Title:To layout() or not to layout().(Desktop Java Viewpoint)
Pub:***Java Developer's Journal***
Detail:Joe Winchester. 9.1 (Jan 2004): p40(1). (904 words)

A problem encountered by any GUI is--if the user resizes the application window at runtime, how should this be handled? The most desirable effect is that the controls flow into the new space to make the best use of it (lists and trees grow while buttons remain a fixed size), and an overall reduction in size doesn't clip or lose in formation.

Using layout managers, this problem is bread and butter for Java. The favored choice among Swing programmers is GridBagLayout where each component is placed in an imaginary grid cell. The height of each row or width of each column is made large enough to accommodate its largest component, and every other component can specify rules as to how it anchors itself to the cell's edges together with any padding, insets, and how may cells or rows it should span, as well as a weighting for how excess space in the overall GUI is divided between its rows and columns.

For some Java developers, learning the nuances of GridBagLayout is a rite of passage easily mastered, while for others it's avoided altogether. I know programmers whose policy is to ignore layout managers entirely and, instead, hard code the coordinates of a GUI's components with setBounds(Rectangle bounds). Because null layout fixes every component's bounds at design time and translated user-visible strings change length at runtime, applications then have to standardize on a single language and explicitly prevent users from resizing windows with setResizeable(false).

One reason this occurs is because developers often use tools to help build their GUI, and there's something very compelling about being able to size and position controls in null layout. The controls go where you tell them. It's a kind of WYPIWTG or "Where you put is where they go." In most Java GUI builders, with anything other than null layout, the experience is closer to "Put it down and watch everything move." It's rather like a car where the driver steers the wheel to where he wants to go and all the roads, buildings, and trees move and change size at the same time. The driver makes another correction only to find he can't go there and instead has to use dashboard buttons to change some of the car's properties before altering course. While this experience may be okay for bicycle riding cartoon clowns, it's not one that Java GUI developers should suffer unnecessarily. Some tools recommend doing everything in null layout (where precise positioning of components can occur) before switching the container to GridBagLayout, whereupon an algorithm kicks in that generates a guestimate GridBagConstraint. Other tools use an alternative view of the container where you manipulate components in an explicit grid and, although this is a very elegant solution, it still requires that you have an understanding of how GridBagLayout works.

It's easy to blame the GUI builders for why layout managers are hard, but they're only working with what's available in the IFC; the original programmers who designed GridBagLayout came up with a good engineering solution to a difficult problem.

What about tackling the problem slightly differently, with the layout logic concerned only with rearranging children when size is altered? Language locales could externalize a completely redesigned GUI class for that country's users. For resizing, there's a general set of rules that could be followed: tables and lists want to grow to occupy new space, buttons and labels stay the same size, text fields can be made wider but not deeper (unless they're multiline text fields), and so forth. The default algorithm might work okay in the 80/20 case, and if further customization were needed this could be achieved by specifying constraints that overrode the defaults.

Having null layout or GridBagLayout is all or nothing at the moment, whereas this would be a shade of gray in between. The developer experience could be flint compo nents are positioned and sized as with null layout, so GUI elements remain where they're put at design time, with the constraints being an additional parameter for what the growth rule is. A colleague recently pointed me to the FormBuilder package from http://www.jgnodies.com that goes some way toward providing this experience with a simple programming API, although I'm not yet aware of its integration in any GUI builders.

In a past life I programmed Windows GOIs and we had a rubberband.vbx, which is analogous to a custom container with all the smarts for resizing logic built in; however, control placement was done on an x and y grid. Another idea once suggested to me was that a GUI builder should allow you to design at two resolutions, and then extrapolate all the intermediate screen sizes.

Whatever the solution might be, what I hope is that we as Java developers can continue to think of new ways to innovate and improve the language and tools. If anyone hasn't already seen it, please visit javadesktop.org, which is part of the java.net community and is a great way to engage the Swing developers in a nice "flame-free" forum, as well as see what's in the pipeline. The other people to talk to about how to make user interfaces better are ... your users. Talk to them, listen to them, and push their feedback down the stack so we can all work to keep improving and enjoying Java clients.

Joe Winchester is a software developer working on WebSphere development tools for IBM in Hursley, UK.

joewinchester@sys-con.com

Source Citation
Winchester, Joe. "To layout() or not to layout()." *Java Developer's Journal* 9.1 (2004): 40. *Academic OneFile*. Web. 10 Nov. 2011.

Document URL
http://find.galegroup.com.ezproxy.lib.swin.edu.au/gtx/infomark.do?&contentSet=IAC-Documents&type=retrieve&tabID=T003&prodId=AONE&docId=A113052662&source=gale&srcprod=AONE&

Gale Document Number:A113052662