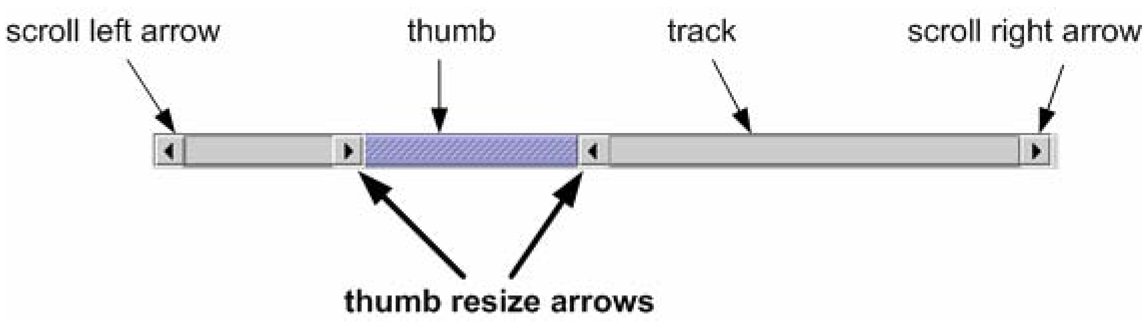
6.831 User Interface Design & Implementation
Fall 2004

PS6: Toolkit Extension

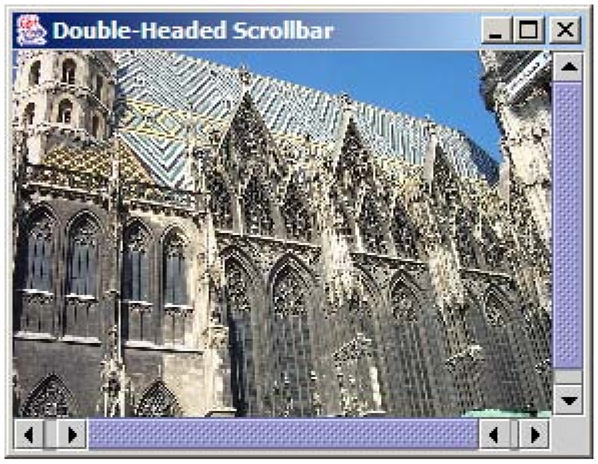
This assignment explores the following topics:

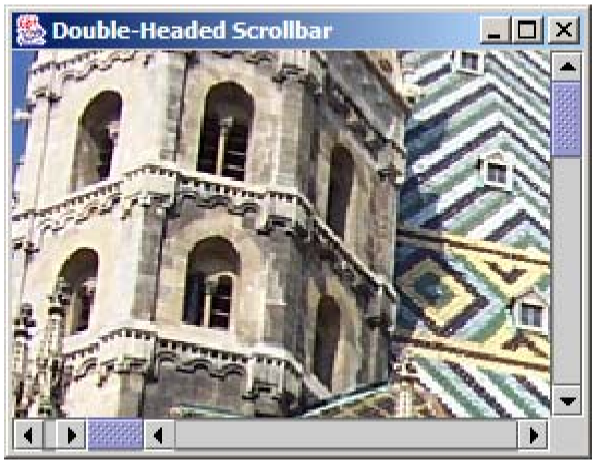
- reusable widgets;
- pluggable look-and-feel;
- extending existing widgets by adding new appearance and new behavior.

In this problem set, you will implement a *double-headed* scrollbar, which allows the user to adjust not only the position of the scroll thumb but also the width of the thumb:



Dragging the arrows at the ends of the scroll thumb resizes the thumb. If the scrollbar is being used to scroll a view, the thumb size maps to the fraction of the view currently visible, so resizing the thumb effectively changes the magnification of the view:





More practically, a double-headed scrollbar can be used to specify a *range query*, such as a range of dates or a range of prices.

Your double-headed scrollbar will be based on JScrollBar. Since Swing uses pluggable look-and-feel, however, it will not be enough to simply extend JScrollBar. JScrollBar knows almost nothing about how to draw itself or handle its input. Those functions are the responsibility of another object, the scrollbar's *look-and-feel delegate*. Most of your work will involve creating a look-and-feel delegate for your double-headed scrollbar, which you will do by extending the standard Java look-and-feel, MetalScrollBarUI.

Provided Resources

We provide the following classes for this assignment:

- DoubleHeadedScrollbar.java: skeleton for the double-headed scrollbar.
- DoubleHeadedScrollbarUI.java: skeleton for the double-headed scrollbar's look-and-feel delegate.
- ImageDisplay.java: a class that tests the double-headed scrollbar by using it to pan and zoom an image.
- Stephansdom.jpg: a sample image used by ImageDisplay

Problem 1: Output (30%)

Initially, the double-headed scrollbar looks and acts no different from a plain JScrollBar. Your first task is to extend its output. Add a button to each end of the scroll thumb, as shown in the screenshot above. The buttons don't have to do anything yet, but they should follow the thumb as the thumb moves. The new buttons should be drawn on top of the thumb, flush against the ends of the thumb.

Use the ImageDisplay class to test your double-headed scrollbar. For simplicity, you only need to implement a **horizontal** scrollbar. ImageDisplay only uses a horizontal double-headed scrollbar. Its vertical scrollbar is just a JScrollBar.

Hints:

- Add the new buttons in DoubleHeadedScrollbar**UI**, not in DoubleHeadedScrollbar.
- You can use MetalScrollButton for the new buttons, so that they look like the buttons at the ends of the track. The constructor for this class is poorly documented, but you can find an example (commented out) in the skeleton of DoubleHeadedScrollbarUI. You can also skip using MetalScrollButton and create your own component, if you prefer.
- If a class isn't sufficiently documented, sometimes the only way to extend it safely is to examine its source code. If you need to, you can look at the source code for any class in Swing by going to your Java installation directory and unpacking src.zip. Relevant classes may be javax.swing.plaf.metal.MetalScrollButton, javax.swing.plaf.metal.MetalScrollBarUI, and javax.swing.JScrollBar.

Problem 2: Input (40%)

Now add input to your double-headed scrollbar. Dragging either of the thumb arrows should resize the thumb **without moving the other end of the thumb**. The other end of the thumb may wiggle back and forth a bit due to rounding errors, but it shouldn't move by more than a pixel. The thumb arrow under the mouse should not make any sudden jumps when the user starts dragging it.

The thumb arrow buttons only need to respond to dragging, not clicking.

There are two boundary cases you need to deal with. The first case is when the user tries to make the thumb too small. The other boundary case is when the user tries to make the thumb too large, filling the entire scrollbar track. When the thumb fills the track, the default MetalScrollBarUI makes the thumb disappear entirely. Resolve these boundary cases as you see fit, following good principles of direct manipulation design. In particular, your scrollbar should not crash, should not enter an unusable state, and should be reversible.

Hints:

- One arrow merely changes the extent of the scrollbar (JScrollbar.setVisibleAmount()), but the other arrow must change both the extent *and* the position (JScrollbar.setValue()). To change the extent and the position simultaneously, use JScrollBar.setValues().
- You can detect whether MetalScrollBarUI is making the thumb invisible, because it calls setThumbBounds() with all zeroes. You can prevent the thumb from disappearing by changing the zeroes to something more sensible, such as the bounds of the track.

Questions (30%)

Answer the following questions in readme.txt.

1. Discuss the advantages and disadvantages of reusing MetalScrollButton for your thumb-resizing controls. Consider both usability and implementation issues.

2. Run ImageDisplay using your platform look-and-feel, instead of the default Java look-and-feel. (ImageDisplay.main() has some commented-out code that does this.) What happens? How would you have to fix your double-headed scrollbar to solve this problem?
3. If you followed our hint, then you used components to implement the new thumb arrows. Suppose you used the stroke model to draw the new arrows instead. Why would it be hard to override the base scrollbar's behavior so you could provide input handling for the arrows?

What to Hand In

Package your completed assignment as a jar file, as described in PS0. Here's a checklist of things you should confirm before you hand in:

☐ all your Java source is included in your jar file (Javadoc documentation isn't necessary)

☐ the main class of your jar file is ImageDisplay

☐ all necessary third-party libraries are included, either inside your jar or as separate jars referenced by your jar's classpath

☐ any resources used by your code (notably the sample image, Stephansdom.jpg) are included in the jar and referenced as resources

☐ readme.txt is included, and it answers the questions above and credits anybody you discussed the assignment with

Before you submit your solution, put all the jar files you plan to submit in an empty directory and make sure you can run it:

```
java -jar yourfile.jar
```