CS 116 - Lab 9

Henry Hsu

May 2, 2012

1 Specification

Create an abstract class Chart that maintains a collection of integer data values stored in a vector. Include a member function draw

```
(Point& p)
```

as a pure virtual member function to display the data at the given point. Create derived classes PieChart and BarChart that each display the data in the indicated format.

2 Design

Per specification, the PieChart and BarChart classes will be derived from the abstract Chart class. 4 values of 10 percent, 20 percent, 30 percent, and 40 percent will be hard coded in as the input value. These values will be stored in a vector, which will then be used as a parameter to instantiate the piechart and barchart objects. Because of the hard coded nature of the project, no user interaction is needed.

Below is code for the Chart abstract class which will serve as the base class for the other derived classes.

```
                                Chart abstract class
.....................................................................................
 class Chart      // abstract class, cannot be instantiated due to pure virtual function
 {
   public:
     Chart(vector <int> chart_data):data(chart_data){}
     virtual void draw(Point& p) = 0;  // pure virtual member function

   protected:
     vector <int> data;
 };
```

Next is the code for my PieChart derived class. the only function is has aside from the constructor, is the draw function which actually provides the implementation detail as opposed to the pure virtual member function version of it from the base class.

PieChart derived class

```
class PieChart:public Chart
{
  public:
    PieChart(vector <int> pie_data):Chart(pie_data){}
    virtual void draw(Point& p)
    {
      double x = p.get_x();
      double y = p.get_y();

      Point center (x, y);

      cwin << Circle(p, 4);

      Point rim1(x - 4, y);
      Point rim2(x - 3, y + 2.65);
      Point rim3(x + 1, y + 3.85);
      Point rim4(x + 3, y - 2.65);

      Line slice1(center, rim1);
      Line slice2(center, rim2);
      Line slice3(center, rim3);
      Line slice4(center, rim4);

      Point msg1(x - 3, y + 1.25);
      Point msg2(x - 1, y + 2.75);
      Point msg3(x + 2, y + 0.5);
      Point msg4(x - 1, y - 2);

      cwin << slice1 << slice2 << slice3 << slice4;
      cwin << Message (msg1, "10%");
      cwin << Message (msg2, "20%");
      cwin << Message (msg3, "30%");
      cwin << Message (msg4, "40%");
    }
};
```

Similar to the PieChart class outlined above; the BarChart derived class shown below is derived from the same Chart base class. The only difference is that the draw member function displays the data in barchart format as opposed to a piechart.

BarChart derived class

```
class BarChart:public Chart
{
  public:
    BarChart(vector <int> bar_data):Chart(bar_data){}

    virtual void draw(Point& p)
    {
      double x = p.get_x();
      double y = p.get_y();
      int bar_num = data.size();

      for (int i = 0; i < bar_num; i++)
      {
        Point ll((x - 0.25) + (1.5*i), y);
        Point lr((x + 0.25) + (1.5*i), y);
        Point ul((x - 0.25) + (1.5*i), y + (data[i]/10));
        Point ur((x + 0.25) + (1.5*i), y + (data[i]/10));
        Point msg((x - 0.25) + (1.5*i), y - 0.25);

        Line left(ll, ul);
        Line right(lr, ur);
        Line top(ul, ur);
        Line bottom(ll, lr);

        cwin << left << right << top << bottom ;
        cwin << Message (msg, data[i]);
      }
    }
};
```

Now that all the classes are defined, all that remains is the main function. In here, I create a vector to hold integer typed data, then fill it with 4 predetermined values.

Next, I create two points where I will display the piechart and barchart objects. Following that I create the two actual piechart and barchart objects using the filled vector as a parameter to the constructor function.

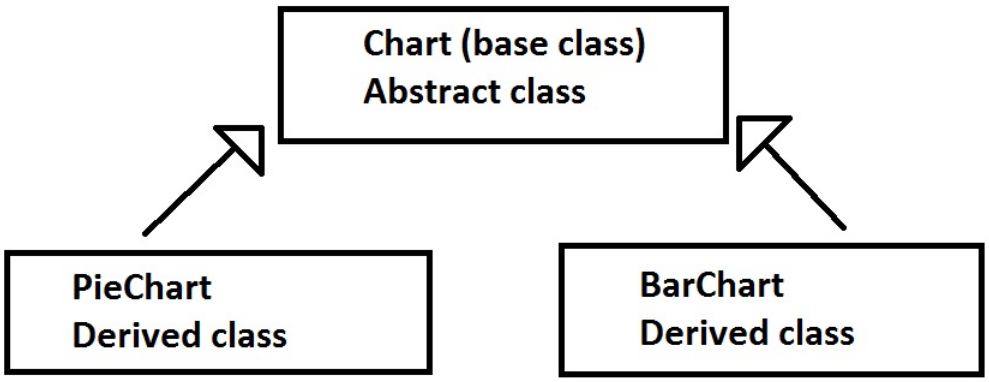
Once the chart objects are successfully created, I call upon each objects' respective draw function to display them.

Main function

```
int ccc_win_main()
{
  vector <int> chart_data;

  chart_data.push_back(10);
  chart_data.push_back(20);
  chart_data.push_back(30);
  chart_data.push_back(40);

  Point p_center (-5, 3), b_center (2, -6);
  PieChart Pie(chart_data);
  BarChart Bar(chart_data);

  Pie.draw(p_center);
  Bar.draw(b_center);
}
```

This lab was very helpful in demonstrating the effect of including a pure virtual member function in a class; that is, turning it into an abstract class which cannot be instantiated.

Per project requirement, here is a UML diagram for my project.



3 screenshot

Here is a screenshot of my program in operation: