The Strategy Pattern

The **Strategy Design Pattern** defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithms vary independently from the clients that use it.

Toni Sellarès
Universitat de Girona

Design Principles

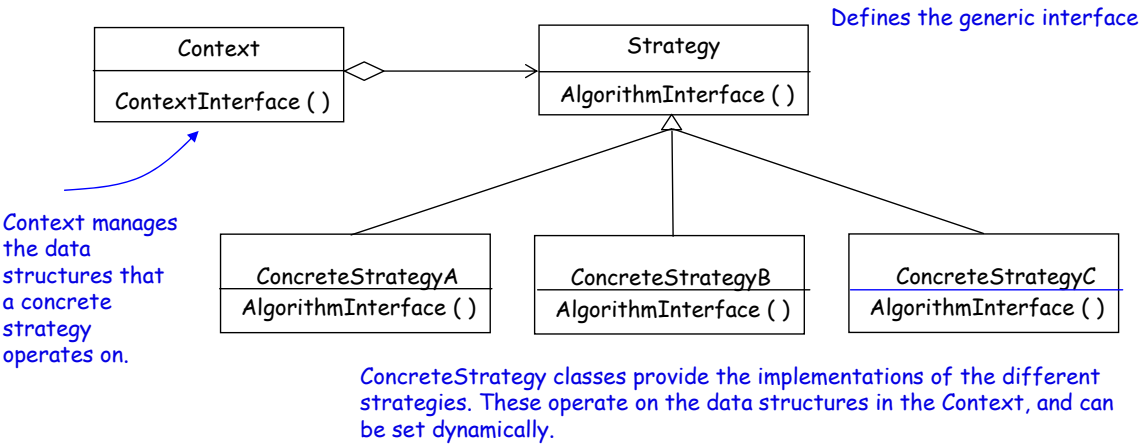
Identify the aspects of your application that vary and separate them from what stays the same.

Program to an interface, not to an implementation.

Favor composition over inheritance.

The Strategy Design Pattern

The **Strategy Design Pattern** defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithms vary independently from the clients that use it.



Strategy: Structural Example

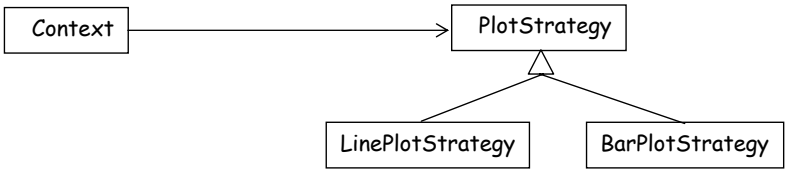
```
public interface Strategy
{
           void algorithmInterface();
}

public class ConcreteStrategyA implements Strategy
{
           public void algorithmInterface()
           {
         System.out.println("Called
                           ConcreteStrategyA.algorithmInterface()");
           }
}

public class ConcreteStrategyB implements Strategy
{
           public void algorithmInterface()
           {
             System.out.println("Called
                           ConcreteStrategyB.algorithmInterface()");

           }
}
```

```
public class ConcreteStrategyC implements Strategy
{
          public void algorithmInterface()
          {
         System.out.println("Called
                        ConcreteStrategyC.algorithmInterface()");

          }
}


class Context
{
  Strategy strategy;

  public Context( Strategy strategy )
  {
    this.strategy = strategy;
  }

  public void contextInterface()
  {
    strategy.algorithmInterface();
  }
}

public class ClientTest
{
  public static void Main( string[] args )
  {
    Context c = new Context( new ConcreteStrategyA() );
    c.ContextInterface();

    Context d = new Context( new ConcreteStrategyB() );
    d.ContextInterface();

    Context e = new Context( new ConcreteStrategyC() );
     e.ContextInterface();
  }
}
```

Example

- Consider a simplified graphing program that can present data as a line graph or a bar chart.
  - 3 classes : the generic PlotStrategy
  - LinePlotStrategy
  - BarPlotStrategy

- Each plot will appear in its own frame -- so PlotStrategy is derived from JFrame



Code to sink your teeth in….

```
public abstract class PlotStrategy extends JFrame {
        protected float[ ]              x, y;
        protected float                 minX, minY, maxX, maxY;
        protected int                   width, height;
        protected Color                 color;

        public PlotStrategy (String title) {
            super (title);
            width =  300;
            height = 200;
            color = Color.black;
            addWindowListener(new WindApp(this));
        }

        public abstract void plot (float xp[], float yp[]);

        public void setSize (Dimension sz) {
               width = sz.width;
               height = sz.height;
        }
        public void setPenColor (Color c) {
               color = c;
        }
}

public class WindAp extends WindowAdapter {
        Jframe fr;
        public WindApp (Jframe f) {
            fr = f;
        }
        public void windowClosing (WindowEvent e) {
             fr.setVisible (false);
        }
}
```

Plot is the only method that the subclasses need to implement.

```
public class Context {
      private PlotStrategy plotStrategy;
      private float   x[], y[];

      public Context ( ){
          setLinePlot ( );
      }
      public void setBarPlot ( ){
          plotStrategy = new BarPlotStrategy ( );
      }
      public void setLinePlot ( ){
          plotStrategy = new LinePlotStrategy ( );
      }
      public void plot ( ) {
          plotStrategy.plot (x, y);
      }
      public void setPenColor (Color c) {
          plotStrategy.setPenColor ( c );
      }
      public void readData (String filename) {
      }
```

The Context class is the traffic cop that decides which strategy must be invoked based on a request from the client program.
-- all the Context does is sets one concrete strategy versus the other.

The Context class is also responsible for reading in the data.

- The Client program…

The Client is a panel with two buttons Bar graph and Line graph that invoke the two different plots.

```
public class JGraphButton extends JButton implements Command {
    Context context;
    public JGraphButton (ActionListener act, Context ctx) {
        super (“Line graph” );
        addActionListener (act);
        context = ctx;
    }
    public void Execute ( ){
        context.setPenColor (Color.red);
        context.setLinePlot ();
        context.readData (“data.txt”);
        context.plot ( );
    }
}
```

This class gives the implementation of the Line Graph Button class.
-- sets the correct strategy and then calls the Context's plot method.
What design pattern is being used here?

Set the Pen color. Set the kind of plot, read the data and then plot the data.

The Two Strategy Classes

```
public class LinePlotStrategy extends PlotStrategy {
    private LinePlotPanel lp;

    public LinePlotStrategy ( ) {
        super(“Line Plot”);
        lp = new LinePlotPanel ( );
        getContentPane().add(lp);
    }
    public void plot (float x[], float y[]){
        this .x = x; this.y = y;
        findBounds ( );
        setSize (width, height);
        setVisible (true);
        setBackground (Color.white);
        lp.setBounds ( minX, minY, maxX, maxY);
        lp.plot (x, y, color);
        repaint ( );
    }
}
```

The two Strategy classes are pretty much the same. They set up the window size for plotting and call a plot method specific for that display panel.

Copy the data to internal variables, set the max and min values. Set up the plot data and call the paint method to plot.

The PlotPanel Classes

```
public class PlotPanel extends Jpanel {
      private float xfactor, yfactor;
      private in xpmin, ypmin, xpmax, ypmax;
      private float minX, maxX, minY, maxY;
      private float x[], y[];
      private Color color;

      public void setBounds (float minx, float miny, float maxx, float maxy){
            minX = minx; minY = miny;
            maxX = maxx; maxY = maxy;
      }
      public void plot ( float[] xp, float [] yp, Color c){
            x = xp;
            y = yp;
            color = c;
            int  w = getWidth ( ) - getInsets().left - getInsets().right;
            int h = getHeight() - getInsets().top - getInsets().bottom;
            xfactor = (0.9f * w)  / (maxX - minX);
            yfactor = (0.9f * h)  /  (maxY - minY);
            xpmin = (int) (0.05f*w);
            ypmin = (int) (0.05f*h);
            xpmax = w - xpmin;
            ypmax =  h - ypmin;
            repaint ( );
      }
}
```

The base class - PlotPanel - contains the common code for scaling the data to the window.

The LinePlotPanel Class

```
public class LinePlotPanel extend PlotPanel {
      public void paint (Graphics g) {
            super.paint ( g);
            int xp = calcx (x[0]);
            int yp = calcy (y[0]);
            g.setColor(color.white);
            g.fillRect ( 0,0, getWidth(), getHeight( ) );
            g.setColor(Color.black);

            g.drawRect (xpmin, ypmin, xpmax, ypmax);
            g.setColor (color);

            for (int j = 1; j < x.length; j++ ) {
                  int xp1 = calcx (x[ j ]);
                  int yp1 = calcy (y[ j ]);
                  g.drawLine (xp, yp, xp1, yp1);
                  xp = xp1;
                  yp = yp1;
            }
      }
}
```

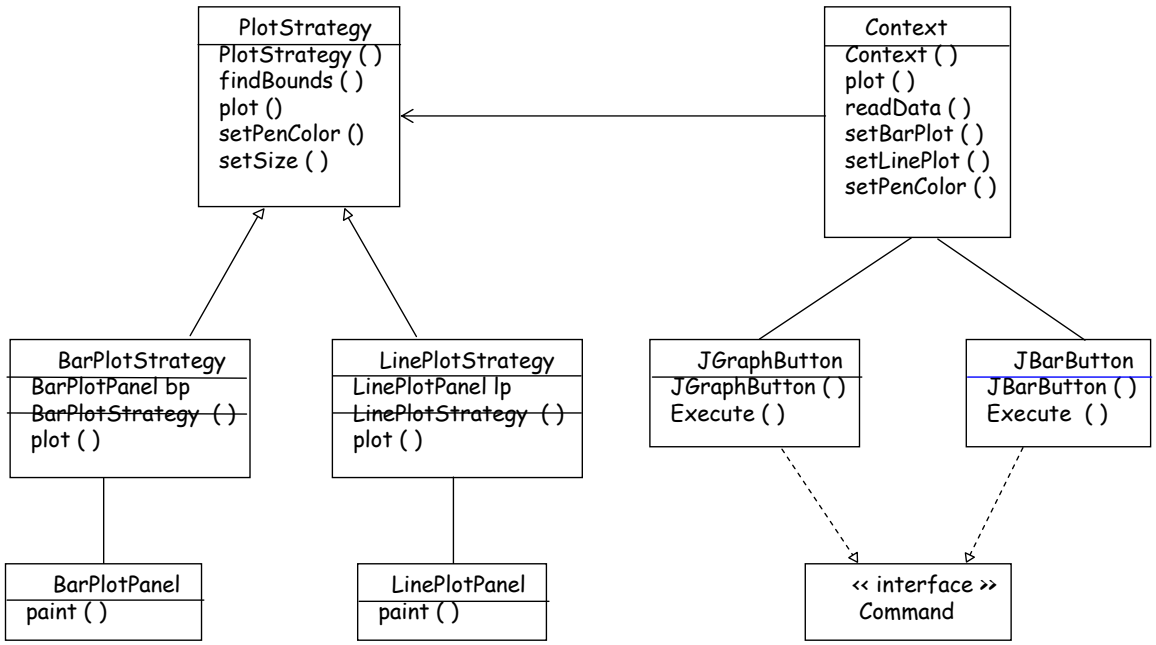
The LinePlotPanel class implements the paint method for its Line graphics. The BarPlotPanel does its specific bar plot graphics.

Getting the first points.

Flood the background.

Plot the graph.

Class Diagram



Summary

- Strategy pattern allows selection of one of several algorithms dynamically.
- Algorithms may be related via an inheritance hierarchy or unrelated [must implement the same interface]
- Strategies don't hide everything -- client code is typically aware that there are a number of strategies and has some criteria to choose among them -- shifts the algorithm decision to the client.

The Abstract Factory Method is a specialization of the Strategy Method!