Labs 4 and 5

This lab exercise is lengthy and will take two lab sessions--this week and next week. If you can finish it this week, you do not have to attend lab next week.

In Lecture 11, we saw the adapter pattern. In this lab, you will encounter a situation that calls for an adapter, and take the steps to convert the code to use an adapter. The code is for creating bar graphs of a data array.

Step One

First, create an empty directory for lab4, then download and unpack this zip file into it. This file contains starter code for the lab:

- main.cpp, which contains the program's main routine, which prints two horizontal bar graphs using the HorizBarChart class.
- HorizBarChart.h, which contains the public and private parts of the API for the existing bar chart drawer.
- HorizBarChart.cpp, which contains an implementation of this basic API.
- VectorGraph.h, which contains the API for a class which will draw a vertical bar chart of an vector of data.
- VectorGraph.cpp, which contains the implementation for drawing a vertical bar chart.

Step Two

Compile main.cpp and HorizBarChart.cpp together, and run the resulting program to see how the horizontal bar charts are drawn.

Step Three

The idea here is that we wish to use the VectorGraph class with the code in main.cpp, without changing much about main.cpp and without changing VectorGraph at all.

The first step of doing this is to create a pure virtual interface class that both HorizBarChart and your (as yet unwritten) adapter class will be derived from.

Step Three-A

To create this interface, copy HorizBarChart.h to BarChart.h. Then, in BarChart.h:

- change the class name to BarChart
- remove the constructor. We will use the C++-supplied default constructor for this abstract class.
- remove the private members and private member functions (and the line "private:")
- change all of the public methods (except the constructor) to be pure virtual. This involves inserting the keyword **virtual** before each declaration and the text "= 0" before the semicolon at the end of each declaration. For instance, the line for setN should look like:

```
    virtual void setN(int n) = 0;       // sets the number of data items
```

Step Three-B

Next, change HorizBarChart to be a subclass of BarChart. This means adding *#include "BarChart.h" at the top and *: public BarChart* after the class name in HorizBarChart.h.

```
#include <string>
#include "BarChart.h"
using namespace std;

class HorizBarChart : public BarChart {
    …
};
```

Step Three-C

Finally, change main.cpp to use the new interface.

- change the argument type for makeGraph1 and makeGraph2 to be BarChart& rather than HorizBarChart&
- in the main function, change chart to be a BarChart* rather than a HorizBarChart*. Do not change *new HorizBarChart()* to *new BarChart()* as that would be an error (because BarChart is an abstract class).

Note that now, main.cpp refers to the class HorizBarChart only twice, and in all other places it refers to BarChart. (I'm counting the #include "HorizBarChart.h" line as a reference to the class.)

Step Three-D

After these changes, compile main.cpp and HorizBarChart.cpp again and verify that the program produces the same output as before.

Step Four

Now we construct the adapter. Have a look at the interface for the adaptee, in VectorGraph.h. Only the public part is important. We see that only two member functions are provided: a constructor and the function draw(). The constructor must be provided with the data, column headings, and scale factor. The draw function must be provided with an ostream.

During this step, I will not tell you when you need to put a #include in your adapter. If your code is like mine, there will end up being five #include lines at the top of your file.

Step Four-A

We'll call the adapter VertBarChart. So edit a file called VertBarChart.h, and put a class declaration body for VertBarChart, which should be a subclass of BarChart.

```
class VertBarChart : public BarChart {
    …
};
```

Inside the class body, we will override each of BarChart's member functions.

Step Four-B

Since we cannot create a VectorGraph until we have all of the data, our strategy will be to store the data in VertBarChart as it comes in (during calls to setN, setData, setLabels, and setScale) and then create the VectorGraph when we get a call to draw(). To accomplish this strategy, we will need private member variables for n, the data, the labels, and the scale. Go ahead and create the declarations for these in VertBarChart.h. You should have something like the following now, but your variable names must be different.

```
#include <string>
using namespace std;

class VertBarChart : public BarChart {
    …
private:
    int foo;
    int bar;
    int* bletch;
    string* cow;
}
```

Step Four-C

Now go ahead and declare and implement the overrides of functions setN, setData, setLabels, and setScale. Each of these functions should simply store its argument to the appropriate private variable. For instance, setN might look like (again, change the variable name foo to your private variable name):

```
virtual void setN(int n) {
    foo = n;
}
```

Step Four-D

Now we will create the draw() function. It is declared as it was in BarChart.h but with braces instead of the "= 0;". draw() will create an instance of VectorGraph, but to do that it must have the data and the headers each in a vector. The standard library's vector class has a constructor that takes a begin iterator and an end iterator. It turns out we can use pointers as iterators, so we can create a vector for the data as follows (again, change the variable names to match yours):

```
vector<int> vecData(data, data+num);
```

The addition here is *pointer arithmetic*. data+num means the same as &data[num], which is the address one past the end of the array.

We can similarly create a vector for the labels:

```
vector<string> vecLabels(labels, labels+num);
```

After constructing those vectors, we can now construct the VectorGraph. Here you get to fill in the arguments. See VectorGraph.h if you don't know what arguments to use.

```
VectorGraph graph(…);
```

Finally, we should call graph.draw(), passing in the ostream std::cout as its argument.

Step Four-E

To ensure that you have all the #include directives you need, and that you have no syntax errors, try compiling VertBarChart.h. Once that gives no errors, proceed to Step Five.

Step Five

Now we get to the payoff for all of that work we've done. Edit main.cpp, and change *#include "HorizBarChart.h"* to *#include "VertBarChart.h"*. Next change the first line of main() to create a VertBarChart rather than a HorizBarChart. Those are all the changes you need to make to change the code to print vertical bar charts.

Now compile main.cpp, VertBarChart.h, and VectorGraph.cpp together. Run the resulting program. If it now outputs the two vertical bar charts, then show your result to the TA so they can give you your marks.

Step Six (Optional -- if you have time)

Step Six-A

HorizBarChart prints a blank line before the bar chart, and two blank lines after. VectorGraph prints no blank lines before or after. Change VertBarChart so that it prints blank lines like HorizBarChart does. Do not change VectorGraph.

Step Six-B

Having two different implementations of the same interface (BarChart) means that you can mix-and-match the implementations even within one running program. Change main so that it prints Graph1 horizontally, then Graph1 again vertically, then Graph2 horizontally, then Graph2 vertically.

Updated Fri June 14 2019, 23:18 by shermer.