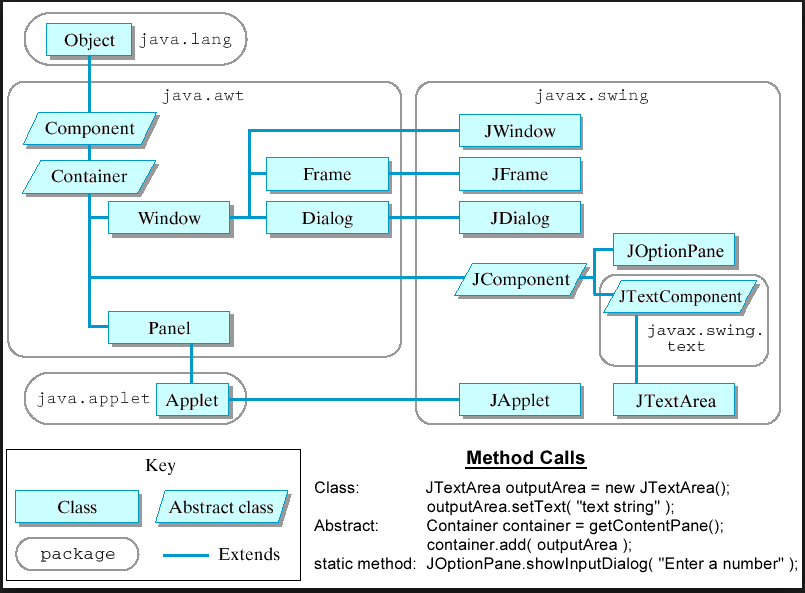
# Summary points you should/hopefully acquired through experimentation and code samples

(In the order I thought of them, for the most part in the sequence I outlined)

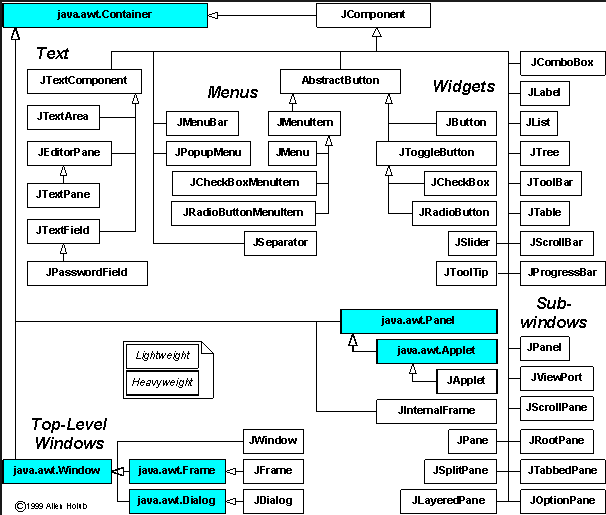
* Every container has a layout manager used to position and place components in the container. The three most common layout managers are **FlowLayout, GridLayout**  and **BorderLayout.**

* You CAN use **jPanel** as a subcontainer to group components
* Use the **add** method to place component to a **JFrame** or a JPanel. The default layout is **BorderLayout** and **JPanel’a** layout is **FlowLayout**
* You can set colours for GUI components using **java.awt.Color** class. Common names, 0 – 255 or RGB values all possible.
* The syntax to create a **Color** object is **Color color = new Color(r, g, b)**. Alternatively you can use the 13 standard colours as defined as constants in **java.awt.Color**
* Every Swing GUI component is a subclass of **javax.swing.JComponent** and **JComponent** is subclasss of **java.awt.Component.**  Inherited is; **font, background, foreground, height, width, preferredSize** in **Component, toolTipText** and **border.**
* You can use borders on any Swing component. Note: You can create an image icon using **ImageIcon** class and display it in a label and a button. Icons and borders can be shared.
* Each component has its own coordinate system, origin (0,0) at the upper left corner of the window. X- coordinate increases to the right as expected, and the y-coordinate downward.
* The **Graphics** class is an abstract class for displaying figures and images on the screen on *different* platforms. When you use the **paintComponent(g)** method to paint on a GUI component, the g is an instance of a concrete subclass of the abstract **Graphics** class for the specific platform. The **Graphics** class encapsulates the platform details and enables you to draw things uniformly without concern for the specific platform.
* Invoking **super.paintComponent(g)** is necessary to ensure that the viewing area is cleared before a new drawing is displayed. The user can request the component to be redisplayed by invoking the **repaint( )** method defined in the **Component** class. Invoking **repaint( )** causes **printComponent** to be invoked by the JVM (Java virtual machine). The user should never invoke **paintComponent** directly. This is the reason the **protected** visibility is sufficient for **paintComponent.**
* Normally you use **JPanel** as a canvas. To draw on a **JPanel**, you create a new class that extends **JPanel** and overrides the **paintComponent** method to tell the panel how to draw things.
* You can set fonts for the components of subjects you draw, and use font metrics to mearure font size. **Font** class and **FontMetrics** hold length, width and size fo a string which helps to place it in the correct position
* The **Component** class has the **setBackground, setForground,**  and **setFont** methods. These methods are used to set colours and fonts for the entire component. If you want different or several settings, you must use the **setColor** and **setFont** methods in **Graphics** class.
* To display an image, first create an image icon. You can then use **ImageIcons’s, getImage()** method to get an **image** object for the iage and draw the image using the **drawImage** method in the **java.awt.Graphics** class.

Swing:



JComponent



Javax.swing.JFrame

