Object-Oriented Programming with Java: An Exploration of Windows, Graphics, and User Interface Fundamentals using the Abstract Window Toolkit

1. Introduction to AWT and Object-Oriented Programming

- ### 1.1. The Abstract Window Toolkit (AWT): A Foundation for Java GUIs

The Abstract Window Toolkit (AWT) stands as Java's original framework for constructing graphical user interfaces (GUIs).[1] Predating the more versatile Swing toolkit, AWT provides a set of classes and interfaces for developing window-based applications. It is a crucial part of the Java Foundation Classes (JFC), which collectively offer a standard API for creating graphical applications in Java.[1] Since its introduction with the first release of Java in 1995, AWT has played a pivotal role in enabling developers to build interactive desktop applications.[1]

A defining characteristic of AWT is that its components are "heavyweight".[2] This term signifies that AWT components rely on the native GUI components of the underlying operating system. For instance, when an AWT application creates a button, it utilizes the operating system's native button widget.[1] Consequently, the appearance of AWT applications can vary significantly across different platforms, as the components are rendered according to the look and feel of the specific operating system (such as Windows, macOS, or Linux).[2]

The core classes that constitute the AWT are contained within the java.awt package.[5] This package is one of Java's largest and provides a wide array of classes for creating and managing windows, handling events, and performing graphical operations.[5] Starting with Java Development Kit (JDK) version 9, the java.awt package became part of the java.desktop module, reflecting its role in desktop application development.[6]

The platform-dependent nature of AWT illustrates a fundamental decision in the early stages of cross-platform GUI development. The choice was between leveraging the native look and feel of each operating system, providing a sense of integration with the user's environment, versus ensuring a consistent visual appearance across all platforms, which would simplify development and user experience across different systems. AWT opted for the former, aiming for seamless integration by using native widgets. However, this approach led to inconsistencies in how applications looked and behaved on different operating systems. To address this, Sun Microsystems later introduced Swing, a GUI toolkit that offers "lightweight" components rendered entirely in Java, providing a consistent look and feel independent of the underlying operating system.[1]

Despite being superseded by Swing for many modern GUI development tasks, AWT's foundational role in Java's graphical capabilities remains significant. Many core classes from the java.awt package, such as Color for managing colors, Font for handling text styles, and the basic event handling mechanism, are still utilized by Swing and the Java 2D API.[1] Swing, in essence, builds upon the event handling and basic drawing capabilities provided by AWT, demonstrating a layered architecture where newer frameworks extend and enhance the functionalities of older ones. The continued use of these core AWT classes underscores their robustness and fundamental importance in the broader Java GUI ecosystem.

- **1.2. Connecting AWT to Object-Oriented Principles**

The architecture of the Abstract Window Toolkit is deeply rooted in the principles of Object-Oriented Programming (OOP), which are fundamental to the Java language.[5] AWT's design leverages key OOP concepts such as inheritance, abstraction, encapsulation, and polymorphism to provide a structured and extensible framework for GUI development.

One of the most evident connections to OOP is through AWT's well-defined **inheritance** hierarchy.[5] The relationships between classes like java.awt.Component, java.awt.Container, java.awt.Window, and java.awt.Frame clearly illustrate inheritance. Subclasses in this hierarchy inherit attributes and methods from their superclasses, allowing for code reuse and the creation of more specialized objects. For example, the Frame class inherits basic windowing functionalities from the Window class and then extends it by adding features specific to application windows, such as a title bar, border, and the ability to host a menu bar. This hierarchical organization avoids code duplication and promotes a logical structure for GUI elements. The Component class serves as the abstract base for all visual entities in AWT [7], defining common properties like size, position, and visibility, as well as fundamental methods like setSize(), setLocation(), and setVisible().[3] The Container class extends Component and introduces the capability to hold other Component objects, along with methods for adding and removing them.[3] Window further extends Container to represent a top-level window on the screen.[5] Finally, Frame builds upon Window to create a standard application window with decorations.[5] This layered structure is a prime example of how inheritance in OOP enables the creation of specialized classes from more general ones, promoting code reuse and a clear organization of responsibilities.

Abstraction is another key OOP principle reflected in AWT's design.[5] Abstract classes like java.awt.Component and java.awt.Graphics provide abstract interfaces, defining methods that must be implemented by concrete subclasses to provide specific platform-dependent behavior. This abstraction hides the underlying complexity of interacting with the native GUI system of different operating systems. For instance, the Graphics class is an abstract class that offers methods for drawing various shapes, text, and images.[6] The actual implementation of these drawing operations (how pixels are manipulated on the screen) is handled by platform-specific subclasses of Graphics provided by the operating system. When a developer calls a method like drawLine() on a Graphics object, they do not need to be concerned with the low-level details of how the line is rendered by the particular operating system the application is running on. This level of abstraction simplifies the development process significantly and ensures that the same Java code can achieve graphical output on different platforms without requiring modifications for each specific operating system.

Encapsulation is also a core aspect of AWT, where classes like java.awt.Font and java.awt.Color encapsulate data and the methods to manipulate that data.[5] The Font class, for example, encapsulates the font's name, style (such as bold or italic), and size.[5] It provides methods to retrieve information about the font and to create new font instances with different properties.[15] Similarly, the Color class encapsulates the red, green, and blue components that define a color.[5] These classes control access to their internal data through well-defined methods, promoting better code organization and reducing the risk of unintended modifications or errors. By encapsulating related data and functionality within cohesive units, AWT makes the codebase more modular, easier to understand, and simpler to maintain over time.

Finally, AWT's event handling mechanism demonstrates **polymorphism** through the use of interfaces like java.awt.event.ActionListener and java.awt.event.WindowListener.[3] Different AWT components can generate the same type of event. For example, both a Button and a MenuItem can generate an ActionEvent when they are activated by the user.[20] Multiple classes can implement the ActionListener interface, each providing its own specific implementation of the actionPerformed() method.[3] When the button or menu item is activated, the appropriate actionPerformed() method of the registered listener is invoked. The source of the event (the button or menu item) does not need to know the concrete class of the listener; it only needs to know that the listener implements the ActionListener interface. This allows for a highly decoupled design, where different parts of the application can respond to the same event in ways that are specific to their functionality, without the event source needing to have knowledge of these specific implementations. This flexibility and extensibility are hallmarks of polymorphism in object-oriented programming.

2. Understanding the AWT Window Hierarchy

- **2.1. The Foundation: Component and Container**

The Abstract Window Toolkit's structure for building graphical user interfaces is based on a hierarchy of classes, with java.awt.Component and java.awt.Container serving as fundamental building blocks.[5]

The Component class is an abstract class that forms the base for all AWT GUI elements, encompassing both simple controls like buttons and labels, as well as more complex containers such as windows and panels.[5] It acts as a blueprint defining the essential attributes and behaviors that all visual elements share. These attributes include properties such as the component's size (width and height), its position on the screen (x and y coordinates), its visibility (whether it is currently displayed), the foreground and background colors used for rendering, and the font used for displaying text.[5] Beyond these attributes, the Component class also declares a set of methods that provide fundamental functionalities for managing the component's lifecycle and interactions. These methods include those for handling various types of events, such as mouse clicks and keyboard input, for controlling the component's positioning and sizing on the screen, and for managing the repainting process when the component needs to be redrawn.[5] In essence, the Component class establishes the common interface and behavior for all visual objects within an AWT application, making it the cornerstone of the toolkit's structure.

Extending the capabilities of Component is the Container class, which is a direct subclass of Component.[3] The primary role of a Container is to hold other Component objects, thereby enabling the creation of complex and hierarchical GUI layouts.[3] The Container class provides specific methods for managing the components it holds. These include the add() method, which is used to insert a Component into the Container, and the remove() method, which allows for the removal of a Component.[3] Furthermore, Container introduces the concept of a layout manager, which is responsible for determining the size and position of each component within the container.[3] The layout manager can be set using the setLayout() method of the Container. AWT provides several predefined layout managers, such as FlowLayout, BorderLayout, and GridLayout, each with its own rules for arranging components.[3] Notably, several key AWT classes that represent windowing elements, such as Window, Frame, Dialog, and the lightweight container Panel, are all subclasses of Container [3], highlighting the fundamental role of Container in structuring AWT applications.

- **2.2. Top-Level Windows: Window, Frame, and Dialog**

In the AWT hierarchy, several classes represent top-level windows, which are windows that appear directly on the desktop and are not contained within another window. The primary abstract class for this concept is java.awt.Window, with its concrete subclasses java.awt.Frame and java.awt.Dialog providing more specific functionalities.[5]

The Window class is an abstract class that serves as the base for creating top-level windows that have no borders and no menu bar.[5] A crucial characteristic of a Window object is that it must have an owner, which can be an instance of Frame, Dialog, or even another Window. This owner is specified during the construction of the Window.[8] The default layout manager for a Window is BorderLayout, which divides the window's content area into five regions: North, South, East, West, and Center.[8] While the Window class provides the fundamental ability to display content in an independent window on the screen, it is generally not instantiated directly. Instead, developers typically use its more specialized subclasses, Frame and Dialog, which add standard window decorations and behaviors.[5] The Window class offers essential methods for managing the window's state, such as its visibility, focus, and the events it can generate.[8] The requirement for an owner establishes a clear relationship between the Window and other top-level windows in the application, influencing aspects like modality and window disposal.

Class	**Top-Level**	**Borders**	**Menu Bar**	**Owner Required**	**Default Layout**	**Primary Use**
Window	Yes	No	No	Yes	BorderLayout	Basic top-level window; base for Frame, Dialog
Frame	Yes	Yes	Yes	No	BorderLayout	Standard application windows
Dialog	Yes	Yes	No	Yes	BorderLayout	Modal or modeless dialog boxes for user input or

						messages
Panel	No	No	No	No	FlowLayout	Grouping components within a window/frame to manage layout

The Frame class is a concrete subclass of Window and represents what is commonly thought of as a standard application window.[3] Unlike Window, a Frame object has a title bar, where the application's name or the current document title is typically displayed, and a border, which allows the user to resize the window.[6] Importantly, a Frame can also have a menu bar attached to it, providing a standard interface for accessing application commands.[6] The default layout manager for a Frame is also BorderLayout.[25] Frame objects are frequently used as the main window for GUI applications built with AWT, serving as the primary container for all other user interface components. The Frame class provides a robust and flexible way to create application windows with the essential features that users expect.

The Dialog class is another concrete subclass of Window and represents a top-level window that is typically used to take some form of input from the user or to display important information.[5] Similar to Frame, a Dialog has a title and a border, but unlike Frame, it does not typically have a menu bar.[33] A key characteristic of a Dialog is that it must have an owner, which can be either a Frame or another Dialog, specified when the Dialog is constructed.[33] Dialogs can be either modeless or modal. A modal dialog is one that blocks input to all other top-level windows in the application until it is closed, ensuring that the user interacts with the dialog before continuing with the rest of the application.[33] The default layout manager for a Dialog is also BorderLayout.[33] Dialogs are commonly used for tasks such as displaying error messages, prompting the user for input (e.g., in a login screen), or showing settings and preferences.

- **2.3. Lightweight Containers: Panel**

In contrast to the top-level window classes, the java.awt.Panel class is a concrete subclass of java.awt.Container that does not create an independent window.[2] Instead, a Panel provides a space within an existing window (like a Frame or another Panel) where other components, including other panels, can be attached.[5] The Panel class itself does not add any new methods beyond those inherited from Container; it primarily serves as a concrete implementation of the Container interface.[5] A Panel can be thought of as a versatile, nestable screen component that allows for the logical grouping and organization of GUI elements.[5] Notably, the java.applet.Applet class, used for creating applets that run within web browsers, is a subclass of Panel.[5] The default layout manager for a Panel is FlowLayout, which arranges components in a left-to-right, top-to-bottom flow.[5] Unlike Window, Frame, and Dialog, a Panel does not have its own title bar, border, or menu bar; it relies on its parent container for these decorations.[3] The primary use of Panel is to act as an intermediate container to help structure more complex GUI layouts by dividing a window into logical sections, each of which can have its own layout manager.

3. Working with Frame Windows

- ### 3.1. Purpose and Characteristics of the Frame Class

The Frame class in Java's Abstract Window Toolkit serves as the foundation for creating the main application windows in a graphical user interface.[6] It encapsulates all the essential features of a standard, top-level window, including a title bar that typically displays the application's name or the title of the current view, a border that allows the user to resize the window, and system-specific controls for operations like closing, minimizing, and maximizing the window.[6]

When working with a Frame, it's important to understand that the total size of the frame, as reported by methods like getSize(), includes the area occupied by its border.[26] The dimensions of this border area are platform-dependent and can be obtained using the getInsets() method. However, a valid insets value, which represents the space taken up by the border on each side of the frame, is only available after the frame has been made displayable, typically by calling the pack() or show() methods.[26] This distinction is crucial when you are laying out components within the frame, as you need to account for the insets to ensure that your components are placed within the usable content area and are not obscured by the frame's border.

The default layout manager for a Frame is BorderLayout.[25] This layout manager divides the frame's content area into five distinct regions: North, South, East, West, and Center. You can add components to the frame, specifying which region they should occupy. This layout is particularly useful for structuring applications with a header at the top (North), a footer at the bottom (South), sidebars on the left (West) and right (East), and the main content in the middle (Center).

Frame objects are capable of generating various types of WindowEvents in response to actions performed on the window or by the user.[25] These events include

WINDOW_OPENED (when the frame is first displayed), WINDOW_CLOSING (when the user attempts to close the window), WINDOW_CLOSED (after the window has been closed), WINDOW_ICONIFIED (when the window is minimized), WINDOW_DEICONIFIED (when the window is restored), WINDOW_ACTIVATED (when the window becomes the active window), and WINDOW_DEACTIVATED (when the window is no longer the active window). By adding a WindowListener to a Frame, an application can intercept these events and perform appropriate actions, such as saving data before closing or pausing operations when the window is deactivated.

For more specialized use cases, Frame windows can have their native decorations, specifically the title bar and border, turned off by using the setUndecorated() method.[28] However, this method must be called before the frame is made displayable (i.e., before pack() or setVisible(true) is called). Removing the decorations can be useful for creating custom window appearances, such as splash screens or full-screen applications, where the standard window controls are not desired.

- **3.2. Constructors for Creating Frame Objects**

The Frame class offers several constructors that allow for different ways to instantiate a Frame object, providing flexibility based on the specific requirements of the application.[25]

The simplest constructor is Frame(), which creates a new Frame object that is initially invisible and has an empty title.[25] After creating a frame with this constructor, you would typically call the setTitle() method to set a meaningful title for the window.

Another common constructor is Frame(String title), which creates a new Frame object that is also initially invisible but is initialized with the title specified by the title argument.[5] This constructor is often preferred for creating standard application windows as it allows you to set the title at the time of the frame's creation.

For applications that need to run in multi-screen environments, the Frame(GraphicsConfiguration gc) constructor is available.[27] This constructor creates a new, initially invisible Frame that is associated with the specified GraphicsConfiguration. The GraphicsConfiguration object describes the characteristics of a specific graphics device, such as a screen, allowing you to control on which screen the frame will appear.

Finally, the constructor Frame(String title, GraphicsConfiguration gc) combines the functionality of the previous two, creating a new, initially invisible Frame with the specified title and associated with the given GraphicsConfiguration.[27] This offers the most control over the initial state of the Frame in multi-screen setups.

It is important to note that all of these constructors for the Frame class may throw a java.awt.HeadlessException if the Java application is being run in a "headless" environment.[6] A headless environment is one in which a display, keyboard, or mouse is not present, such as when running an application on a server without a graphical interface. In such cases, attempting to create a Frame instance will result in this exception being thrown.

- **3.3. Key Methods for Managing Frame Properties**

Once a Frame object has been created, the java.awt.Frame class provides a variety of methods that allow developers to manage its properties and behavior.[6] These methods are essential for customizing the frame to suit the specific needs of an application.

To control the size of the frame, the setSize(int width, int height) and setSize(Dimension newSize) methods are used.[4] Both methods allow you to specify the frame's dimensions in pixels. The first takes the width and height as integer arguments, while the second takes a java.awt.Dimension object that encapsulates both width and height. It is generally recommended to call setSize() after the frame has been created and before making it visible.[6]

The visibility of the frame is controlled by the setVisible(boolean visibleFlag) method.[3] Passing true to this method makes the frame visible on the screen, while passing false hides it. By default, a Frame is created as invisible, so you must call setVisible(true) to display it to the user.

The title that appears in the frame's title bar can be set using the setTitle(String newTitle) method.[3] This method takes a String argument that will be displayed as the title of the window.

To control whether the user can resize the frame, the setResizable(boolean resizable) method is available.[25] By default, frames are resizable, allowing users to adjust their size. Passing false to this method will prevent the user from resizing the window.

For applications that use menus, a menu bar can be associated with a Frame using the setMenuBar(MenuBar mb) method.[25] This method takes a java.awt.MenuBar object as an argument, which will then be displayed at the top of the frame.

Finally, when it comes to closing a Frame, the approach depends on whether it is the main application window or a secondary window. For the main application window, you can terminate the entire program by calling System.exit(0).[6] However, to handle the event of a user attempting to close the window (e.g., by clicking the close button), you need to implement the windowClosing() method of the java.awt.event.WindowListener interface.[6] This involves adding a WindowListener to the Frame and defining the behavior you want to occur when the window is closing. If you are using javax.swing.JFrame (which extends Frame in the Swing toolkit), a more convenient way to handle window closing is by using the setDefaultCloseOperation(int operation) method, where you can specify actions like JFrame.EXIT_ON_CLOSE to automatically terminate the application when the window is closed.[49]

4. Creating Frame Windows in Applets (Historical Perspective)

- **4.1. Applets as Subclasses of Panel**

Java applets, small programs designed to be embedded within web pages and run by a Java-enabled web browser, are fundamentally based on the java.applet.Applet class.[5] The Applet class itself is part of the AWT and is a subclass of java.awt.Panel.[5] This inheritance is significant because it means that an applet, at its core, is a specialized type of panel, inheriting the properties and behaviors of a panel, such as the ability to contain other AWT components and using FlowLayout as its default layout manager.[42]

To include an applet in a web page, developers use the <applet> tag within the HTML markup.[30] This tag specifies the location of the applet's compiled code (typically a .class file), as well as the width and height that the applet should occupy within the web page's layout. The web browser, upon encountering this tag, will then load and execute the applet within the designated area.

Applets have a well-defined lifecycle that is managed by the web browser.[66] This lifecycle includes several key methods that are called by the browser at different stages of the applet's existence. The init() method is invoked only once when the applet is first loaded into the browser. It is typically used for one-time initialization tasks, such as setting up variables or loading resources.[66] The start() method is called each time the applet becomes visible on the screen, for example, when the user navigates to the web page containing the applet or when the browser window is restored. It is often used to start animations or other processes that should run while the applet is visible.[66] The paint(Graphics g) method is called whenever the applet needs to be drawn or redrawn on the screen. This is where the applet's visual content is rendered using the Graphics object provided by the browser.[66] The stop() method is called when the applet is no longer visible, such as when the user navigates away from the page. It is used to pause any ongoing processes or animations.[66] Finally, the destroy() method is called just before the applet is unloaded from memory. It is used for any final cleanup tasks, such as releasing resources.[66] Understanding this lifecycle is crucial for developing applets that behave correctly within the web browser environment.

- **4.2. Instantiating and Displaying Frame Windows from Applets**

Historically, Java applets, running within the confines of a web browser, had the capability to create and display separate, top-level Frame windows using the java.awt.Frame class.[5] This was often done to provide a more traditional application-like interface or to display content outside the applet's designated area within the web page.

To create a Frame window from an applet, the process typically involved instantiating the Frame class within one of the applet's lifecycle methods, such as the init() method, or in response to a user action.[5] For example, an applet might create a new Frame object when a user clicks a specific button within the applet.

Once a Frame object was created, it was necessary to set its dimensions using the setSize() method.[6] This determined the initial width and height of the separate frame window, allowing the developer to control its size on the user's screen. Without setting the size, the frame might appear very small or not at all.

To make the newly created Frame window visible to the user, the setVisible(true) method had to be called.[6] This action would cause the frame to be displayed as a separate window on the user's desktop, often appearing outside the bounds of the web browser window that contained the applet.

To properly manage the lifecycle of these separate Frame windows, especially handling the user's attempt to close them, it was common practice to add a WindowListener to the Frame object.[6] This listener would implement methods to respond to window events, such as the windowClosing() event, which is triggered when the user clicks the close button. Often, a java.awt.event.WindowAdapter was used as a convenient base class for this listener, allowing developers to override only the methods they were interested in, such as windowClosing(), where they could define the action to take when the frame was closed, like hiding or disposing of the frame.

In the context of web development history, applets provided a way to embed interactive content within web pages. However, they were constrained by the browser's environment and security sandbox. Creating separate Frame windows from applets was a technique used to overcome some of these limitations, allowing for the development of more complex user interfaces that could exist as independent windows on the user's desktop, providing a more traditional application experience despite being initiated from within a web browser.[64]

Despite this historical use, modern web development has largely shifted away from Java applets due to various security concerns and the emergence of alternative technologies like JavaScript and WebAssembly that offer more integrated and secure ways to create interactive web content.[1] Nevertheless, understanding the capability of applets to create and manage Frame windows provides valuable historical context in the evolution of web-based applications and reinforces the fundamental principles of window management within the AWT framework. The core AWT concepts of creating, sizing, making visible, and handling events for Frame windows remain relevant even in standalone AWT applications developed today.

5. Displaying Information: Graphics and Text in AWT

- ### 5.1. Introduction to the Graphics Class

The java.awt.Graphics class is an abstract superclass that serves as the foundation for all graphics contexts in Java's Abstract Window Toolkit.[6] It provides a set of methods that enable an application to draw and paint on various Component objects, such as windows, panels, and canvases, as well as onto off-screen images. Essentially, a Graphics object represents a drawing surface with a defined area where graphical operations can be performed.

A Graphics object encapsulates a significant amount of state information that is required for performing basic rendering operations.[10] This state includes several key properties: the specific Component object on which the drawing should occur, a translation origin that defines the coordinate system for rendering, the current clipping area that restricts drawing to a certain region, the current color that will be used for drawing, the current font for rendering text, the logical pixel operation function (which determines how new pixels are combined with existing ones), and the XOR alternation color (used in XOR paint mode). All subsequent drawing operations performed using a particular Graphics object will be affected by its current state.

The most common way to obtain a Graphics object is within the paint(Graphics g) method of a Component.[6] This method is called automatically by the AWT event dispatching thread whenever the component needs to be drawn on the screen or when it has been invalidated and requires repainting. The Graphics object passed as an argument to the paint() method provides the necessary context for drawing within the bounds of that specific component. It is within this paint() method that developers typically implement their custom drawing logic, using the methods provided by the Graphics class to render shapes, text, images, and other graphical elements. It is generally not recommended to try to create Graphics objects directly; instead, one should rely on the framework to provide them through the paint() method or other appropriate mechanisms.

- **5.2. Drawing Shapes and Lines**

The java.awt.Graphics class provides a fundamental set of methods that allow developers to draw basic geometric shapes and lines on AWT components.[9] These methods utilize the current color set in the Graphics object.

To draw a straight line between two points, the drawLine(int x1, int y1, int x2, int y2) method is used.[9] The parameters (x1, y1) specify the starting coordinates of the line, and (x2, y2) specify the ending coordinates.

The drawRect(int x, int y, int width, int height) method draws the outline of a rectangle.[9] The top-left corner of the rectangle is at (x, y), and its dimensions are defined by width and height. To draw a filled rectangle, the fillRect(int x, int y, int width, int height) method is used.[9] This method fills the interior of the specified rectangle with the current color.

To draw the outline of an oval, the drawOval(int x, int y, int width, int height) method is available.[9] The oval is inscribed within the bounding rectangle defined by the top-left corner (x, y) and the dimensions width and height. To draw a filled oval, the fillOval(int x, int y, int width, int height) method is used [9], filling the oval with the current color.

Arcs, which are portions of an oval, can be drawn using the drawArc(int x, int y, int width, int height, int startAngle, int arcAngle) method.[9] This method draws the outline of an arc defined by the bounding rectangle, the starting angle in degrees (startAngle), and the angular extent of the arc in degrees (arcAngle). The fillArc(int x, int y, int width, int height, int startAngle, int arcAngle) method fills the arc.[9]

Polygons, which are closed shapes with multiple sides, can be drawn using the drawPolygon(int xPoints, int yPoints, int nPoints) method, where xPoints and yPoints are arrays of x and y coordinates of the polygon's vertices, and nPoints is the number of vertices.[9] There is also an overloaded version, drawPolygon(Polygon p), that takes a java.awt.Polygon object. To draw a filled polygon, the fillPolygon(int xPoints, int yPoints, int nPoints) and fillPolygon(Polygon p) methods are used.[9]

- **5.3. Working with Color**

The java.awt.Color class plays a fundamental role in AWT, allowing developers to define and manipulate colors for use in drawing operations.[5] It can represent colors in the default sRGB color space or in arbitrary color spaces.

The Color class provides a set of predefined color constants, which are static final fields representing commonly used colors. These include Color.RED, Color.BLUE, Color.GREEN, Color.BLACK, Color.WHITE, and many others.[11] Using these constants is a convenient way to specify standard colors in your AWT applications.

For more specific color requirements, you can create custom colors using the Color(int red, int green, int blue) constructor.[17] This constructor takes three integer arguments, representing the intensity of the red, green, and blue color components, respectively. Each value should be in the range from 0 to 255, where 0 indicates no intensity of that color component, and 255 indicates the maximum intensity. By combining different values for red, green, and blue, you can create a vast spectrum of colors.

Once you have a Color object, either a predefined constant or a custom-created instance, you can set the current drawing color of a Graphics object using the setColor(Color c) method.[9] This method takes a Color object as an argument and sets it as the color that will be used for all subsequent drawing operations performed with that Graphics object until the color is changed again by another call to setColor(). This allows for flexibility in drawing different parts of a GUI in different colors.

- **5.4. Paint Modes**

The java.awt.Graphics class provides two methods for controlling how colors are applied during drawing operations: setPaintMode() and setXORMode(Color c1).[5]

The setPaintMode() method sets the graphics context to the default paint mode, where new drawing operations will simply overwrite the existing pixels in the drawing area with the current color.[5] This is the most common and straightforward way of drawing, where anything drawn subsequently will appear on top of what was already there.

The setXORMode(Color c1) method, on the other hand, sets the paint mode to use an exclusive OR (XOR) operation when drawing.[5] This method takes a Color object (c1) as an argument, which is used in the XOR operation. In XOR mode, when a pixel is drawn, its color is determined by performing a bitwise XOR operation between the current color, the specified color (c1), and the color of the pixel that is already at that location on the screen.[9] A notable characteristic of drawing in XOR mode is that if you draw the same shape in the same location twice, it will effectively erase itself and restore the original background.[9] This behavior makes XOR mode particularly useful for temporary drawing effects, such as highlighting selections or drawing moving objects like a cursor, where the underlying content should not be permanently altered.

- **5.5. Working with Fonts**

The java.awt.Font class is essential for rendering text within AWT applications, allowing developers to specify the typeface, style, and size of the text being displayed.[5] A Font object represents a specific font to the system and is defined by three key attributes: its name (which can be a logical font name like "Serif", "SansSerif", "Monospaced" or a specific font face name available on the system, such as "Arial" or "Times New Roman"), its style (which can be Font.PLAIN, Font.BOLD, Font.ITALIC, or a combination of Font.BOLD and Font.ITALIC using the bitwise OR operator), and its point size (an integer value indicating the height of the font in points, where one point is approximately 1/72 of an inch).[13]

To create an instance of the Font class, you typically use the constructor Font(String name, int style, int size).[14] For example, to create a bold, 12-point Arial font, you would write: new Font("Arial", Font.BOLD, 12). The style parameter accepts integer constants defined in the Font class: Font.PLAIN for regular text, Font.BOLD for bold text, Font.ITALIC for italic text, and you can combine styles using the bitwise OR operator, for instance, Font.BOLD | Font.ITALIC for bold and italic text.

Once you have created a Font object, you need to set it as the current font in a Graphics context before drawing any text. This is done using the setFont(Font font) method of the Graphics class.[9] For instance, if you have a Graphics object named g and a Font object named myFont, you would call g.setFont(myFont) to use that font for all subsequent text drawing operations performed with that Graphics object.

- **5.6. Managing Text Output and Using FontMetrics**

To display text within an AWT window, the java.awt.Graphics class provides the drawString(String message, int x, int y) method.[1] This method draws the specified message string using the current font and color of the Graphics object. The (x, y) coordinates define the starting position for the text, specifically the baseline of the leftmost character in the string. In the AWT coordinate system, the upper-left corner of a window or component is at the location (0,0).[6]

For more precise control over text layout and to determine the dimensions of text rendered in a specific font, AWT provides the java.awt.FontMetrics class.[5] FontMetrics is an abstract class that encapsulates various measurements about a font, such as its height, the width of individual characters and strings, the ascent (the height of the font above the baseline), the descent (the depth of the font below the baseline), and the leading (the interline spacing).[103]

To obtain a FontMetrics object for a particular Font, you use the getFontMetrics(Font f) method of the Graphics class.[5] You need a Graphics object to get the font metrics because the exact metrics can depend on the rendering context, such as the screen resolution. For example, if you have a Graphics object g and a Font object myFont, you can get the corresponding FontMetrics object by calling FontMetrics fm = g.getFontMetrics(myFont).

Once you have a FontMetrics object, you can use its methods to get detailed information about the font:

- getHeight(): Returns the total height of the font, which is typically the sum of the ascent, descent, and leading.[103]
- stringWidth(String str): Returns the total width of the specified string when rendered in this font, in pixels.[104]
- charWidth(char ch): Returns the width of the specified character in pixels.[103]
- getAscent(): Returns the ascent of the font, which is the distance from the baseline to the top of most alphanumeric characters.[103]
- getDescent(): Returns the descent of the font, which is the distance from the baseline to the bottom of characters with descenders.[103]
- getLeading(): Returns the leading, which is the interline spacing recommended for this font.[103]

Beyond these, FontMetrics provides a wide range of other methods for obtaining more specific measurements, such as getMaxAscent(), getMaxDescent(), getMaxAdvance(), and methods for getting the widths of byte arrays, character arrays, and the advance widths of the first 256 characters in the font.[15] It also includes methods for getting line metrics, string bounds, and font render context, as well as for checking if the font has uniform line metrics.[15]

6. Working with AWT Controls, Layout Managers, and Menus

- **6.1. AWT Controls: Building Blocks of Interactive GUIs**

AWT controls are essential Component subclasses that enable users to interact with

Java applications through graphical interfaces.[2] These controls provide a variety of ways for users to input data, trigger actions, and make selections.

Commonly used AWT controls include:

- Label: A component for displaying static text or informational messages.[2]
- Button: A control that triggers an action when clicked by the user.[2]
- TextField: A component that allows the user to enter and edit a single line of text.[2]
- TextArea: A component that allows the user to enter and edit multiple lines of text.[2]
- Checkbox: A component that presents a binary choice (on or off) to the user.[2]
- CheckboxGroup: A utility class for grouping Checkbox components, ensuring that only one checkbox in the group can be selected at any time (used for creating radio button-like behavior).[2]
- Choice: A component that provides a drop-down list of options from which the user can select a single item.[2]
- List: A component that displays a scrollable list of text items, allowing the user to select one or multiple items.[2]
- Scrollbar: A component that allows the user to select a value from a continuous range by dragging a thumb along a bar.[2]
- Canvas: A blank rectangular area on which the application can draw custom graphics or handle user input events directly.[2]
- Image: A component used to display graphical images.[2]
- Dialog and FileDialog: Specialized top-level windows used for displaying messages to the user or for allowing them to select files from the file system.[2]

To use these controls in an AWT application, you typically create instances of the appropriate control classes and then add them to a Container object (such as a Frame, Panel, or Dialog) using the add() method.[3] Each AWT control inherits fundamental properties and behaviors from the abstract Component class, including attributes like size, position, and visibility, as well as the ability to handle various types of events.[2]

- **6.2. Layout Managers: Arranging Components Within Containers**

AWT employs layout managers to automate the process of positioning and sizing components within a Container.[1] These managers handle the arrangement of components in a way that is generally independent of the underlying platform and can adapt to different screen sizes and resolutions.

Commonly used AWT layout managers include:

- FlowLayout: Arranges components in a directional flow, similar to text in a paragraph. It is the default layout for Panel and Applet.[2]
- BorderLayout: Arranges components in five regions: North, South, East, West, and Center. It is the default for Window, Frame, and Dialog.[2]
- GridLayout: Arranges components in a grid with a specified number of rows and columns, where each cell in the grid has the same size.[2]
- CardLayout: Treats each component as a card, where only one card is visible at a time. This is useful for creating interfaces with multiple views that the user can navigate through.[2]
- GridBagLayout: A highly flexible layout manager that arranges components within a grid but allows them to span multiple rows and columns and to have different sizes.[2]

To apply a layout manager to a Container, you use the setLayout(LayoutManager m) method, passing an instance of the desired layout manager class as the argument.[3]

- **6.3. Menus: Providing Application Functionality**

AWT provides a set of classes for creating and managing menus in GUI applications, allowing developers to offer a structured way for users to access application features.[3]

The MenuBar class represents the menu bar that is typically displayed at the top of a Frame.[4] You associate a MenuBar with a Frame using the setMenuBar(MenuBar mb) method.[25]

A Menu object represents a pull-down menu that appears when an item in the menu bar is selected.[4] You add Menu objects to a MenuBar using the add(Menu m) method.[163]

Individual items within a Menu are represented by the MenuItem class.[4] You add MenuItem objects to a Menu using the add(MenuItem mi) method.[159]

A specialized type of menu item is CheckboxMenuItem, which represents a menu item that can be checked or unchecked.[16]

MenuItem objects can have labels and can be associated with keyboard shortcuts using the MenuShortcut class.[157] To make a menu item perform an action when selected, you add an ActionListener to it.[157]

7. Conclusion: AWT and Object-Oriented Principles in GUI Development

- **7.1. Recap of Key AWT Concepts**

This report has explored the fundamental concepts of GUI development in Java using the Abstract Window Toolkit (AWT). We examined the core AWT classes for windowing, including Component, Container, Window, Frame, Dialog, and Panel, highlighting their roles and relationships within the AWT hierarchy. We detailed the process of working with Frame windows, covering their purpose, characteristics, constructors, and essential methods for managing their properties. We also discussed the historical context of creating Frame windows from within applets. The report further explored the Graphics class and its methods for drawing shapes, lines, and text, as well as techniques for working with color and paint modes. We emphasized the importance of the Font class for managing text appearance and the FontMetrics class for precisely measuring text dimensions. Finally, we provided an overview of common AWT controls, layout managers, and the classes used for creating application menus.

- **7.2. Reinforcing the Connection Between AWT and Object-Oriented Programming**

The design of the Abstract Window Toolkit is deeply intertwined with the principles of Object-Oriented Programming. The clear **inheritance** hierarchy, with Component serving as the root and specialized classes like Frame and Button extending its functionality, exemplifies how inheritance promotes code reuse and the creation of specialized GUI elements. **Abstraction** is evident in the abstract classes such as Component and Graphics, which define high-level interfaces for interacting with GUI elements and drawing operations, effectively hiding the complexities of platform-specific implementations. **Encapsulation** is demonstrated by classes like Font and Color, which bundle together data and the methods for manipulating that data, leading to more modular and maintainable code. Lastly, the event handling mechanism in AWT, which relies on interfaces like ActionListener and WindowListener, showcases **polymorphism** by allowing different components to generate similar events and different listeners to handle those events in their own specific ways. Understanding these fundamental OOP principles is not only crucial for effectively working with the AWT framework but also for designing well-structured, reusable, and maintainable GUI applications. While AWT is an older toolkit, its foundational concepts and the object-oriented design patterns it employs have significantly influenced subsequent GUI frameworks in Java, including Swing and JavaFX. Therefore, gaining a solid understanding of AWT provides a valuable stepping stone for learning and working with these more modern GUI toolkits as well.

Works cited

1. Abstract Window Toolkit - Wikipedia, accessed on April 21, 2025, https://en.wikipedia.org/wiki/Abstract_Window_Toolkit
2. Java AWT Tutorial - GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/java-awt-tutorial/
3. Java AWT Tutorial - Tpoint Tech, accessed on April 21, 2025, https://www.tpointtech.com/java-awt
4. Java AWT - Hansraj College, accessed on April 21, 2025, https://www.hansrajcollege.ac.in/hCPanel/uploads/elearning/elearning_document/8_Java_AWT_ButtonNew.pdf
5. Chapter 01 Introduction the Abstract Window Toolkit (AWT) - GitHub Pages, accessed on April 21, 2025, https://cwidevs.github.io/practice_files/AJPFILES/AJP_CH1_cwipedia.pdf
6. Chapter 25 Introducing the AWT: Working with Windows, Graphics, and Text - Hansraj College, accessed on April 21, 2025, https://www.hansrajcollege.ac.in/hCPanel/uploads/elearning/elearning_document/01_04_2020_15_Chapter_25Introducing_the_AWT_Working_with_windows.pdf
7. AWT Hierarchy, accessed on April 21, 2025, https://www.csub.edu/~ychoi2/MIS%20260/NotesJava/chap55/ch55_10.html
8. AWT Window in Java - Tutorialspoint, accessed on April 21, 2025, https://www.tutorialspoint.com/awt/awt_window.htm
9. What is Java AWT Graphics? - GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/what-is-java-awt-graphics/
10. AWT Graphics Class Overview - Tutorialspoint, accessed on April 21, 2025, https://www.tutorialspoint.com/awt/awt_graphics_class.htm
11. Other AWT Classes - MIT, accessed on April 21, 2025, https://web.mit.edu/java_v1.0.2/www/tutorial/ui/overview/otherClasses.html
12. java.awt Class Font, accessed on April 21, 2025, https://www.beg.utexas.edu/lmod/agi.servlet/doc/detail/java/awt/Font.html
13. 3: Fonts and Colors - Java AWT Reference [Book] - O'Reilly, accessed on April 21, 2025, https://www.oreilly.com/library/view/java-awt-reference/9781565922402/06_chapter-03.html
14. Class java.awt.Font - Washington, accessed on April 21, 2025, https://courses.cs.washington.edu/courses/cse341/98au/java/jdk1.2beta4/docs/api/java/awt/Font.html
15. Font (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/Font.html
16. java.awt (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/package-summary.html
17. Add RGB Values Into setColor() in Java | Baeldung, accessed on April 21, 2025, https://www.baeldung.com/java-setcolor-rgb-values
18. Add RGB Values Into setColor() in Java - Java Code Geeks, accessed on April 21, 2025, https://www.javacodegeeks.com/add-rgb-values-into-setcolor-in-java.html

19. AWT Event Listeners - Tutorialspoint, accessed on April 21, 2025, https://www.tutorialspoint.com/awt/awt_event_listeners.htm
20. Event Handling in Java | GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/event-handling-in-java/
21. AWT Event Handling - Tutorialspoint, accessed on April 21, 2025, https://www.tutorialspoint.com/awt/awt_event_handling.htm
22. Package java.awt.event - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/event/package-summary.html
23. Class java.awt.Window - Lysator, accessed on April 21, 2025, http://www.lysator.liu.se/java/apidocs/java.awt.Window.html
24. Window (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/Window.html
25. Class java.awt.Frame - Washington, accessed on April 21, 2025, https://courses.cs.washington.edu/courses/cse341/98au/java/jdk1.2beta4/docs/api/java/awt/Frame.html
26. java.awt Class Frame, accessed on April 21, 2025, https://www.beg.utexas.edu/lmod/agi.servlet/doc/detail/java/awt/Frame.html
27. AWT Frame in Java - Tutorialspoint, accessed on April 21, 2025, https://www.tutorialspoint.com/awt/awt_frame.htm
28. Class Frame - java.awt, accessed on April 21, 2025, https://download.java.net/java/early_access/valhalla/docs/api/java.desktop/java/awt/Frame.html
29. Frame (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/Frame.html
30. Java AWT tutorial, accessed on April 21, 2025, https://www.cse.lehigh.edu/~glennb/oose/java/javaawt.htm
31. Java Examples: Awt Frame Window - Owlcation, accessed on April 21, 2025, https://owlcation.com/stem/Java-Examples-Awt-Frame-Window
32. Working with frames and Java AWT - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/694971/working-with-frames-and-java-awt
33. java.awt Class Dialog, accessed on April 21, 2025, https://www.beg.utexas.edu/lmod/agi.servlet/doc/detail/java/awt/Dialog.html
34. java.awt Class Dialog, accessed on April 21, 2025, https://resources.mpi-inf.mpg.de/d5/teaching/ss05/is05/javadoc/java/awt/Dialog.html
35. : Uses of Class java.awt.Dialog, accessed on April 21, 2025, https://www.beg.utexas.edu/lmod/agi.servlet/doc/detail/java/awt/class-use/Dialog.html
36. AWT Dialog in Java - Tutorialspoint, accessed on April 21, 2025, https://www.tutorialspoint.com/awt/awt_dialog.htm
37. Uses of Class java.awt.Dialog - Washington, accessed on April 21, 2025, https://courses.cs.washington.edu/courses/cse341/98au/java/jdk1.2beta4/docs/api/java/awt/class-use/Dialog.html
38. The Java AWT: Dialogs - Jan Newmarch, accessed on April 21, 2025, https://www.jan.newmarch.name/java/xadvisor/dialogs/dialogs.html

39. Dialog (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/Dialog.html
40. How do I get this dialog class to work? - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/34910184/how-do-i-get-this-dialog-class-to-work
41. Java AWT Panel | GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/java-awt-panel/
42. AWT Panel in Java - Tutorialspoint, accessed on April 21, 2025, https://www.tutorialspoint.com/awt/awt_panel.htm
43. Class java.awt.Panel, accessed on April 21, 2025, https://www.eskimo.com/support/java/docs/api/java.awt.Panel.html
44. Class java.awt.Panel - Lysator, accessed on April 21, 2025, http://www.lysator.liu.se/java/apidocs/java.awt.Panel.html
45. Uses of Class java.awt.Panel (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/class-use/Panel.html
46. Panel (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/Panel.html
47. Class java.awt.Panel, accessed on April 21, 2025, http://homepage.divms.uiowa.edu/~slonnegr/13/qref11/java.awt.Panel.html
48. Class Frame - java.awt, accessed on April 21, 2025, https://download.java.net/java/early_access/genzgc/docs/api/java.desktop/java/awt/Frame.html
49. java - Writing correct JFrames for all Window Manager - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/8908313/writing-correct-jframes-for-all-window-manager
50. Class Window, accessed on April 21, 2025, https://courses.cs.vt.edu/~cs2114/Spring2016/GraphWindow/JavaDocs/CS2114/Window.html
51. Frame (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/Frame.html#setSize-int-int-
52. Java JFrame Class: Syntax Guide and Code Examples - IOFLOOD.com, accessed on April 21, 2025, https://ioflood.com/blog/jframe/
53. Java JFrame | GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/java-jframe/
54. creating frame by extending frame class - Java - OneCompiler, accessed on April 21, 2025, https://onecompiler.com/java/3wtp95e5u
55. Basic JPanel/JFrame class problem in Java - swing - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/70009328/basic-jpanel-jframe-class-problem-in-java
56. Advanced Java AWT Applet Frame Part 1 - YouTube, accessed on April 21, 2025, https://www.youtube.com/watch?v=WPLDPL2kwuk

57. Creating Frames using Swings in Java | GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/creating-frames-using-swings-java/
58. How to make Frame from Applet? - java - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/34785367/how-to-make-frame-from-applet
59. How to build a frame - Swing / AWT / SWT - Code Ranch, accessed on April 21, 2025, https://coderanch.com/t/556179/java/build-frame
60. Frame (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/Frame.html#setVisible-boolean-
61. Frame (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/Frame.html#getTitle--
62. Java WindowListener in AWT - GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/java-windowlistener-in-awt/
63. How to Make Frames (Main Windows) (The Java™ Tutorials > Creating a GUI With Swing > Using Swing Components) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/tutorial/uiswing/components/frame.html
64. How to open new applet window from a applet - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/2434483/how-to-open-new-applet-window-from-a-applet
65. java - How can I create a frame with a BorderLayout and assign each space a component?, accessed on April 21, 2025, https://stackoverflow.com/questions/72237283/how-can-i-create-a-frame-with-a-borderlayout-and-assign-each-space-a-component
66. Java Applet Basics | GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/java-applet-basics/
67. Section 5.1 The Basic Java Applet, accessed on April 21, 2025, https://math.hws.edu/eck/cs124/javanotes1/c5/s1.html
68. The "Hello World" Applet, accessed on April 21, 2025, https://www.whitman.edu/mathematics/java_tutorial/getStarted/applet/index.html
69. Creating A Frame Window in An Applet | PDF - Scribd, accessed on April 21, 2025, https://www.scribd.com/doc/71556635/Frame
70. using an applet to open another window (JFrame) - Code Ranch, accessed on April 21, 2025, https://coderanch.com/t/555324/java/applet-open-window-JFrame
71. How to create a JFrame within a Java applet or vice versa - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/14499270/how-to-create-a-jframe-within-a-java-applet-or-vice-versa
72. Graphics (Java SE 11 & JDK 11) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/en/java/javase/11/docs/api/java.desktop/java/awt/Graphics.html
73. in java.awt package Graphics is an abstract class so how we can create the Graphics class object and call the methods. | Sololearn: Learn to code for FREE!, accessed on April 21, 2025,

https://www.sololearn.com/en/Discuss/2061608/in-java-awt-package-graphics-is-an-abstract-class-so-how-we-can-create-the-graphics-class-object-and
74. How to get an instance of java.awt.Graphics in a java.applet.Applet? - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/30687327/how-to-get-an-instance-of-java-awt-graphics-in-a-java-applet-applet
75. Where are methods in java.awt.Graphics defined? - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/9010588/where-are-methods-in-java-awt-graphics-defined
76. Graphics (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html
77. 2: Simple Graphics - Java AWT Reference [Book] - O'Reilly, accessed on April 21, 2025, https://www.oreilly.com/library/view/java-awt-reference/9781565922402/05_chapter-02.html
78. Component (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/Component.html#paint-java.awt.Graphics-
79. Class Graphics - java.awt - developer.classpath.org!, accessed on April 21, 2025, https://developer.classpath.org/doc/java/awt/Graphics.html
80. Graphics (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html#drawLine-int-int-int-int-
81. How to add RGB values into setColor() in Java? - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/42855224/how-to-add-rgb-values-into-setcolor-in-java
82. graphics.setColor(color); not working - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/41321471/graphics-setcolorcolor-not-working
83. Appendix B: Java 2D Graphics - Interactive Textbooks hosted by Trinket, accessed on April 21, 2025, https://books.trinket.io/thinkjava/appendix-b.html
84. : Uses of Package java.awt, accessed on April 21, 2025, https://www.beg.utexas.edu/lmod/agi.servlet/doc/detail/java/awt/package-use.html
85. Java - Applet - setColor - eVidhya, accessed on April 21, 2025, https://evidhya.com/subjects/java/applet-setcolor
86. Fill and Set Color in Java Graphics - YouTube, accessed on April 21, 2025, https://www.youtube.com/watch?v=VuGc6vgMN4Y
87. setColor() method in the Graphics class (Beginning Java forum at Coderanch), accessed on April 21, 2025, https://coderanch.com/t/411903/java/setColor-method-Graphics-class

88. Class java.awt.Graphics2D - Washington, accessed on April 21, 2025, https://courses.cs.washington.edu/courses/cse341/98au/java/jdk1.2beta4/docs/api/java/awt/Graphics2D.html
89. DebugGraphics (Java Platform SE 7) - SciJava Javadoc, accessed on April 21, 2025, https://javadoc.scijava.org/Java7/javax/swing/DebugGraphics.html
90. DXFGraphics, accessed on April 21, 2025, https://jsevy.com/java/jdxf/doc/com/jsevy/jdxf/DXFGraphics.html
91. I made a function that uses graphics and I wanted to call it in the main it did not work, accessed on April 21, 2025, https://stackoverflow.com/questions/30757944/i-made-a-function-that-uses-graphics-and-i-wanted-to-call-it-in-the-main-it-did
92. AWT Font in Java - Tutorialspoint, accessed on April 21, 2025, https://www.tutorialspoint.com/awt/awt_font.htm
93. Fonts Available in Java AWT - GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/fonts-available-in-java-awt/
94. Font.createFont(..) set color and size (java.awt.Font) - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/16761630/font-createfont-set-color-and-size-java-awt-font
95. Java - Applet - setFont - eVidhya, accessed on April 21, 2025, https://evidhya.com/subjects/java/applet-setfont
96. 7.7: Handling Text in a Graphics Context (Optional) - Engineering LibreTexts, accessed on April 21, 2025, https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_Languages/Java_Java_Java_-_Object-Oriented_Programming_(Morelli_and_Walde)/07%3A_Strings_and_String_Processing/7.07%3A_Handling_Text_in_a_Graphics_Context_(Optional)
97. Can't get Graphics.setFont() to work - Code Ranch, accessed on April 21, 2025, https://coderanch.com/t/516451/java/Graphics-setFont-work
98. Fonts and Text Layout - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/technotes/guides/2d/spec/j2d-fonts.html
99. Proper Way Of Setting The Font Size for Text Rendering In Image - Code Ranch, accessed on April 21, 2025, https://coderanch.com/t/754939/java/Proper-Setting-Font-Size-Text
100. Graphics2D setfont() heavily slows down the start up of java application - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/24971336/graphics2d-setfont-heavily-slows-down-the-start-up-of-java-application
101. How to Change Font Size in drawString Java - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/18249592/how-to-change-font-size-in-drawstring-java
102. Graphics (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html#drawString-jav

a.lang.String-int-int-
103. FontMetrics (Java SE 17 & JDK 17) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/FontMetrics.html
104. Differences Between Font and FontMetrics in Java - Tutorialspoint, accessed on April 21, 2025, https://www.tutorialspoint.com/what-are-the-differences-between-a-font-and-a-fontmetrics-in-java
105. FontMetrics Class In Java - C# Corner, accessed on April 21, 2025, https://www.c-sharpcorner.com/article/fontmetrics-class-in-java2/
106. Class java.awt.FontMetrics - Washington, accessed on April 21, 2025, https://courses.cs.washington.edu/courses/cse341/98au/java/jdk1.2beta4/docs/api/java/awt/FontMetrics.html
107. FontMetrics (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/FontMetrics.html
108. Uses of Class java.awt.FontMetrics (Java SE 19 & JDK 19 [build 1]), accessed on April 21, 2025, https://download.java.net/java/early_access/panama/docs/api/java.desktop/java/awt/class-use/FontMetrics.html
109. Getting FontMetrics StackOverflowError - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/22177948/getting-fontmetrics-stackoverflowerror
110. Font metrics and centering a character in a box - Code Ranch, accessed on April 21, 2025, https://coderanch.com/t/753449/java/Font-metrics-centering-character-box
111. Java: Friendlier way to get an instance of FontMetrics - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/2753514/java-friendlier-way-to-get-an-instance-of-fontmetrics
112. Uses of Class java.awt.FontMetrics - SciJava Javadoc, accessed on April 21, 2025, https://javadoc.scijava.org/Java7/index.html?java/awt/class-use/FontMetrics.html
113. Measuring Text (The Java™ Tutorials > 2D Graphics > Working with Text APIs), accessed on April 21, 2025, https://www.cs.auckland.ac.nz/references/java/java1.5/tutorial/2d/text/measuringtext.html
114. FontMetrics (Eclipse Platform API Specification), accessed on April 21, 2025, https://help.eclipse.org/latest/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/swt/graphics/FontMetrics.html
115. About fonts:, accessed on April 21, 2025, https://www2.seas.gwu.edu/~rhyspj/fall05cs143/lab8/lab850.html
116. Measuring Text - The Java™ Tutorials, accessed on April 21, 2025, https://docs.oracle.com/javase/tutorial/2d/text/measuringtext.html

117. How to measure a character width with a Font? - Code Ranch, accessed on April 21, 2025, https://coderanch.com/t/346045/java/measure-character-width-Font
118. awt - Java - FontMetrics without Graphics - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/2843601/java-fontmetrics-without-graphics
119. Java AWT Controls - EDUCBA, accessed on April 21, 2025, https://www.educba.com/java-awt-controls/
120. Using AWT controls, Layout Managers, and Menus, accessed on April 21, 2025, https://dducollegedu.ac.in/Datafiles/cms/ecourse%20content/java_lecture_13_%20AWT_Controls.pdf
121. AWT Controls in Java - Tutorialspoint, accessed on April 21, 2025, https://www.tutorialspoint.com/awt/awt_controls.htm
122. Java AWT Button | GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/java-awt-button/
123. java.awt (Java SE 17 & JDK 17) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/package-summary.html
124. Event Handling, accessed on April 21, 2025, https://www.cs.cmu.edu/afs/cs/academic/class/15212-s98/www/java/tutorial/ui/overview/events.html
125. Introduction to Frame, Panel, Applet, Button, Layout In Awt - C# Corner, accessed on April 21, 2025, https://www.c-sharpcorner.com/UploadFile/fd0172/introduction-to-frame-panel-applet-button-layout-in-awt/
126. FlowLayout - Learning Java, 4th Edition [Book] - O'Reilly, accessed on April 21, 2025, https://www.oreilly.com/library/view/learning-java-4th/9781449372477/ch19s01.html
127. BorderLayout & FlowLayout - Swing / AWT / SWT - Code Ranch, accessed on April 21, 2025, https://coderanch.com/t/337659/java/BorderLayout-FlowLayout
128. BorderLayout - Java - CS-Rutgers University, accessed on April 21, 2025, https://www.cs.rutgers.edu/courses/111/classes/fall_2011_tjang/texts/notes-java/GUI/layouts/20borderlayout.html
129. Java AWT | BorderLayout Class - GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/java-awt-borderlayout-class/
130. Java GUI BorderLayout application 4 buttons Left, right, up and down each move location, accessed on April 21, 2025, https://coderanch.com/t/649426/java/Java-GUI-BorderLayout-application-buttons
131. java.awt Class TextField, accessed on April 21, 2025, https://www.beg.utexas.edu/lmod/agi.servlet/doc/detail/java/awt/TextField.html
132. Java AWT TextField - GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/java-awt-textfield/

133. How can we display the output of function on JTextArea in java ? | Sololearn, accessed on April 21, 2025, https://www.sololearn.com/en/Discuss/874984/how-can-we-display-the-output-of-function-on-jtextarea-in-java-
134. JTextField (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/javax/swing/JTextField.html
135. Using Text Components - Java™ Tutorials - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/tutorial/uiswing/components/text.html
136. What is the purpose of `setLayout` method in Swing/AWT components? - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/49553205/what-is-the-purpose-of-setlayout-method-in-swing-awt-components
137. How to get my AWT frame to do more stuff? : r/learnjava - Reddit, accessed on April 21, 2025, https://www.reddit.com/r/learnjava/comments/n1hvvt/how_to_get_my_awt_frame_to_do_more_stuff/
138. How to print text to a text area - java - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/7375827/how-to-print-text-to-a-text-area
139. Exploring the AWT Layout Managers - Oracle, accessed on April 21, 2025, https://www.oracle.com/technical-resources/articles/javase/awtlayoutmgr.html
140. java - How to add an image to a frame in AWT? - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/75776789/how-to-add-an-image-to-a-frame-in-awt
141. 7: Layouts - Java AWT Reference [Book] - O'Reilly, accessed on April 21, 2025, https://www.oreilly.com/library/view/java-awt-reference/9781565922402/10_chapter-07.html
142. Interface java.awt.LayoutManager - MIT, accessed on April 21, 2025, http://web.mit.edu/java_v1.0.2/www/javadoc/java.awt.LayoutManager.html
143. AWT Layouts in Java - Tutorialspoint, accessed on April 21, 2025, https://www.tutorialspoint.com/awt/awt_layouts.htm
144. Uses of Interface java.awt.LayoutManager, accessed on April 21, 2025, https://download.java.net/java/early_access/valhalla/docs/api/java.desktop/java/awt/class-use/LayoutManager.html
145. Java Platform 1.2 API Specification: Interface LayoutManager2, accessed on April 21, 2025, https://javaalmanac.io/jdk/1.2/api/java/awt/LayoutManager2.html
146. LayoutManager (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/LayoutManager.html
147. Class java.awt.FlowLayout, accessed on April 21, 2025, https://legacy.ifa.hawaii.edu/users/gmm/java/docs/java.awt.FlowLayout.html
148. [Chapter 7] 7.2 FlowLayout - Litux, accessed on April 21, 2025, https://litux.nl/Books/Books/www.leothreads.com/e-book/oreillybookself/java/awt/ch07_02.htm

149. Java AWT | FlowLayout - GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/java-awt-flowlayout/
150. FlowLayout (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/FlowLayout.html
151. Java FlowLayout inside FlowLayout - Stack Overflow, accessed on April 21, 2025, https://stackoverflow.com/questions/52234113/java-flowlayout-inside-flowlayout
152. Java8-Source-Code/src/main/jdk8/java/awt/FlowLayout.java at master - GitHub, accessed on April 21, 2025, https://github.com/mynawang/Java8-Source-Code/blob/master/src/main/jdk8/java/awt/FlowLayout.java
153. Class java.awt.BorderLayout - MIT, accessed on April 21, 2025, http://web.mit.edu/java_v1.0.2/www/javadoc/java.awt.BorderLayout.html
154. SWING - BorderLayout Class - Tutorialspoint, accessed on April 21, 2025, https://www.tutorialspoint.com/swing/swing_borderlayout.htm
155. BorderLayout (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase%2F8%2Fdocs%2Fapi%2F%2F/index.html?java/awt/BorderLayout.html
156. BorderLayout - Swing / AWT / SWT - Code Ranch, accessed on April 21, 2025, https://coderanch.com/t/505294/java/BorderLayout
157. AWT MenuItem Control - Tutorialspoint, accessed on April 21, 2025, https://www.tutorialspoint.com/awt/awt_menuitem_control.htm
158. java.awt Class MenuItem, accessed on April 21, 2025, https://www.beg.utexas.edu/lmod/agi.servlet/doc/detail/java/awt/MenuItem.html
159. java.awt Class Menu, accessed on April 21, 2025, https://www.beg.utexas.edu/lmod/agi.servlet/doc/detail/java/awt/Menu.html
160. Uses of Class java.awt.MenuItem, accessed on April 21, 2025, https://cr.openjdk.org/~jjg/8239816/api.00/java.desktop/java/awt/class-use/MenuItem.html
161. Java AWT MenuItem & Menu - GeeksforGeeks, accessed on April 21, 2025, https://www.geeksforgeeks.org/java-awt-menuitem-menu/
162. 29 Java AWT Creating Menu bar, Menu and MenuItems - YouTube, accessed on April 21, 2025, https://www.youtube.com/watch?v=cpOhKw9qHug
163. MenuBar (Java Platform SE 8) - Oracle Help Center, accessed on April 21, 2025, https://docs.oracle.com/javase/8/docs/api/java/awt/MenuBar.html
164. Change Color of MenuBar, Menu, and MenuItem in Java using the AWT Components, accessed on April 21, 2025, https://stackoverflow.com/questions/30825824/change-color-of-menubar-menu-and-menuitem-in-java-using-the-awt-components