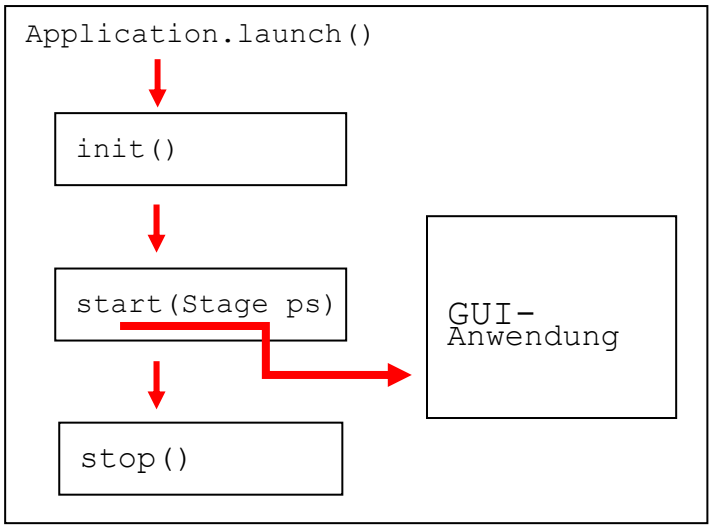
JavaFX GUI-Applikationen

Der Start einer FX – GUI-Applikation erfolgt immer über eine von "Application" abgeleitete "Main"-Klasse.
In der main(…)-Methode dieser Klasse erfolgt der Aufruf von launch(args).
Zuerst wird der Konstruktor abgearbeitet, danach die init()-Methode aufgerufen, dann die start(Stage primaryStage), und zum Schluss die stop().

Der JavaFX Lebenszyklus:



Die Klasse ***Application*** ist für jede JavaFX-Anwendung der Eintrittspunkt; sie wird über den Methoden-Aufruf ***launch()*** der in der "Ausgangs"-Klasse **Main** implementierten ***static void main(…)*** instanziert.

Hierbei geschieht folgendes:

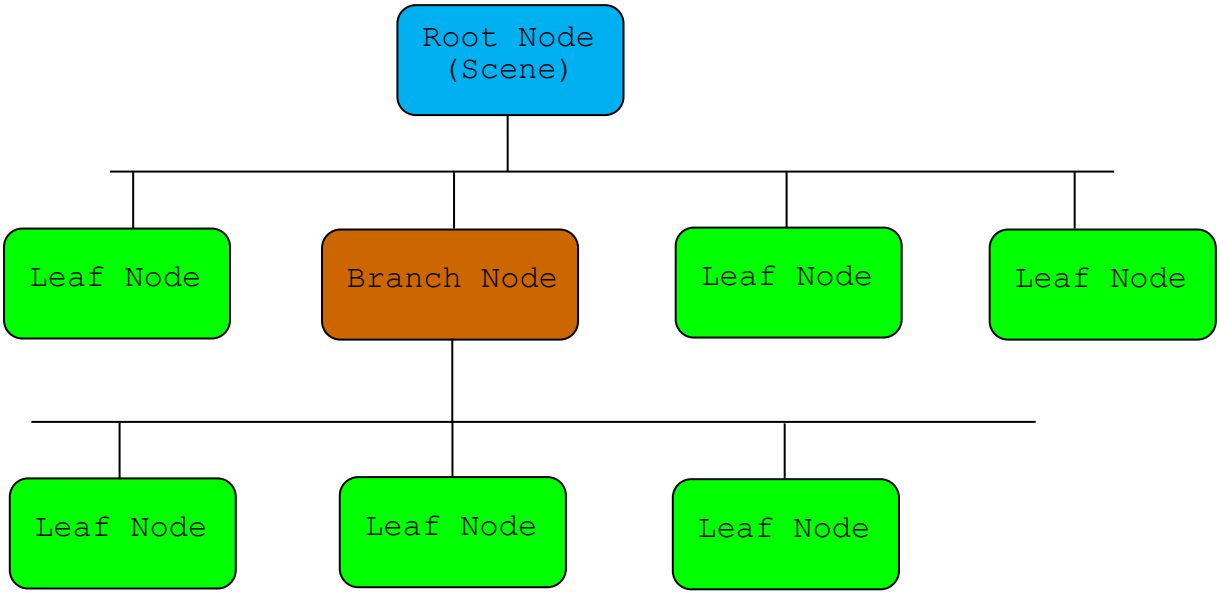
1. Die ***Application*** - Instanz wird erzeugt.
2. Die leer implementierte ***init()***-Methode der erzeugten Instanz wird ausgeführt (diese kann überschrieben werden; es dürfen hier jedoch noch keine **UI**-Elemente erzeugt werden!!)
3. Danach wird die ***start****(Stage primaryStage)* - Methode ausgeführt; in dieser wird die eigentliche FX - GUI-Applikation gestartet.
4. Diese "blockiert" die weitere Ausführung der ***start****(…)*-Methode; d.h. es wird solange gewartet, bis diese Applikation beendet ist. Das passiert, wenn
   - die Methode *Platfrom.exit()* aufgerufen wurde oder
   - wenn das letzte Fenster geschlossen wurde und das *implicitExit*-Attribut von *Platform* den Wert *true* hat.
5. Zuletzt wird dann die ***stop****()* Methode aufgerufen (hat in der Klasse **Application** wie die ***init()*** ebenfalls nur eine Leer-Implementierung, die aber genauso überschrieben werden kann).

Das Benutzen der *init()*-Methode ist dann besonders sinnvoll, wenn zu Beginn z.B. große Datenmengen geladen werden müssen. Die *start()*-Methode baut in der Zwischenzeit das eigentliche Applikationsfenster auf; somit kann man dem Benutzer schon einmal ein Willkommen-Fenster anzeigen, während die benötigten Daten bereitgestellt werden.

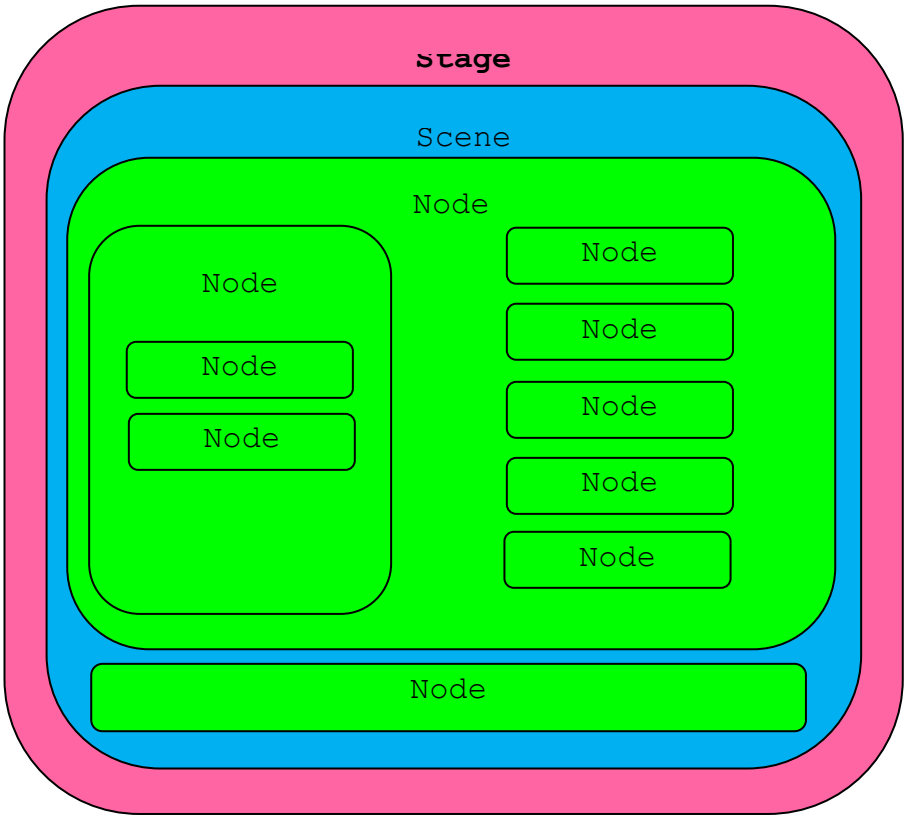
Der JavaFX Scene-Graph:

Die JavaFX **Stage** Klasse ist der eigentliche "Top-level JavaFX Container".
Die **"primary stage"** wird hierbei durch die jeweilige Plattform konstruiert und an die **start**(…)-Methode übergeben, zusätzliche Stage-Objekte können über die Applikation erzeugt werden..

Die JavaFX **Scene** ist der Container für jeglichen "Content" eines sogenannten JavaFX – Scene-Graphs.



Schematische Darstellung einer JavaFX - GUI:



Weitere bedeutende Features in JavaFX sind die sogenannten Properties und Bindings. Letztere setzen das Konzept um, die Werte von Variablen an andere Variablen zu binden. Ist der Wert der Variablen A an den Wert der Variablen B gebunden, wird A automatisch den Wert von Variable B erhalten, ohne dass man A explizit setzen muss.

Entsprechungen JavaFX <-> Swing

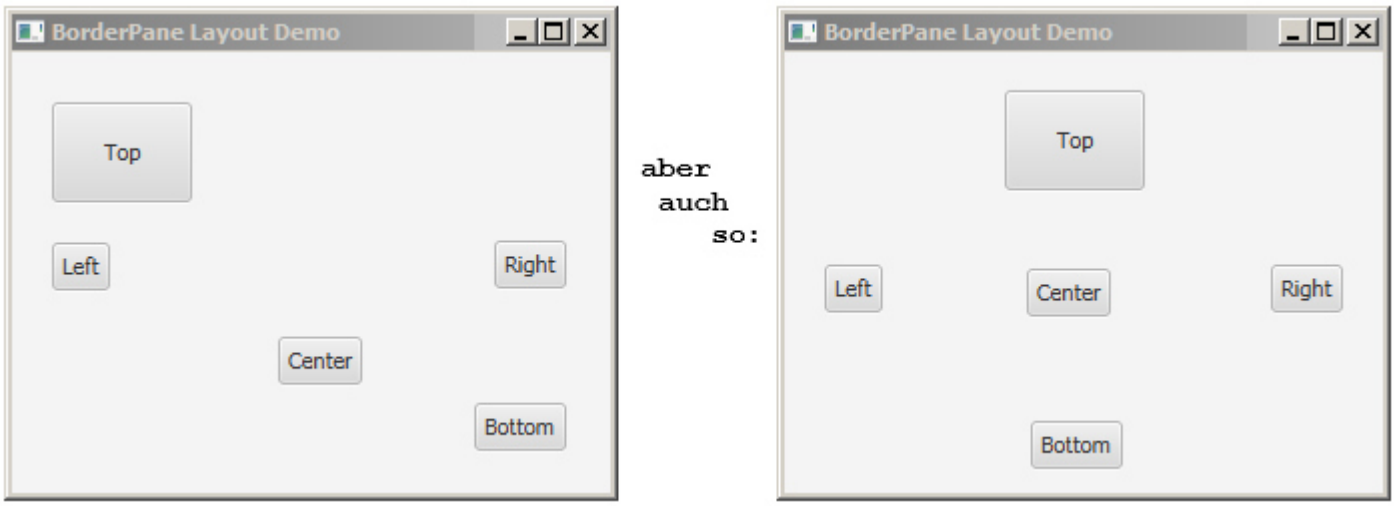
Properties	<->	Wrapper-Klassen
Stage	<->	JFrame oder JDialog (mit Modality etc.)
BorderPane	<->	JPanel (mit BorderLayout)
FlowPane	<->	JPanel (mit FlowLayout)
GridPane	<->	JPanel (mit GridLayout)
StackPane	<->	JFrame oder JPanel(mit CardLayout)

Neuerungen gegenüber Swing:

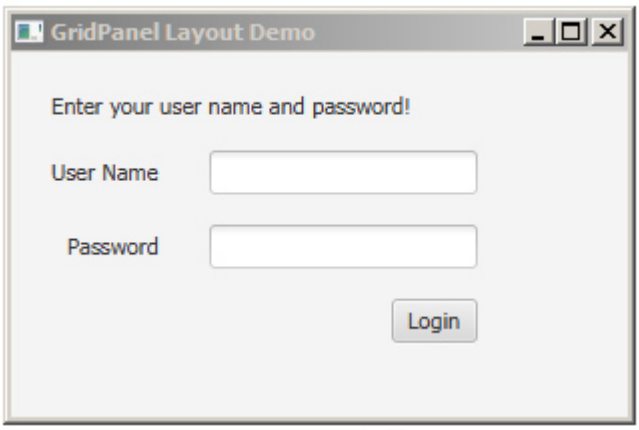
Layout-Panes statt Panels (denen man in Swing einen LayoutManager setzen muss)
Properties- und Bindings
CSS
Charts (Torten-, Balken, etc.-Diagramme)
Multi-Touch-Anwendungen
Animation und Multimedia (*Transitions*, *Interpolation*, *Timeline*-Animation)
FXML
⋮

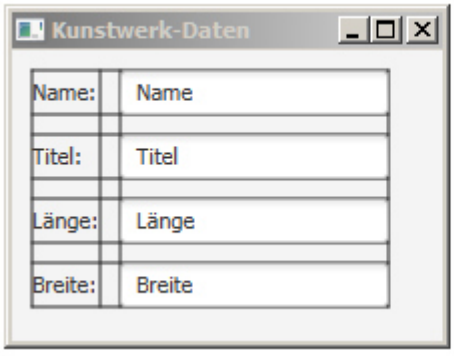
Container-Klassen in JavaFX

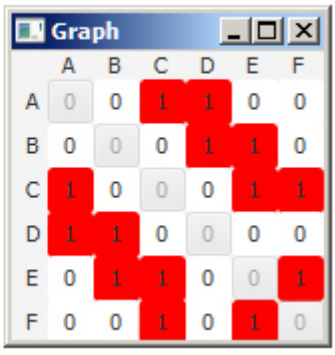
Eine **BorderPane** verteilt ihre "Content Nodes" in die Regionen **top**, **bottom**, **right**, **left** oder **center**. Deren Position in der entsprechenden Region kann jedoch weiter verändert werden, z.B. den Button **Top** in die Mitte oder nach rechts, dazu können Abstände an den Rändern kommen, Höhe und Breite beeinflusst werden usw.



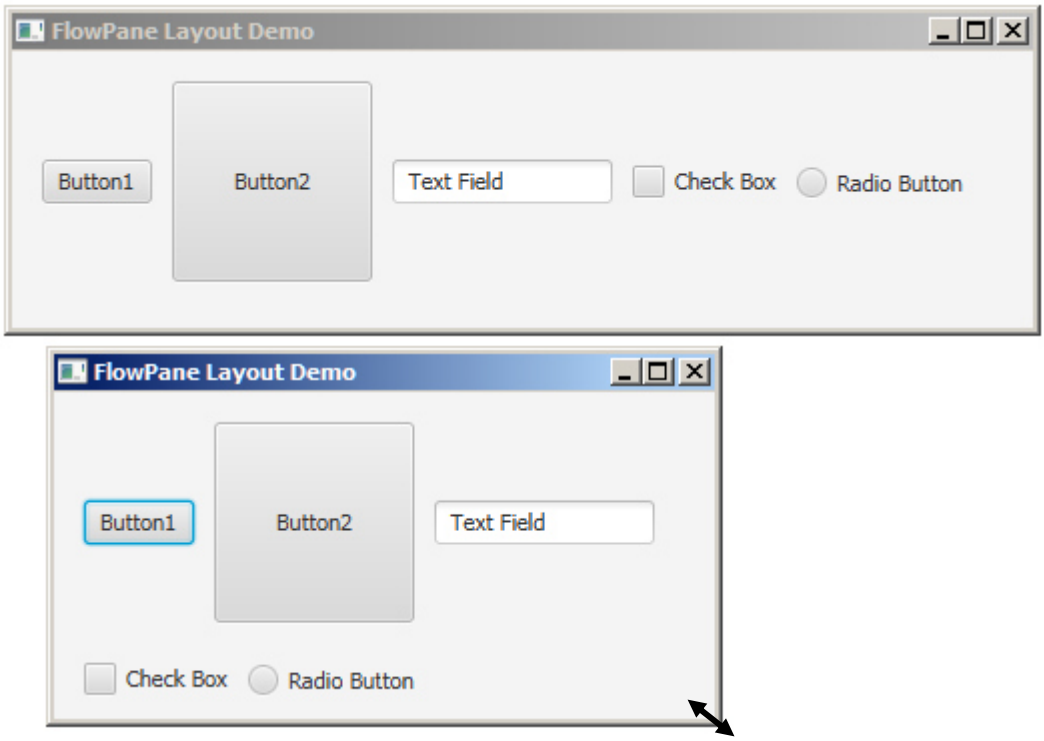
Die **GridPane** ermöglicht es, die Content Nodes in einem flexiblen Raster von Zeilen und Spalten anzuordnen.
Beispiele:



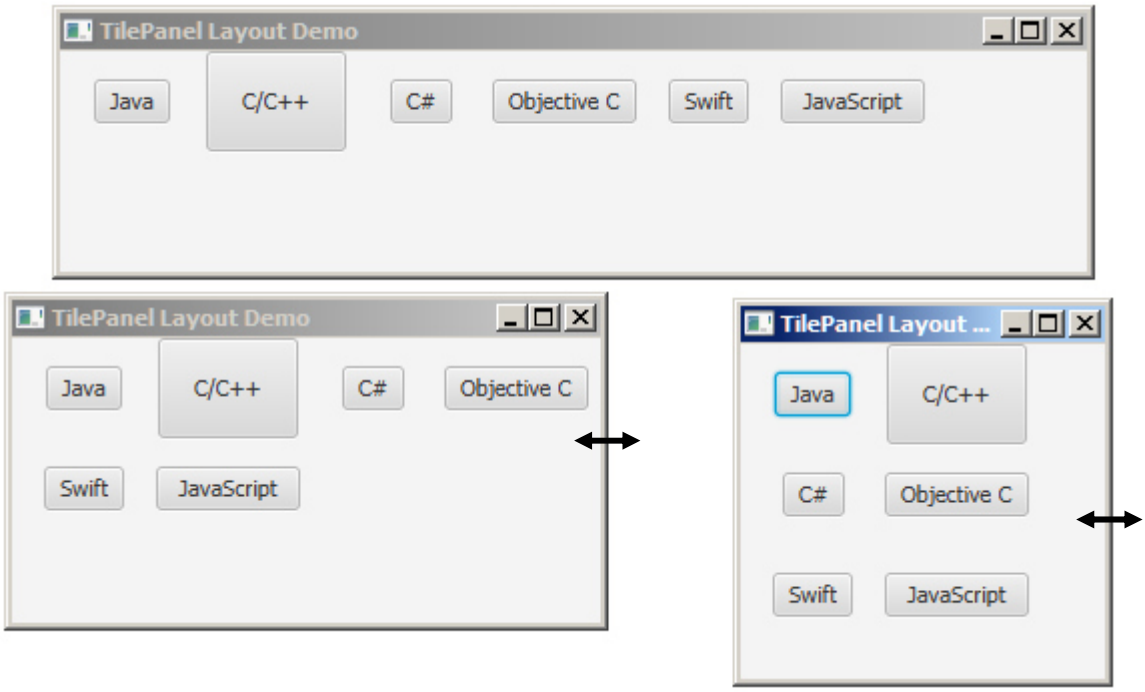




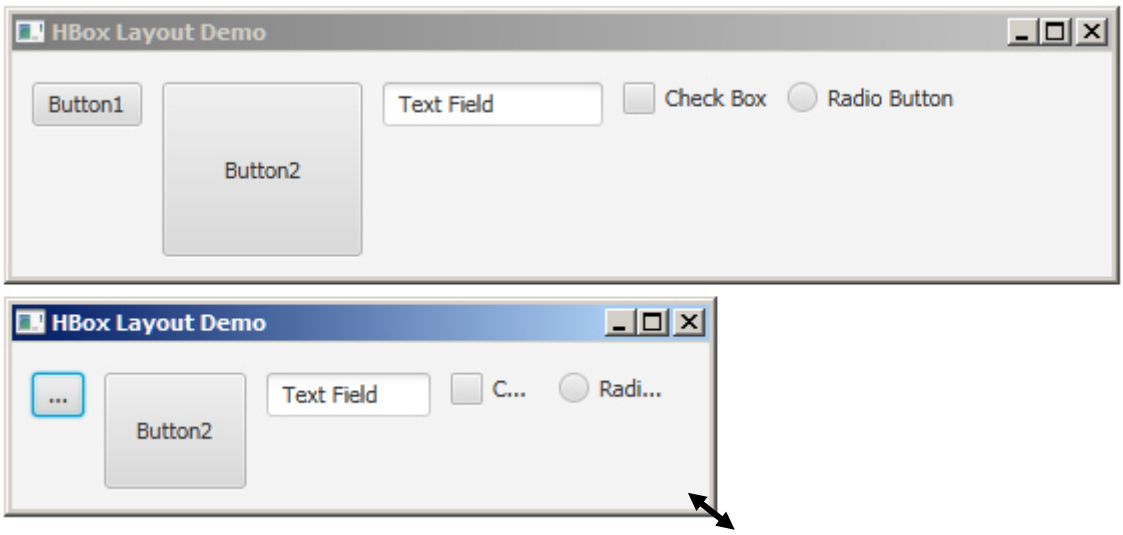
Eine **FlowPane** arrangiert ihre Content Nodes entweder in einem horizontalen oder vertikal "flow" unter Berücksichtigung einer "width" (horizontaler) oder "height" (vertikaler Abgrenzung).
Verhalten der FlowPane beim Verkleinern des Fensters:



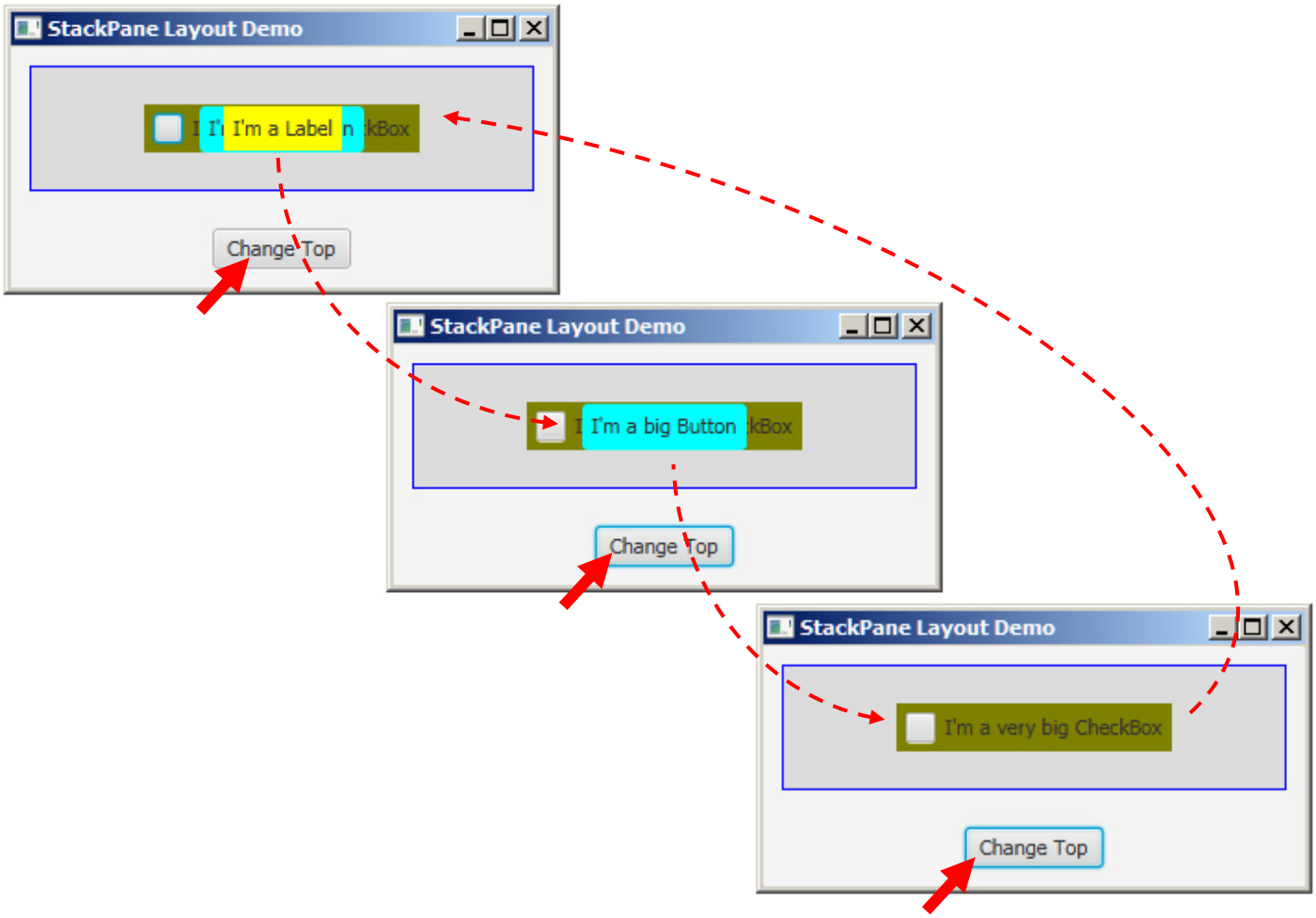
Eine **TilePane** funktioniert ähnlich wie eine **FlowPane**, allerdings ordnet sie die ebenfalls "fließenden" Nodes in gleich großen "Layout-Zellen" oder Kacheln an.
Verhalten einer TilePane beim Verkleinern des Fensters:



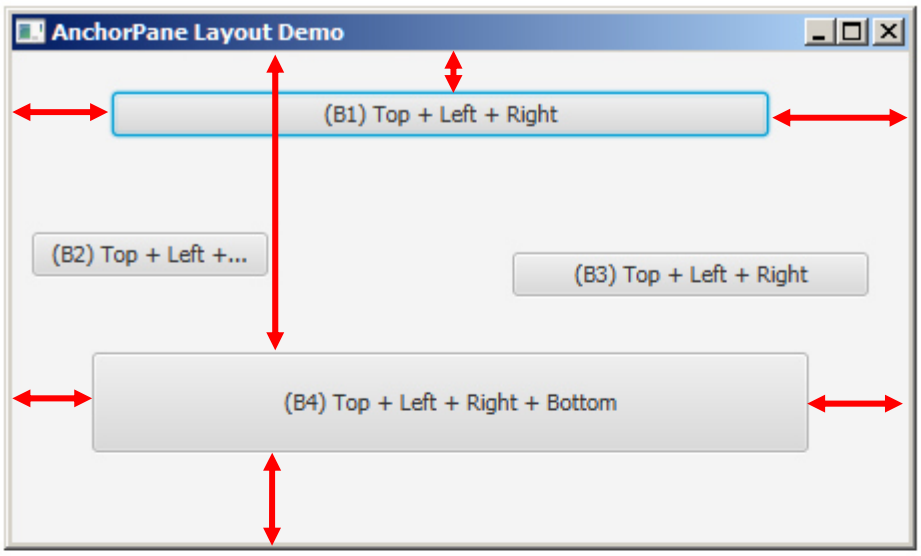
Eine **HBox** arrangiert ihre Content Nodes horizontal in einer einzelnen **"Zeile"**, eine **VBox** vertikal in einer **"Spalte"**; beim Verkleinern des Fensters werden die darin enthaltenen Nodes **NICHT** umgebrochen!

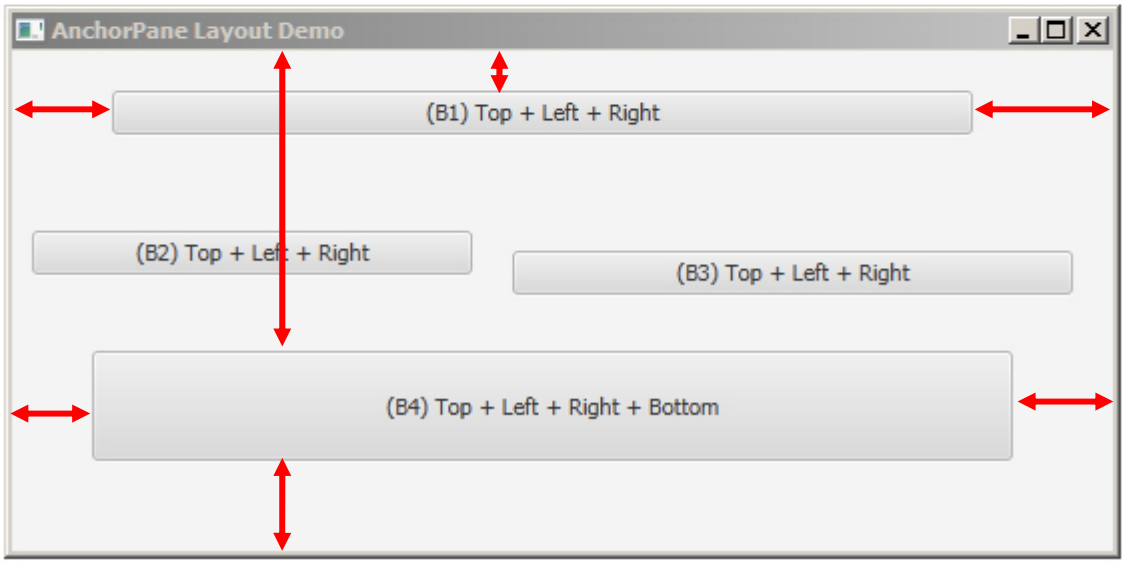


Eine **StackPane** platziert die Nodes **übereinander** in einem **"back-to-front stack"** bzw. einem **unten-nach-oben - Stapel**.
Beispiel: Eine **Checkbox**, ein **Button** und ein **Label** übereinander, drücken des Buttons "*Change Top*" rückt jeweils diee vorderste Komponente in den Hintergrund, wodurch die dahinterliegende vollständig sichtbar wird.



Die **AnchorPane** ermöglicht es, den Nodes Anker zu setzen, die sie an ihrem Erscheinungs-Ort fixieren, so dass sie sich beim Ändern der Fenster-Größe nur in jenen Dimensionen **nicht** verändern, in denen ihnen keine Anker gesetzt wurden.





usw.....

Viele diesr Panes ermöglichen es darüberhinaus, z.B. sogenannte **Paddings** ("Polsterungen", das bedeutet die Abstände nach außen) festzulegen, indem sog. **Insets** (Einfügungen) gesetzt werden - z.B. setPadding(new Insets(10,10,10,10)). Man kann aber beispielsweise auch horizontale und vertikale Zwischenräume zwischen den Nodes definieren(setHgap,setVgap) usw.

Warnung (!!!):

Wenn man beim Implementieren den Code-Vorschlägen von Eclipse folgt, immer **extrem(!!)** darauf achten, die richtigen Klassen aus einem der **FX-Packages** auszuwählen!!
Es gibt nämlich im alten AWT-Package viele ident benannte Klassen, die man rasch einmal irrtümlich auswählt, und dann lange über die nachfolgenden Syntax-Errors grübelt…