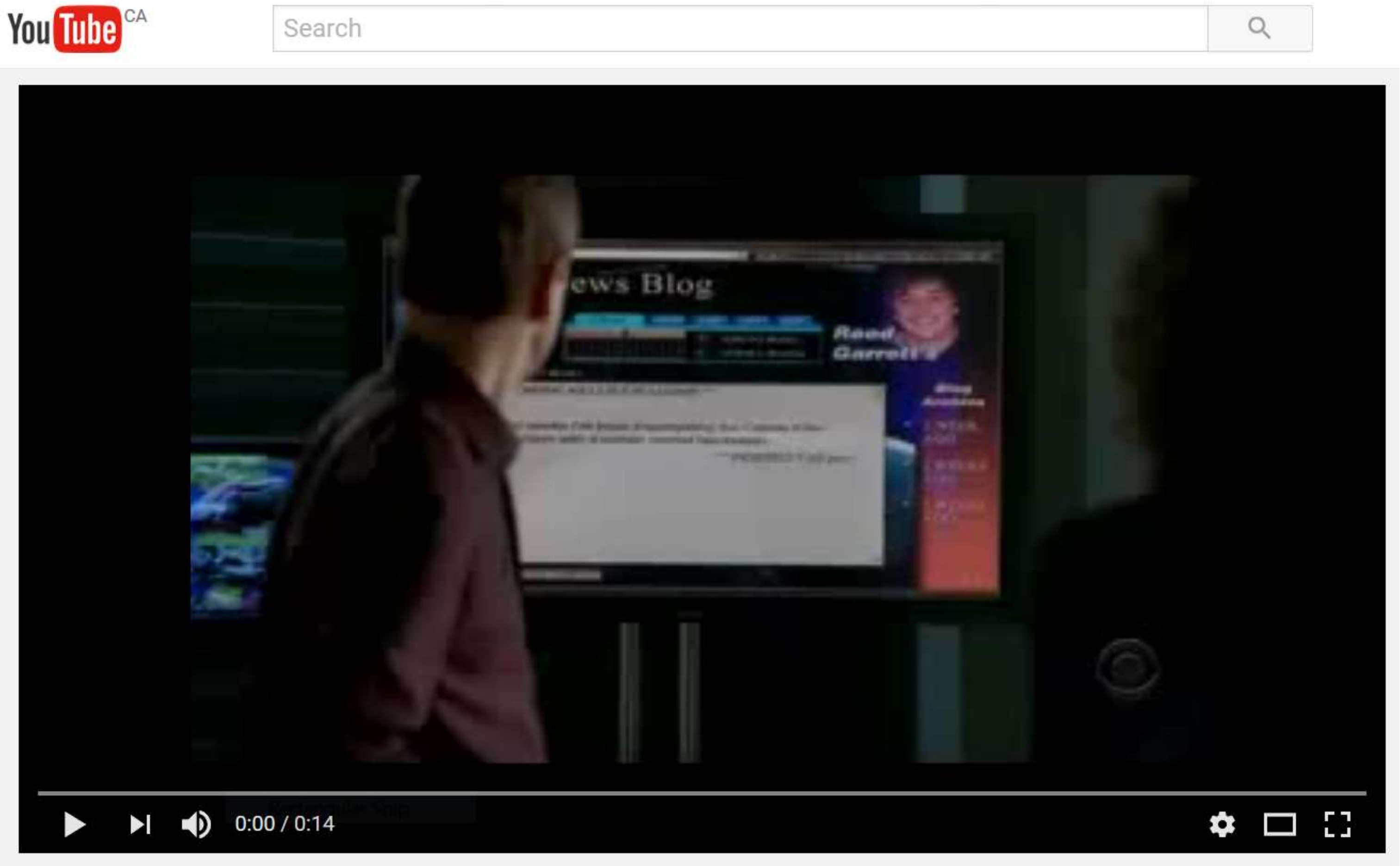
LAYOUT ALGORITHMS

CMPT 381

But first...



Comments from YouTube…

- GUI Interface = Graphical user interface interface
- I'll flash my ROM with avocados, see if I can reconnect a severed artery.
- I'll use GIMP to create a command line interface to backtrace the killer's DNS ping in HTML
- Good for her. She could have just tracked down the IP address herself, but instead she's taking time out of her busy schedule to create a user friendly gui that everyone can use later on

Comments from YouTube…

- I'll install adobe reader and combine it with a winrar license I made in C++ and hook it up to a 3d printer and print out the murder weapon which we can dust for fingerprints
- You do that. I'll open the internet with my remote NES controller and backtrack his algorithms
- Okay I tracked him, he is at 127.0.0.1, wait….. OH MY GOD HE'S IN THE BUILDING!!!!

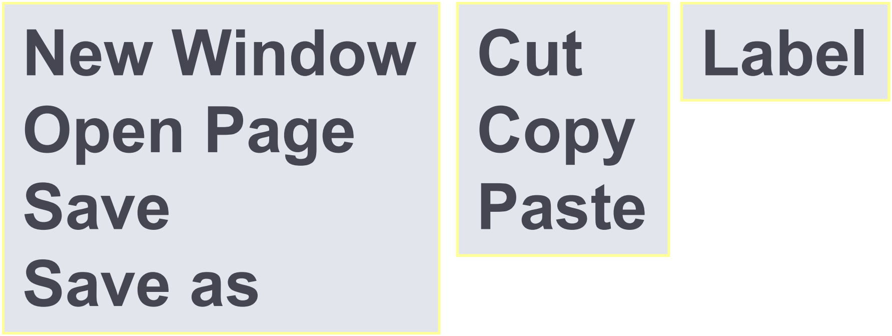
Overview

Variable intrinsic size examples

Row layout algorithm

Variable Intrinsic Size

- Size of widget determined by sizes of items within
  - e.g. Menus, most Java widgets

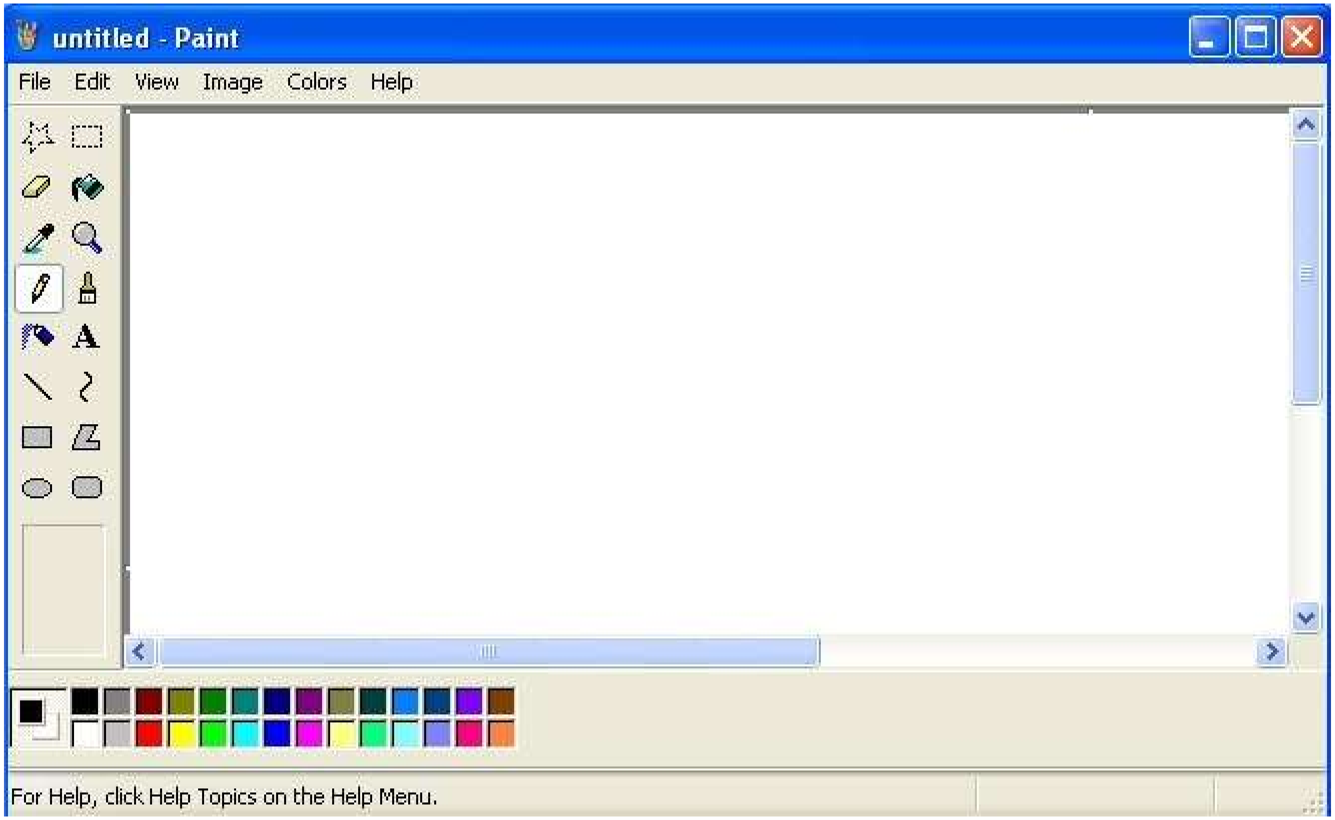


- Intrinsic size does not handle resizing
- Variable Intrinsic Size
  - Each widget reports its size needs (recursively if necessary)
  - Each widget also reports how much it can be reasonably squeezed or expanded

Variable Intrinsic Size: JavaFX Control

- Min size
  - Widget will not shrink below this size
  - setMinSize(), setMinWidth(), setMinHeight()
- Preferred size
  - Used for initial layout
  - setPrefSize(), setPrefWidth(), setPrefHeight()
- Max size
  - Widget will not grow larger than this size
  - setMaxSize(), setMaxWidth(), setMaxHeight()
  - Value of Double.MAX_VALUE means unbounded

Infer the layout / VIS

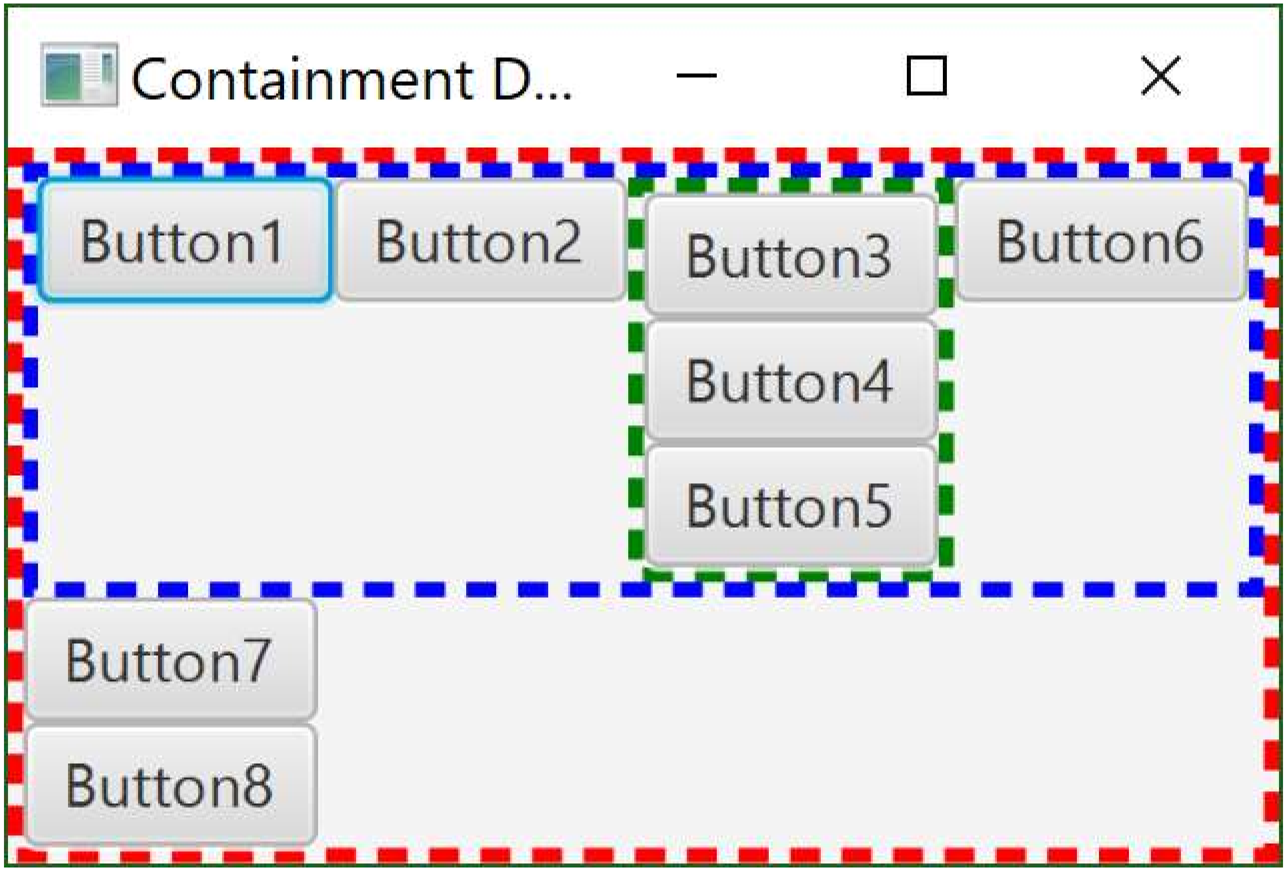


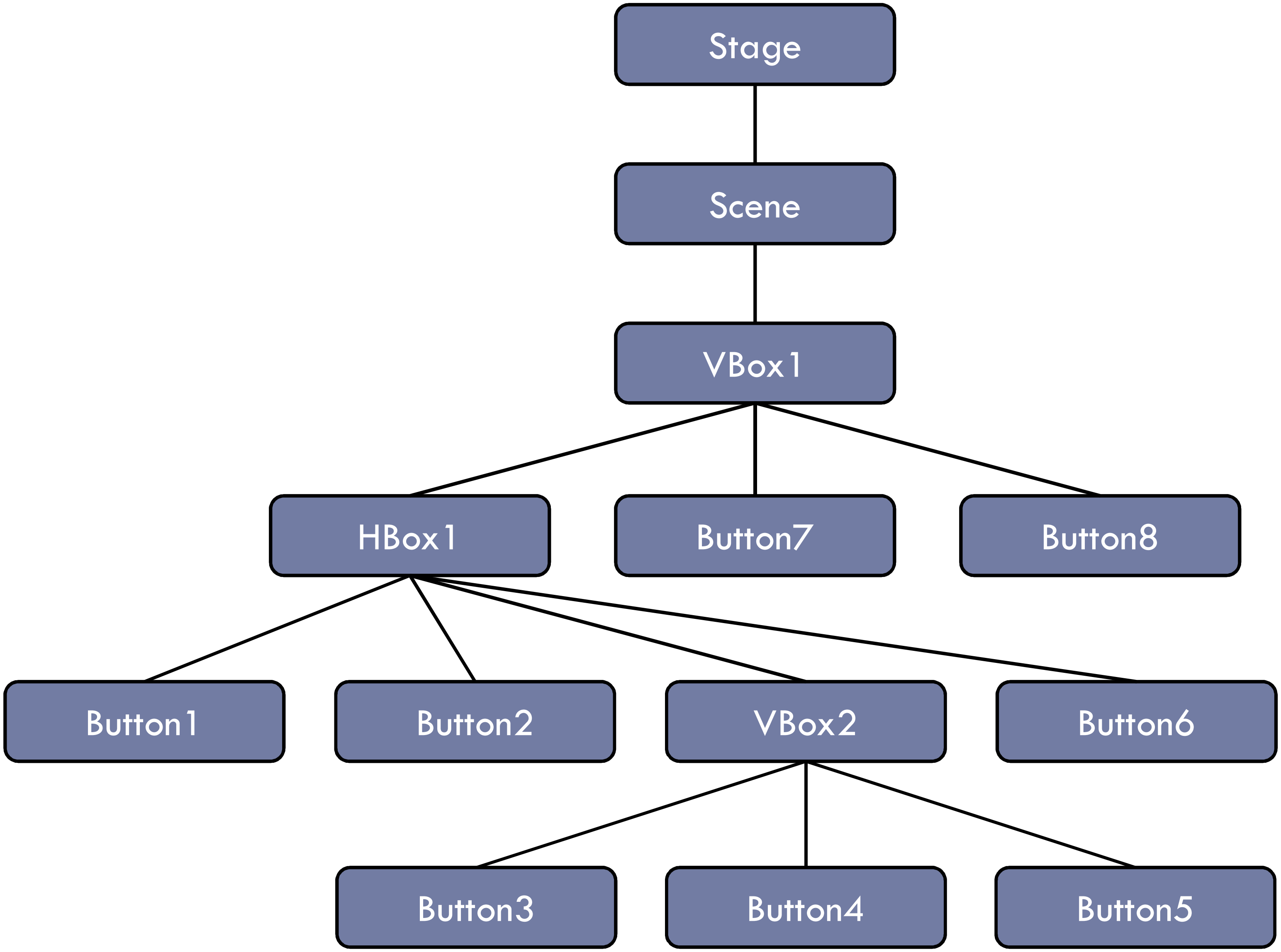
For Help, click Help Topics on the Help Menu.

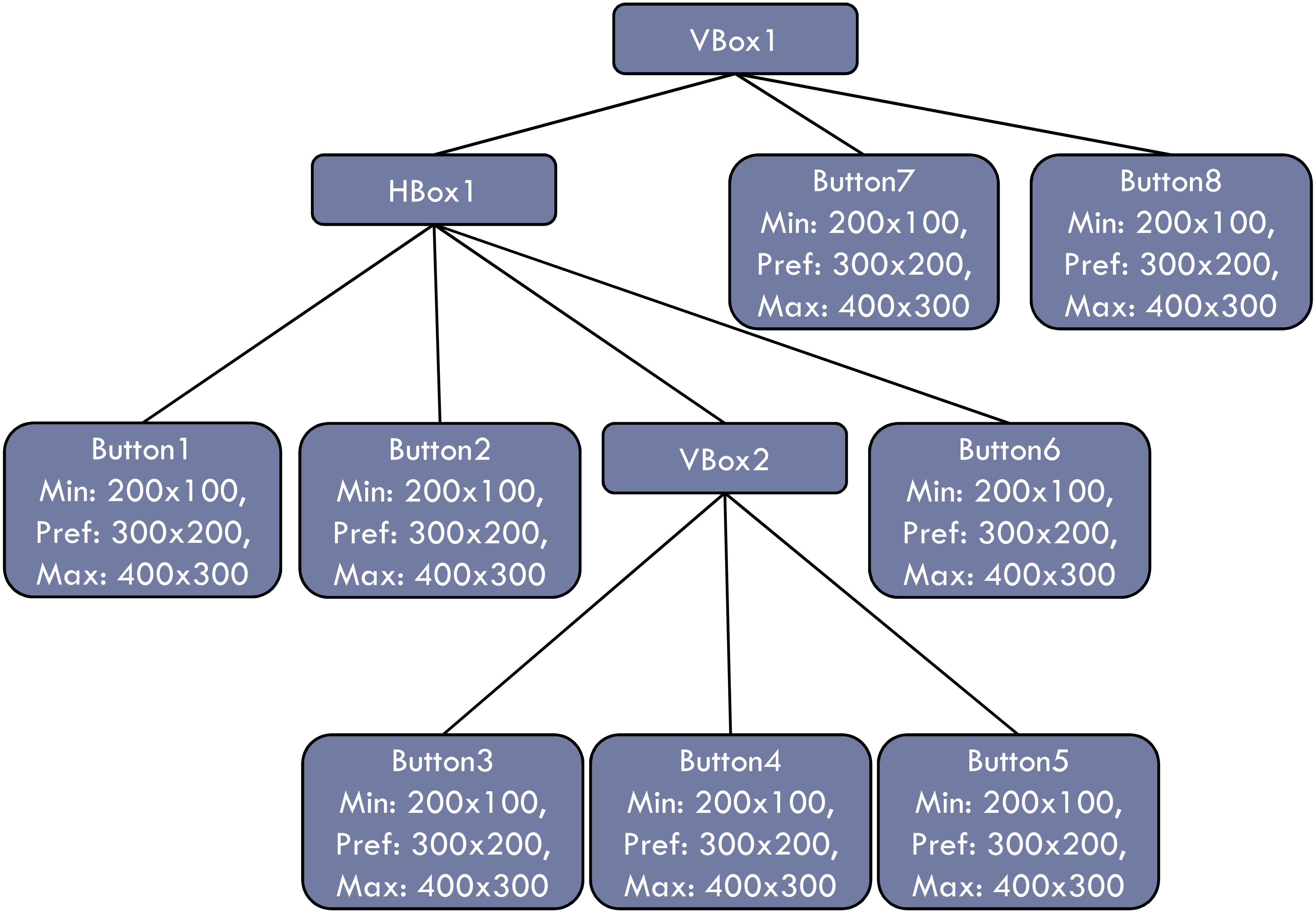
Basic layout algorithm

```
public void doLayout(Rectangle myBounds)
    foreach child C:
        get max / min / pref size of C
    update my max / min / pref size
    foreach child C:
        allocate bounds for C, based on layout
        approach, desired sizes, and myBounds
    foreach child C:
        C.doLayout(new bounds for C)
```

Containment hierarchy




Stage
Scene
VBox1
HBox1
Button7
Button8
Button1
Button2
VBox2
Button6
Button3
Button4
Button5



VBox1

HBox1

Button7
Min: 200x100,
Pref: 300x200,
Max: 400x300

Button8
Min: 200x100,
Pref: 300x200,
Max: 400x300

Button1
Min: 200x100,
Pref: 300x200,
Max: 400x300

Button2
Min: 200x100,
Pref: 300x200,
Max: 400x300

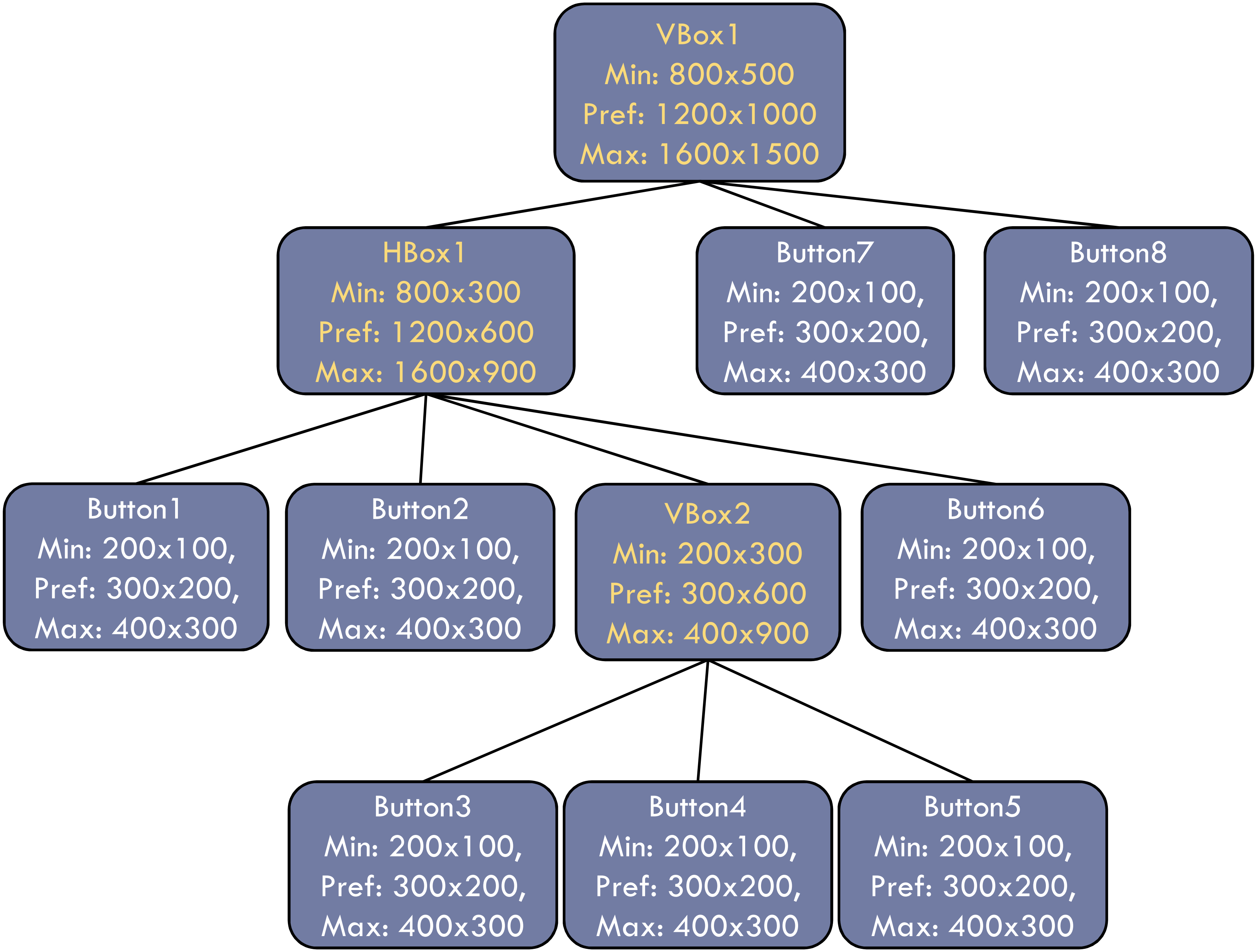
VBox2

Button6
Min: 200x100,
Pref: 300x200,
Max: 400x300

Button3
Min: 200x100,
Pref: 300x200,
Max: 400x300

Button4
Min: 200x100,
Pref: 300x200,
Max: 400x300

Button5
Min: 200x100,
Pref: 300x200,
Max: 400x300



VBox1
Min: 800x500
Pref: 1200x1000
Max: 1600x1500

HBox1
Min: 800x300
Pref: 1200x600
Max: 1600x900

Button7
Min: 200x100,
Pref: 300x200,
Max: 400x300

Button8
Min: 200x100,
Pref: 300x200,
Max: 400x300

Button1
Min: 200x100,
Pref: 300x200,
Max: 400x300

Button2
Min: 200x100,
Pref: 300x200,
Max: 400x300

VBox2
Min: 200x300
Pref: 300x600
Max: 400x900

Button6
Min: 200x100,
Pref: 300x200,
Max: 400x300

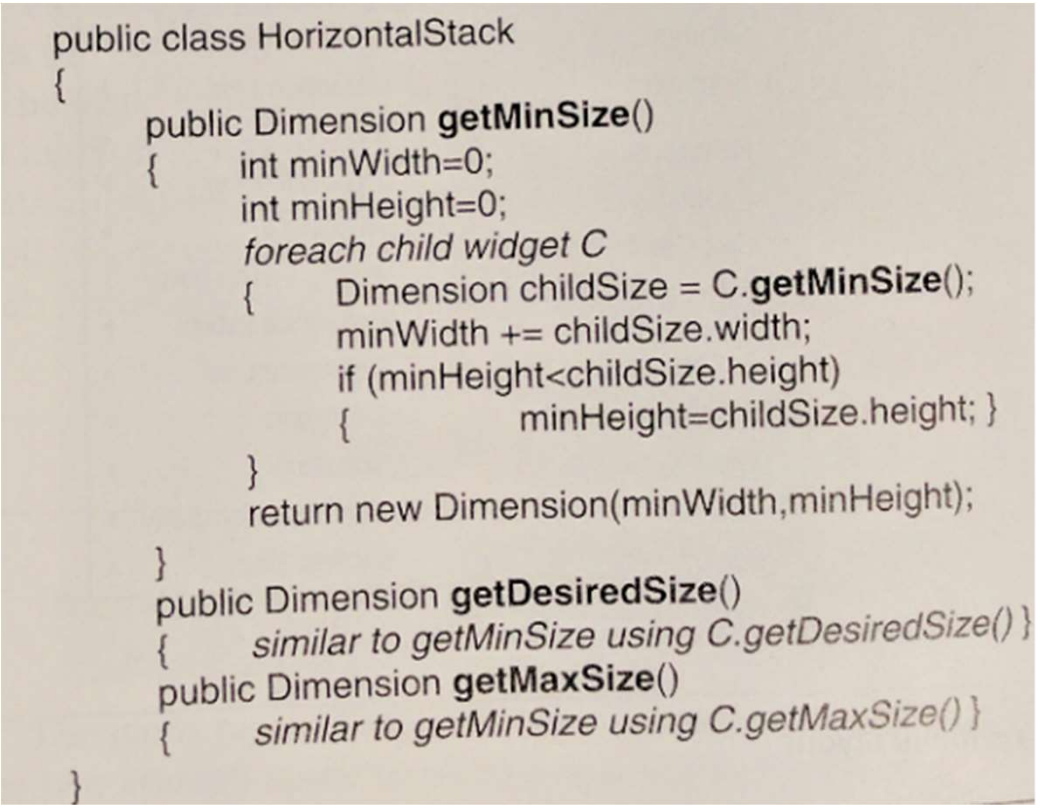
Button3
Min: 200x100,
Pref: 300x200,
Max: 400x300

Button4
Min: 200x100,
Pref: 300x200,
Max: 400x300

Button5
Min: 200x100,
Pref: 300x200,
Max: 400x300

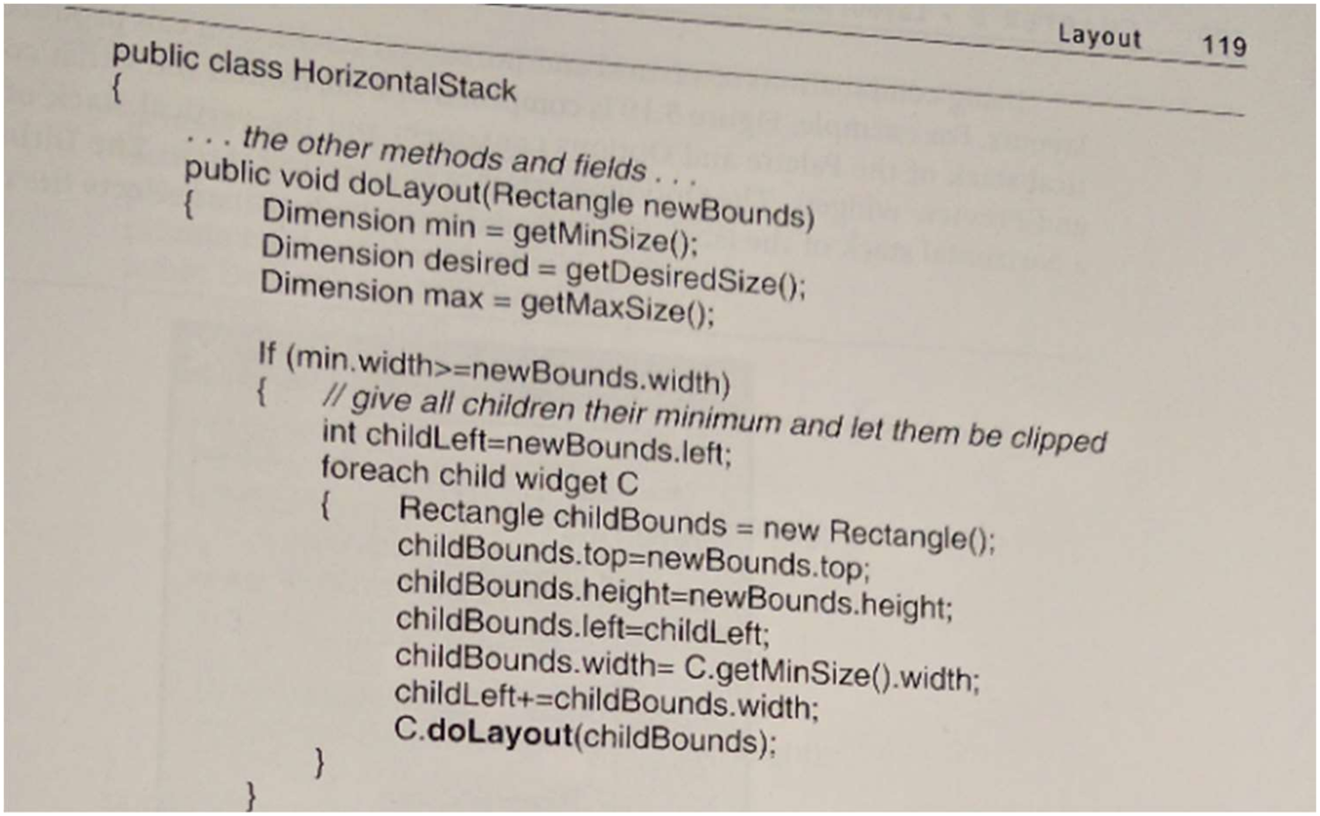
Calculate all VISs

Calculate size – row layout



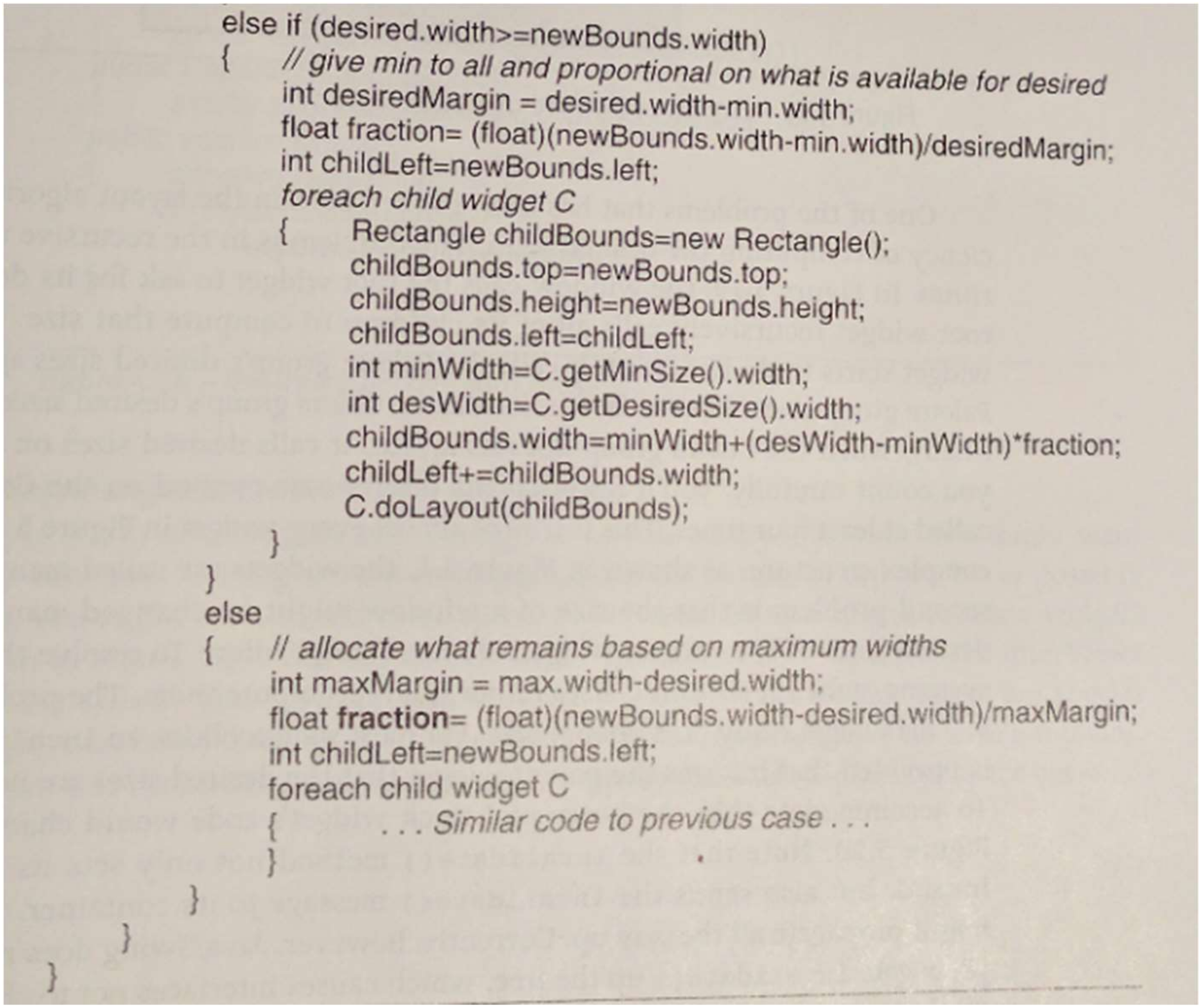
```
public class HorizontalStack
{
        public Dimension getMinSize()
        {       int minWidth=0;
                int minHeight=0;
                foreach child widget C
                {       Dimension childSize = C.getMinSize();
                        minWidth += childSize.width;
                        if (minHeight<childSize.height)
                        {               minHeight=childSize.height; }
                }
                return new Dimension(minWidth,minHeight);
        }
        public Dimension getDesiredSize()
        {       similar to getMinSize using C.getDesiredSize() }
        public Dimension getMaxSize()
        {       similar to getMinSize using C.getMaxSize() }
}
```

Do layout – row layout



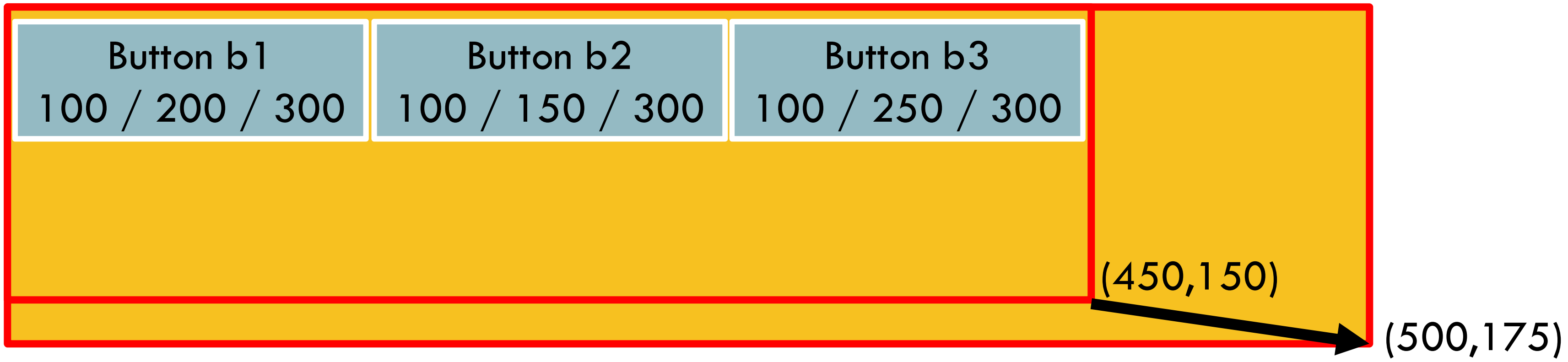
```
public class HorizontalStack
{
        . . . the other methods and fields . . .
        public void doLayout(Rectangle newBounds)
        {       Dimension min = getMinSize();
                Dimension desired = getDesiredSize();
                Dimension max = getMaxSize();

                If (min.width>=newBounds.width)
                {       // give all children their minimum and let them be clipped
                        int childLeft=newBounds.left;
                        foreach child widget C
                        {       Rectangle childBounds = new Rectangle();
                                childBounds.top=newBounds.top;
                                childBounds.height=newBounds.height;
                                childBounds.left=childLeft;
                                childBounds.width= C.getMinSize().width;
                                childLeft+=childBounds.width;
                                C.doLayout(childBounds);
                        }
                }
```

Do layout – row layout



```
            else if (desired.width>=newBounds.width)
            {       // give min to all and proportional on what is available for desired
                    int desiredMargin = desired.width-min.width;
                    float fraction= (float)(newBounds.width-min.width)/desiredMargin;
                    int childLeft=newBounds.left;
                    foreach child widget C
                    {       Rectangle childBounds=new Rectangle();
                            childBounds.top=newBounds.top;
                            childBounds.height=newBounds.height;
                            childBounds.left=childLeft;
                            int minWidth=C.getMinSize().width;
                            int desWidth=C.getDesiredSize().width;
                            childBounds.width=minWidth+(desWidth-minWidth)*fraction;
                            childLeft+=childBounds.width;
                            C.doLayout(childBounds);
                    }
            }
            else
            {       // allocate what remains based on maximum widths
                    int maxMargin = max.width-desired.width;
                    float fraction= (float)(newBounds.width-desired.width)/maxMargin;
                    int childLeft=newBounds.left;
                    foreach child widget C
                    {       . . . Similar code to previous case . . .
                    }
            }
    }
}
```

Example



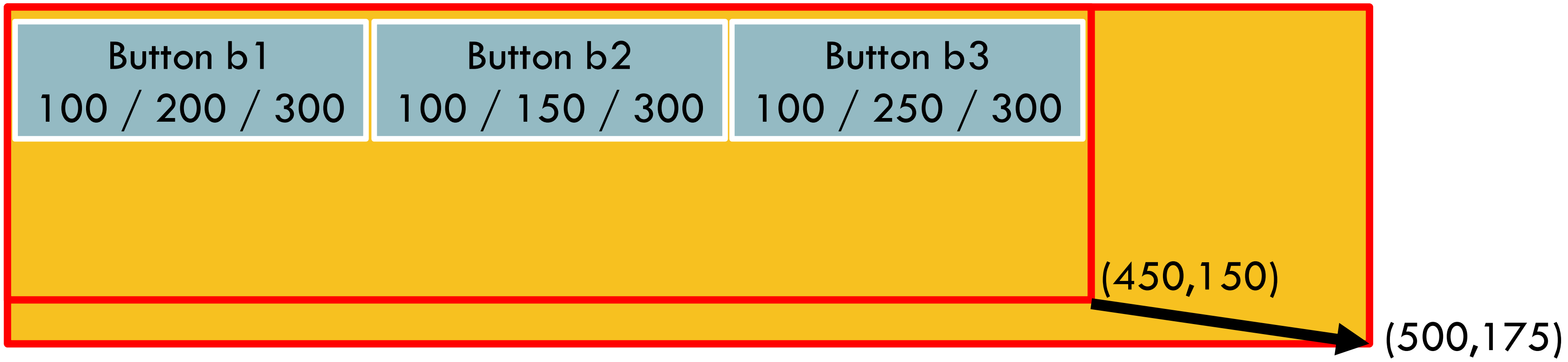
Container:

MinWidth:

PrefWidth:

MaxWidth:

Example

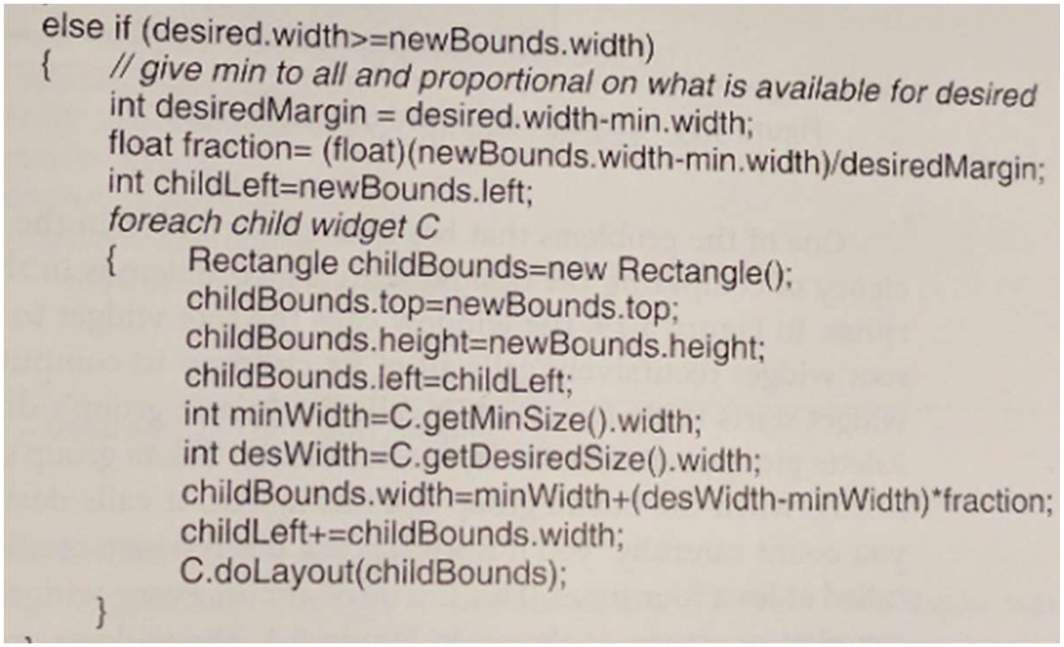


Container:

MinWidth:

PrefWidth:

MaxWidth:



```
else if (desired.width>=newBounds.width)
{       // give min to all and proportional on what is available for desired
        int desiredMargin = desired.width-min.width;
        float fraction= (float)(newBounds.width-min.width)/desiredMargin;
        int childLeft=newBounds.left;
        foreach child widget C
        {       Rectangle childBounds=new Rectangle();
                childBounds.top=newBounds.top;
                childBounds.height=newBounds.height;
                childBounds.left=childLeft;
                int minWidth=C.getMinSize().width;
                int desWidth=C.getDesiredSize().width;
                childBounds.width=minWidth+(desWidth-minWidth)*fraction;
                childLeft+=childBounds.width;
                C.doLayout(childBounds);
        }
```