(); The Button class has two constructors. The first constructor creates a Button object with the given label painted over the button. The second constructor creates a Button object with no label. Public Methods public String getLabel(); public void setLabel(String btnLabel); public void setEnable(boolean enable); The getLabel() and setLabel() methods can be used to read the current label and modify the label of a button, respectively. Note: The latest Swing's JButton replaces getLabel()/setLabel() with getText()/setText() to be consistent with all the components. We will describe Swing later. Event Clicking a button fires a so-called ActionEvent and triggers a certain programmed action. I will explain event-handling later. Example Button btnColor = new Button("Red"); add(btnColor); ...btnColor.setLabel("Green"); btnColor.getLabel(); AWT GUI Component: java.awt.TextField A java.awt.TextField is single-line text box for users to enter texts. (There is a multiple-line text box called TextArea.) Hitting the "ENTER" key on a TextField object fires an ActionEvent. Constructors public TextField(String initialText, int columns); public TextField(String initialText); public TextField(int columns); Public Methods public String getText(); public void setText(String strText); public void setEditable(boolean editable); Event Hitting the "ENTER" key on a TextField fires a ActionEvent, and triggers a certain programmed action. Example TextField tfInput = new TextField(30); add(tfInput); TextField tfResult = new TextField(); tfResult.setEditable(false) ; add(tfResult); ......int number = Integer.parseInt(tfInput.getText());number *= number;tfResult.setText(number + ""); Take note that getText()/SetText() operates on String. You can convert a String to a primitive, such as int or double via static method Integer.parseInt() or Double.parseDouble(). To convert a primitive to a String, simply concatenate the primitive with an empty String. Example 1: AWTCounter Let's assemble a few components together into a simple GUI counter program, as illustrated. It has a top-level container Frame, which contains three components - a Label "Counter", a non-editable TextField to display the current count, and a "Count" Button. The TextField shall display count of 0 initially. Each time you click the button, the counter's value increases by 1. 1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859606162636465666768 import java.awt.*; import java.awt.event.*; public class AWTCounter extends Frame { private Label lblCount; private TextField tfCount; private Button btnCount; private int count = 0; public AWTCounter () { setLayout(new FlowLayout()); lblCount = new Label("Counter"); add(lblCount); tfCount = new TextField(count + "", 10); tfCount.setEditable(false); add(tfCount); btnCount = new Button("Count"); add(btnCount); btnCount.addActionListener(new BtnCountListener()); //BtnCountListener listener = new BtnCountListener(); //btnCount.addActionListener(listener); setTitle("AWT Counter"); setSize(300, 100); setVisible(true); } public static void main(String[] args) { AWTCounter app = new AWTCounter(); } private class BtnCountListener implements ActionListener { @Override public void actionPerformed(ActionEvent evt) { ++count; tfCount.setText(count + ""); } } } To exit this program, you have to close the CMD-shell (or press "control-c" on the CMD console); or push the "red" close button in Eclipse's Application Console. This is because we have yet to write the handler for the Frame's close button. We shall do that in the later example. Dissecting the AWTCounter.java The import statements (Lines 1-2) are needed, as AWT container and component classes, such as Frame, Button, TextField, and Label, are kept in the java.awt package; while AWT events and event-listener interfaces, such as ActionEvent and ActionListener are kept in the java.awt.event package. A GUI program needs a top-level container, and is often written as a subclass of Frame (Line 5). In other words, this class AWTCounter is a Frame, and inherits all the attributes and behaviors of a Frame, such as the title bar and content pane. Lines 11 to 47 define a constructor, which is used to setup the GUI components and event handlers. In Line 13, the setLayout() (inherited from the superclass Frame) is used to set the layout of the container. FlowLayout is used which arranges the components in left-to-right and flows into next row in a top-to-bottom manner. A Label, TextField (non-editable), and Button are constructed. We invoke the add() method (inherited from the superclass Frame) to add these components into container. In Line 36-37, we invoke the setSize() and the setTitle() (inherited from the superclass Frame) to set the initial size and the title of the Frame. The setVisible(true) method (Line 42) is then invoked to show the display. Line 27 (or Line 29-30) is used to setup the callback event-handler, which will be discussed in length later. In brief, whenever the button is clicked, the actionPerformed() will be called. In the actionPerformed() (Lines 61-66), the counter value increases by 1 and displayed on the TextField. In the entry main() method (Lines 52-56), an instance of AWTCounter is constructed. The constructor is executed to initialize the GUI components and setup the event-handlers. The GUI program then waits for the user action. Inspecting Container/Components via toString() It is interesting to inspect the GUI objects via the toString(), to gain an insight to these classes. (Alternatively, use a graphic debugger in Eclipse/NetBeans or study the JDK source code.) For example, if we insert the following code before and after the setVisible(): System.out.println(this); System.out.println(lblCount); System.out.println(tfCount); System.out.println(btnCount); setVisible(true); System.out.println(this);System.out.println(lblCount);System.out.println(tfCount);System.out.println(btnCount); Example 2: AWTAccumulator In this example, the top-level container is again the typical java.awt.Frame. It contains 4 components: a Label "Enter an Integer", a TextField for accepting user input, another Label "The Accumulated Sum is", and another non-editable TextField for displaying the sum. The components are arranged in GridLayout of 2 rows 2 columns. The program shall accumulate the number entered into the input TextField and display the sum in the output TextField. 1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859 60 import java.awt.*; import java.awt.event.*; public class AWTAccumulator extends Frame { private TextField tfInput; private TextField tfOutput; private int sum = 0; public AWTAccumulator() { setLayout(new GridLayout(2, 2)); add(new Label("Enter an Integer: ")); tfInput = new TextField(10); add(tfInput); tfInput.addActionListener(new TFInputListener()); add(new Label("The Accumulated Sum is: ")); tfOutput = new TextField(10); tfOutput.setEditable(false); add(tfOutput); setTitle("AWT Accumulator"); setSize(350, 120); setVisible(true); } public static void main(String[] args) { new AWTAccumulator(); } private class TFInputListener implements ActionListener { @Override public void actionPerformed(ActionEvent evt) { int numberIn = Integer.parseInt(tfInput.getText()); sum += numberIn; tfInput.setText(""); tfOutput.setText(sum + ""); } }} Dissecting the AWTAccumulator.java An AWT GUI program extends from java.awt.Frame (Line 6) - the top-level window container. In the constructor (Line 14), we constructs 4 components - 2 anonymous java.awt.Labels and 2 java.awt.TextFields. The Frame adds the components, in GridLayout. tfInput (TextField) is the source object, which fires an ActionEvent upon hitting the Enter key. tfInput adds an anonymous instance of TFInputListener as an ActionEvent handler (Line 23). The listener class needs to implement ActionListener interface and provides implementation to method actionPerformed(). Whenever an user hits Enter on the tfInput (TextField), the actionPerformed() will be called back. AWT Event-Handling Java adopts the so-called "Event-Driven" (or "Event-Delegation") programming model for event-handling, similar to most of the visual programming languages like Visual Basic. In event-driven programming, a piece of event-handling codes is executed (or called back by the graphics subsystem) when an event was fired in response to an user input (such as clicking a mouse button or hitting the ENTER key in a text field). Callback Methods In the above examples, the method actionPerformed() is known as a callback method. In other words, you never invoke actionPerformed() in your codes explicitly. The actionPerformed() is called back by the graphics subsystem under certain circumstances in response to certain user actions. JavaScript can attach a Callback method to an Event Directly In some languages, you can directly attach a method (or function) to an event (such as mouse-click). For example, the following JavaScript code (called JSCounter.html) implement a counter similar to the AWTCounter, with a text label, text field and button: JavaScript Counter Count: function myBtnOnClick() { tfCount.value++; } function myBtnMouseOver() { btnCount.value = "Count Up"; } function myBtnMouseOut() { btnCount.value = "Click Me"; } btnCount.onmouseout = myBtnMouseOut; In Java, we CANNOT attach a method to a source object directly, as method is not a first-class object in Java. For example, a Java method cannot accept methods as its arguments and it cannot return a method; you cannot assign a method to a variable, etc. (JavaScript and C language CAN!). Source, Event and Listener Objects The AWT's event-handling classes are kept in package java.awt.event. Three kinds of objects are involved in the event-handling: a source, listener(s) and an event object. The source object (such as Button and Textfield) interacts with the user. Upon triggered, the source object creates an event object to capture the action (e.g., mouse-click x and y, texts entered, etc). This event object will be messaged to all the registered listener object(s), and an appropriate event-handler method of the listener(s) is called-back to provide the response. In other words, triggering a source fires an event to all its listener(s), and invoke an appropriate event handler of the listener(s). To express interest for a certain source's event, the listener(s) must be registered with the source. In other words, the listener(s) "subscribes" to a source's event, and the source "publishes" the event to all its subscribers upon activation. This is known as subscribe-publish or observable-observer design pattern. The sequence of steps is illustrated above: The source object registers its listener(s) for a certain type of event. A source fires an event when triggered. For example, clicking a Button fires an ActionEvent, clicking a mouse button fires MouseEvent, typing a key fires KeyEvent, and etc. How the source and listener understand each other? The answer is via an agreed-upon interface. For example, if a source is capable of firing an event called XxxEvent (e.g., ActionEvent). Firstly, we need to declare an interface called XxxListener (e.g., ActionListener) containing the names of the handler methods (recall that an interface contains only abstract methods without implementation). For example, the ActionListener interface is declared as follows: public interface ActionListener { public void actionPerformed(ActionEvent evt); } Secondly, all XxxEvent listeners must implement the XxxListener interface. That is, the listeners must provide their own implementations (i.e., programmed responses) to all the abstract methods declared in the XxxListener interface. In this way, the listener(s) can response to these events appropriately. For example, class MyActionListener implement ActionListener { @Override public void actionPerformed(ActionEvent evt) { System.out.println("ActionEvent detected!"); }} Thirdly, in the source, we need to maintain a list of XxxEvent listener object(s), and define two methods: addXxxListener() and removeXxxListener() to add and remove a XxxEvent listener from this list. For the ActionEvent, the signature of the methods are: public void addActionListener(ActionListener lis);public void removeActionListener(ActionListener lis); Take note that the addXxxListener() takes a XxxListener object as its sole parameter. In other words, it can only add objects of the type XxxListener, and its sub-type. Since XxxListener is an interface, you cannot create instance of XxxListener, but need to create instance of a subclass implementing the XxxListener interface. In summary, we identify the source, the event-listener interface, and the listener object. The listener must implement the event-listener interface. The source object then registers listener object via the addXxxListener(XxxListener lis) method. The source is triggered by a user. The source create a XxxEvent object, which encapsulates the necessary information about the activation. For example, the (x, y) position of the mouse pointer, the text entered, etc. Finally, for each of the XxxEvent listeners in the listener list, the source invokes the appropriate handler on the listener(s), which provides the programmed response. In summary, triggering a source fires an event to all its registered listeners, and invoke an appropriate handler of the listener. Revisit Example 1 AWTCounter: ActionEvent and ActionListener Interface Clicking a Button (or hitting the "Enter" key on a TextField) fires an ActionEvent to all its ActionEvent listener(s). An ActionEvent listener must implement the ActionListener interface, which declares one abstract method called actionPerformed() as follow: public interface ActionListener { public void actionPerformed(ActionEvent evt); } Here are the event-handling steps: We identify btnCount (of Button) as the source object. Clicking Button fires an ActionEvent to all its ActionEvent listener(s). The listener(s) is required to implement ActionListener interface, and override the actionPerformed() method to provide the response. In Line 56-65, we write an inner class called BtnCountListener, which override the actionPerformed() to increment and display the count. An inner class is a class defined inside an outer class, and it can access the private entities of the outer class. We will elaborate on the inner class in the next section. The source object registers listener via the addActionListener(). In this example, the source btnCount (Button) adds an instance of BtnCountListener as a listener via: BtnCountListener listener = new BtnCountListener();btnCount.addActionListener(listener); Note that addActionListener() takes an argument of the type ActionListener. BtnCountListener, which implements ActionListener interface (i.e., a subclass of ActionListener), is upcasted and passed to the addActionListener() method. Upon button-click, the btnCount creates an ActionEvent object, and calls back the actionPerformed(ActionEvent) method of all its registered listener(s) with the ActionEvent object created: ActionEvent evt = new ActionEvent(......); listener.actionPerformed(evt); The sequence diagram is as follows: Revisit Example 2 AWTAccumulator: ActionEvent and ActionListener Interface In this example, We identify the tfInput (of TextField) as the source object. Hitting the "Enter" key on a TextField fires an ActionEvent to all its ActionEvent listener(s). In Line 46-59, we define an inner class called TFInputListener as the ActionEvent listener.The ActionEvent listener is required to implement the ActionListener interface, and override the actionPerformed() method to provide the programmed response upon activation. The source object tfInput (of TextField) registers an anonymous instance of TFInputListener as its ActionEvent listener via the tfInput.addActionListener(new TFInputListener()) (Line 23). Example 3: WindowEvent and WindowListener Interface A WindowEvent is fired (to all its WindowEvent listeners) when a window (e.g., Frame) has been opened/closed, activated/deactivated, iconified/deiconified via the 3 buttons at the top-right corner or other means. The source of WindowEvent shall be a top-level window-container such as Frame. A WindowEvent listener must implement WindowListener interface, which declares 7 abstract event-handling methods, as follows. Among them, the windowClosing(), which is called back upon clicking the window-close button, is the most commonly-used. public void windowClosing(WindowEvent evt) public void windowOpened(WindowEvent evt) public void windowClosed(WindowEvent evt) public void windowActivated(WindowEvent evt) public void windowDeactivated(WindowEvent evt) public void windowIconified(WindowEvent evt) public void windowDeiconified(WindowEvent evt) The following program added support for "close-window button" to "Example 1: AWTCounter". 1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859606162636465666768 6970 import java.awt.*; import java.awt.event.*; public class WindowEventDemo extends Frame { private TextField tfCount; private Button btnCount; private int count = 0; public WindowEventDemo() { setLayout(new FlowLayout()); add(new Label("Counter")); tfCount = new TextField("0", 10); tfCount.setEditable(false); add(tfCount); btnCount = new Button("Count"); add(btnCount); btnCount.addActionListener(new BtnCountListener()); addWindowListener(new MyWindowListener()); setTitle("WindowEvent Demo"); setSize(300, 100); setVisible(true); } public static void main(String[] args) { new WindowEventDemo(); } private class BtnCountListener implements ActionListener { @Override public void actionPerformed(ActionEvent evt) { ++count; tfCount.setText(count + ""); } } private class MyWindowListener implements WindowListener { @Override public void windowClosing(WindowEvent evt) { System.exit(0); } @Override public void windowOpened(WindowEvent evt) { } @Override public void windowClosed(WindowEvent evt) { } @Override public void windowIconified(WindowEvent evt) { System.out.println("Window Iconified"); } @Override public void windowDeiconified(WindowEvent evt) { System.out.println("Window Deiconified"); } @Override public void windowActivated(WindowEvent evt) { System.out.println("Window Activated"); } @Override public void windowDeactivated(WindowEvent evt) { System.out.println("Window Deactivated"); } }} In this example, we shall modify the earlier AWTCounter example to handle the WindowEvent. Recall that pushing the "close-window" button on the AWTCounter has no effect, as it did not handle the WindowEvent of windowClosing(). We included the WindowEvent handling codes in this example. We identify the super Frame as the source object. The Frame fires the WindowEvent to all its registered WindowEvent listener(s). In Line 53-69, we define an inner class called MyWindowListener as the WindowEvent listener. It is required to implement the WindowListener interface, which declares 7 abstract methods: windowOpened(), windowClosed(), windowClosing(), windowActivated(), windowDeactivated(), windowIconified() and windowDeiconified(). We register an anonymous instance of MyWindowListener as the WindowEvent listener to the source Frame via method addWindowListener(new MyWindowListener()). We override the windowClosing() handler to terminate the program using System.exit(0). We ignore the other 6 handlers, but required to provide an empty body for compilation. The sequence diagram is as follow: Example 4: MouseEvent and MouseListener Interface A MouseEvent is fired when you press, release, or click (press followed by release) a mouse-button (left or right button) at the source object; or position the mouse-pointer at (enter) and away (exit) from the source object. A MouseEvent listener must implement the MouseListener interface, which declares the following five abstract methods: public void mouseClicked(MouseEvent evt) public void mousePressed(MouseEvent evt) public void mouseReleased(MouseEvent evt) public void mouseEntered(MouseEvent evt) public void mouseExited(MouseEvent evt) 123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657 import java.awt.*; import java.awt.event.*; public class MouseEventDemo extends Frame { private TextField tfMouseX; private TextField tfMouseY; public MouseEventDemo() { setLayout(new FlowLayout()); add(new Label("X-Click: ")); tfMouseX = new TextField(10); tfMouseX.setEditable(false); add(tfMouseX); add(new Label("Y-Click: ")); tfMouseY = new TextField(10); tfMouseY.setEditable(false); add(tfMouseY); addMouseListener(new MyMouseListener()); setTitle("MouseEvent Demo"); setSize(350, 100); setVisible(true); } public static void main(String[] args) { new MouseEventDemo(); } private class MyMouseListener implements MouseListener { @Override public void mouseClicked(MouseEvent evt) { tfMouseX.setText(evt.getX() + ""); tfMouseY.setText(evt.getY() + ""); } @Override public void mousePressed(MouseEvent evt) { } @Override public void mouseReleased(MouseEvent evt) { } @Override public void mouseEntered(MouseEvent evt) { } @Override public void mouseExited(MouseEvent evt) { } }} In this example, we setup a GUI with 4 components (two anonymous Labels and two non-editable TextFields) inside a top-level container Frame, arranged in FlowLayout. To demonstrate the MouseEvent: We identity super Frame as the source object. The Frame fires a MouseEvent to all its MouseEvent listener(s) when you click/press/release a mouse-button or enter/exit with the mouse-pointer. In Line 42-56, we define an inner class called MyMouseListener as the MouseEvent listener. It is required to implement the MouseListener interface, which declares 5 abstract methods: mouseClicked(), mousePressed(), mouseReleased(), mouseEntered(), and mouseExit(). We override the mouseClicked() to display the (x, y) coordinates of the mouse click on the two displayed TextFields. We ignore all the other handlers (for simplicity - but you need to provide an empty body for compilation). We register an anonymous instance of MyMouseListener as the MouseEvent listener to super Frame (source) via the method addMouseListener(new MyMouseListener()). Try: Include a WindowListener to handle the close-window button. Example 5: MouseEvent and MouseMotionListener Interface A MouseEvent is also fired when you move and drag the mouse pointer at the source object. But you need to use MouseMotionListener to handle the mouse-move and mouse-drag. The MouseMotionListener interface declares the following two abstract methods: public void mouseDragged(MouseEvent e) public void mouseMoved(MouseEvent e) 1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859606162636465666768697071727374757677787980 81 import java.awt.*; import java.awt.event.*; public class MouseMotionDemo extends Frame { private TextField tfMouseClickX; private TextField tfMouseClickY; private TextField tfMousePositionX; private TextField tfMousePositionY; public MouseMotionDemo() { setLayout(new FlowLayout()); add(new Label("X-Click: ")); tfMouseClickX = new TextField(10); tfMouseClickX.setEditable(false); add(tfMouseClickX); add(new Label("Y-Click: ")); tfMouseClickY = new TextField(10); tfMouseClickY.setEditable(false); add(tfMouseClickY); add(new Label("X-Position: ")); tfMousePositionX = new TextField(10); tfMousePositionX.setEditable(false); add(tfMousePositionX); add(new Label("Y-Position: ")); tfMousePositionY = new TextField(10); tfMousePositionY.setEditable(false); add(tfMousePositionY); MyMouseListener listener = new MyMouseListener(); addMouseListener(listener); addMouseMotionListener(listener); setTitle("MouseMotion Demo"); setSize(400, 120); setVisible(true); } public static void main(String[] args) { new MouseMotionDemo(); } private class MyMouseListener implements MouseListener, MouseMotionListener { @Override public void mouseClicked(MouseEvent evt) { tfMouseClickX.setText(evt.getX() + ""); tfMouseClickY.setText(evt.getY() + ""); } @Override public void mousePressed(MouseEvent evt) { } @Override public void mouseReleased(MouseEvent evt) { } @Override public void mouseEntered(MouseEvent evt) { } @Override public void mouseExited(MouseEvent evt) { } @Override public void mouseMoved(MouseEvent evt) { tfMousePositionX.setText(evt.getX() + ""); tfMousePositionY.setText(evt.getY() + ""); } @Override public void mouseDragged(MouseEvent evt) { } }} In this example, we shall illustrate both the MouseListener and MouseMotionListener. We identify the super Frame as the source, which fires the MouseEvent to its registered MouseListener and MouseMotionListener. In Line 53-80, we define an inner class called MyMouseListener as both the MouseListener and MouseMotionListener. We register an instance of MyMouseListener as the listener to super Frame via method addMouseListener() and addMouseMotionListener(). The MouseMotionListener needs to implement 2 abstract methods: mouseMoved() and mouseDragged() declared in the MouseMotionListener interface. We override the mouseMoved() to display the (x, y) position of the mouse pointer. We ignore the MouseDragged() handler by providing an empty body for compilation. Try: Include a WindowListener to handle the close-window button. Example 6: KeyEvent and KeyListener Interface A KeyEvent is fired when you pressed, released, and typed (pressed followed by released) a key on the source object. A KeyEvent listener must implement KeyListener interface, which declares three abstract methods: public void keyTyped(KeyEvent e) public void keyPressed(KeyEvent e) public void keyReleased(KeyEvent e) 1234567891011121314151617181920212223242526272829303132333435363738394041424344454647 import java.awt.*; import java.awt.event.*; public class KeyEventDemo extends Frame { private TextField tfInput; private TextArea taDisplay; public KeyEventDemo() { setLayout(new FlowLayout()); add(new Label("Enter Text: ")); tfInput = new TextField(10); add(tfInput); taDisplay = new TextArea(5, 40); add(taDisplay); tfInput.addKeyListener(new MyKeyListener()); setTitle("KeyEvent Demo"); setSize(400, 200); setVisible(true); } public static void main(String[] args) { new KeyEventDemo(); } private class MyKeyListener implements KeyListener { @Override public void keyTyped(KeyEvent evt) { taDisplay.append("You have typed " + evt.getKeyChar() + ""); } @Override public void keyPressed(KeyEvent evt) { } @Override public void keyReleased(KeyEvent evt) { } }} In this example: We identify the tfInput (of TextField) as the source object. The source fires a KeyEvent when you press/release/type a key to all its KeyEvent listener(s). In Line 35-46, we define an inner class called MyKeyListener as the KeyEvent listener. We register an anonymous instance of MyKeyListener as the KeyEvent listener to the source TextField via method input.addKeyListener(). The KeyEvent listener needs to implement the KeyListener interface, which declares 3 abstract methods: keyTyped(), keyPressed(), keyReleased(). We override the keyTyped() to display key typed on the display TextArea. We ignore the keyPressed() and keyReleased(). Nested (Inner) Classes Without Inner classes In our AWTCounter example, suppose we would like to write an external ordinary class (say MyExternalBtnListener) as our ActionEvent listener. This class shall implement ActionListener interface and override the actionPerformed() method. An example is as follows: import java.awt.*; import java.awt.event.*; public class AWTCounterExternal extends Frame { private Label lblCount; private TextField tfCount; private Button btnCount; private int count = 0; public AWTCounterExternal () { setLayout(new FlowLayout()); lblCount = new Label("Counter"); add(lblCount); tfCount = new TextField(count + "", 10); tfCount.setEditable(false); add(tfCount); btnCount = new Button("Count"); add(btnCount); MyExternalBtnListener listener = new MyExternalBtnListener(); btnCount.addActionListener(listener); setTitle("AWT Counter"); setSize(250, 100); setVisible(true); } public static void main(String[] args) { AWTCounterExternal app = new AWTCounterExternal(); } } class MyExternalBtnListener implements ActionListener { @Override public void actionPerformed(ActionEvent evt) { System.out.println("You clicked the button!"); ++count; tfCount.setText(count + ""); } } Can you see the problem? This external class cannot access the variables such as count and tfCount in the AWTCounterExternal class. We can fix this problem, but the solution is messy. An easy solution is to use an inner class instead of an ordinary external class (to be explained in the following sections). What are Inner classes? A nested class (or commonly called inner class) is a class defined inside another class - introduced in JDK 1.1. As an illustration, two nested classes MyNestedClass1 and MyNestedClass2 are defined inside the definition of an outer class called MyOuterClass. public class MyOuterClass { ...... private class MyNestedClass1 { ...... } public static class MyNestedClass2 { ...... } ......} A nested class has these properties: A nested class is a proper class. That is, it could contain constructors, member variables and member methods. You can create an instance of a nested class via the new operator and constructor. A nested class is a member of the outer class, just like any member variables and methods defined inside a class. Most importantly, a nested class can access the private members (variables/methods) of the enclosing outer class, as it is at the same level as these private members. This is the property that makes inner class useful. A nested class can have private, public, protected, or the default access, just like any member variables and methods defined inside a class. A private inner class is only accessible by the enclosing outer class, and is not accessible by any other classes. [An top-level outer class cannot be declared private, as no one can use a private outer class.] A nested class can also be declared static, final or abstract, just like any ordinary class. A nested class is NOT a subclass of the outer class. That is, the nested class does not inherit the variables and methods of the outer class. It is an ordinary self-contained class. [Nonetheless, you could declare it as a subclass of the outer class, via keyword "extends OuterClassName", in the nested class's definition.] The usages of nested class are: To control visibilities (of the member variables and methods) between inner/outer class. The nested class, being defined inside an outer class, can access private members of the outer class. To place a piece of class definition codes closer to where it is going to be used, to make the program clearer and easier to understand. For namespace management. Example 7: A Named Inner Class as Event Listener (Revisit Example 1 AWTCounter) A nested class is useful if you need a small class which relies on the enclosing outer class for its private variables and methods. It is ideal in an event-driven environment for implementing event handlers. This is because the event handling methods (in a listener) often require access to the private variables (e.g., a private TextField) of the outer class. In this example (revisit Example 1 AWTCounter), we define an inner class called BtnCountListener, and create an instance of BtnCountListener as the ActionEvent listener for the btnCount. The BtnCountListener needs to implement the ActionListener interface, and override the actionPerformed() handler. BtnCountListener needs to be defined as an inner class, as it needs to access private variables (count and tfCount) of the outer class. 123456789101112131415161718192021222324252627282930313233343536373839404142434445464748 import java.awt.*; import java.awt.event.*; public class AWTCounter extends Frame { private TextField tfCount; private Button btnCount; private int count = 0; public AWTCounter() { setLayout(new FlowLayout()); add(new Label("Counter")); tfCount = new TextField("0", 10); tfCount.setEditable(false); add(tfCount); btnCount = new Button("Count"); add(btnCount); btnCount.addActionListener(new BtnCountListener()); setTitle("AWT Counter"); setSize(250, 100); setVisible(true); } public static void main(String[] args) { new AWTCounter(); } private class BtnCountListener implements ActionListener { @Override public void actionPerformed(ActionEvent evt) { ++count; tfCount.setText(count + ""); } }} Dissecting the Program An inner class named BtnCountListener is used as the ActionListener. An anonymous instance of the BtnCountListener inner class is constructed. The btnCount source object adds this instance as a listener, as follows: btnCount.addActionListener(new BtnCountListener()); The inner class can access the private variable tfCount and count of the outer class. The inner class is compiled into AWTCount$BtnCountListener.class, in the format of OuterClassName$InnerClassName.class. (Advanced) Using an Ordinary (Outer) Class as Listener Try moving the BtnCountListener class outside, and define it as an ordinary class. You would need to pass a reference of the AWTCounter into the constructor of BtnCountListener, and use this reference to access variables tfCount and count, through public getters or granting them to public access. public class BtnCountListener implements ActionListener { AWTCounter frame; public BtnCountListener(AWTCounter frame) { this.frame = frame; } @Override public void actionPerformed(ActionEvent evt) { frame.count++; frame.tfCount.setText(frame.count + ""); }} This code is messy! Inner class provides a much cleaner solution! Example 8: An Anonymous Inner Class as Event Listener Instead of using a named inner class (called BtnCountListener in the previous example), we shall use an inner class without a name, known as anonymous inner class as the ActionListener in this example. 123456789101112131415161718192021222324252627282930313233343536373839404142 import java.awt.*; import java.awt.event.*; public class AWTCounterAnonymousInnerClass extends Frame { private TextField tfCount; private Button btnCount; private int count = 0; public AWTCounterAnonymousInnerClass () { setLayout(new FlowLayout()); add(new Label("Counter")); tfCount = new TextField("0", 10); tfCount.setEditable(false); add(tfCount); btnCount = new Button("Count"); add(btnCount); btnCount.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent evt) { ++count; tfCount.setText(count + ""); } }); setTitle("AWT Counter"); setSize(250, 100); setVisible(true); } public static void main(String[] args) { new AWTCounterAnonymousInnerClass(); }} Dissecting the Program The anonymous inner class is given a name generated by the compiler, and compiled into OuterClassName$n.class, where n is a running number of the inner classes of this outer class. An anonymous instance of an anonymous inner class is constructed, and passed as the argument of the addActionListener() method as follows: btnCount.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent evt) { ++count; tfCount.setText(count + ""); } }); The above codes is equivalent to and compiled as: private class N implements ActionListener { @Override public void actionPerformed(ActionEvent evt) { ++count; tfCount.setText(count + ""); } } btnCount.addActionListener(new N()); N n = new N() btnCount.addActionListener(n); From JDK 8, you can write the event handler using "Lambda Expression" in a one-liner as follows: btnCount.addActionListener(evt -> tfCount.setText(++count + "")); Properties of Anonymous Inner Class The anonymous inner class is define inside a method, instead of a member of the outer class (class member). It is local to the method and cannot be marked with access modifier (such as public, private) or static, just like any local variable of a method. An anonymous inner class must always extend a superclass or implement an interface. The keyword "extends" or "implements" is NOT required in its declaration. An anonymous inner class must implement all the abstract methods in the superclass or in the interface. An anonymous inner class always uses the default (no-arg) constructor from its superclass to create an instance. If an anonymous inner class implements an interface, it uses the java.lang.Object(). An anonymous inner class is compiled into a class named OuterClassName$n.class, where n is a running number of inner classes within the outer class. An instance of an anonymous inner class is constructed via this syntax: new SuperClassName/InterfaceName() { } The created instance can be assigned to a variable or used as an argument of a method. Example 9: An Anonymous Inner Class for Each Source Let's modify our AWTCounter example to include 3 buttons for counting up, counting down, and reset the count, respectively. We shall attach an anonymous inner class as the listener to each of buttons. 1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859 import java.awt.*; import java.awt.event.*; public class AWTCounter3ButtonsAnonymousIC extends Frame { private TextField tfCount; private Button btnCountUp, btnCountDown, btnReset; private int count = 0; public AWTCounter3ButtonsAnonymousIC() { setLayout(new FlowLayout()); add(new Label("Counter")); tfCount = new TextField("0", 10); tfCount.setEditable(false); add(tfCount); btnCountUp = new Button("Count Up"); add(btnCountUp); btnCountUp.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent evt) { ++count; tfCount.setText(count + ""); } }); btnCountDown = new Button("Count Down"); add(btnCountDown); btnCountDown.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent evt) { count--; tfCount.setText(count + ""); } }); btnReset = new Button("Reset"); add(btnReset); btnReset.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent evt) { count = 0; tfCount.setText("0"); } }); setTitle("AWT Counter"); setSize(400, 100); setVisible(true); } public static void main(String[] args) { new AWTCounter3ButtonsAnonymousIC(); }} Dissecting the Program Each of the Buttons uses one anonymous instance of an anonymous inner class as its ActionEvent listener. Example 10: Using the Same Listener Instance for All the Buttons If you use the same instance as the listener for all the 3 buttons, you need to determine which button has fired the event. It is because all the 3 buttons trigger the same event-handler method. Using ActionEvent's getActionCommand() In the following example, we use the same instance of a "named" inner class as the listener for all the 3 buttons. The listener needs to determine which button has fired the event. This can be accomplished via the ActionEvent's getActionCommand() method, which returns the button's label. 1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859606162 import java.awt.*; import java.awt.event.*; public class AWTCounter3Buttons1Listener extends Frame { private TextField tfCount; private Button btnCountUp, btnCountDown, btnReset; private int count = 0; public AWTCounter3Buttons1Listener () { setLayout(new FlowLayout()); add(new Label("Counter")); tfCount = new TextField("0", 10); tfCount.setEditable(false); add(tfCount); btnCountUp = new Button("Count Up"); add(btnCountUp); btnCountDown = new Button("Count Down"); add(btnCountDown); btnReset = new Button("Reset"); add(btnReset); AllButtonsListener listener = new AllButtonsListener(); btnCountUp.addActionListener(listener); btnCountDown.addActionListener(listener); btnReset.addActionListener(listener); setTitle("AWT Counter"); setSize(400, 100); setVisible(true); } public static void main(String[] args) { new AWTCounter3Buttons1Listener(); } private class AllButtonsListener implements ActionListener { @Override public void actionPerformed(ActionEvent evt) { String btnLabel = evt.getActionCommand(); if (btnLabel.equals("Count Up")) { ++count; } else if (btnLabel.equals("Count Down")) { --count; } else { count = 0; } tfCount.setText(count + ""); } }} Using getSource() of EventObject Besides the getActionCommand(), which is only available for ActionEvent, you can use the getSource() method, which is available to all event objects, to retrieve a reference to the source object that has fired the event. getSource() returns a java.lang.Object. You may need to downcast it to the proper type of the source object. For example, 1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859606162 import java.awt.*; import java.awt.event.*; public class AWTCounter3ButtonsGetSource extends Frame { private TextField tfCount; private Button btnCountUp, btnCountDown, btnReset; private int count = 0; public AWTCounter3ButtonsGetSource () { setLayout(new FlowLayout()); add(new Label("Counter")); tfCount = new TextField("0", 10); tfCount.setEditable(false); add(tfCount); btnCountUp = new Button("Count Up"); add(btnCountUp); btnCountDown = new Button("Count Down"); add(btnCountDown); btnReset = new Button("Reset"); add(btnReset); AllButtonsListener listener = new AllButtonsListener(); btnCountUp.addActionListener(listener); btnCountDown.addActionListener(listener); btnReset.addActionListener(listener); setTitle("AWT Counter"); setSize(400, 100); setVisible(true); } public static void main(String[] args) { new AWTCounter3ButtonsGetSource(); } private class AllButtonsListener implements ActionListener { @Override public void actionPerformed(ActionEvent evt) { Button source = (Button)evt.getSource(); if (source == btnCountUp) { ++count; } else if (source == btnCountDown) { --count; } else { count = 0; } tfCount.setText(count + ""); } }} Event Listener's Adapter Classes Example 11: WindowAdapter for WindowListener Using WindowListener Interface Refer to the WindowEventDemo, a WindowEvent listener is required to implement the WindowListener interface, which declares 7 abstract methods. Although we are only interested in windowClosing(), we need to provide an empty body to the other 6 abstract methods in order to compile the program. This is tedious, e.g., we can rewrite the WindowEventDemo using an inner class implementing ActionListener as follows: 123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354 import java.awt.*; import java.awt.event.*; public class WindowEventDemoWithInnerClass extends Frame { private TextField tfCount; private Button btnCount; private int count = 0; public WindowEventDemoWithInnerClass () { setLayout(new FlowLayout()); add(new Label("Counter")); tfCount = new TextField("0", 10); tfCount.setEditable(false); add(tfCount); btnCount = new Button("Count"); add(btnCount); btnCount.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent evt) { ++count; tfCount.setText(count + ""); } }); addWindowListener(new WindowListener() { @Override public void windowClosing(WindowEvent evt) { System.exit(0); } @Override public void windowOpened(WindowEvent evt) { } @Override public void windowClosed(WindowEvent evt) { } @Override public void windowIconified(WindowEvent evt) { } @Override public void windowDeiconified(WindowEvent evt) { } @Override public void windowActivated(WindowEvent evt) { } @Override public void windowDeactivated(WindowEvent evt) { } }); setTitle("WindowEvent Demo"); setSize(250, 100); setVisible(true); } public static void main(String[] args) { new WindowEventDemoWithInnerClass(); }} Using WindowAdapter Superclass An adapter class called WindowAdapter is therefore provided, which implements the WindowListener interface and provides default implementations to all the 7 abstract methods. You can then derive a subclass from WindowAdapter and override only methods of interest and leave the rest to their default implementation. For example, 1234567891011121314151617181920212223242526272829303132333435363738394041424344454647 import java.awt.*; import java.awt.event.*; public class WindowEventDemoAdapter extends Frame { private TextField tfCount; private Button btnCount; private int count = 0; public WindowEventDemoAdapter () { setLayout(new FlowLayout()); add(new Label("Counter")); tfCount = new TextField("0", 10); tfCount.setEditable(false); add(tfCount); btnCount = new Button("Count"); add(btnCount); btnCount.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent evt) { ++count; tfCount.setText(count + ""); } }); addWindowListener(new WindowAdapter() { @Override public void windowClosing(WindowEvent evt) { System.exit(0); } }); setTitle("WindowEvent Demo"); setSize(250, 100); setVisible(true); } public static void main(String[] args) { new WindowEventDemoAdapter(); }} Clearly, the adapter greatly simplifies the codes. Other Event-Listener Adapter Classes Similarly, adapter classes such as MouseAdapter, MouseMotionAdapter, KeyAdapter, FocusAdapter are available for MouseListener, MouseMotionListener, KeyListener, and FocusListener, respectively. There is no ActionAdapter for ActionListener, because there is only one abstract method (i.e. actionPerformed()) declared in the ActionListener interface. This method has to be overridden and there is no need for an adapter. [SKIP] The Legacy "this" Listener If you read some old books, you may find many examples that use "this" object as the event listener. For example, 1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859606162636465666 67 import java.awt.*; import java.awt.event.*; public class ThisListenerDemo extends Frame implements ActionListener, WindowListener { private TextField tfCount; private Button btnCount; private int count = 0; public ThisListenerDemo() { setLayout(new FlowLayout()); add(new Label("Counter")); tfCount = new TextField("0", 10); tfCount.setEditable(false); add(tfCount); btnCount = new Button("Count"); add(btnCount); btnCount.addActionListener(this); addWindowListener(this); setTitle("WindowEvent Demo"); setSize(250, 100); setVisible(true); } public static void main(String[] args) { new ThisListenerDemo(); } @Override public void actionPerformed(ActionEvent evt) { ++count; tfCount.setText(count + ""); } @Override public void windowClosing(WindowEvent evt) { System.exit(0); } @Override public void windowOpened(WindowEvent evt) { } @Override public void windowClosed(WindowEvent evt) { } @Override public void windowIconified(WindowEvent evt) { System.out.println("Window Iconified"); } @Override public void windowDeiconified(WindowEvent evt) { System.out.println("Window Deiconified"); } @Override public void windowActivated(WindowEvent evt) { System.out.println("Window Activated"); } @Override public void windowDeactivated(WindowEvent evt) { System.out.println("Window Deactivated"); }} There is only ONE class in this code. But this code is much harder to understand and seldom used nowadays. Using inner class is a better solution. Layout Managers and Panel A container has a so-called layout manager to arrange its components. The layout managers provide a level of abstraction to map your user interface on all windowing systems, so that the layout can be platform-independent. AWT provides the following layout managers (in package java.awt): FlowLayout, GridLayout, BorderLayout, GridBagLayout, BoxLayout, CardLayout, and others. Swing added more layout manager in package javax.swing, to be described later. Container's setLayout() method A container has a setLayout() method to set its layout manager: public void setLayout(LayoutManager mgr) To set up the layout of a Container (such as Frame, JFrame, Panel, or JPanel), you have to: Construct an instance of the chosen layout object, via new and constructor, e.g., new FlowLayout()) Invoke the setLayout() method of the Container, with the layout object created as the argument; Place the GUI components into the Container using the add() method in the correct order; or into the correct zones. For example, Panel pnl = new Panel(); pnl.setLayout(new FlowLayout());pnl.add(new JLabel("One"));pnl.add(new JLabel("Two"));pnl.add(new JLabel("Three"));...... Container's getLayout() method You can get the current layout via Container's getLayout() method. Panel pnl = new Panel();System.out.println(pnl.getLayout()); Panel's Initial Layout Panel

(and Swing's JPanel) provides a constructor to set its initial layout manager. It is because a primary function of Panel is to layout a group of component in a particular layout. public void Panel(LayoutManager layout) Panel pnl = new Panel(new BorderLayout()); FlowLayout In the java.awt.FlowLayout, components are arranged from left-to-right inside the container in the order that they are added (via method aContainer.add(aComponent)). When one row is filled, a new row will be started. The actual appearance depends on the width of the display window. Constructors public FlowLayout(); public FlowLayout(int alignment); public FlowLayout(int alignment, int hgap, int vgap); Example 123456789101112131415161718192021222324252627282930313233343536 import java.awt.*; import java.awt.event.*; public class AWTFlowLayoutDemo extends Frame { private Button btn1, btn2, btn3, btn4, btn5, btn6; public AWTFlowLayoutDemo () { setLayout(new FlowLayout()); btn1 = new Button("Button 1"); add(btn1); btn2 = new Button("This is Button 2"); add(btn2); btn3 = new Button("3"); add(btn3); btn4 = new Button("Another Button 4"); add(btn4); btn5 = new Button("Button 5"); add(btn5); btn6 = new Button("One More Button 6"); add(btn6); setTitle("FlowLayout Demo"); setSize(280, 150); setVisible(true); } public static void main(String[] args) { new AWTFlowLayoutDemo(); }} GridLayout In java.awt.GridLayout, components are arranged in a grid (matrix) of rows and columns inside the Container. Components are added in a left-to-right, top-to-bottom manner in the order they are added (via method aContainer.add(aComponent)). Constructors public GridLayout(int rows, int columns); public GridLayout(int rows, int columns, int hgap, int vgap); Example 123456789101112131415161718192021222324252627282930313233343536 import java.awt.*; import java.awt.event.*; public class AWTGridLayoutDemo extends Frame { private Button btn1, btn2, btn3, btn4, btn5, btn6; public AWTGridLayoutDemo () { setLayout(new GridLayout(3, 2, 3, 3)); btn1 = new Button("Button 1"); add(btn1); btn2 = new Button("This is Button 2"); add(btn2); btn3 = new Button("3"); add(btn3); btn4 = new Button("Another Button 4"); add(btn4); btn5 = new Button("Button 5"); add(btn5); btn6 = new Button("One More Button 6"); add(btn6); setTitle("GridLayout Demo"); setSize(280, 150); setVisible(true); } public static void main(String[] args) { new AWTGridLayoutDemo(); }} BorderLayout In java.awt.BorderLayout, the container is divided into 5 zones: EAST, WEST, SOUTH, NORTH, and CENTER. Components are added using method aContainer.add(aComponent, zone), where zone is either BorderLayout.NORTH (or PAGE_START), BorderLayout.SOUTH (or PAGE_END), BorderLayout.WEST (or LINE_START), BorderLayout.EAST (or LINE_END), or BorderLayout.CENTER. You need not place components to all the 5 zones. The NORTH and SOUTH components may be stretched horizontally; the EAST and WEST components may be stretched vertically; the CENTER component may stretch both horizontally and vertically to fill any space left over. Constructors public BorderLayout(); public BorderLayout(int hgap, int vgap); Example 1234567891011121314151617181920212223242526272829303132333435 import java.awt.*; import java.awt.event.*; public class AWTBorderLayoutDemo extends Frame { private Button btnNorth, btnSouth, btnCenter, btnEast, btnWest; public AWTBorderLayoutDemo () { setLayout(new BorderLayout(3, 3)); btnNorth = new Button("NORTH"); add(btnNorth, BorderLayout.NORTH); btnSouth = new Button("SOUTH"); add(btnSouth, BorderLayout.SOUTH); btnCenter = new Button("CENTER"); add(btnCenter, BorderLayout.CENTER); btnEast = new Button("EAST"); add(btnEast, BorderLayout.EAST); btnWest = new Button("WEST"); add(btnWest, BorderLayout.WEST); setTitle("BorderLayout Demo"); setSize(280, 150); setVisible(true); } public static void main(String[] args) { new AWTBorderLayoutDemo(); }} Using Panels as Sub-Container to Organize Components An AWT Panel is a rectangular pane, which can be used as sub-container to organized a group of related components in a specific layout (e.g., FlowLayout, BorderLayout). Panels are secondary containers, which shall be added into a top-level container (such as Frame), or another Panel. For example, the following figure shows a Frame in BorderLayout containing two Panels - panelResult in FlowLayout and panelButtons in GridLayout. panelResult is added to the NORTH, and panelButtons is added to the CENTER. 1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859 import java.awt.*; import java.awt.event.*; public class AWTPanelDemo extends Frame { private Button[] btnNumbers; private Button btnHash, btnStar; private TextField tfDisplay; public AWTPanelDemo () { Panel panelDisplay = new Panel(new FlowLayout()); tfDisplay = new TextField("0", 20); panelDisplay.add(tfDisplay); Panel panelButtons = new Panel(new GridLayout(4, 3)); btnNumbers = new Button[10]; btnNumbers[1] = new Button("1"); panelButtons.add(btnNumbers[1]); btnNumbers[2] = new Button("2"); panelButtons.add(btnNumbers[2]); btnNumbers[3] = new Button("3"); panelButtons.add(btnNumbers[3]); btnNumbers[4] = new Button("4"); panelButtons.add(btnNumbers[4]); btnNumbers[5] = new Button("5"); panelButtons.add(btnNumbers[5]); btnNumbers[6] = new Button("6"); panelButtons.add(btnNumbers[6]); btnNumbers[7] = new Button("7"); panelButtons.add(btnNumbers[7]); btnNumbers[8] = new Button("8"); panelButtons.add(btnNumbers[8]); btnNumbers[9] = new Button("9"); panelButtons.add(btnNumbers[9]); btnStar = new Button("*"); panelButtons.add(btnStar); btnNumbers[0] = new Button("0"); panelButtons.add(btnNumbers[0]); btnHash = new Button("#"); panelButtons.add(btnHash); setLayout(new BorderLayout()); add(panelDisplay, BorderLayout.NORTH); add(panelButtons, BorderLayout.CENTER); setTitle("BorderLayout Demo"); setSize(200, 200); setVisible(true); } public static void main(String[] args) { new AWTPanelDemo(); }} GridBagLayout Reference: Read "How to Use GridBagLayout" @ . Example 1 12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455565758596061626364 import java.awt.*;import javax.swing.*;@SuppressWarnings("serial")public class GridBagLayoutExample extends JPanel { JButton button1, button2, button3, button4, button5; public GridBagLayoutExample() { super.setLayout(new GridBagLayout()); GridBagConstraints gbc = new GridBagConstraints(); button1 = new JButton("B1"); gbc.gridx = 0; gbc.gridy = 0; super.add(button1, gbc); button2 = new JButton("B2"); gbc.gridx = 1; gbc.gridy = 0; super.add(button2, gbc); button3 = new JButton("B3"); gbc.gridx = 2; gbc.gridy = 0; super.add(button3, gbc); button4 = new JButton("Long-Named Button 4"); gbc.gridx = 0; gbc.gridy = 1; gbc.gridwidth = 3; gbc.ipady = 40; super.add(button4, gbc); button5 = new JButton("5"); gbc.gridx = 1; gbc.gridy = 2; gbc.gridwidth = 2; gbc.ipady = 0; gbc.anchor = GridBagConstraints.PAGE_END; gbc.insets = new Insets(10, 0, 0, 0); super.add(button5, gbc); } public static void main(String[] args) { javax.swing.SwingUtilities.invokeLater(new Runnable() { public void run() { JFrame frame = new JFrame("GridBagLayoutDemo"); frame.setContentPane(new GridBagLayoutExample()); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); frame.setVisible(true); } }); }} Run the program, the output is in (a), where all components have their natural width and height, placed in center (anchor=CENTER) with extra x and y spaces at the 4 margins. The ipady increases the internal y-padding for button-4 (spans 3 columns). Button 5 (spans second and third columns) has top margin specified by insets, and anchor at the bottom-right corner (anchor=PAGE_END). Uncomment the gbc.fill = GridBagConstraints.HORIZONTAL (line 14). The output is in (b), where ALL components fill horizontally with equal column width. Uncomment all weightx (lines 20, 26, 32). The output is in (c). The extra x spaces are distributed according to the weightx of (0.5, 1.0, 0.5). Uncomment the weighty (line 49). The output is in (d). The extra y spaces are given to row 3, as row 1 and 2 have weighty of 0 and non-participating in the distribution of extra y-spaces. BoxLayout BoxLayout arrange components in a single row or column. It respects components' requests on the minimum sizes. [TODO] Example and diagram Swing Introduction Swing is part of the so-called "Java Foundation Classes (JFC)" (have you heard of MFC?), which was introduced in 1997 after the release of JDK 1.1. JFC was subsequently included as an integral part of JDK since JDK 1.2. JFC consists of: Swing API: for advanced graphical programming. Accessibility API: provides assistive technology for the disabled. Java 2D API: for high quality 2D graphics and images. Pluggable look and feel supports. Drag-and-drop support between Java and native applications. The goal of Java GUI programming is to allow the programmer to build GUI that looks good on ALL platforms. JDK 1.0's AWT was awkward and non-object-oriented (using many event.getSource()). JDK 1.1's AWT introduced event-delegation (event-driven) model, much clearer and object-oriented. JDK 1.1 also introduced inner class and JavaBeans a component programming model for visual programming environment (similar to Visual Basic). Swing appeared after JDK 1.1. It was introduced into JDK 1.1 as part of an add-on JFC (Java Foundation Classes). Swing is a rich set of easy-to-use, easy-to-understand JavaBean GUI components that can be dragged and dropped as "GUI builders" in visual programming environment. Swing is now an integral part of Java since JDK 1.2. Swing's Features Swing is huge (consists of 18 packages of 737 classes as in JDK 1.8) and has great depth. Compared with AWT, Swing provides a huge and comprehensive collection of reusable GUI components, as shown in the Figure below (extracted form Swing Tutorial). The main features of Swing are (extracted from the Swing website): Swing is written in pure Java (except a few classes) and therefore is 100% portable. Swing components are lightweight. The AWT components are heavyweight (in terms of system resource utilization). Each AWT component has its own opaque native display, and always displays on top of the lightweight components. AWT components rely heavily on the underlying windowing subsystem of the native operating system. For example, an AWT button ties to an actual button in the underlying native windowing subsystem, and relies on the native windowing subsystem for their rendering and processing. Swing components (JComponents) are written in Java. They are generally not "weight-down" by complex GUI considerations imposed by the underlying windowing subsystem. Swing components support pluggable look-and-feel. You can choose between Java look-and-feel and the look-and-feel of the underlying OS (e.g., Windows, UNIX or macOS). If the later is chosen, a Swing button runs on the Windows looks like a Windows' button and feels like a Window's button. Similarly, a Swing button runs on the UNIX looks like a UNIX's button and feels like a UNIX's button. Swing supports mouse-less operation, i.e., it can operate entirely using keyboard. Swing components support "tool-tips". Swing components are JavaBeans a Component-based Model used in Visual Programming (like Visual Basic). You can drag-and-drop a Swing component into a "design form" using a "GUI builder" and double-click to attach an event handler. Swing application uses AWT event-handling classes (in package java.awt.event). Swing added some new classes in package javax.swing.event, but they are not frequently used. Swing application uses AWT's layout manager (such as FlowLayout and BorderLayout in package java.awt). It added new layout managers, such as Springs, Struts, and BoxLayout (in package javax.swing). Swing implements double-buffering and automatic repaint batching for smoother screen repaint. Swing introduces JLayeredPane and JInternalFrame for creating Multiple Document Interface (MDI) applications. Swing supports floating toolbars (in JToolBar), splitter control, "undo". Others - check the Swing website. Using Swing API If you understood the AWT programming (in particular, container/component and event-handling), switching over to Swing (or any other Graphics packages) is straight-forward. Swing's Components Compared with the AWT component classes (in package java.awt), Swing component classes (in package javax.swing) begin with a prefix "J", e.g., JButton, JTextField, JLabel, JPanel, JFrame, or JApplet. The above figure shows the class hierarchy of the swing GUI classes. Similar to AWT, there are two groups of classes: containers and components. A container is used to hold components. A container can also hold containers because it is a (subclass of) component. As a rule, do not mix heavyweight AWT components and lightweight Swing components in the same program, as the heavyweight components will always be painted on top of the lightweight components. Swing's Top-Level and Secondary Containers Just like AWT application, a Swing application requires a top-level container. There are three top-level containers in Swing: JFrame: used for the application's main window (with an icon, a title, minimize/maximize/close buttons, an optional menu-bar, and a content-pane), as illustrated. JDialog: used for secondary pop-up window (with a title, a close button, and a content-pane). JApplet: used for the applet's display-area (content-pane) inside a browsers window. Similarly to AWT, there are secondary containers (such as JPanel) which can be used to group and layout relevant components. The Content-Pane of Swing's Top-Level Container However, unlike AWT, the JComponents shall not be added onto the top-level container (e.g., JFrame, JApplet) directly because they are lightweight components. The JComponents must be added onto the so-called content-pane of the top-level container. Content-pane is in fact a java.awt.Container that can be used to group and layout components. You could: get the content-pane via getContentPane() from a top-level container, and add components onto it. For example, public class SwingDemo extends JFrame { public SwingDemo() { Container cp = getContentPane(); cp.setLayout(new FlowLayout()); cp.add(new JLabel("Hello, world!")); cp.add(new JButton("Button")); ...... } ....} set the content-pane to a JPanel (the main panel created in your application which holds all your GUI components) via JFrame's setContentPane(). public class SwingDemo extends JFrame { public SwingDemo() { JPanel mainPanel = new JPanel(new FlowLayout()); mainPanel.add(new JLabel("Hello, world!")); mainPanel.add(new JButton("Button")); setContentPane(mainPanel); ...... } ....} Notes: If a component is added directly into a JFrame, it is added into the content-pane of JFrame instead, i.e., add(new JLabel("add to JFrame directly"));getContentPane().add(new JLabel("add to JFrame directly")); Event-Handling in Swing Swing uses the AWT event-handling classes (in package java.awt.event). Swing introduces a few new event-handling classes (in package javax.swing.event) but they are not frequently used. Writing Swing Applications In summary, to write a Swing application, you have: Use the Swing components with prefix "J" in package javax.swing, e.g., JFrame, JButton, JTextField, JLabel, etc. A top-level container (typically JFrame) is needed. The JComponents should not be added directly onto the top-level container. They shall be added onto the content-pane of the top-level container. You can retrieve a reference to the content-pane by invoking method getContentPane() from the top-level container. Swing applications uses AWT event-handling classes, e.g., ActionEvent/ActionListener, MouseEvent/MouseListener, etc. Run the constructor in the Event Dispatcher Thread (instead of Main thread) for thread safety, as shown in the following program template. Swing Program Template 123456789101112131415161718192021222324252627282930313233343536373839404142434445 import java.awt.*; import java.awt.event.*; import javax.swing.*; public class SwingTemplate extends JFrame { public SwingTemplate() { Container cp = getContentPane(); cp.setLayout(new ....Layout()); cp.add(....); setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); setTitle("......"); setSize(300, 150); setVisible(true); } public static void main(String[] args) { SwingUtilities.invokeLater(new Runnable() { @Override public void run() { new SwingTemplate(); } }); }} I will explain this template in the following Swing example. Swing Example 1: SwingCounter Let's convert the earlier AWT application example into Swing. Compare the two source files and note the changes (which are highlighted). The display is shown below. Note the differences in look and feel between the AWT GUI components and Swing's. 12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152 import java.awt.*; s import java.awt.event.*; import javax.swing.*; public class SwingCounter extends JFrame { private JTextField tfCount; private JButton btnCount; private int count = 0; public SwingCounter() { Container cp = getContentPane(); cp.setLayout(new FlowLayout()); cp.add(new JLabel("Counter")); tfCount = new JTextField("0"); tfCount.setEditable(false); cp.add(tfCount); btnCount = new JButton("Count"); cp.add(btnCount); btnCount.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent evt) { ++count; tfCount.setText(count + ""); } }); setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); setTitle("Swing Counter"); setSize(300, 100); setVisible(true); } public static void main(String[] args) { SwingUtilities.invokeLater(new Runnable() { @Override public void run() { new SwingCounter(); } }); }} JFrame's Content-Pane The JFrame's method getContentPane() returns the content-pane (which is a java.awt.Container) of the JFrame. You can then set its layout (the default layout is BorderLayout), and add components into it. For example, Container cp = getContentPane(); cp.setLayout(new FlowLayout()); cp.add(new JLabel("Counter")); ......cp.add(tfCount); ......cp.add(btnCount); You can also use the JFrame's setContentPane() method to directly set the content-pane to a JPanel (or a JComponent). For example, JPanel displayPanel = new JPanel();setContentPane(displayPanel); ..... getContentPane().add(displayPanel); JFrame's setDefaultCloseOperation() Instead of writing a WindowEvent listener with a windowClosing() handler to process the "close-window" button, JFrame provides a method called setDefaultCloseOperation() to sets the default operation when the user initiates a "close" on this frame. Typically, we choose the option JFrame.EXIT_ON_CLOSE, which terminates the application via a System.exit(). setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); Running the GUI Construction Codes on the Event-Dispatching Thread In the previous examples, we invoke the constructor directly in the entry main() method to setup the GUI components. For example, public static void main(String[] args) { new SwingCounter();} The constructor will be executed in the so-called "Main-Program" thread. This may cause multi-threading issues (such as unresponsive user-interface and deadlock). It is recommended to execute the GUI setup codes in the so-called "Event-Dispatching" thread, instead of "Main-Program" thread, for thread-safe operations. Event-dispatching thread, which processes events, should be used when the codes updates the GUI. To run the constructor on the event-dispatching thread, invoke static method SwingUtilities.invokeLater() to asynchronously queue the constructor on the event-dispatching thread. The codes will be run after all pending events have been processed. For example, public static void main(String[] args) { SwingUtilities.invokeLater(new Runnable() { @Override public void run() { new SwingCounter(); } });} Note: javax.swing.SwingUtilities.invokeLater() is a cover for java.awt.EventQueue.invokeLater() (which is used in the NetBeans' Visual GUI Builder). At times, for example in game programming, the constructor or the main() may contains non-GUI codes. Hence, it is a common practice to create a dedicated method called initComponents() (used in NetBeans visual GUI builder) or createAndShowGUI() (used in Swing tutorial) to handle all the GUI codes (and another method called initGame() to handle initialization of the game's objects). This GUI init method shall be run in the event-dispatching thread. Warning Message "The serialization class does not declare a static final serialVersionUID field of type long" This warning message is triggered because java.awt.Frame (via its superclass java.awt.Component) implements the java.io.Serializable interface. This interface enables the object to be written out to an output stream serially (via method writeObject()); and read back into the program (via method readObject()). The serialization runtime uses a number (called serialVersionUID) to ensure that the object read into the program is compatible with the class definition, and not belonging to another version. You have these options: Simply ignore this warning message. If a serializable class does not explicitly declare a serialVersionUID, then the serialization runtime will calculate a default serialVersionUID value for that class based on various aspects of the class. Add a serialVersionUID (Recommended), e.g. private static final long serialVersionUID = 1L; Suppress this particular warning via annotation @SuppressWarnings (in package java.lang) (JDK 1.5): @SuppressWarnings("serial")public class MyFrame extends JFrame { ...... } Swing Example 2: SwingAccumulator 12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535 4 import java.awt.*; import java.awt.event.*; import javax.swing.*; public class SwingAccumulator extends JFrame { private JTextField tfInput, tfOutput; private int sum = 0; public SwingAccumulator() { Container cp = getContentPane(); cp.setLayout(new GridLayout(2, 2, 5, 5)); cp.add(new JLabel("Enter an Integer: ")); tfInput = new JTextField(10); cp.add(tfInput); cp.add(new JLabel("The Accumulated Sum is: ")); tfOutput = new JTextField(10); tfOutput.setEditable(false); cp.add(tfOutput); tfInput.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent evt) { int numberIn = Integer.parseInt(tfInput.getText()); sum += numberIn; tfInput.setText(""); tfOutput.setText(sum + ""); } }); setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); setTitle("Swing Accumulator"); setSize(350, 120); setVisible(true); } public static void main(String[] args) { SwingUtilities.invokeLater(new Runnable() { @Override public void run() { new SwingAccumulator(); } }); }} Using Visual GUI Builder - NetBeans/Eclipse If you have a complicated layout for your GUI application, you should use a GUI Builder, such as NetBeans or Eclipse to layout your GUI components in a drag-and-drop manner, similar to the popular visual languages such as Visual Basic. NetBeans For using NetBeans GUI Builder, read my "Writing Java GUI (AWT/Swing) Application in NetBeans"; or Swing Tutorial's "Learning Swing with the NetBeans IDE". Eclipse For using Eclipse GUI Builder, read "Writing Swing Applications using Eclipse GUI Builder". LINK TO JAVA REFERENCES & RESOURCES MORE REFERENCES & RESOURCES Swing in Java is a Graphical User Interface (GUI) toolkit that includes the GUI components. Swing provides a rich set of widgets and packages to make sophisticated GUI components for Java applications. Swing is a part of Java Foundation Classes(JFC), which is an API for Java GUI programing that provide GUI. The Java Swing library is built on top of the Java Abstract Widget Toolkit (AWT), an older, platform dependent GUI toolkit. You can use the Java simple GUI programming components like button, textbox, etc., from the library and do not have to create the components from scratch.In this Java Swing tutorial, you will learn Java GUI basics like- Java Swing class Hierarchy Diagram Java Swing Class Hierarchy Diagram All components in Java Swing are JComponent which can be added to container classes. What is a Container Class?Container classes are classes that can have other components on it. So for creating a Java Swing GUI, we need at least one container object. There are 3 types of Java Swing containers. Panel: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window. Frame: It is a fully functioning window with its title and icons. Dialog: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame. GUI (Graphical User Interface) in Java is an easy-to-use visual experience builder for Java applications. It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application. GUI plays an important role to build easy interfaces for Java applications.How to Make a GUI in Java with ExampleNow in this Java GUI Tutorial, lets understand how to create a GUI in Java with Swings in Java examples.Step 1) Copy code into an editorIn first step Copy the following code into an editor.import javax.swing.*;class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button = new JButton("Press"); frame.getContentPane().add(button); // Adds Button to content pane of frame frame.setVisible(true); }} Step 2) Run the codeNext step, Save, Compile, and Run the codeStep 3) Copy following code into an editorNow lets Add a Button to our frame. Copy following code into an editor from given Java UI Exampleimport javax.swing.*; class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button1 = new JButton("Press"); frame.getContentPane().add(button1); frame.setVisible(true); }} Step 4) Execute the codeNext, Execute the code. You will get a big button.Step 5) Add two buttonsHow about adding two buttons? Copy the following code into an editor.import javax.swing.*;class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button1 = new JButton("Button 1"); JButton button2 = new JButton("Button 2"); frame.getContentPane().add(button1); frame.getContentPane().add(button2); frame.setVisible(true); }}Step 6) Save & Run the programNext, Save, Compile, and Run the program.Step 7) Check outputUnexpected output =? Buttons are getting overlapped.Java Layout ManagerThe Layout manager is used to layout (or arrange) the GUI Java components inside a container. There are many layout managers, but the most frequently used are-Java BorderLayoutA BorderLayout places components in up to five areas: top, bottom, left, right, and center. It is the default layout manager for every java JFrameJava BorderLayout Java FlowLayoutFlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row one after the other.Java FlowLayoutJava GridBagLayoutIt is the more sophisticated of all layouts. It aligns components by placing them within a grid of cells, allowing components to span more than one cell. Java GridBagLayout Step 8) Create chat frameHow about creating a chat frame like below?Try to code yourself before looking at the program below.//Usually you will require both swing and awt packages// even if you are working with just swings.import javax.swing.*;import java.awt.*;class gui { public static void main(String args[]) { //Creating the Frame JFrame frame = new JFrame("Chat Frame"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(400, 400); //Creating the MenuBar and adding components JMenuBar mb = new JMenuBar(); JMenu m1 = new JMenu("FILE"); JMenu m2 = new JMenu("Help"); mb.add(m1); mb.add(m2); JMenuItem m11 = new JMenuItem("Open"); JMenuItem m22 = new JMenuItem("Save as"); m1.add(m11); m1.add(m22); //Creating the panel at bottom and adding components JPanel panel = new JPanel(); // the panel is not visible in output JLabel label = new JLabel("Enter Text"); JTextField tf = new JTextField(10); // accepts upto 10 characters JButton send = new JButton("Send"); JButton reset = new JButton("Reset"); panel.add(label); // Components Added using Flow Layout panel.add(tf); panel.add(send); panel.add(reset); // Text Area at the Center JTextArea ta = new JTextArea(); //Adding Components to the frame. frame.getContentPane().add(BorderLayout.SOUTH, panel); frame.getContentPane().add(BorderLayout.NORTH, mb); frame.getContentPane().add(BorderLayout.CENTER, ta); frame.setVisible(true); }} Read More Swing in Java is a Graphical User Interface (GUI) toolkit that includes the GUI components. Swing provides a rich set of widgets and packages to make sophisticated GUI components for Java applications. Swing is a part of Java Foundation Classes(JFC), which is an API for Java GUI programing that provide GUI. The Java Swing library is built on top of the Java Abstract Widget Toolkit (AWT), an older, platform dependent GUI toolkit. You can use the Java simple GUI programming components like button, textbox, etc., from the library and do not have to create the components from scratch.In this Java Swing tutorial, you will learn Java GUI basics like- Java Swing class Hierarchy Diagram Java Swing Class Hierarchy Diagram All components in Java Swing are JComponent which can be added to container classes. What is a Container Class?Container classes are classes that can have other components on it. So for creating a Java Swing GUI, we need at least one container object. There are 3 types of Java Swing containers. Panel: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window. Frame: It is a fully functioning window with its title and icons. Dialog: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame. GUI (Graphical User Interface) in Java is an easy-to-use visual experience builder for Java applications. It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application. GUI plays an important role to build easy interfaces for Java applications.How to Make a GUI in Java with ExampleNow in this Java GUI Tutorial, lets understand how to create a GUI in Java with Swings in Java examples.Step 1) Copy code into an editorIn first step Copy the following code into an editor.import javax.swing.*;class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button = new JButton("Press"); frame.getContentPane().add(button); // Adds Button to content pane of frame frame.setVisible(true); }} Step 2) Run the codeNext step, Save, Compile, and Run the codeStep 3) Copy following code into an editorNow lets Add a Button to our frame. Copy following code into an editor from given Java UI Exampleimport javax.swing.*; class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button1 = new JButton("Press"); frame.getContentPane().add(button1); frame.setVisible(true); }} Step 4) Execute the codeNext, Execute the code. You will get a big button.Step 5) Add two buttonsHow about adding two buttons? Copy the following code into an editor.import javax.swing.*;class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button1 = new JButton("Button 1"); JButton button2 = new JButton("Button 2"); frame.getContentPane().add(button1); frame.getContentPane().add(button2); frame.setVisible(true); }}Step 6) Save & Run the programNext, Save, Compile, and Run the program.Step 7) Check outputUnexpected output =? Buttons are getting overlapped.Java Layout ManagerThe Layout manager is used to layout (or arrange) the GUI Java components inside a container. There are many layout managers, but the most frequently used are-Java BorderLayoutA BorderLayout places components in up to five areas: top, bottom, left, right, and center. It is the default layout manager for every java JFrameJava BorderLayout Java FlowLayoutFlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row one after the other.Java FlowLayoutJava GridBagLayoutIt is the more sophisticated of all layouts. It aligns components by placing them within a grid of cells, allowing components to span more than one cell. Java GridBagLayout Step 8) Create chat frameHow about creating a chat frame like below? Try to code yourself before looking at the program below.//Usually you will require both swing and awt packages// even if you are working with just swings.import javax.swing.*;import java.awt.*;class gui { public static void main(String args[]) { //Creating the Frame JFrame frame = new JFrame("Chat Frame"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(400, 400); //Creating the MenuBar and adding components JMenuBar mb = new JMenuBar(); JMenu m1 = new JMenu("FILE"); JMenu m2 = new JMenu("Help"); mb.add(m1); mb.add(m2); JMenuItem m11 = new JMenuItem("Open"); JMenuItem m22 = new JMenuItem("Save as"); m1.add(m11); m1.add(m22); //Creating the panel at bottom and adding components JPanel panel = new JPanel(); // the panel is not visible in output JLabel label = new JLabel("Enter Text"); JTextField tf = new JTextField(10); // accepts upto 10 characters JButton send = new JButton("Send"); JButton reset = new JButton("Reset"); panel.add(label); // Components Added using Flow Layout panel.add(tf); panel.add(send); panel.add(reset); // Text Area at the Center JTextArea ta = new JTextArea(); //Adding Components to the frame. frame.getContentPane().add(BorderLayout.SOUTH, panel); frame.getContentPane().add(BorderLayout.NORTH, mb); frame.getContentPane().add(BorderLayout.CENTER, ta); frame.setVisible(true); }} Read More Swing is a Java Foundation Classes [JFC] library and an extension of the Abstract Window Toolkit [AWT]. Java Swing offers much-improved functionality over AWT, new components, expanded components features, and excellent event handling with drag-and-drop support.Introduction of Java SwingSwing has about four times the number of User Interface [UI] components as AWT and is part of the standard Java distribution. By today's application GUI requirements, AWT is a limited implementation, not quite capable of providing the components required for developing complex GUIs required in modern commercial applications. The AWT component set has quite a few bugs and does take up a lot of system resources when compared to equivalent Swing resources. Netscape introduced its Internet Foundation Classes [IFC] library for use with Java. Its Classes became very popular with programmers creating GUI's for commercial applications.Swing is a Set of API (API- Set of Classes and Interfaces)Swing is Provided to Design Graphical User InterfacesSwing is an Extension library to the AWT (Abstract Window Toolkit)Includes New and improved Components that have been enhancing the looks and Functionality of GUIs'Swing can be used to build (Develop) The Standalone swing GUI Apps as Servlets and AppletsIt Employs model/view design architecture.Swing is more portable and more flexible than AWT, the Swing is built on top of the AWT.Swing is Entirely written in Java.Java Swing Components are Platform-independent, and The Swing Components are lightweight.Swing Supports a Pluggable look and feel and Swing provides more powerful components.such as tables, lists, Scrollpanes, Colourchooser, tabbed pane, etc.Further Swing Follows MVC.Difference between Java Swing and Java AWTThere are certain points from which Java Swing is different than Java AWT as mentioned below:Java AWTJava SwingJava AWT is an API to develop GUI applications in Java.Swing is a part of Java Foundation Classes and is used to create various applications.Components of AWT are heavy weighted.The components of Java Swing are lightweight.Components are platform dependent.Components are platform independent.Execution Time is more than Swing.Execution Time is less than AWT.AWT components require java.awt package.Swing components requires javax.swing package.To know more about the topic, refer to Java Swing vs Java AWT.What is JFC?JFC stands for Java Foundation Classes. JFC is the set of GUI components that simplify desktop Applications. Many programmers think that JFC and Swing are one and the same thing, but that is not so. JFC contains Swing [A UI component package] and quite a number of other items:Cut and paste: Clipboard support.Accessibility features: Aimed at developing GUIs for users with disabilities.The Desktop Colors Features were first introduced in Java 1.1Java 2D: it has Improved colors, images, and text support.Features Of Swing Class Pluggable look and feel.Uses MVC architecture.Lightweight ComponentsPlatform IndependentAdvanced features such as JTable, JTabbedPane, JScollPane, etc.Java is a platform-independent language and runs on any client machine, the GUI look and feel, owned and delivered by a platform-specific O/S, simply does not affect an application's GUI constructed using Swing components.Lightweight Components: Starting with the JDK 1.1, its AWT-supported lightweight component development. For a component to qualify as lightweight, it must not depend on any non-Java [O/s based) system classes. Swing components have their own view supported by Java's look and feel classes.Pluggable Look and Feel: This feature enable the user to switch the look and feel of Swing components without restarting an application. The Swing library supports components' look and feels that remain the same across all platforms wherever the program runs. The Swing library provides an API that gives real flexibility in determining the look and feel of the GUI of an applicationHighly customizable - Swing controls can be customized in a very easy way as visual appearance is independent of internal representation.Rich controls- Swing provides a rich set of advanced controls like Tree TabbedPane, slider, colorpicker, and table controls.Swing Classes HierarchyThe MVC ConnectionIn general, a visual component is a composite of three distinct aspects:The way that the component looks when rendered on the screen.The way such that the component reacts to the user.The state information associated with the component.Over the years, one component architecture has proven itself to be exceptionally effective: - Model-View-Controller or MVC for short.In MVC terminology, the model corresponds to the state information associated with the Component.The view determines how the component is displayed on the screen, including any aspects of the view that are affected by the current state of the model.The controller determines how the component reacts to the user.The simplest Swing components have capabilities far beyond AWT components as follows:Swing buttons and labels can be displaying images instead of or in addition to text.The borders around most Swing components can be changed easily. For example, it is easy to put a 1-pixel border around the outside of a Swing label.Swing components do not have to be rectangular. Buttons, for example, can be round.Now The Latest Assertive technologies such as screen readers can easily get information from Swing components. Example: A screen reader tool can easily capture the text that is displayed on a Swing button or label.Example of Java Swing ProgramsExample 1: Develop a program using label (swing) to display the message GFG WEB Site Click: Java // Java program using label (swing)// to display the message GFG WEB Site Clickimport java.io.*;import javax.swing.*; // Main classclass GFG { // Main driver method public static void main(String[] args) { // Creating instance of JFrame JFrame frame = new JFrame(); // Creating instance of JButton JButton button = new JButton(" GFG WebSite Click"); // x axis, y axis, width, height button.setBounds(150, 200, 220, 50); // adding button in JFrame frame.add(button); // 400 width and 500 height frame.setSize(500, 600); // using no layout managers frame.setLayout(null); // making the frame visible frame.setVisible(true); }}Output:Example 2: Write a program to create three buttons with caption OK, SUBMIT, CANCEL. Java // Java program to create three buttons// with caption OK, SUBMIT, CANCELimport java.awt.*; class button { button() { Frame f = new Frame(); // Button 1 created // OK button Button b1 = new Button("OK"); b1.setBounds(100, 50, 50, 50); f.add(b1); // Button 2 created // Submit button Button b2 = new Button("SUBMIT"); b2.setBounds(100, 101, 50, 50); f.add(b2); // Button 3 created // Cancel button Button b3 = new Button("CANCEL"); b3.setBounds(100, 150, 80, 50); f.add(b3); f.setSize(500, 500); f.setLayout(null); f.setVisible(true); } public static void main(String a[]) { new button(); } }Output:Output of Example 2Example 3:Program to Add Checkbox in the Frame Java // Java Swing Program to Add Checkbox// in the Frameimport java.awt.*; // Driver Classclass Lan { // Main Function Lan() { // Frame Created Frame f = new Frame(); Label l1 = new Label("Select known Languages"); l1.setBounds(100, 50, 120, 80); f.add(l1); // CheckBox created Checkbox c2 = new Checkbox("Hindi"); c2.setBounds(100, 150, 50, 50); f.add(c2); // CheckBox created Checkbox c3 = new Checkbox("English"); c3.setBounds(100, 200, 80, 50); f.add(c3); // CheckBox created Checkbox c4 = new Checkbox("marathi"); c4.setBounds(100, 250, 80, 50); f.add(c4); f.setSize(500, 500); f.setLayout(null); f.setVisible(true); } public static void main(String ar[]) { new Lan(); } }Output:Output of Example 3Components of Swing Class the task's percentageClassDescriptionComponentA Component is the Abstract base class for about the non-menu user-interface controls of Java SWING. Components are representing an object with a graphical representation.ContainerA Container is a component that can container Java SWING ComponentsJComponentA JComponent is a base class for all swing UI Components In order to use a swing component that inherits from JComponent, the component must be in a containment hierarchy whose root is a top-level Java Swing container.JLabelA JLabel is an object component for placing text in a container.JButtonThis class creates a labeled button.JColorChooser A JColorChooser provides a pane of controls designed to allow the user to manipulate and select a color.JCheckBoxA JCheckBox is a graphical (GUI) component that can be in either an on-(true) or off-(false) state.JRadioButtonThe JRadioButton class is a graphical (GUI) component that can be in either an on-(true) or off-(false) state. in the groupJListA JList component represents the user with the scrolling list of text items.JComboBoxA JComboBox component is Presents the User with a show up Menu of choices.JTextFieldA JTextField object is a text component that will allow for the editing of a single line of text.JPasswordField A JPasswordField object it is a text component specialized for password entry.JTextAreaA JTextArea object is a text component that allows for the editing of multiple lines of text.ImageIconA ImageIcon control is an implementation of the Icon interface that paints Icons from ImagesJScrollbarA JScrollbar control represents a scroll bar component in order to enable users to Select from range values.JOptionPane JOptionPane provides set of standard dialog boxes that prompt users for a value or Something.JFileChooser A JFileChooser it Controls represents a dialog window from which the user can select a file.JProgressBar As the task progresses towards completion, the progress bar displays the tasks percentage on its completion.JSliderA JSlider this class is letting the user graphically (GUI) select by using a value by sliding a knob within a bounded interval.JSpinnerA JSpinner this class is a single line input where the field that lets the user select by using a number or an object value from an ordered sequence. In this article, we learn about GUI in Java with Java AWT, Swing, JavaFX, and some other relevant concepts.We'll discuss:IntrdoductionJava AWT (Abstract Window Toolkit)Java SwingJavaFXCreating LayoutBorderLayoutFlowLayoutGridLayoutEvent HandlingWorking with Images and Audio FilesConclusionIntroductionGraphical User Interface (GUI) is an essential part of modern software development. It provides a visual interface for users to interact with the computer and access its functions and features. GUI has become an indispensable aspect of software development, making it easier for users to perform tasks and interact with the computer.Java, one of the most popular programming languages in the world, provides several libraries for GUI development. Over the years, Java's GUI libraries have evolved to cater to the changing needs of software development, and today, Java offers some of the most advanced and sophisticated GUI libraries available.In this blog, we will explore the basics of GUI development in Java, including the history of Java's GUI libraries, and delve into the key features and examples of using the Abstract Window Toolkit (AWT), Swing, and JavaFX. We will also cover important concepts such as layout management, event handling, and working with images and media. By the end of this blog, you will have a solid understanding of GUI development in Java and be able to create your own GUI interfaces with confidence.Java AWT (Abstract Window Toolkit)The Abstract Window Toolkit (AWT) was introduced as part of Java's standard library in its early days and was the first GUI library for Java. AWT provides a set of basic components and widgets, such as buttons, labels, text fields, and more, that developers can use to create simple GUI interfaces.One of the key features of AWT is its cross-platform compatibility. AWT was designed to provide a consistent look and feel across different operating systems, allowing developers to write once and run anywhere. This is achieved by using the underlying operating system's native GUI components, and abstracting the differences between different platforms.AWT is still widely used today, particularly for applications that require a simple and straightforward GUI. Here are a few examples of using AWT to create basic GUI interfaces:Creating a simple window:// Import the required classes from the java.awt and java.awt.event packagesimport java.awt.*;import java.awt.event.*;public class AWTExample { // Declare the components of the GUI private Frame mainFrame; private Label headerLabel; private Label statusLabel; private Panel controlPanel; public AWTExample() { // Call the prepareGUI method to set up the GUI prepareGUI(); } public static void main(String[] args) { // Create an instance of the AWTExample class and call the showEventDemo method AWTExample awtExample = new AWTExample(); awtExample.showEventDemo(); } private void prepareGUI() { // Create the main frame and set its size and layout mainFrame = new Frame("Java AWT Examples"); mainFrame.setSize(400, 400); mainFrame.setLayout(new GridLayout(3, 1)); // Add a window listener to the main frame to close the application when the close button is clicked mainFrame.addWindowListener(new WindowAdapter() { public void windowClosing(WindowEvent windowEvent) { System.exit(0); } }); // Create the header label and set its alignment headerLabel = new Label(); headerLabel.setAlignment(Label.CENTER); // Create the status label and set its alignment and size statusLabel = new Label(); statusLabel.setAlignment(Label.CENTER); statusLabel.setSize(350, 100); // Create the control panel and set its layout controlPanel = new Panel(); controlPanel.setLayout(new FlowLayout()); // Add the header label, control panel, and status label to the main frame mainFrame.add(headerLabel); mainFrame.add(controlPanel); mainFrame.add(statusLabel); // Make the main frame visible mainFrame.setVisible(true); } private void showEventDemo() { // Set the text of the header label headerLabel.setText("Control in action: Button"); // Create the buttons and add action listeners to them Button okButton = new Button("OK"); Button submitButton = new Button("Submit"); Button cancelButton = new Button("Cancel"); okButton.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent e) { // Set the text of the status label when the OK button is clicked statusLabel.setText("Ok Button clicked."); } }); submitButton.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent e) { // Set the text of the status label when the Submit button is clicked statusLabel.setText("Submit Button clicked."); } }); cancelButton.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent e) { // Set the text of the status label when the Cancel button is clicked statusLabel.setText("Cancel Button clicked."); } }); // Add buttons to the control panel controlPanel.add(okButton); controlPanel.add(submitButton); controlPanel.add(cancelButton); // Set the visibility of the main frame to true mainFrame.setVisible(true); }}The output look like this:Creating a simple dialog box:// Import the required libraries for the AWT components and event handlingimport java.awt.*;import java.awt.event.*;public class AWTDialogExample { // Declare the main frame and the required labels and panels private Frame mainFrame; private Label headerLabel; private Label statusLabel; private Panel controlPanel; public AWTDialogExample() { // Call the prepareGUI method to set up the main frame and its components prepareGUI(); } public static void main(String[] args) { // Create a new instance of the AWTDialogExample class and call the showEventDemo method AWTDialogExample awtDialogExample = new AWTDialogExample(); awtDialogExample.showEventDemo(); } private void prepareGUI() { // Initialize the main frame mainFrame = new Frame("Java AWT Dialog Example"); mainFrame.setSize(400, 400); // Set the layout of the main frame to a 3x1 grid layout mainFrame.setLayout(new GridLayout(3, 1)); // Add a window listener to the main frame to exit the program when the window is closed mainFrame.addWindowListener(new WindowAdapter() { public void windowClosing(WindowEvent windowEvent) { System.exit(0); } }); // Initialize the header label and set its alignment to center headerLabel = new Label(); headerLabel.setAlignment(Label.CENTER); // Initialize the status label, set its alignment to center, and set its size statusLabel = new Label(); statusLabel.setAlignment(Label.CENTER); statusLabel.setSize(350, 100); // Initialize the control panel and set its layout to a flow layout controlPanel = new Panel(); controlPanel.setLayout(new FlowLayout()); // Add the header label, control panel, and status label to the main frame mainFrame.add(headerLabel); mainFrame.add(controlPanel); mainFrame.add(statusLabel); // Set the main frame to be visible mainFrame.setVisible(true); } private void showEventDemo() { // Set the text of the header label to "Control in action: Dialog" headerLabel.setText("Control in action: Dialog"); // Initialize a button to show the dialog Button showDialogButton = new Button("Show Dialog"); // Add an action listener to the show dialog button showDialogButton.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent e) { // Initialize a new dialog with the main frame as its parent, a title of "Dialog Example", and modality set to true Dialog d = new Dialog(mainFrame, "Dialog Example", true); // Set the layout of the dialog to a flow layout d.setLayout(new FlowLayout()); // Initialize a button to dispose of the dialog Button b = new Button("OK"); b.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent e) { // Dispose of the dialog when the OK button is clicked d.dispose(); } }); d.add(new Label("This is a Dialog")); //add a label to the Dialog d.add(b); //add the OK button to the Dialog d.setSize(300, 300); //set the size of the Dialog d.setVisible(true); //make the Dialog visible to the user } }); // add the showDialogButton to the mainFrame controlPanel.add(showDialogButton); //make the mainFrame visible to the user mainFrame.setVisible(true); }}The dialog box will look like this:AWT is still a popular choice for GUI development, especially for applications that require a simple and straightforward GUI. However, AWT has several limitations and drawbacks, such as its lack of advanced features and poor performance compared to more modern GUI libraries.In the next section, we will take a look at the Swing library, which was introduced as an improvement over AWT and provides many additional features and improvements.Java SwingJava Swing is a GUI widget toolkit for Java that provides a platform-independent and lightweight solution for building rich graphical user interfaces. It was introduced as a part of the Java Foundation Classes (JFC) and is an improvement over the earlier Abstract Window Toolkit (AWT) in terms of performance, features, and versatility.Some of the key features of Java Swing include:Lightweight components: Swing components are lightweight, meaning they are drawn entirely in software, which makes them more flexible and versatile.Platform-independent: Java Swing is platform-independent, which means that applications built using Swing can run on any operating system that supports Java without the need for any modification.Rich set of components: Swing provides a rich set of components, including buttons, labels, text boxes, tables, trees, and more.Customizable look and feel: The look and feel of the Swing components can be customized, either by using the built-in look and feel options or by creating a custom look and feel.Model-View-Controller (MVC) architecture: Swing components are based on the Model-View-Controller (MVC) architecture, which separates the data representation from the user interface, making it easier to develop, maintain, and update the application.Here is an example of a simple Java Swing program that creates a basic window with a button:import javax.swing.;import java.awt.;import java.awt.event.*;// Main classpublic class SwingExample { // Variables for frame, labels and panel private JFrame mainFrame; private JLabel headerLabel; private JLabel statusLabel; private JPanel controlPanel; // Constructor public SwingExample() { prepareGUI(); } public static void main(String[] args) { // Create instance of the class SwingExample swingExample = new SwingExample(); swingExample.showEventDemo(); } // Method to initialize the GUI components private void prepareGUI() { // Create a frame and set its properties mainFrame = new JFrame("Java Swing Examples"); mainFrame.setSize(400, 400); // Set layout to be a BoxLayout with Y_AXIS orientation mainFrame.setLayout(new BoxLayout(mainFrame.getContentPane(), BoxLayout.Y_AXIS)); // Add a window listener to exit the program when the window is closed mainFrame.addWindowListener(new WindowAdapter() { public void windowClosing(WindowEvent windowEvent) { System.exit(0); } }); // Initialize the header label and set its properties headerLabel = new JLabel("", JLabel.CENTER); // Initialize the status label and set its properties statusLabel = new JLabel("", JLabel.CENTER); statusLabel.setSize(350, 100); // Initialize the control panel controlPanel = new JPanel(); controlPanel.setLayout(new FlowLayout()); // Add the header label, control panel, and status label to the main frame mainFrame.add(headerLabel); mainFrame.add(controlPanel); mainFrame.add(statusLabel); mainFrame.setVisible(true); } // Method to display event demo private void showEventDemo() { // Set the text for the header label headerLabel.setText("Control in action: Button"); // Create an "OK" button JButton okButton = new JButton("OK"); // Set the action command for the button to "OK" okButton.setActionCommand("OK"); // Add an action listener to the button okButton.addActionListener(new ButtonClickListener()); // Add the button to the control panel controlPanel.add(okButton); mainFrame.setVisible(true); } // Inner class for the button click listener private class ButtonClickListener implements ActionListener { public void actionPerformed(ActionEvent e) { // Get the action command from the button String command = e.getActionCommand(); // Check if the action command is "OK" if (command.equals("OK")) { // If it is, set the text of the status label to "OK Button clicked." statusLabel.setText("OK Button clicked."); } } }}The output of the code is:In this example, a JFrame is created, which serves as the main window for the application. A JLabel is used to display text at the top of the window, a JPanel is used to contain the buttons, and a JButton is used to trigger an action when clicked. The button's action is defined by an ActionListener, which is a separate class in this example. When the button is clicked, the text in the JLabel changes to reflect the action.This is just a simple example, but it demonstrates the basic building blocks of a Java Swing application and how they work together to create a functional GUI. Java Swing provides many more components and features that can be used to create more complex and sophisticated user interfaces.Difference between Java AWT and Java SwingJava AWT (Abstract Window Toolkit) is the original Java GUI toolkit, introduced in the early days of Java. It provides a basic set of graphical user interface (GUI) components, such as buttons, text boxes, and labels, among others.Java Swing, on the other hand, is a more advanced and feature-rich GUI toolkit that builds on top of AWT. It provides a wider range of components, including table, tree, and tabbed panels, as well as advanced components such as progress bars and sliders.In terms of performance, Swing is faster and more efficient than AWT because it is implemented entirely in Java and does not rely on native components. AWT components, on the other hand, are implemented as native components, which can lead to performance issues and platform-specific quirks.When it comes to look and feel, AWT components have a basic and plain look, whereas Swing components can be customized to match the look and feel of the underlying operating system or a specific look and feel.In conclusion, both Java AWT and Java Swing have their own strengths and weaknesses, and the choice between the two largely depends on the specific requirements of your project. For simple GUI applications, AWT may be sufficient, but for more complex applications with advanced requirements, Swing is the better choice.JavaFXJavaFX is a modern and powerful GUI toolkit that was introduced as a part of Java SE in Java 8. It provides a rich set of graphical user interface components, animation, and multimedia capabilities. JavaFX is designed to provide a consistent and unified look and feel across all platforms, regardless of the underlying operating system.Creating a simple JavaFX interface involves several steps:Setting up the environment: JavaFX is bundled with Java SE, so you don't need to install any additional libraries. You'll need to make sure that you have the latest version of Java installed, and then set up your development environment with a suitable Java IDE, such as Eclipse or IntelliJ IDEA.Creating a new JavaFX project: In your IDE, create a new JavaFX project and add a new JavaFX class to the project.Writing the code: In the JavaFX class, you'll write code to create the interface. JavaFX uses a declarative programming style, where you specify the components and their properties, and the underlying system takes care of rendering the components.Here is an example code to create a simple JavaFX interface with a button and a label:import javafx.application.Application;import javafx.event.ActionEvent;import javafx.event.EventHandler;import javafx.scene.Scene;import javafx.scene.control.Button;import javafx.scene.control.Label;import javafx.scene.layout.StackPane;import javafx.stage.Stage;// JavaFXExample class extends the Application classpublic class JavaFXExample extends Application { public static void main(String[] args) { launch(args); } @Override public void start(Stage primaryStage) { // Set the title for the primary stage primaryStage.setTitle("JavaFX Example"); // Create a button and set its text Button btn = new Button(); btn.setText("Click Me"); // Add an action event to the button btn.setOnAction(new EventHandler() { @Override public void handle(ActionEvent event) { System.out.println("Hello JavaFX"); } }); // Create a label and set its text Label label = new Label(); label.setText("Welcome to JavaFX"); // Create a StackPane layout and add the button and label StackPane root = new StackPane(); root.getChildren().add(btn); root.getChildren().add(label); // Set the scene to the primary stage and show the stage primaryStage.setScene(new Scene(root, 300, 250)); primaryStage.show(); }}In the code above, we create a button and a label, and add them to a StackPane layout. The StackPane layout allows components to be stacked on top of each other. We also set the text for the button and label, and attach an event handler to the button to print a message when it's clicked. Finally, we create a scene and set it as the content of the primary stage, and show the stage.Here's a brief overview of the difference between the two:Architecture: Swing is based on the Model-View-Controller (MVC) architecture, while JavaFX uses a Scene Graph architecture. The Scene Graph architecture allows for more sophisticated UI design and animation capabilities compared to the MVC architecture used by Swing.Look and Feel: JavaFX provides a more modern and aesthetically pleasing look and feel compared to Swing. JavaFX also offers a wider range of UI components and more flexible styling options.Performance: JavaFX has better performance compared to Swing, especially when it comes to UI animations and graphics-intensive applications.Features: JavaFX offers a number of advanced features such as built-in support for 3D graphics, video and audio playback, and multi-touch gestures. These features are not available in Swing.Ease of Development: JavaFX provides a more streamlined and easy-to-use API compared to Swing, making it easier for developers to create complex and sophisticated UIs.Overall, JavaFX provides a more modern and advanced approach to UI design compared to Swing, and is better suited for complex and sophisticated UI applications. However, Swing is still widely used and has a large user community, and may be a better choice for projects with specific requirements or constraints.Creating LayoutWhen building graphical user interfaces in Java, it is important to consider the layout of the interface. A well-designed layout not only improves the look and feel of the application, but also makes it more user-friendly. Java offers several built-in layout managers to arrange components in a GUI, such as BorderLayout, FlowLayout, and GridLayout. In this section, we will introduce these layout managers and show examples of how they can be used to arrange components in a GUI.BorderLayoutBorderLayout is one of the most commonly used layout managers in Java. It divides a container into five areas: north, south, east, west, and center. Components can be added to each of these areas, and they will be positioned accordingly. For example, a button can be added to the north region, and it will be positioned at the top of the container.import java.awt.*;import javax.swing.*;public class BorderLayoutExample { public static void main(String[] args) { // create the JFrame with title "BorderLayout Example" JFrame frame = new JFrame("BorderLayout Example"); // set the default close operation of the frame frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // get the contentPanel of the frame Container contentPanel = frame.getContentPane(); // set the layout manager of the contentPane to BorderLayout contentPanel.setLayout(new BorderLayout()); // create button1 and set its text to "North" JButton button1 = new JButton("North"); // add button1 to the north region of the contentPane contentPanel.add(button1, BorderLayout.NORTH); // create button2 and set its text to "South" JButton button2 = new JButton("South"); // add button2 to the south region of the contentPane contentPanel.add(button2, BorderLayout.SOUTH); // create button3 and set its text to "East" JButton button3 = new JButton("East"); // add button3 to the east region of the contentPane contentPanel.add(button3, BorderLayout.EAST); // create button4 and set its text to "West" JButton button4 = new JButton("West"); // add button4 to the west region of the contentPane contentPanel.add(button4, BorderLayout.WEST); // create button5 and set its text to "Center" JButton button5 = new JButton("Center"); // add button5 to the center region of the contentPane contentPanel.add(button5, BorderLayout.CENTER); // pack the frame so that all its components are at or above their preferred sizes frame.pack(); // set the visibility of the frame to true frame.setVisible(true); }}Border Layout will look like this:FlowLayoutFlowLayout is another popular layout manager in Java. It positions components in a left-to-right, top-to-bottom flow. When the components reach the end of the container, they will be positioned on a new row.import java.awt.*;import javax.swing.*;public class FlowLayoutExample { public static void main(String[] args) { // Create a new JFrame instance with the title "FlowLayout Example" JFrame frame = new JFrame("FlowLayout Example"); // Set the default close operation for the frame to exit the application frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Get the content pane of the frame and set its layout to FlowLayout Container contentPane = frame.getContentPane(); contentPane.setLayout(new FlowLayout()); // Create three buttons with text "Button 1", "Button 2", and "Button 3" JButton button1 = new JButton("Button 1"); contentPane.add(button1); JButton button2 = new JButton("Button 2"); contentPane.add(button2); JButton button3 = new JButton("Button 3"); contentPane.add(button3); // Pack the frame and set it to be visible frame.pack(); frame.setVisible(true); }}Flow Layout will look like this:GridLayoutGridLayout is used to arrange components in a grid of equally sized cells. The number of rows and columns can be specified, and components will be positioned in the cells based on their index.import java.awt.GridLayout;import javax.swing.JButton;import javax.swing.JFrame;import javax.swing.JPanel;public class GridLayoutExample { public static void main(String[] args) { JFrame frame = new JFrame("GridLayout Example"); JPanel panel = new JPanel(); panel.setLayout(new GridLayout(2, 2)); panel.add(new JButton("Button 1")); panel.add(new JButton("Button 2")); panel.add(new JButton("Button 3")); panel.add(new JButton("Button 4")); frame.add(panel); frame.setSize(300, 300); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setVisible(true); }}Grid Layout will look like this:Event HandlingEvent handling is a fundamental part of creating GUI interfaces in Java. An event is a user-generated action such as a mouse click, button press, or key press. In Java GUI, event handling is implemented using event listeners, which listen for events and take the appropriate action when the event occurs.Java provides a wide range of event listeners to handle different types of events, including mouse events, keyboard events, and action events. In this section, we'll cover some of the most commonly used event listeners and demonstrate how to use them to handle events in Java GUI.ActionEvent Listener: The ActionEvent listener is used to handle action events such as button clicks. To use an ActionEvent listener, you must create an action listener class that implements the ActionListener interface and register it with the button using the addActionListener() method.Here is an example of how to use an ActionEvent listener to handle a button click:import java.awt.FlowLayout;import java.awt.event.ActionEvent;import java.awt.event.ActionListener;import javax.swing.JButton;import javax.swing.JFrame;import javax.swing.JLabel;public class ActionEventHandlingExample { public static void main(String[] args) { // Create a JFrame with the title "Event Handling Example" JFrame frame = new JFrame("Event Handling Example"); // Set the layout of the frame to FlowLayout frame.setLayout(new FlowLayout()); // Set the size of the frame to 400x200 frame.setSize(400, 200); // Set the default close operation to exit the program when the frame is closed frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Create a JButton with the text "Click Me" JButton button = new JButton("Click Me"); // Create a JLabel to display the button click event JLabel label = new JLabel(); // Add an action listener to the button button.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent e) { // When the button is clicked, set the text of the label to "Button Clicked" label.setText("Button Clicked"); } }); // Add the button and label to the frame frame.add(button); frame.add(label); // Make the frame visible frame.setVisible(true); }}The output of this code is:MouseEvent Listener: The MouseEvent listener is used to handle mouse events such as mouse clicks and mouse movements. To use a MouseEvent listener, you must create a mouse listener class that implements the MouseListener interface and register it with the component using the addMouseListener() method.Here is an example of how to use a MouseEvent listener to handle a mouse click:import java.awt.event.MouseAdapter;import java.awt.event.MouseEvent;import java.awt.*;import javax.swing.JFrame;import javax.swing.JLabel;public class MouseEventHandlingExample extends JFrame { // Declare a JLabel instance variable to display status private JLabel statusBar; // constructor to set up the GUI components public MouseEventHandlingExample() { // Set the layout to FlowLayout setLayout(new FlowLayout()); // Initialize the JLabel with default text statusBar = new JLabel("default"); // Add the JLabel to the frame add(statusBar); // Add mouse listener to the frame addMouseListener(new MouseClass()); } // Inner class to handle mouse events private class MouseClass extends MouseAdapter { public void mouseClicked(MouseEvent event) { // Set the text of the JLabel to the mouse click coordinates statusBar.setText(String.format("Clicked at %d, %d", event.getX(), event.getY())); } } // Main method to create the frame and display it public static void main(String[] args) { // Create an instance of MouseEventHandlingExample MouseEventHandlingExample mouseEventExample = new MouseEventHandlingExample(); // Set the default close operation to EXIT_ON_CLOSE mouseEventExample.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Set the size of the frame mouseEventExample.setSize(300, 100); // Set the visibility of the frame to true mouseEventExample.setVisible(true); // Set the title of the frame mouseEventExample.setTitle("MouseEvent Example"); }}The output of this code is:There are many more event listeners that can be used in Java GUI applications to handle different types of events. Some of the common event listeners include KeyListener, WindowListener, FocusListener, ComponentListener, and ContainerListener.here is a brief description of each of the listeners:KeyListener: A KeyListener listens for events related to the user's interaction with the keyboard. It is used to track when a key is pressed, released, or typed.WindowListener: A WindowListener listens for events related to windows, such as when a window is opened or closed, minimized or maximized, or when it gains or loses focus.FocusListener: A FocusListener listens for events related to the focus of a component, such as when a component gains or loses focus. This is useful for tracking which component has the focus at any given time.ContainerListener: A ContainerListener listens for events related to the components contained within a container, such as when a component is added or removed from the

container. This is useful for tracking changes to the structure of a container and updating the GUI accordingly. Java provides built-in support for working with images and media in GUI applications. You can easily display images and play audio files in your Java GUI applications by using the appropriate classes and methods provided by the Java libraries.For displaying images, you can use the javax.swing.ImageIcon class. This class provides a simple way to load an image and display it in your application. To display the image, you simply need to create an instance of the ImageIcon class, passing the path to the image file as a parameter, and then set the icon of a JLabel to the instance of the ImageIcon class.ImageIcon image = new ImageIcon("path/to/image.jpg");JLabel imageLabel = new JLabel(image);For playing audio files, you can use the javax.sound.sampled.AudioSystem class. This class provides a simple way to play audio files in your application. To play an audio file, you simply need to create an instance of the AudioClip class, passing the path to the audio file as a parameter, and then call the start() method on the instance.AudioClip audio = AudioSystem.getClip();audio.open(AudioSystem.getAudioInputStream(new File("path/to/audio.wav")));audio.start();In conclusion, working with images and media in Java GUI is a simple and straightforward process, thanks to the built-in support provided by the Java libraries. With a few lines of code, you can easily display images and play audio files in your applications, adding an extra layer of interactivity and visual appeal.ConclusionIn this article at OpenGenus, we have covered various aspects of creating graphical user interfaces in Java, including the history and importance of GUI, the key features and differences between the various GUI libraries in Java (AWT, Swing, and JavaFX), the different layout managers in Java GUI, event handling in Java GUI, and working with images and audio files in Java GUI.In conclusion, Java offers several options for creating interactive and visually appealing graphical user interfaces, each with its own strengths and limitations. Whether you are creating a basic GUI interface using AWT, a more advanced interface using Swing, or a visually appealing interface using JavaFX, Java provides the tools and resources necessary to get the job done. In addition, the event handling and media support provided by Java make it a powerful and versatile platform for GUI development. Swing in Java is a Graphical User Interface (GUI) toolkit that includes the GUI components. Swing provides a rich set of widgets and packages to make sophisticated GUI components for Java applications. Swing is a part of Java Foundation Classes(JFC), which is an API for Java GUI programing that provide GUI. The Java Swing library is built on top of the Java Abstract Widget Toolkit (AWT), an older, platform dependent GUI toolkit. You can use the Java simple GUI programming components like button, textbox, etc., from the library and do not have to create the components from scratch.In this Java Swing tutorial, you will learn Java GUI basics like- Java Swing class Hierarchy Diagram Java Swing Class Hierarchy Diagram All components in Java Swing are JComponent which can be added to container classes. What is a Container Class?Container classes are classes that can have other components on it. So for creating a Java Swing GUI, we need at least one container object. There are 3 types of Java Swing containers. Panel: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window. Frame: It is a fully functioning window with its title and icons. Dialog: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame. GUI (Graphical User Interface) in Java is an easy-to-use visual experience builder for Java applications. It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application. GUI plays an important role to build easy interfaces for Java applications.How to Make a GUI in Java with ExampleNow in this Java GUI Tutorial, lets understand how to create a GUI in Java with Swings in Java examples.Step 1) Copy code into an editorIn first step Copy the following code into an editor.import javax.swing.*;class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button = new JButton("Press"); frame.getContentPane().add(button); // Adds Button to content pane of frame frame.setVisible(true); }} Step 2) Run the codeNext step, Save, Compile, and Run the codeStep 3) Copy following code into an editorNow lets Add a Button to our frame. Copy following code into an editor from given Java UI Exampleimport javax.swing.*; class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button1 = new JButton("Press"); frame.getContentPane().add(button1); frame.setVisible(true); }} Step 4) Execute the codeNext, Execute the code. You will get a big button.Step 5) Add two buttonsHow about adding two buttons? Copy the following code into an editor.import javax.swing.*;class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button1 = new JButton("Button 1"); JButton button2 = new JButton("Button 2"); frame.getContentPane().add(button1); frame.getContentPane().add(button2); frame.setVisible(true); }}Step 6) Save & Run the programNext, Save, Compile, and Run the program.Step 7) Check outputUnexpected output =? Buttons are getting overlapped.Java Layout ManagerThe Layout manager is used to layout (or arrange) the GUI Java components inside a container. There are many layout managers, but the most frequently used are-Java BorderLayoutA BorderLayout places components in up to five areas: top, bottom, left, right, and center. It is the default layout manager for every java JFrameJava BorderLayout Java FlowLayoutFlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row one after the other.Java FlowLayoutJava GridBagLayoutIt is the more sophisticated of all layouts. It aligns components by placing them within a grid of cells, allowing components to span more than one cell. Java GridBagLayout Step 8) Create chat frameHow about creating a chat frame like below?Try to code yourself before looking at the program below.//Usually you will require both swing and awt packages// even if you are working with just swings.import javax.swing.*;import java.awt.*;class gui { public static void main(String args[]) { //Creating the Frame JFrame frame = new JFrame("Chat Frame"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(400, 400); //Creating the MenuBar and adding components JMenuBar mb = new JMenuBar(); JMenu m1 = new JMenu("FILE"); JMenu m2 = new JMenu("Help"); mb.add(m1); mb.add(m2); JMenuItem m11 = new JMenuItem("Open"); JMenuItem m22 = new JMenuItem("Save as"); m1.add(m11); m1.add(m22); //Creating the panel at bottom and adding components JPanel panel = new JPanel(); // the panel is not visible in output JLabel label = new JLabel("Enter Text"); JTextField tf = new JTextField(10); // accepts upto 10 characters JButton send = new JButton("Send"); JButton reset = new JButton("Reset"); panel.add(label); // Components Added using Flow Layout panel.add(tf); panel.add(send); panel.add(reset); // Text Area at the Center JTextArea ta = new JTextArea(); //Adding Components to the frame. frame.getContentPane().add(BorderLayout.SOUTH, panel); frame.getContentPane().add(BorderLayout.NORTH, mb); frame.getContentPane().add(BorderLayout.CENTER, ta); frame.setVisible(true); }} Read More Swing in Java is a Graphical User Interface (GUI) toolkit that includes the GUI components. Swing provides a rich set of widgets and packages to make sophisticated GUI components for Java applications. Swing is a part of Java Foundation Classes(JFC), which is an API for Java GUI programing that provide GUI. The Java Swing library is built on top of the Java Abstract Widget Toolkit (AWT), an older, platform dependent GUI toolkit. You can use the Java simple GUI programming components like button, textbox, etc., from the library and do not have to create the components from scratch.In this Java Swing tutorial, you will learn Java GUI basics like- Java Swing class Hierarchy Diagram Java Swing Class Hierarchy Diagram All components in Java Swing are JComponent which can be added to container classes. What is a Container Class?Container classes are classes that can have other components on it. So for creating a Java Swing GUI, we need at least one container object. There are 3 types of Java Swing containers. Panel: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window. Frame: It is a fully functioning window with its title and icons. Dialog: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame. GUI (Graphical User Interface) in Java is an easy-to-use visual experience builder for Java applications. It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application. GUI plays an important role to build easy interfaces for Java applications.How to Make a GUI in Java with ExampleNow in this Java GUI Tutorial, lets understand how to create a GUI in Java with Swings in Java examples.Step 1) Copy code into an editorIn first step Copy the following code into an editor.import javax.swing.*;class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button = new JButton("Press"); frame.getContentPane().add(button); // Adds Button to content pane of frame frame.setVisible(true); }} Step 2) Run the codeNext step, Save, Compile, and Run the codeStep 3) Copy following code into an editorNow lets Add a Button to our frame. Copy following code into an editor from given Java UI Exampleimport javax.swing.*; class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button1 = new JButton("Press"); frame.getContentPane().add(button1); frame.setVisible(true); }} Step 4) Execute the codeNext, Execute the code. You will get a big button.Step 5) Add two buttonsHow about adding two buttons? Copy the following code into an editor.import javax.swing.*;class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button1 = new JButton("Button 1"); JButton button2 = new JButton("Button 2"); frame.getContentPane().add(button1); frame.getContentPane().add(button2); frame.setVisible(true); }}Step 6) Save & Run the programNext, Save, Compile, and Run the program.Step 7) Check outputUnexpected output =? Buttons are getting overlapped.Java Layout ManagerThe Layout manager is used to layout (or arrange) the GUI Java components inside a container. There are many layout managers, but the most frequently used are-Java BorderLayoutA BorderLayout places components in up to five areas: top, bottom, left, right, and center. It is the default layout manager for every java JFrameJava BorderLayout Java FlowLayoutFlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row one after the other.Java FlowLayoutJava GridBagLayoutIt is the more sophisticated of all layouts. It aligns components by placing them within a grid of cells, allowing components to span more than one cell. Java GridBagLayout Step 8) Create chat frameHow about creating a chat frame like below?Try to code yourself before looking at the program below.//Usually you will require both swing and awt packages// even if you are working with just swings.import javax.swing.*;import java.awt.*;class gui { public static void main(String args[]) { //Creating the Frame JFrame frame = new JFrame("Chat Frame"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(400, 400); //Creating the MenuBar and adding components JMenuBar mb = new JMenuBar(); JMenu m1 = new JMenu("FILE"); JMenu m2 = new JMenu("Help"); mb.add(m1); mb.add(m2); JMenuItem m11 = new JMenuItem("Open"); JMenuItem m22 = new JMenuItem("Save as"); m1.add(m11); m1.add(m22); //Creating the panel at bottom and adding components JPanel panel = new JPanel(); // the panel is not visible in output JLabel label = new JLabel("Enter Text"); JTextField tf = new JTextField(10); // accepts upto 10 characters JButton send = new JButton("Send"); JButton reset = new JButton("Reset"); panel.add(label); // Components Added using Flow Layout panel.add(tf); panel.add(send); panel.add(reset); // Text Area at the Center JTextArea ta = new JTextArea(); //Adding Components to the frame. frame.getContentPane().add(BorderLayout.SOUTH, panel); frame.getContentPane().add(BorderLayout.NORTH, mb); frame.getContentPane().add(BorderLayout.CENTER, ta); frame.setVisible(true); }} Read More There are two sets of Java APIs for graphics programming: AWT (Abstract Windowing Toolkit) and Swing.1. AWT API was introduced in JDK 1.0. Most of the AWT components have become obsolete and should be replaced by newer Swing components.2. Swing API, a much more comprehensive set of graphics libraries that enhances the AWT, was introduced as part of Java Foundation Classes (JFC) after the release of JDK 1.1. JFC consists of Swing, Java2D, Accessibility, Internationalization, and Pluggable Look-and-Feel Support APIs. JFC was an add-on to JDK 1.1 but has been integrated into core Java since JDK 1.2.Other than AWT/Swing Graphics APIs provided in JDK, others have also provided Graphics APIs that work with Java, such as Eclipses Standard Widget Toolkit (SWT) (used in Eclipse), Google Web Toolkit (GWT) (used in Android), 3D Graphics API such as Java bindings for OpenGL (JOGL) and Java3D.2 GUI CreationIn this example we will learn how to create a GUI using eclipse in JAVA.2.1 Setup Prerequisite: This example is developed on Eclipse therefore a compatible Eclipse IDE is required to be installed on the system.We also need WindowBuilder tool to be installed on Eclipse IDE for the easiness of the work.Following Steps are required to install the WindowBuilder tool.Go to Eclipse Help Install New SoftwareInstalling WindowBuilder ToolSelect your version of eclipse version/download/eclipse.org/release/eclipse version, For example, Mars General purpose tools from the dropdown and click next.Installing WindowBuilder ToolThis will take some time to install the software, so you have to restart eclipse in order to see the changes.2.2 Programming GUI with AWTJava Graphics APIs AWT and Swing provide a huge set of reusable GUI components, such as button, text field, label, choice, panel and frame for building GUI applications. You can simply reuse these classes rather than re-invent the wheels. I shall start with the AWT classes before moving into Swing to give you a complete picture. I have to stress that AWT component classes are now obsoleted by Swings counterparts.2.2.1 AWT PackagesAWT is huge! It consists of 12 packages (Swing is even bigger, with 18 packages as of JDK 1.8). Fortunately, only 2 packages java.awt and java.awt.event are commonly-used.1.The java.awt package contains the core AWT graphics classes:GUI Component classes (such as Button, TextField, and Label),GUI Container classes (such as Frame, Panel, Dialog and ScrollPane),Layout managers (such as FlowLayout, BorderLayout and GridLayout),Custom graphics classes (such as Graphics, Color and Font).2.The java.awt.event package supports event handling:Event classes (such as ActionEvent, MouseEvent, KeyEvent and WindowEvent),Event Listener Interfaces (such as ActionListener, MouseListener, KeyListener and WindowListener),Event Listener Adapter classes (such as MouseAdapter, KeyAdapter, and WindowAdapter).AWT provides a platform-independent and device-independent interface to develop graphic programs that runs on all platforms, such as Windows, Mac, and Linux.2.2.2 Containers and ComponentsThere are two types of GUI elements:1. Component: Components are elementary GUI entities (such as Button, Label, and TextField.)2. Container: Containers (such as Frame and Panel) are used to hold components in a specific layout (such as flow or grid). A container can also hold sub-containers.GUI components are also called controls (Microsoft ActiveX Control), widgets (Eclipses Standard Widget Toolkit, Google Web Toolkit), which allow users to interact with (or control) the application through these components (such as button-click and text-entry).A Frame is the top-level container of an AWT program. A Frame has a title bar (containing an icon, a title, and the minimize/maximize/close buttons), an optional menu bar and the content display area. A Panel is a rectangular area used to group related GUI components in a certain layout. In the above figure, the top-level Frame contains two Panels. There are five components: a Label (providing description), a TextField (for users to enter text), and three Buttons (for user to trigger certain programmed actions).In a GUI program, a component must be kept in a container. You need to identify a container to hold the components. Every container has a method called add(Component c). A container (say aContainer) can invoke aContainer.add(aComponent) to add aComponent into itself. For example,MyGUIProgram.javaPanel panel = new Panel(); // Panel is a ContainerButton btn = new Button("Press"); // Button is a Componentpanel.add(btn); // The Panel Container adds a Button Component2.2.2 AWT Container Classes Top-Level Containers: Frame, Dialog and Applet Each GUI program has a top-level container. The commonly-used top-level containers in AWT are Frame, Dialog and Applet:A Frame provides the main window for the GUI application, which has a title bar (containing an icon, a title, the minimize, maximize/restore-down and close buttons), an optional menu bar, and the content display area. To write a GUI program, we typically start with a subclass extending from java.awt.Frame to inherit the main window as follows:MyGUIProgram.javaimport java.awt.Frame; // Using Frame class in package java.awt// A GUI program is written as a subclass of Frame - the top-level container// This subclass inherits all properties from Frame, e.g., title, icon, buttons, content-panepublic class MyGUIProgram extends Frame { // Constructor to setup the GUI components public MyGUIProgram() { ...... } // Other methods .... // The entry main() method public static void main(String[] args) { // Invoke the constructor (to setup the GUI) by allocating an instance new MyGUIProgram(); }}An AWT Dialog is a pop-up window used for interacting with the users. A Dialog has a title-bar (containing an icon, a title and a close button) and a content display area, as illustrated.An AWT Applet (in package java.applet) is the top-level container for an applet, which is a Java program running inside a browser. Applet will be discussed in the later chapter.Secondary Containers: Panel and ScrollPane Secondary containers are placed inside a top-level container or another secondary container. AWT also provide these secondary containers:Panel: a rectangular box under a higher-level container, used to layout a set of related GUI components in pattern such as grid or flow.ScrollPane: provides automatic horizontal and/or vertical scrolling for a single child component.2.2.2 AWT Component ClassesAWT provides many ready-made and reusable GUI components. The frequently-used are: Button, TextField, Label, Checkbox, CheckboxGroup (radio buttons), List, and Choice, as illustrated below.AWT GUI Component:java.awt.Label A java.awt.Label provides a text description message. Take note that System.out.println() prints to the system console, not to the graphics screen. You could use a Label to label another component (such as text field) or provide a text description.Check the JDK API specification for java.awt.Label.Below is how a label will look like:Inserting a LabelConstructors public Label(String strLabel, int alignment); // Construct a Label with the given text String, of the text alignmentpublic Label(String strLabel); // Construct a Label with the given text Stringpublic Label(); // Construct an initially empty LabelThe Label class has three constructors:1.The first constructor constructs a Label object with the given text string in the given alignment. Note that three static constants Label.LEFT, Label.RIGHT, and Label.CENTER are defined in the class for you to specify the alignment (rather than asking you to memorize arbitrary integer values).2.The second constructor constructs a Label object with the given text string in default of left-aligned.3.The third constructor constructs a Label object with an initially empty string. You could set the label text via the setText() method later.Constructing a Component and Adding the Component into a ContainerThree steps are necessary to create and place a GUI component:1.Declare the component with an identifier (name);2.Construct the component by invoking an appropriate constructor via the new operator;3.Identify the container (such as Frame or Panel) designed to hold this component. The container can then add this component onto itself via aContainer.add(aComponent) method. Every container has an add(Component) method. Take note that it is the container that actively and explicitly adds a component onto itself, instead of the other way.Below is the code to add a label(component) to a frame(container)MyGUIProgram.java JLabel lblName = new JLabel("Name");lblName.setBounds(93, 67, 46, 14);frame.getContentPane().add(lblName);AWT GUI Component: java.awt.Button A java.awt.Button is a GUI component that triggers a certain programmed action upon clicking.Constructorspublic Button(String buttonLabel);// Construct a Button with the given labelpublic Button();// Construct a Button with empty labelThe Button class has two constructors. The first constructor creates a Button object with the given label painted over the button. The second constructor creates a Button object with no label.Here is how button will look like:Insertion of a buttonclicking of a button generates an event. code for adding a button is described below :MyGUIProgram.javaJButton btnSubmit = new JButton("Submit"); btnSubmit.addActionListener(new ActionListener() {public void actionPerformed(ActionEvent arg0) {JOptionPane.showMessageDialog(null, "Data Submitted");}});btnSubmit.setBounds(93, 121, 89, 23);frame.getContentPane().add(btnSubmit); AWT GUI Component: java.awt.TextField A java.awt.TextField is single-line text box for users to enter texts. (There is a multiple-line text box called TextArea.) Hitting the ENTER key on a TextField object triggers an action-event.Constructorspublic TextField(String strInitialText, int columns);// Construct a TextField instance with the given initial text string with the number of columns.public TextField(String strInitialText);// Construct a TextField instance with the given initial text string.public TextField(int columns);// Construct a TextField instance with the number of columns.Below is how a TextField will look like :Inserting a TextField3. AWT Event-HandlingJava adopts the so-called Event-Driven (or Event-Delegation) programming model for event-handling, similar to most of the visual programming languages (such as Visual Basic and Delphi).In event-driven programming, a piece of event-handling codes is executed (or called back by the graphics subsystem) when an event has been fired in response to an user input (such as clicking a mouse button or hitting the ENTER key). This is unlike the procedural model, where codes are executed in a sequential manner.The AWTs event-handling classes are kept in package java.awt.event.Three objects are involved in the event-handling: a source, listener(s) and an event object.The source object (such as Button and Textfield) interacts with the user. Upon triggered, it creates an event object. This event object will be messaged to all the registered listener object(s), and an appropriate event-handler method of the listener(s) is called-back to provide the response. In other words, triggering a source fires an event to all its listener(s), and invoke an appropriate handler of the listener(s).The sequence of steps is illustrated above:The source object registers its listener(s) for a certain type of event.Source object fires event event upon triggered. For example, clicking an Button fires an ActionEvent, mouse-click fires MouseEvent, key-type fires KeyEvent, etc.How the source and listener understand each other? The answer is via an agreed-upon interface. For example, if a source is capable of firing an event called XxxEvent (e.g., MouseEvent) involving various operational modes (e.g., mouse-clicked, mouse-entered, mouse-exited, mouse-pressed, and mouse-released). Firstly, we need to declare an interface called XxxListener (e.g., MouseListener) containing the names of the handler methods. Recall that an interface contains only abstract methods without implementationSecondly, all the listeners interested in the XxxEvent must implement the XxxListener interface. That is, the listeners must provide their own implementations (i.e., programmed responses) to all the abstract methods declared in the XxxListener interface. In this way, the listenser(s) can response to these events appropriately.Thirdly, in the source, we need to maintain a list of listener object(s), and define two methods: addXxxListener() and removeXxxListener() to add and remove a listener from this list.Take note that all the listener(s) interested in the XxxEvent must implement the XxxListener interface. That is, they are sub-type of the XxxListener. Hence, they can be upcasted to XxxListener and passed as the argument of the above methods.In summary, we identify the source, the event-listener interface, and the listener object. The listener must implement the event-listener interface. The source object then registers listener object via the addXxxListener() methodThe source is triggered by a user.The source create an XxxEvent object, which encapsulates the necessary information about the activation. For example, the (x, y) position of the mouse pointer, the text entered, etc.Finally, for each of the listeners in the listener list, the source invokes the appropriate handler on the listener(s), which provides the programmed response.4. DownloadThis was an example of How to create a GUI in JAVA.DownloadYou can download the full source code of this example here: MyGUIProgram Swing in Java is a Graphical User Interface (GUI) toolkit that includes the GUI components. Swing provides a rich set of widgets and packages to make sophisticated GUI components for Java applications. Swing is a part of Java Foundation Classes(JFC), which is an API for Java GUI programing that provide GUI. The Java Swing library is built on top of the Java Abstract Widget Toolkit (AWT), an older, platform dependent GUI toolkit. You can use the Java simple GUI programming components like button, textbox, etc., from the library and do not have to create the components from scratch.In this Java Swing tutorial, you will learn Java GUI basics like- Java Swing class Hierarchy Diagram Java Swing Class Hierarchy Diagram All components in Java Swing are JComponent which can be added to container classes. What is a Container Class?Container classes are classes that can have other components on it. So for creating a Java Swing GUI, we need at least one container object. There are 3 types of Java Swing containers. Panel: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window. Frame: It is a fully functioning window with its title and icons. Dialog: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame. GUI (Graphical User Interface) in Java is an easy-to-use visual experience builder for Java applications. It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application. GUI plays an important role to build easy interfaces for Java applications.How to Make a GUI in Java with ExampleNow in this Java GUI Tutorial, lets understand how to create a GUI in Java with Swings in Java examples.Step 1) Copy code into an editorIn first step Copy the following code into an editor.import javax.swing.*;class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button = new JButton("Press"); frame.getContentPane().add(button); // Adds Button to content pane of frame frame.setVisible(true); }} Step 2) Run the codeNext step, Save, Compile, and Run the codeStep 3) Copy following code into an editorNow lets Add a Button to our frame. Copy following code into an editor from given Java UI Exampleimport javax.swing.*; class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button1 = new JButton("Press"); frame.getContentPane().add(button1); frame.setVisible(true); }} Step 4) Execute the codeNext, Execute the code. You will get a big button.Step 5) Add two buttonsHow about adding two buttons? Copy the following code into an editor.import javax.swing.*;class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button1 = new JButton("Button 1"); JButton button2 = new JButton("Button 2"); frame.getContentPane().add(button1); frame.getContentPane().add(button2); frame.setVisible(true); }}Step 6) Save & Run the programNext, Save, Compile, and Run the program.Step 7) Check outputUnexpected output =? Buttons are getting overlapped.Java Layout ManagerThe Layout manager is used to layout (or arrange) the GUI Java components inside a container. There are many layout managers, but the most frequently used are-Java BorderLayoutA BorderLayout places components in up to five areas: top, bottom, left, right, and center. It is the default layout manager for every java JFrameJava BorderLayout Java FlowLayoutFlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row one after the other.Java FlowLayoutJava GridBagLayoutIt is the more sophisticated of all layouts. It aligns components by placing them within a grid of cells, allowing components to span more than one cell. Java GridBagLayout Step 8) Create chat frameHow about creating a chat frame like below?Try to code yourself before looking at the program below.//Usually you will require both swing and awt packages// even if you are working with just swings.import javax.swing.*;import java.awt.*;class gui { public static void main(String args[]) { //Creating the Frame JFrame frame = new JFrame("Chat Frame"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(400, 400); //Creating the MenuBar and adding components JMenuBar mb = new JMenuBar(); JMenu m1 = new JMenu("FILE"); JMenu m2 = new JMenu("Help"); mb.add(m1); mb.add(m2); JMenuItem m11 = new JMenuItem("Open"); JMenuItem m22 = new JMenuItem("Save as"); m1.add(m11); m1.add(m22); //Creating the panel at bottom and adding components JPanel panel = new JPanel(); // the panel is not visible in output JLabel label = new JLabel("Enter Text"); JTextField tf = new JTextField(10); // accepts upto 10 characters JButton send = new JButton("Send"); JButton reset = new JButton("Reset"); panel.add(label); // Components Added using Flow Layout panel.add(tf); panel.add(send); panel.add(reset); // Text Area at the Center JTextArea ta = new JTextArea(); //Adding Components to the frame. frame.getContentPane().add(BorderLayout.SOUTH, panel); frame.getContentPane().add(BorderLayout.NORTH, mb); frame.getContentPane().add(BorderLayout.CENTER, ta); frame.setVisible(true); }} Read More ,the free encyclopedia that anyone can edit.110,331 active editors 7,023,650 articles in EnglishGame Boy, platform of Donkey Kong LandDonkey Kong Land is a platform game developed by Rare and published by Nintendo for the Game Boy (pictured). Released on June26, 1995, it condenses the side-scrolling gameplay of Donkey Kong Country with a different level design and boss fights. The player controls Donkey Kong and Diddy Kong as they recover their stolen banana hoard from King K. Rool. Development began in 1994: Rare's Game Boy programmer, Paul Machacek, developed Land as an original game rather than a port of Country, believing that it would be a better use of resources. Land features pre-rendered graphics converted to sprites through a compression technique. Rare retooled Country's gameplay to account for the lower-quality display, and David Wise and Graeme Norgate converted the soundtrack to the Game Boy's sound chip. Critics praised it as successfully translating Country's gameplay, visuals, and music to the Game Boy. Land was rereleased for the Nintendo 3DS and the Nintendo Switch. (Fullarticle...)Recently featured: History of education in Wales (17011870)White dwarfBattle of GroixArchiveBy emailMore featured articlesAboutMendel Catholic High School... that Mendel Catholic High School (pictured) raised $15million through hosting weekly house dances from 1975?... that singer Karin Ann made headlines after performing live on Polish state television wrapped in a rainbow flag?... that voters approved a 600-percent increase in property taxes to fund programs at Valley Medical Center?... that the Diocese of Pore and Pula was nearly bankrupted over a dispute with the Italian Benedictines, a case that Draen Kutlea is credited with quietly resolving?... that the Dutch government considered converting the incomplete Java-class cruisers into English Channel ferries?... that Jack Reardon received a heart transplant while serving as the mayor of Kansas City?... that the symbol for equality in mathematics was not used for 61 years after its introduction, and was later popularized by Isaac Newton?... that stand-up comedian Dustin Nickerson has said that his children have veto power over any jokes he might tell about them?... that Saint-Sans's Phaton was described by a critic after its premiere as "the noise of a hack coming down from Montmartre"?ArchiveStart a new articleNominate an articleTrifid and Lagoon nebulaeThe Vera C. Rubin Observatory in Chile releases the first light images (example shown) from its new 8.4-metre (28ft) telescope.In basketball, the Oklahoma City Thunder defeat the Indiana Pacers to win the NBA Finals.An attack on a Greek Orthodox church in Damascus, Syria, kills at least 25 people.The United States conducts military strikes on three nuclear facilities in Iran.In rugby union, the Crusaders defeat the Chiefs to win the Super Rugby Pacific final.Ongoing: Gaza warIranIsrael warRussian invasion of UkrainetimelineSudanese civil wartimelineRecent deaths: Maria VoceWes HildrethLucien NedziAnne BurrellFrederick W. SmithRon TaylorNominate an articleJune 26Douglas Skymaster plane Amana1740 War of Jenkins' Ear: Spanish troops stormed the British-held strategically crucial position of Fort Mose in Spanish Florida.1945 At a conference in San Francisco, delegates from 50nations signed a charter establishing the United Nations.1950 A Douglas DC-4 Skymaster aircraft (pictured) crashed after departing from Perth, becoming the worst peacetime aviation accident in Australia's history.2010 A G20 summit, the largest and most expensive security operation in Canadian history, began in downtown Toronto.2015 The U.S. Supreme Court ruled in Obergefell v. Hodges that the right of same-sex couples to marry is guaranteed by the Fourteenth Amendment.Robert the Lotharingian (d.1095)George IV of the United Kingdom (d.1830)Walter C. Root (d.1925)Pavel Belyayev (b.1925)More anniversaries: June 25June 26June 27ArchiveBy emailList of days of the yearAboutAtacamite is a copper halide mineral: a copper(II) chloride hydroxide with the chemical formula $Cu_2Cl(OH)_3$. It was first described in 1802 by Dmitri Alekseyevich Golitsyn from deposits in Chile's Atacama Desert, after which it is named. Atacamite is a comparatively rare mineral, formed from primary copper minerals in the oxidation or weathering zone of arid climates. It has also been reported as a volcanic sublimate from fumarole deposits, as sulfide alteration products in black smokers. This photograph shows a specimen of atacamite, on a malachite matrix, from the Mount Gunson Mines in South Australia. The picture was focus-stacked from 42 separate images.Photograph credit: Ivar LeidusRecently featured: Turban Head eagleSpringbokGeraldine UlmarArchiveMore featured picturesCommunity portal The central hub for editors, with resources, links, tasks, and announcements.Village pump Forum for discussions about Wikipedia itself, including policies and technical issues.Site news Sources of news about Wikipedia and the broader Wikimedia movement.Teahouse Ask basic questions about using or editing Wikipedia.Help desk Ask questions about using or editing Wikipedia.Reference desk Ask research questions about encyclopedic topics.Content portals A unique way to navigate the encyclopedia.Wikipedia is written by volunteer editors and hosted by the Wikimedia Foundation, a non-profit organization that also hosts a range of other volunteer projects: CommonsFree media repository MediaWikiWiki software development Meta-WikiWikimedia project coordination WikibooksFree textbooks and manuals WikidataFree knowledge base WikinewsFree-content news WikiquoteCollection of quotations WikisourceFree-content library WikispeciesDirectory of species WikiversityFree learning tools WikivoyageFree travel guide WiktionaryDictionary and thesaurusThis Wikipedia is written in English. Many other Wikipedias are available; some of the largest are listed below. 1,000,000+ articles DeutschEspaolFranaisItalianoNederlandsPolskiPortugusSvenskaTing Vit 250,000+ articles Bahasa IndonesiaBahasa MelayuBn-lm-gCataletinaDanskEestiEsperantoEuskaraMagyarNorsk bokmlRomnSimple EnglishSloveninaSrpskiSrpskohrvatskiSuomiTrkeOzbekcha 50,000+ articles AsturianuAzrbaycancaBosanskiFryskGaeilgeGalegoHrvatskiKurdLatvieuLietuviNorsk nynorskShqipSlovenina Retrieved from " 2Calendar yearYearsMillennium2ndmillenniumCenturies17thcentury18thcentury 19thcenturyDecades1720s1730s1740s 1750s1760sYears1737173817391740 174117421743vteOctober 9: The Batavia Massacre by the Dutch East India Company of at least 5,000 Chinese Indonesians begins in what is now Jakarta.1740 by topicArts and scienceArchaeologyArchitectureArtLiteraturePoetryMusicScienceCountriesCanadaDenmarkFranceGreat BritainIrelandJapanNorwayRussiaScotlandSpainSwedenLists of leadersState leadersColonial governorsReligious leadersBirth and death categoriesBirthsDeathsEstablishments and disestablishments categoriesEstablishmentsDisestablishmentsWorks categoryWorksvte1740 in various calendarsGregorian calendar1740MDCCXLAb urbe condita2493Armenian calendar1189 Assyrian calendar6490Balinese saka calendar16611662Bengali calendar11461147Berber calendar2690British Regnal year13Geo.214Geo.2Buddhist calendar2284Burmese calendar1102Byzantine calendar72487249Chinese calendar (EarthGoat)4437 or 4230to (MetalMonkey)4438 or 4231Coptic calendar14561457Discordian calendar2906Ethiopian calendar17321733Hebrew calendar55005501Hindu calendars- Vikram Samvat17961797- Shaka Samvat16611662- Kali Yuga48404841Holocene calendar11740Igbo calendar740741Iranian calendar11181119Islamic calendar11521153Japanese calendarGenbun 5()Javanese calendar16641665Julian calendarGregorian minus 11 daysKorean calendar4073Minguo calendar172 before ROC172Nanakshahi calendar272Thai solar calendar22822283Tibetan calendar(female Earth-Goat)1866 or 1485 or 713to(male Iron-Monkey)1867 or 1486 or 714Wikimedia Commons has media related to 1740.1740 (MDCCXL) was a leap year starting on Friday of the Gregorian calendarand a leap year starting on Tuesday of the Julian calendar, the 1740th year of the Common Era (CE) and Anno Domini (AD) designations, the 740th year of the 2ndmillennium, the 40th year of the 18thcentury, and the 1st year of the 1740s decade. As of the start of 1740, the Gregorian calendar was 11 days ahead of the Julian calendar, which remained in localized use until 1923. Calendar year January 8 All 237 crewmen on the Dutch East India Company ship Rooswijk are drowned when the vessel strikes the shoals of Goodwin Sands, off of the coast of England, as it is beginning its second voyage to the Indies. The wreckage is discovered more than 250 years later, in 2004.[1]February 20 The North Carolina General Assembly incorporates the town of Newton as Wilmington, North Carolina, named for Spencer Compton, 1st Earl of Wilmington and patron of Royal Governor Gabriel Johnston.March 16 King Edward of the Miskito Indians signs a treaty making his kingdom, located on the coast of modern-day Nicaragua, a protectorate of Great Britain.[2]March 25 Construction begins on Bethesda Orphanage for boys near Savannah, Georgia, founded by George Whitefield.April 8 War of the Austrian Succession: The Royal Navy captures the Spanish ship of the line Princesa off Cape Finisterre and takes her into British service.May 31 Frederick II becomes King in Prussia upon the death of his father, Frederick William I.June 1 Plantation Act 1740 or Naturalization Act 1740 of the Parliament of Great Britain comes into effect providing for Protestant alien immigrants (including Huguenots, and also Jews) residing in the American colonies for 7 years to receive British nationality.June 16 Pour le Mrite first awarded in Prussia as a military honour.June 26 War of Jenkins' Ear: Siege of Fort Mose A Spanish column of 300 regular troops, free Black militia and Indian auxiliaries storms Britain's strategically crucial position of Fort Mose, Florida.July 7 Adam Smith sets out from Scotland to take up a scholarship at Balliol College, Oxford.[3]July 11 Pogrom: Jews are expelled from Little Russia.August 1 The song Rule, Britannia! is first performed at Cliveden, the country home of Frederick, Prince of Wales, in England.[4]August 17 Pope Benedict XIV succeeds Pope Clement XII, as the 247th pope.September 8 Hertford College, Oxford, England, is founded for the first time.[5]October 922 Batavia Massacre: Troops of the Dutch East India Company massacre 5,00010,000 Chinese Indonesians in Batavia.[6]October 20 Maria Theresa inherits the hereditary dominions of the Habsburg monarchy (Austria, Bohemia, Hungary and modern-day Belgium) under the terms of the Pragmatic Sanction of 1713 on the death of her father, Charles VI. Her succession to the Holy Roman Empire is contested widely because she is a woman, but she will reign for 40 years.November 6 Samuel Richardson's popular and influential epistolary novel, Pamela; or, Virtue Rewarded, is published anonymously in London.November 14 The University of Pennsylvania is officially established.December 16 Frederick II of Prussia invades the Habsburg possession of Silesia, starting the War of the Austrian Succession.Enfield, North Carolina, is founded.Spain begins construction on Fort Matanzas in the Matanzas Inlet, approximately 15 miles (24km) south of St. Augustine, Florida.The fairy-tale Beauty and the Beast by French novelist Gabrielle-Suzanne Barbot de Villeneuve was published.February 4 Carl Michael Bellman, Swedish poet, composer (d. 1795)February 15 Juan Andrs, Spanish Jesuit (d. 1817)February 16 Giambattista Bodoni, Italian publisher and engraver (d. 1813)February 17 John Sullivan, American General in the American Revolutionary War, delegate in the Continental Congress (d. 1795)March Johann van Beethoven, German musician, father of Ludwig van Beethoven (d. 1792)March 16 Johann Jacob Schweppe, German-born inventor, founder of the Schweppes Company (d. 1821)April 7 Haym Salomon, Polish-Jewish American financier of the American Revolution (d. 1785)April 14 Anna Strong, Patriot spy during the American Revolutionary War (d. 1812)May 7 Nikolai Arkharov, Russian police chief (d. 1814)Marquis de SadeJune 2 Marquis de Sade, French author, for whom sadism is named (d. 1814)June 24 Juan Ignacio Molina, Spanish-Chilean Jesuit priest, naturalist, historian, translator, geographer, botanist, ornithologist and linguist (d. 1829)June 27 James Woodforde, English clergyman and diarist (d. 1803)July 27 Jeanne Bar, French explorer (d. 1803)August 23 Emperor Ivan VI of Russia (d. 1764)August 26 Joseph-Michel Montgolfier, French inventor (d. 1810)September 12 Johann Heinrich Jung, German writer (d. 1817)September 23 Empress Go-Sakuramachi of Japan (d. 1813)September 25 Hercules Mulligan, tailor and spy during the American Revolutionary War (d. 1825)October 29 James Boswell, Scottish author (d. 1795)October 31 Philip James de Loutherbourg, English artist (d. 1812)December Elisabeth Olin, Swedish opera singer (d. 1828)Ali Pasha of Ioannina, Albanian ruler (d. 1822)Margaret Bingham, Countess of Lucan, born Margaret Smith, English portrait miniature painter and writer (d. 1814)[7]John Milton, American politician and officer of the Continental Army (d. 1817) (earliest estimated date of birth)Septimanie d'Egmont, French salonist (d. 1773)Pope Clement XIIFrederick William I, King in PrussiaSaint Theophilus of CorteCharles VI, Holy Roman EmperorAnna, Empress of RussiaJanuary Louise lisabeth de Joybert, politically active Canadian governors' wife (b. 1673)January 5 Antonio Lotti, Italian composer (b. 1667)January 17 Matthias Buchinger, German artist (b. 1674)January 20 Niccol Comneno Papadopoli, Italian jurist of religious law and historian (b. 1655)January 21 Nicholas Trott, colonial magistrate, South Carolina Chief Justice (b. 1663)January 27 Louis Henri, Duke of Bourbon, Prime Minister of France (b. 1692)January 29 Richard Lumley, 2nd Earl of Scarbrough (b. 1686)February 6 Pope Clement XII (b. 1652)[8]February 23 Massimiliano Soldani Benzi, Italian artist (b. 1656)February 29 Pietro Ottoboni, Italian cardinal (b. 1667)March 23 Olof Rudbeck the Younger, Swedish scientist and explorer (b. 1660)April 28 Bajirao I, Great Maratha warrior and Prime Minister of Marartha Empire (b.1700)April 23 Thomas Tickell, English writer (b. 1685)May 17 Jean Cavalier, French Protestant rebel leader (b. 1681)May 31 Frederick William I, King in Prussia (b. 1688)June 1 Samuel Werenfels, Swiss theologian (b. 1657)June 6 Alexander Spotswood, British governor of Virginia Colony (b. 1676)June 17Theophilus of Corte, Italian Roman Catholic priest, preacher and missionary, canonized (b. 1676)William Wyndham, English politician (b. 1687)June 18 Piers Butler, 3rd Viscount Galmoye, Anglo-Irish nobleman (b. 1652)July 2 Thomas Baker, English antiquarian (b. 1656)October 5 Johann Philipp Baratier, German scholar (b. 1721)October 11 Princess Magdalena Augusta of Anhalt-Zerbst, Duchess of Saxe-Gotha-Altenburg (b. 1679)October 20 Charles VI, Holy Roman Emperor (b. 1685)October 28 Anna, Empress of Russia (b. 1693)December 1 John Abernethy, Irish Protestant minister (b. 1680)December 20 Richard Boyle, 2nd Viscount Shannon, British military officer and statesman (b. 1675)December 30 John Senex, English geographer (b. ca. 1678)[9]^ Wendy van Duivenvoorde, Dutch East India Company Shipbuilding: The Archaeological Study of Batavia and Other Seventeenth-Century VOC Ships (Texas A&M University Press, 2015) p145^ "Mosquito Coast", in Historical Dictionary of the British Empire, ed. by Kenneth J. Panton (Rowman & Littlefield, 2015) p384^ "On this day in 1740..." Adam Smith Institute. July 7, 2010. Retrieved November 19, 2019.^ Williams, Hywel (2005). Cassell's Chronology of World History. London: Weidenfeld & Nicolson. p.308. ISBN0-304-35730-8.^ Hamilton, Sidney Graves (1903). Hertford College. University of Oxford college histories. London: Robinson.^ "Image: Bird's eye view of Batavia showing the massacre of the Chinese". Archived from the original on September 21, 2009. Retrieved November 12, 2006.^ This article incorporates text from a publication now in the public domain:Stephen, Leslie, ed. (1886). "Bingham, Margaret". Dictionary of National Biography. Vol.5. London: Smith, Elder & Co. ^ "Clement XII | pope". Encyclopedia Britannica. Retrieved April 22, 2021.^ "The Historical Theater in the Year 400 AD, in Which Both Romans and Barbarians Resided Side by Side in the Eastern Part of the Roman Empire". World Digital Library. 1725. Retrieved July 27, 2013.Retrieved from " 3One hundred years, from 1601 to 1700Millennia2ndmillenniumCenturies16thcentury17thcentury18thcenturyTimelines16thcentury17thcentury18thcenturyState leaders16thcentury17thcentury18thcenturyDecades1600s1610s1620s1630s1640s1650s1660s1670s1680s1690sCategories:Births Deaths Establishments DisestablishmentsvteThe 17th century lasted from January 1, 1601 (represented by the Roman numerals MDCI), to December 31, 1700 (MDCC).It falls into the early modern period of Europe and in that continent (whose impact on the world was increasing) was characterized by the Baroque cultural movement, the latter part of the Spanish Golden Age, the Dutch Golden Age,[1] the French Grand Sicle dominated by Louis XIV, the Scientific Revolution, the world's first public company and megacorporation known as the Dutch East India Company, and according to some historians, the General Crisis.From the mid-17th century, European politics were increasingly dominated by the Kingdom of France of Louis XIV, where royal power was solidified domestically in the civil war of the Fronde. The semi-feudal territorial French nobility was weakened and subjugated to the power of an absolute monarchy through the reinvention of the Palace of Versailles from a hunting lodge to a gilded prison, in which a greatly expanded royal court could be more easily kept under surveillance. With domestic peace assured, Louis XIV caused the borders of France to be expanded. It was during this century that the English monarch became increasingly involved in conflicts with the Parliament - this would culminate in the English civil war and an end to the dominance of the English monarchy.A scene on the ice, Dutch Republic, first half of the 17th centuryBy the end of the century, Europeans were masters of logarithms, electricity, the telescope and microscope, calculus, universal gravitation, Newton's Laws of Motion, air pressure, and calculating machines due to the work of the first scientists of the Scientific Revolution, including Galileo Galilei, Johannes Kepler, Ren Descartes, Pierre Fermat, Blaise Pascal, Robert Boyle, Christiaan Huygens, Antonie van Leeuwenhoek, Robert Hooke, Isaac Newton, and Gottfried Wilhelm Leibniz. It was also a period of development of culture in general (especially theater, music, visual arts and philosophy). Some of the greatest inventions took place in this century.It was during this period that the European colonization of the Americas began in earnest, including the exploitation of the silver deposits, which resulted in bouts of inflation as wealth was drawn into Europe.[2] Also during this period, there would be a more intense European presence in Southeast Asia and East Asia (such as the colonization of Taiwan). These foreign elements would contribute to a revolution in Ayutthaya. The Mataram Sultanate and the Aceh Sultanate would be the major powers of the region, especially during the first half of the century.[2]In the Islamic world, the gunpowder empires the Ottoman, Safavid, and Mughal grew in strength as well. The southern half of India would see the decline of the Deccan Sultanates and extinction of the Vijayanagara Empire. The Dutch would colonize Ceylon and endure hostilities with Kandy. The end of the 17th century saw the first major surrender of Ottoman territory in Europe when the Treaty of Karlowitz ceded most of Hungary to the Habsburgs in 1699.In Japan, Tokugawa Ieyasu established the Tokugawa shogunate at the beginning of the century, beginning the Edo period; the isolationist Sakoku policy began in the 1630s and lasted until the 19th century. In China, the collapsing Ming dynasty was challenged by a series of conquests led by the Manchu warlord Nurhaci, which were consolidated by his son Hong Taiji and finally consummated by his grandson, the Shunzhi Emperor, founder of the Qing dynasty.[3] Qing China spent decades of this century with economic problems (results of civil wars between the Qing and former Ming dynasty loyalists), only recovering well at the end of the century.The greatest military conflicts of the century were the Thirty Years' War,[4] DutchPortuguese War,[5] the Great Turkish War, the Nine Years' War, MughalSafavid Wars, and the Qing annexation of the Ming.For a chronological guide, see Timeline of the 17th century.Main articles: 1600s, 1610s, 1620s, 1630s, and 1640sPersian Ambassador during his entry into Krakw for the wedding ceremonies of King Sigismund III of Poland in 1605.1601: 4th Spanish Armada; in the Battle of Kinsale, England defeats Irish and Spanish forces, driving the Gaelic aristocracy out of Ireland and destroying the Gaelic clan system.16011603: The Russian famine of 16011603 kills perhaps one-third of Russia.[6]1602: Matteo Ricci produces the Map of the Myriad Countries of the World (, Kny Wngu Qunt), a world map that will be used throughout East Asia for centuries.1602: The Dutch East India Company (VOC) is established by merging competing Dutch trading companies.[7] Its success contributes to the Dutch Golden Age.1603: Elizabeth I of England dies and is succeeded by her cousin King James VI of Scotland, uniting the crowns of Scotland and England.1603: Tokugawa Ieyasu takes the title of shgun, establishing the Tokugawa shogunate. This begins the Edo period, which will last until 1868.1603: In Nagasaki, the Portuguese Jesuit missionary Joo Rodrigues publishes Nippo Jisho, the first dictionary of Japanese to a European (Portuguese) language.1605: The King of Gowa, a Makassarese kingdom in South Sulawesi, converts to Islam.Tsar Michael I of Russia reigned 1613164516051627: The reign of Mughal emperor Jahangir after the death of emperor Akbar.1606: The Long Turkish War between the Ottoman Empire and Austria is ended with the Peace of ZsitvatorokAustria abandons Transylvania.1606: Treaty of Vienna ends an anti-Habsburg uprising in Royal Hungary.1606: Willem Janszoon captained the first recorded European landing on the Australian continent, sailing from Bantam, Java, in the Duyfken.1607: Flight of the Earls (the fleeing of most of the native Gaelic aristocracy) occurs from County Donegal in the west of Ulster in Ireland.1607: Iskandar Muda becomes the Sultan of Aceh for 30 years. He will launch a series of naval conquests that will transform Aceh into a great power in the western Malay Archipelago.1610: The PolishLithuanian Commonwealth army defeats combined RussianSwedish forces at the Battle of Klushino and conquers Moscow.1610: King Henry IV of France is assassinated by Franois Ravaillac.1611: The Pontifical and Royal University of Santo Tomas, the oldest existing university in Asia, is established by the Dominican Order in Manila[8]1611: The first publication of the King James Bible.1612: The first Cotswold Olympic Games, an annual public celebration of games and sports begins in the Cotswolds, England.1613: The Time of Troubles in Russia ends with the establishment of the House of Romanov, which rules until 1917.16131617: PolishLithuanian Commonwealth is invaded by the Tatars dozens of times.[9]James I of England and VI of Scotland ruled in the first quarter of the 17th century1613: The Dutch East India Company is forced to evacuate Gresik due to the Mataram siege in neighboring Surabaya. The dutch negotiates with Mataram and is allowed to set up a trading post in Jepara.16141615: The Siege of Osaka (last major threat to Tokugawa shogunate) ends.1616: The last remaining Moriscos (Moors who had nominally converted to Christianity) in Spain are expelled.1616: English poet and playwright William Shakespeare dies.1618: The Defenestration of Prague.1618: The Bohemian Revolt precipitates the Thirty Years' War, which devastates Europe in the years 161848.1618: The Manchus start invading China. Their conquest eventually topples the Ming dynasty.1619: European slaving reaches America when the first Africans are brought to the present-day United States.1619: The Dutch East India Company storm Jayakarta and withstand a months-long siege by the combined English, Bantenese and Jayakartan forces. They are relieved by Jan Pieterszoon Coen and a fleet of ships from Ambon. The dutch destroys Jayakarta and builds its new headquarters, Batavia, on top of it.16201621: PolishOttoman War over Moldavia.1620: Bethlen Gabor allies with the Ottomans and an invasion of Moldavia takes place. The Polish suffer a disaster at Cecora on the River Prut.1620: The Mayflower sets sail from Plymouth, England to what became the Plymouth Colony in New England.The 1622 massacre was instrumental in causing English colonists to view all natives as enemies1621: The Battle of Chocim: Poles and Cossacks under Jan Karol Chodkiewicz defeat the Ottomans.1622: Jamestown massacre: Algonquian natives kill 347 English settlers outside Jamestown, Virginia (approximately one-third of the colony's population)[10][11] and burn the Henricus settlement.16241642: As chief minister, Cardinal Richelieu centralises power in France.1626: St. Peter's Basilica in the Vatican completed.1627: Aurochs go extinct.[12]16281629: Sultan Agung of Mataram launches a failed campaign to conquer Dutch Batavia.1629: Abbas I, the Safavids king, died.1629: Cardinal Richelieu allies with Swedish Protestant forces in the Thirty Years' War to counter Ferdinand II's expansion.1630: Birth of Shivaji at Shivneri fort, in present day Maharashtra, India, who later founded Maratha Empire in year 1674.[13]1631: Mount Vesuvius erupts.1632: Battle of Ltzen, death of king of Sweden Gustav II Adolf.Battle of Nrdlingen (1634). The Catholic Imperial army, bolstered by professional Habsburg Spanish troops won a great victory in the battle over the combined Protestant armies of Sweden and their German allies1632: Taj Mahal building work started in Agra, India.1633: Galileo Galilei arrives in Rome for his trial before the Inquisition.16331639: Japan transforms into "locked country".1634: Battle of Nrdlingen results in Catholic victory.1636: Harvard University is founded in Cambridge, Massachusetts.1637: Shimabara Rebellion of Japanese Christians, rnin and peasants against Edo.1637: The first opera house, Teatro San Cassiano, opens in Venice.1637: Qing dynasty attacked the Joseon dynasty.1639: Naval Battle of the Downs Republic of the United Provinces fleet decisively defeats a Spanish fleet in English waters.1639: Disagreements between the Farnese and Barberini Pope Urban VIII escalate into the Wars of Castro and last until 1649.16391651: Wars of the Three Kingdoms, civil wars throughout Scotland, Ireland, and England.16401668: The Portuguese Restoration War led to the end of the Iberian Union.The inauguration of the Royal Academy of Turku in 1640.1641: The Irish Rebellion, by Irish Catholics who wanted an end to discrimination, greater self-governance, and reverse ownership of the plantations of Ireland.1641: Ren Descartes publishes Meditationes de prima philosophia Meditations on First Philosophy.1642: Beginning of English Civil War, conflict will end in 1649 with the execution of King Charles I, the abolition of the monarchy and the establishment of the supremacy of Parliament over the king.1643: L'incoronazione di Poppea, Monterverdi1644: The Manchu conquer China ending the Ming dynasty. The subsequent Qing dynasty rules until 1912.16441674: The Mauritanian Thirty-Year War.16451669: Ottoman war with Venice. The Ottomans invade Crete and capture Canea.16471652: The Great Plague of Seville.1648: The Peace of Westphalia ends the Thirty Years' War and the Eighty Years' War and marks the ends of Spain and the Holy Roman Empire as major European powers.Map of Europe in 1648 at the end of the Thirty Years' War16481653: Fronde civil war in France.16481657: The Khmelnytsky Uprising a Cossack rebellion in Ukraine which turned into a Ukrainian war of liberation from Poland.16481667: The Deluge wars leave PolishLithuanian Commonwealth in ruins.16481669: The Ottomans capture Crete from the Venetians after the Siege of Candia.1649: King Charles I is executed for high treason, the first and only English king to be subjected to legal proceedings in a High Court of Justice and put to death.16491653: The Cromwellian conquest of Ireland.Main articles: 1650s, 1660s, 1670s, 1680s, 1690s, and 1700sThe Night Watch or The Militia Company of Captain Frans Banning Cocq, 1642. Oil on canvas; on display at the Rijksmuseum, Amsterdam1651: English Civil War ends with the Parliamentarian victory at the Battle of Worcester.16561661: Mehmed Kprl is Grand Vizier.16551661: The Northern Wars cement Sweden's rise as a Great Power.1657: Sambhaji, the second King of Maratha Empire and eldest son of King Shivaji was born at Purandar Fort on 14 May.[citation needed]1658: After his father Shah Jahan completes the Taj Mahal, his son Aurangzeb deposes him as ruler of the Mughal Empire.1659: King Shivaji killed Adil Shahi dynasty's general Afzal Khan at Pratapgad fort on 9 November.[14]1660: The Commonwealth of England ends and the monarchy is brought back during the English Restoration.1660: The Royal Society is founded.1660: The Bruneian Civil War begins1661: The reign of the Kangxi Emperor of China begins.1663: Ottoman war against Habsburg Hungary.1664: The Battle of St. Gotthard: count Raimondo Montecuccoli defeats the Ottomans. The Peace of Vasvar intended to keep the peace for 20 years.1665: Maratha King Shivaji signed the Treaty of Purandar with Mughal general Jai Singh I after Battle of Purandar.[citation needed]1665: Robert Hooke discovers cells using a microscope.1665: Portugal defeats the Kongo Empire at the Battle of Mbwila.Taj Mahal, completed by 1653 and commissioned by Shah Jahan, one of the Wonders of the World16651667: The Second Anglo-Dutch War fought between England and the United Provinces.1666: The Great Fire of London.1666: Shivaji visited Aurangzeb at Agra Fort and forced him into house arrest. Shivaji later escaped and returned to the Maratha kingdom.[citation needed]1667: The Raid on the Medway during the Second Anglo-Dutch War.16671668: The War of Devolution: France invades the Netherlands. The Peace of Aix-la-Chapelle (1668) brings this to a halt.16671699: The Great Turkish War halts the Ottoman Empire's expansion into Europe.16721673: Ottoman campaign to help the Ukrainian Cossacks. John Sobieski defeats the Ottomans at the second battle of Khotyn (1673).16721674: The Third Anglo-Dutch War fought between England and the United Provinces16721676: PolishOttoman War.French invasion of the Netherlands, which Louis XIV initiated in 1672, starting the Franco-Dutch War16721678: Franco-Dutch War.1673: The Bruneian Civil War ends with Muhiyiddin winning the war.1674: Shivaji founded the Maratha Empire and crowned himself as first Chatrapati of the empire.16761681: Russia and the Ottoman Empire commence the Russo-Turkish Wars.1678: The Treaty of Nijmegen ends various interconnected wars among France, the Dutch Republic, Spain, Brandenburg, Sweden, Denmark, the Prince-Bishopric of Mnster, and the Holy Roman Empire.Claiming Louisiana for France in 16821680: The Pueblo Revolt drives the Spanish out of New Mexico until 1692.1680: Prince Sambhaji crowned himself as the second Chatrapati of Maratha Empire 20 July.[citation needed]1682: French explorer Robert La Salle claims all the land east of the Mississippi River.[15]1683: China conquers the Kingdom of Tungning and annexes Taiwan.1683: The Ottoman Empire is defeated in the second Siege of Vienna.16831699: The Great Turkish War leads to the conquest of most of Ottoman Hungary by the Habsburgs.1687: Isaac Newton publishes Philosophiae Naturalis Principia Mathematica.1688: The Siege of Derry, the first major event in the Williamite War in Ireland.1688: Siamese revolution of 1688 ousted French influence and virtually severed all ties with the West until the 19th century.16881689: The Glorious Revolution starts with the Dutch Republic invading England, England becomes a constitutional monarchy.16881691: The War of the Two Kings in Ireland.16881697: The Grand Alliance sought to stop French expansion during the Nine Years' War.1689: The Battle of Killiecrankie is fought between Jacobite and Williamite forces in Highland Perthshire.1689: The Karposh rebellion is crushed in present-day North Macedonia, Skopje is retaken by the Ottoman Turks. Karposh is killed, and the rebels are defeated.The Battle of Vienna (1683) marked the historic end of the expansion of the Ottoman Empire into Europe1689: Bill of Rights gains royal consent.1689: John Locke publishes Two Treatises of Government and A Letter Concerning Toleration.1690: The Battle of the Boyne in Ireland.1692: Port Royal in Jamaica is struck by an earthquake and a tsunami. Approximately 2,000 people die and 2,300 are injured.16921694: Famine in France kills two million.[16]1693: College of William & Mary is founded in Williamsburg, Virginia, by a royal charter.1694: The Bank of England is established.1695: The Mughal Empire nearly bans the East India Company in response to pirate Henry Every's capture of the trading ship Ganj-i-Sawai.16961697: Famine in Finland wipes out almost one-third of the population.[17]16971699: Grand Embassy of Peter the Great to Western Europe.1699: Thomas Savery demonstrates his first steam engine to the Royal Society.Catholic general Albrecht von Wallenstein (15831634), supreme commander of the armies of the Imperial Army during the Thirty Years WarJan Pieterszoon Coen (15871629), the founder of Batavia, was an officer of the Dutch East India Company (VOC), holding two terms as its Governor-General of the Dutch East IndiesRen Descartes (15961650) with Queen Christina of Sweden (16261689)Cardinal Mazarin (16021661), who served as the chief minister to the kings of France Louis XIII and Louis XIVMughal Emperor Aurangzeb (16181707), who ruled over almost the entire Indian subcontinent for a period of 49 yearsChhatrapati Shivaji (16301680) founder of Maratha Empire is widely regarded as one of the greatest Hindu rulersKangxi Emperor (16611722) one of the most influential emperors of the Qing dynastyShgun Tokugawa Ieyasu was the founder of Japan's final shogunate, which lasted well into the 19th century See also: Timeline of historic inventions 17th centuryMajor changes in philosophy and science take place, often characterized as the Scientific Revolution.Banknotes reintroduced in Europe.Ice cream.Tea and coffee become popular in Europe.Central Banking in France and modern Finance by Scottish economist John Law.Minarets, Jam Mosque of Isfahan, Isfahan, Persia (Iran), are built.1604: Supernova SN 1604 is observed in the Milky Way.1605: Johannes Kepler starts investigating elliptical orbits of planets.1605: Johann Carolus of Germany publishes the 'Relation', the first newspaper.1608: Refracting telescopes first appear. Dutch spectacle-maker Hans Lippershey tries to obtain a patent on one, spreading word of the invention.1610: The Orion Nebula is identified by Nicolas-Claude Fabri de Peiresc of France.1610: Galileo Galilei and Simon Marius observe Jupiter's Galilean moons.1611: King James Bible or 'Authorized Version' first published.1612: The first flintlock musket likely created for Louis XIII of France by gunsmith Marin Bourgeois.1614: John Napier introduces the logarithm to simplify calculations.1616: Niccol Zucchi describes experiments with a bronze parabolic mirror trying to make a reflecting telescope.1620: Cornelis Drebbel, funded by James I of England, builds the first 'submarine' made of wood and greased leather.1623: The third English dictionary, English Dictionarie, is published by Henry Cockeram, listing difficult words with definitions.1628: William Harvey publishes and elucidates his earlier discovery of the circulatory system.1637: Dutch Bible published.1637: Teatro San Cassiano, the first public opera house, opened in Venice.1637: Pierre de Fermat formulates his so-called Last Theorem, unsolved until 1995.1637: Although Chinese naval mines were earlier described in the 14th century Huolongjing, the Tian Gong Kai Wu book of Ming dynasty scholar Song Yingxing describes naval mines wrapped in a lacquer bag and ignited by an ambusher pulling a rip cord on the nearby shore that triggers a steel-wheel flint mechanism.1642: Blaise Pascal invents the mechanical calculator called Pascal's calculator.1642: Mezzotint engraving introduces grey tones to printed images.1643: Evangelista Torricelli of Italy invents the mercury barometer.1645: Giacomo Torelli of Venice, Italy invents the first rotating stage.1651: Giovanni Riccioli renames the lunar maria.1656: Christiaan Huygens describes the true shape of the rings of Saturn.1657: Christiaan Huygens develops the first functional pendulum clock based on the learnings of Galileo Galilei.1659: Christiaan Huygens first to observe surface details of Mars.1662: Christopher Merret presents first paper on the production of sparkling wine.1663: James Gregory publishes designs for a reflecting telescope.1669: The first known operational reflecting telescope is built by Isaac Newton.1676: Antonie van Leeuwenhoek discovers Bacteria.1676: First measurement of the speed of light.1679: Binary system developed by Gottfried Wilhelm Leibniz.1684: Calculus independently developed by both Gottfried Wilhelm Leibniz and Sir Isaac Newton and used to formulate classical mechanics.^ "Exchange History NL 400 years: the story". Exchange History NL. Archived from the original on 6 October 2022. Retrieved 6 October 2022.^ a b "The Seventeenth-Century Decline". The Library of Iberian resources online. Retrieved 13 August 2008.^ "5 of the 10 Deadliest Wars Began in China". Business Insider. 6 October 2014.^ "The Thirty-Years-War". Western New England College. Archived from the original on 1999-10-09. Retrieved 2008-05-24.^ Ames, Glenn J. (2008). The Globe Encompassed: The Age of European Discovery, 15001700. pp.102103.^ Turchin, Peter (2009). Secular Cycles. Princeton University Press. pp.256257. ISBN9780691136967.^ Ricklefs (1991), page 28^ History of UST UST.edu.ph. Retrieved December 21, 2008.^ "The Tatar Khanate of Crimea". Archived from the original on 2016-03-23. Retrieved 2008-06-05.^ Mark, Joshua J. "Indian Massacre of 1622". World History Encyclopedia. Retrieved 2022-09-13.^ Campbell, B.C (2008). Disasters, accidents, and crises in American history: A reference guide to the nation's most catastrophic events. Infobase Publishing. pp.1112.^ Rokosz, M. (1995). "History of the Aurochs (Bos taurus primigenius) in Poland" (PDF). Animal Genetics Resources Information. 16: 512. doi:10.1017/S1014233900004582. Archived from the original (PDF) on 14 January 2013.^ Itihas [History, class fourth] (in Marathi). Maharashtra, India: Pathyapustak nirmiti madal, Pune. 2019.^ "London museum correct record on general disembowelment by Chhatrapati Shivaji". The Tribune.^ "Ren-Robert Cavelier, sieur de La Salle". Britannica. 30 March 2021. Retrieved 21 March 2021.^ Alan Macfarlane (1997). The savage wars of peace: England, Japan and the Malthusian trap. Wiley . p. 64. ISBN0-631-18117-2^ Karen J. Cullen (2010). "Famine in Scotland: The 'Ill Years' of the 1690s". Edinburgh University Press. p. 20. ISBN0-7486-3887-3Detail of a 17th-century Tekke Turkmen carpetChang, Chun-shu, and Shelley Hsueh-lun Chang. Crisis and Transformation in Seventeenth-Century China (1998).Langer, William. An Encyclopedia of World History (5th ed. 1973); highly detailed outline of events online freeReid, A. J. S. Trade and State Power in 16th & 17th Century Southeast Asia (1977).Spence, J. D. The

Death of Woman Wang: Rural Life in China in the 17th Century (1978).Clark, George. The Seventeenth Century (2nd ed. 1945).Hampshire, Stuart. The Age of Reason the 17th Century Philosophers, Selected, with Introduction and Interpretive Commentary (1961).Hugon, Ccile (1997) [1911]. "Social Conditions in 17th-Century France (1649-1652)". In Halsall, Paul (ed.). Social France in the XVII Century. London: Methuen. pp.171172, 189. ISBN9780548161944. Archived from the original on 23 August 2016. Retrieved 7 August 2021.Lewitter, Lucian Ryszard. "Poland, the Ukraine and Russia in the 17th Century." The Slavonic and East European Review (1948): 157171. in JSTOROgg, David. Europe in the Seventeenth Century (6th ed. 1965).Rowbotham, Sheila. Hidden from history: Rediscovering women in history from the 17th century to the present (1976).Trevor-Roper, Hugh R. "The general crisis of the 17th century." Past & Present 16 (1959): 3164.Wikimedia Commons has media related to 17th century.Vistorica: Timelines of 17th century events, science, culture and personsRetrieved from " 4The following pages link to 17th century External tools(link counttransclusion countsorted list) See help page for transcluding these entriesShowing 50 items.View (previous 50 | next 50) (20 | 50 | 100 | 250 | 500)Astrology (links | edit)Alessandro Scarlatti (links | edit)Aurochs (links | edit)Bagpipes (links | edit)Drink (links | edit)List of decades, centuries, and millennia (links | edit)Kaohsiung (links | edit)Suffolk (links | edit)Waltz (links | edit)20th century (links | edit)15th century (links | edit)16th century (links | edit)18th century (links | edit)1624 (links | edit)1626 (links | edit)1642 (links | edit)1661 (links | edit)1756 (links | edit)1791 (links | edit)1608 (links | edit)1743 (links | edit)14th century (links | edit)1788 (links | edit)1st century (links | edit)13th century (links | edit)1787 (links | edit)4th century (links | edit)12th century (links | edit)11th century (links | edit)1564 (links | edit)1648 (links | edit)1572 (links | edit)1623 (links | edit)1662 (links | edit)1640s (links | edit)1770s (links | edit)1780s (links | edit)1789 (links | edit)1782 (links | edit)1707 (links | edit)1700s (decade) (links | edit)1597 (links | edit)1690 (links | edit)1764 (links | edit)1760s (links | edit)1742 (links | edit)1735 (links | edit)1730s (links | edit)1740s (links | edit)1790s (links | edit)View (previous 50 | next 50) (20 | 50 | 100 | 250 | 500)Retrieved from " WhatLinksHere/17th_century" Swing in Java is a Graphical User Interface (GUI) toolkit that includes the GUI components. Swing provides a rich set of widgets and packages to make sophisticated GUI components for Java applications. Swing is a part of Java Foundation Classes(JFC), which is an API for Java GUI programing that provide GUI. The Java Swing library is built on top of the Java Abstract Widget Toolkit (AWT), an older, platform dependent GUI toolkit. You can use the Java simple GUI programming components like button, textbox, etc., from the library and do not have to create the components from scratch.In this Java Swing tutorial, you will learn Java GUI basics like- Java Swing class Hierarchy Diagram Java Swing Class Hierarchy Diagram All components in Java Swing are JComponent which can be added to container classes. What is a Container Class?Container classes are classes that can have other components on it. So for creating a Java Swing GUI, we need at least one container object. There are 3 types of Java Swing containers. Panel: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window. Frame: It is a fully functioning window with its title and icons. Dialog: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame. GUI (Graphical User Interface) in Java is an easy-to-use visual experience builder for Java applications. It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application. GUI plays an important role to build easy interfaces for Java applications.How to Make a GUI in Java with ExampleNow in this Java GUI Tutorial, lets understand how to create a GUI in Java with Swings in Java examples.Step 1) Copy code into an editorIn first step Copy the following code into an editor.import javax.swing.*;class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button = new JButton("Press"); frame.getContentPane().add(button); // Adds Button to content pane of frame frame.setVisible(true); }} Step 2) Run the codeNext step, Save, Compile, and Run the codeStep 3) Copy following code into an editorNow lets Add a Button to our frame. Copy following code into an editor from given Java UI Exampleimport javax.swing.*; class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button1 = new JButton("Press"); frame.getContentPane().add(button1); frame.setVisible(true); }} Step 4) Execute the codeNext, Execute the code. You will get a big button.Step 5) Add two buttonsHow about adding two buttons? Copy the following code into an editor.import javax.swing.*;class gui{ public static void main(String args[]){ JFrame frame = new JFrame("My First GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,300); JButton button1 = new JButton("Button 1"); JButton button2 = new JButton("Button 2"); frame.getContentPane().add(button1); frame.getContentPane().add(button2); frame.setVisible(true); }}Step 6) Save & Run the programNext, Save, Compile, and Run the program.Step 7) Check outputUnexpected output =? Buttons are getting overlapped.Java Layout ManagerThe Layout manager is used to layout (or arrange) the GUI Java components inside a container. There are many layout managers, but the most frequently used are-Java BorderLayoutA BorderLayout places components in up to five areas: top, bottom, left, right, and center. It is the default layout manager for every java JFrameJava BorderLayout Java FlowLayoutFlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row one after the other.Java FlowLayoutJava GridBagLayoutIt is the more sophisticated of all layouts. It aligns components by placing them within a grid of cells, allowing components to span more than one cell. Java GridBagLayout Step 8) Create chat frameHow about creating a chat frame like below?Try to code yourself before looking at the program below.//Usually you will require both swing and awt packages// even if you are working with just swings.import javax.swing.*;import java.awt.*;class gui { public static void main(String args[]) { //Creating the Frame JFrame frame = new JFrame("Chat Frame"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(400, 400); //Creating the MenuBar and adding components JMenuBar mb = new JMenuBar(); JMenu m1 = new JMenu("FILE"); JMenu m2 = new JMenu("Help"); mb.add(m1); mb.add(m2); JMenuItem m11 = new JMenuItem("Open"); JMenuItem m22 = new JMenuItem("Save as"); m1.add(m11); m1.add(m22); //Creating the panel at bottom and adding components JPanel panel = new JPanel(); // the panel is not visible in output JLabel label = new JLabel("Enter Text"); JTextField tf = new JTextField(10); // accepts upto 10 characters JButton send = new JButton("Send"); JButton reset = new JButton("Reset"); panel.add(label); // Components Added using Flow Layout panel.add(tf); panel.add(send); panel.add(reset); // Text Area at the Center JTextArea ta = new JTextArea(); //Adding Components to the frame. frame.getContentPane().add(BorderLayout.SOUTH, panel); frame.getContentPane().add(BorderLayout.NORTH, mb); frame.getContentPane().add(BorderLayout.CENTER, ta); frame.setVisible(true); }} Read More Share copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. GUI Programming in Java is an exciting venture that transforms bland lines of code into vibrant, interactive interfaces. Ever wondered how your favourite desktop applications are visually brought to life? Javas Graphical User Interface (GUI) is the magic behind it all. Whether youre a newbie or looking to up your coding game, youll find this blogs insights useful. Stick around, as well unravel the complexities of GUI, making it simple and fun to integrate into real-world projects! Dive in, and lets decode Java together. Graphical User Interface (GUI) Programming in Java allows developers to create applications that users can interact with visuallythrough windows, buttons, menus, text fields, and other graphical elementsrather than using the command line. This approach enhances user experience by making applications intuitive and visually appealing. A GUI (Graphical User Interface) is a type of interface that lets users interact with software applications through graphical components like windows, icons, buttons, and menus. Java developers create GUIs using pre-defined classes and libraries that simplify the process of building interactive front ends. Java offers a wide range of GUI components that you can add to your application to collect input and display information: JButton for clickable buttons JLabel to display text or images JTextField to accept single-line text input JTextArea for multi-line text input JCheckBox to allow users to make multiple selections JRadioButton for selecting a single option among many JComboBox drop-down menus JPanel containers to group components JFrame the main application window These components are typically arranged using layout managers to ensure the GUI looks organized on different screen sizes. GUI programming in Java is event-driven, meaning the program waits for user actions (events) like clicks, typing, or selections, and then responds accordingly. Java uses event listeners to handle these interactions. For example, clicking a button might trigger an ActionListener that runs a specific block of code. Java provides two primary libraries for building GUIs: AWT and Swing. AWT was Javas original GUI toolkit. It provides a set of components that map directly to native (platform-specific) GUI components. This means an AWT button on Windows looks different from one on macOS. While AWT is lightweight and simple to use, it lacks flexibility and consistency across platforms. Key characteristics of AWT: Platform-dependent look and feel Limited GUI components Faster but less customizable Swing is a more advanced and flexible GUI toolkit introduced as part of Java Foundation Classes (JFC). Unlike AWT, Swing components are written entirely in Java and are platform-independent, offering a consistent look and feel across operating systems. Swing provides more sophisticated components like: JTable for data grids JTree for hierarchical data Pluggable Look and Feel (can change the appearance of components) Swing is preferred over AWT for most Java GUI applications because: It offers more powerful and customizable components Ensures consistent appearance across platforms Supports MVC (Model-View-Controller) architecture Allows developers to design richer user interfaces While JavaFX is now another option for modern GUI development, Swing remains widely used due to its maturity, community support, and integration with legacy systems. Creating a GUI in Java involves setting up a window and placing components like buttons and text fields inside it. Lets walk through a simple example using Swing. import javax.swing.JFrame;public class SimpleGUI { public static void main(String[] args) { // Create a window (JFrame) JFrame frame = new JFrame("My First GUI App"); frame.setSize(400, 300); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setLayout(null); // For absolute positioning (not recommended for complex UIs) frame.setVisible(true); }} This code creates a basic window titled My First GUI App. Well now add a label, a text field, and a button to the window: import javax.swing.*;public class SimpleGUI { public static void main(String[] args) { JFrame frame = new JFrame("GUI Example"); frame.setSize(400, 300); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setLayout(null); JLabel label = new JLabel("Enter your name:"); label.setBounds(50, 50, 150, 30); frame.add(label); JTextField textField = new JTextField(); textField.setBounds(180, 50, 150, 30); frame.add(textField); JButton button = new JButton("Submit"); button.setBounds(150, 100, 100, 30); frame.add(button); frame.setVisible(true); }} To respond to button clicks, you use an ActionListener: button.addActionListener(e -> { String name = textField.getText(); JOptionPane.showMessageDialog(frame, "Hello, " + name + "!");}); This displays a popup with the users name when the button is clicked. Java Swing provides many reusable components to build rich user interfaces. ComponentDescriptionJButtonA clickable buttonJLabelDisplays text or imagesJTextFieldSingle-line text inputJTextAreaMulti-line text inputJCheckBoxCheckbox for multiple selectionsJRadioButtonRadio button for single selection among a groupJComboBoxDrop-down menuJListList boxJTableTable for tabular dataJPanelContainer to group componentsJFrameThe main application window Layout managers control the arrangement of components in a container. Here are the most common ones: FlowLayout (default for JPanel)Arranges components in a row, left to right.Wraps to the next line as needed.frame.setLayout(new FlowLayout()); BorderLayoutDivides the container into five regions: North, South, East, West, Center.frame.setLayout(new BorderLayout()); frame.add(button, BorderLayout.SOUTH); GridLayoutOrganizes components in a grid (rows and columns).frame.setLayout(new GridLayout(2, 2)); Each layout manager serves different use cases. Choosing the right one helps in creating flexible and scalable interfaces. javaimport javax.swing.*;import java.awt.*;import java.awt.event.ActionEvent;import java.awt.event.ActionListener;public class SimpleGUI { public static void main(String[] args) { JFrame frame = new JFrame("Simple GUI"); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300, 200); JPanel panel = new JPanel(); frame.add(panel); placeComponents(panel); frame.setVisible(true); } private static void placeComponents(JPanel panel) { panel.setLayout(null); JLabel userLabel = new JLabel("User:"); userLabel.setBounds(10, 20, 80, 25); panel.add(userLabel); JTextField userText = new JTextField(20); userText.setBounds(100, 20, 165, 25); panel.add(userText); JLabel passwordLabel = new JLabel("Password:"); passwordLabel.setBounds(10, 50, 80, 25); panel.add(passwordLabel); JPasswordField passwordText = new JPasswordField(20); passwordText.setBounds(100, 50, 165, 25); panel.add(passwordText); JButton loginButton = new JButton("Login"); loginButton.setBounds(10, 80, 80, 25); panel.add(loginButton); JButton registerButton = new JButton("Register"); registerButton.setBounds(100, 80, 100, 25); panel.add(registerButton); loginButton.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent e) { System.out.println("Login pressed"); } }); registerButton.addActionListener(new ActionListener() { @Override public void actionPerformed(ActionEvent e) { System.out.println("Register pressed"); } }); }} Explanation of the CodeHeres a breakdown of the Java code snippet provided for a simple graphical user interface (GUI):This code begins by importing necessary libraries from Swing and AWT, which help build the GUI and handle events.A `JFrame` is created, which is essentially the window where your GUI components will live. The `frame` is given a title Simple GUI and a default close operation to exit the program when the window closes.We set the size of the `frame` to 300200 pixels. Then, a `JPanel` is added to the `frame` to act as our main container for all components.The `placeComponents` method is called to add various elements like text fields and buttons onto the `panel`. This method positions labels, text fields, and buttons with boundaries using `setBounds` for precise placement.Action listeners are added to buttons to print messages to the console when clicked, demonstrating event handling in Java. GUI Programming in Java As youre venturing into the realm of GUI Programming in Java, its fascinating to see how companies and brands leverage these skills to solve real-world problems. Here are some authentic scenarios:Healthcare Systems Management: Many hospitals have utilized Java Swing, a GUI toolkit in Java, to develop patient management systems. These applications allow staff to easily input, access, and modify patient records with a user-friendly interface, ensuring efficient and intuitive data management.Banking Applications: Financial institutions have created desktop applications using Java FX, another Java GUI library, for managing customer accounts and transactions. These applications provide secure and interactive banking experiences to clients while enhancing teller operations.Educational Software: An example is an online learning platform using GUI elements to create interactive educational tools. These tools help teachers design quizzes and exercises, making learning more engaging for students.Retail Point of Sale (POS) Systems: Many retail companies have implemented POS systems developed with Java, allowing seamless integration of inventory management, sales tracking, and payment processing.Weather Forecast Platforms: Software developed to process and display real-time weather data with complex visual graphs and updates. The GUI components make it easier for users to interact with large datasetsEach of these examples showcases how JAVA GUI programming cleverly meets both industrys and consumers needs! To create efficient, maintainable, and responsive Java GUI applications, its important to follow certain best practices. These principles ensure your application is not only user-friendly but also scalable and bug-resistant. One of the golden rules in application development is the separation of concerns. You should divide your code into at least two parts: GUI Layer handles the interface and user interactions (e.g., button clicks, displaying data). Business Logic Layer performs the actual operations (e.g., calculations, file operations, database handling). This approach makes your code: Easier to read and maintain More testable (business logic can be tested independently) Reusable across different types of interfaces (e.g., GUI and CLI) Tip: Consider using the MVC (Model-View-Controller) pattern for larger projects. Avoid using absolute positioning (setLayout(null)) in production-level applications. Instead, use built-in layout managers to create responsive and clean designs: FlowLayout for left-to-right alignment BorderLayout for placing components in five predefined regions GridLayout for uniform grid-based layouts BoxLayout for vertical or horizontal stacking Combining multiple layout managers using JPanel sub-containers allows for more complex and organized layouts. Tip: Always test your layout with different screen sizes and window resizing. In Java Swing, all GUI updates must occur on the Event Dispatch Thread (EDT). Long-running tasks (like file reading, network calls, or database access) should not be performed directly in the ActionListener or any event-handling code. Doing so will freeze the interface and degrade user experience. To prevent this: Use a separate thread for background tasks Or, better yet, use the SwingWorker class Example using SwingWorker: SwingWorker worker = new SwingWorker() { @Override protected Void doInBackground() throws Exception { // Long-running task Thread.sleep(3000); // simulate work return null; } @Override protected void done() { // Update UI after task completion JOptionPane.showMessageDialog(null, "Task Completed!"); }};worker.execute(); Tip: Never call Thread.sleep() or long loops directly inside GUI event methods. Following these best practices will help you build Java GUI applications that are user-friendly, robust, and easy to maintain. Our AI-powered java online compiler lets users instantly write, run, and test their code seamlessly. No more waiting aroundexperience fast results with AI assistance. Dive right into coding with confidence, knowing youve got a speedy tool that adapts to your needs effortlessly. Transform your coding experience today! Lets get a bit hands-on with a quiz, shall we? Quizzes are fantastic for reinforcing your learning and ensuring the concepts stick. Heres an example set of questions you might encounter while learning GUI programming in Java:What is the main class used to create a window in Swing? Which library provides advanced GUI components with rich visualizations in Java? What method is used to add components to a container in Swing? addComponent() attach() add() In JavaFX, which class is used as the primary stage for the applications GUI? What is an event in the context of GUI programming? A background process A user action A component GUI Programming in Java equips you with vital skills to create interactive applications, sharpening both your technical savvy and problem-solving abilities. Why not give it a go and experience the satisfaction of building something yourself? Check out Newtum for more programming languages like Java, Python, C, C++, and more! This article was compiled and edited by @rasikadeshpande, who has over 4 years of experience in writing. Shes passionate about helping beginners understand technical topics in a more interactive way. About The Author

What is gui programming in java. Java form gui. Gui tutorial java. What is gui in java with examples.