13 Chapter

Theme Mapping and Analysis

Thematic mapping is a powerful way to analyze and visualize your data. You give graphic form to your data so that you can see it on a map. Patterns and trends that are almost impossible to detect in lists of data reveal themselves clearly when you use thematic shading to display data on a map.

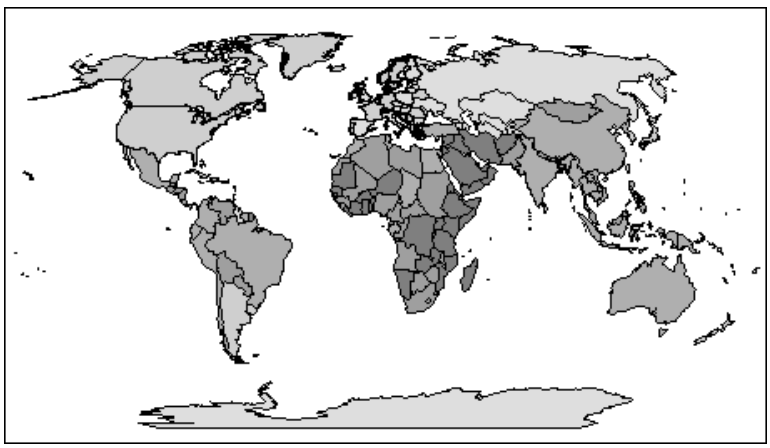
With MapXtreme, you can create applications with thematic maps.

What Is Thematic Mapping?

Thematic mapping is the process of shading your map according to a particular theme. The theme is usually some piece or pieces of your data.You thematically shade a map using data from a layer. The most commonly known example of a thematic map is a weather map. When you see red, you know it is hot (high number of degrees); where you see blue, it is cold (low number of degrees).

Themes represent your data with shades of color, fill patterns, or symbols. There are many uses for thematic maps to display your data. You create different thematic maps by assigning these colors, patterns, or symbols to map objects according to specific *values* in your data.

Thematic mapping allows you to visualize and highlight trends in data that would be difficult to see through tabular data. Using the methods in the ThemeList and the Theme Interface, you can create and define your own thematic shading.



Theme Types

MapXtreme Java Edition offers four types of thematic maps:

- **OverrideTheme** – changes the rendition of an entire layer
- **RangedTheme** – groups data into ranges and shading based on range value
- **IndividualValueTheme** – shades groups of features which share a specific attribute value
- **SelectionTheme** – applies a rendition to a user-defined list of selected features

IDSelectionTheme, a type of theme available in previous releases, has been deprecated. It has been replaced by IndividualValueTheme and SelectionTheme.

All layers that have thematics use the ThemeList collection, the Theme interface, and the Rendition object. Each specific theme type may use additional objects. They are explained in detail later in this chapter.

Themes can be created programmatically or, in the case of RangedTheme and IndividualValueThemes, through the AddTheme Wizard Bean. Themes can now be displayed and controlled through the Layer Control.

Theme Legends

MapXtreme Java now can generate legends for Ranged and IndividualValue themes. The information in the legend is tied directly to the theme. When the theme changes, the legend is updated. More on theme legends is presented later in this chapter.

AddTheme Wizard

MapXtreme Java provides a wizard to assist you when creating a RangedTheme or IndividualValueTheme. This wizard is a JavaBean that you can add to the MapToolbar for easy access. See AddTheme Wizard on page 216.

General Objects for Themes

ThemeList

The ThemeList collection is accessible from the Layer object and contains Theme objects. The ThemeList collection has methods used to perform operations such as adding, removing, and reordering Theme object(s) from the collection.

The ThemeList allows several thematic shades to exist for one layer. It is important to keep the themes ordered correctly so that they display on the map. Themes are rendered in reverse order, like layers, from the bottom up. For example, if you had two themes that both changed the fill color of objects, one would obscure the other when the map is rendered. The theme at the top of the ThemeList takes precedence.

The ThemeList works well for certain situations. For example, if you had a map of the world and wanted to shade the countries by population and also by literacy rate, the ThemeList allows this. You could first create a theme to shade by population using a different fill color for the regions and add that to the ThemeList. You could then create

a theme to shade by literacy using a hatch pattern for the regions and add that to the ThemeList.

ThemeList objects are used even if you only have one theme.

Theme

Theme is an interface. The methods of this interface allow you to retrieve the column on which the theme is based and the theme's descriptive name. If either the column or the name do not exist for a particular theme, a null value will be returned.

Rendition

The Rendition object is used throughout MapXtreme. It gives all of the style characteristics to features. When creating thematic maps, you will use Renditions to specify the appearance of the objects.

The following sections describe the four types of themes.

OverrideTheme

An OverrideTheme can be used to change the rendition of an entire layer. For example, if you wanted the world table to display with a red fill pattern and green line color, you would use an OverrideTheme.

To make an OverrideTheme for a layer, you only need to pass a Rendition object in its constructor.

```
// Assume myLayer is a Layer object.

// Assume myRend is a Rendition object.


OverrideTheme myOTheme = new OverrideTheme(myRend,"My
   Theme");

myLayer.getThemeList.add(myOTheme);
```

RangedTheme

A RangedTheme is a more complex type of thematic map. When you create a ranged thematic map, all features are grouped into ranges and each assigned a rendition for its corresponding range. For example, you have a table of weather stations for your television viewing area, and you want to shade the locations according to their reported snowfall amounts.

With the Ranged map feature, MapXtreme groups the snowfall amounts into ranges. For instance, all weather stations that received between zero and five inches of snowfall in the past month are grouped into one range. Stations receiving between five and 10 inches are in a separate range. Sites that received between 10 and 15 inches are in a third range, while those stations reporting greater than 15 inch snowfall amounts are in a fourth range. Each range is referred to as a Bin. Each Bin has an upper-bound cut-off value.

All records in the layer are assigned to a range and then assigned a rendition based on that range. For instance the weather stations reporting 15 plus inches of snow ar shaded red. The other ranges are shaded in lighter shades of red with the last range in gray (default colors). When you display the map, the colors make it readily apparent which locations received the most and least snow accumulation.

MapXtreme includes several utility objects that help create a RangedTheme.

ColumnStatistics

A ColumnStatistics object is returned when you use the **fetchColumnStatistics** method of the Layer object. The ColumnStatistics object contains information on the minimum, maximum, mean, and standard deviation of the values in a column. When you use the **fetchColumnStatistics** method, you pass the column on which you want the map shaded. You will not need to use the methods of the ColumnStatistics object directly to create a RangedTheme. Once the object has been retrieved, it is used in the Bucketer object to create a vector of breakpoints.

Bucketer

The Bucketer class is responsible for calculating the breakpoints for the Bins in a RangedTheme. Continuing the snowfall example above, you have four ranges that represent weather stations receiving 0-5 inches, 5-10 inches, 10-15 inches, and 15 inches or more of annual snowfall. Each of these ranges is a Bin.

The Bucketer calculates the breakpoints of these Bins using the **computeDistribution** methods. These methods all return a vector of breakpoints. Each value in the vector is an Attribute object. All of the **computeDistribution** methods pass the number of ranges and a ColumnStatistics object. You may also pass a Distribution Type and a RoundOff object.

Distribution Types

DISTRIBUTION_TYPE_EQUAL_COUNT has approximately the same number of records in each range. If you want the Bucketer to group 100 records into 4 ranges using equal count, it computes the ranges so that approximately 25 records fall into each range, depending on the rounding factor you set.

When using equal count (or any other range method), it's important to watch out for any extreme data values that might affect your thematic map (in statistics, these values are referred to as *outliers*). For example, if you shade according to equal count with this database:

John	5000	Andrea	7000
Penny	6000	Kyle	5500
Miguel	4500	Angela	7500
Ben	100	Mark	7000

Ben and Miguel are grouped in the same range (since they have the two lowest values). This may not produce the results you want since the value for Ben is so much lower than any of the other values.

DISTRIBUTION_TYPE_EQUAL_RANGES divides records across ranges of equal size. For example, you have a field in your table with data values ranging from 1 to 100. You want to create a thematic map with four equal size ranges. The Bucketer produces ranges 1–25, 26–50, 51–75, and 76–100.

Keep in mind that the Bucketer may create ranges with no data records, depending on the distribution of your data. For example, if you shade the following database according to Equal Ranges:

John	100	Andrea	90
Penny	6	Kyle	1
Miguel	4	Angela	92
Linda	95	Elroy	89
Ben	10	Mark	10

The Bucketer creates four ranges (1–25, 26–50, 51–75, and 76–100). Notice, however, that only two of those ranges (1–25 and 76–100) actually contain records.

DISTRIBUTION_TYPE_STANDARD_DEVIATION breaks at the middle range of the mean of your values, and the ranges above and below the middle range are one standard deviation above or below the mean.

RoundOff

The RoundOff object is use to create clean breakpoints for ranges. For example, if you were shading a map with values that ranged from 101 to 397, the range breaks would be cleaner if the range was 100 to 400. RoundOff can round down the lower end of your range, and round up the higher end of your range.

LinearRenditionSpreader

An important part of creating a useful thematic map is to represent the values with renditions that gradually go from one value to another. The example in the introduction to RangedTheme discussed shading snowfall amounts. One end of the values was represented with red, and the next range was a lighter red, and so forth. The **spread** method of the LinearRenditionSpreader will return a vector of Renditions that spread the style from one given rendition to another for the number of elements given. The number of elements should match the number of ranges passed to the Bucketer object. For example, if you passed a rendition that was a red fill, a rendition that was a white fill, and the number five, the LinearRenditionSpreader would create a vector of five renditions with the red fill at the beginning, the white fill at the end, and an even spread of fill types in between.

Creating a RangedTheme

The following example demonstrates the code for creating a RangedTheme:

```
Layer lyr=null;
   Rendition yellow=new Rendition(), red=new Rendition();
   lyr = m_map.getLayers().getLayer("States.tab");

   String colName = "Pop_1990";
   ColumnStatistics colStats =
   lyr.fetchColumnStatistics(colName);

   // Set number of breaks for data
   int numBreaks=5;

   // Compute the distribution of data with 5 breaks and
   // Equal Ranges
   Vector rBreaks = Bucketer.computeDistribution
   (numBreaks,colStats,
   Bucketer.DISTRIBUTION_TYPE_EQUAL_RANGES);

   // Set up a red and a yellow rendition and then
   // spread the colors
   yellow.setValue(Rendition.FILL, Color.yellow);
   yellow.setValue(Rendition.STROKE_WIDTH, 2);

   red.setValue(Rendition.FILL, Color.red);
   red.setValue(Rendition.STROKE_WIDTH, 4);

   Vector rends = LinearRenditionSpreader.spread
   (numBreaks, yellow, red);

   // Create Theme object
   RangedTheme rTheme = new RangedTheme
   (colName, rBreaks, rends, "States by Pop_1990");

   // Get ThemeList class object
   ThemeList tList=lyr.getThemeList();

   // Add theme to Layers themeList
   tList.add(rTheme);
```

A Ranged theme can also be constructed through the AddTheme Wizard. Associated theme legends can be created, as well. Both are discussed later in this chapter.

SelectionTheme

A SelectionTheme applies a rendition to all features referenced in a selection object. This type of theme is commonly used to store features returned by a search method on a layer using add(FeatureSet fs). SelectionTheme replaces the deprecated IDSelectionTheme.

For example, given a Layer object and X and Y coordinates, the code below demonstrates the selection of the layer's feature(s) at the specified location, and the creation of a SelectionTheme to display the selected features in red.

```
void selectFeatureAtPoint(Layer layer, double x, double
   y) {
  Vector v = new Vector();
  DoublePoint dp = new DoublePoint(x, y);
  FeatureSet fs = null;
  try {
    // Select a feature at the specified location
    fs = layer.searchAtPoint(v, dp, null);
    // Create a SelectionTheme
    SelectionTheme selTheme = new
   SelectionTheme("PointSelection");
    // Create a Selection object, and add the selected
   features
    Selection sel = new Selection();
    sel.add(fs);
    // Assign the Selection object to the SelectionTheme
    selTheme.setSelection(sel);
    // Assign the display style of the SelectionTheme
    Rendition rend = new Rendition();
    rend.setValue(Rendition.FILL, Color.red);
    selTheme.setRendition(rend);
    // Add the SelectionTheme to the layer's list of
   themes
    layer.getThemeList().add(selTheme);
  }
  catch (Exception e) {
  }
}
```

IndividualValueTheme

This type of theme allows the thematic shading of features based on specific attribute values for a specified column. For example, use Individual Value theme to modify the renditions of region features in a "Coverage" table based on the values in the "Territories" column.Territories with type values of "SouthWest" could be red, while values of "SouthEast" could be shaded blue.

To create an IndividualValueTheme, follow this code sample:

```
lyr = lyrs.add(dpr, ttdh, "Territories");

   IndividualValueTheme iValThm = new
   IndividualValueTheme("CoverageTerritory");
   Rendition rend=new Rendition();

   rend.setValue(Rendition.FILL, Color.red);
   iValThm.add(new Attribute("SouthWest"), rend);

   rend.setValue(Rendition.FILL, Color.blue);
   iValThm.add(new Attribute("SouthEast"), rend);

   rend.setValue(Rendition.FILL, Color.green);
   iValThm.add(new Attribute("Central"), rend);

   lyr.getThemeList().add(iValThm);
```

An IndividualValueTheme can also be constructed through the AddThemeWizard. Associated theme legends can be created, as well. See AddTheme Wizard and Theme Legends sections, below.

The IndividualValueTheme and the SelectionTheme replace the deprecated IDSelectionTheme.

Theme Legends

You can create a legend for your Ranged or Individual Value theme based on the data. You have a lot of control over how the legend will look. You can change the title, fonts, and insets, as well as modify the descriptive text and colors.

Legends can be used on the client or server to export the legend as an image (e.g.,GIF, JPEG). Theme legends are Swing components and can be used with MapXtreme's JavaBeans.

To enable theme legends, the Theme interface has been enhanced to support a theme's Legend object. Every Theme will have an associated ThemeLegend object, which will initially be null. Only RangedThemes and IndividualValueThemes have legends that contain meaningful information. (The other theme types, OverrideThemes and SelectionThemes, return empty legends.) To set the legend associated with a Theme, the Theme object's **setLegend(ThemeLegend)** method must be invoked. A ThemeLegend can be obtained by creating one explicitly, or via the Theme's **createDefaultLegend()** method.

The following example demonstrates the code for creating a RangedThemeLegend:

```
// Create Theme object
rTheme = new RangedTheme(colName, rBreaks, rends, "States
   by Pop_1990");
// Create a default legend
   RangedThemeLegend rThmLeg =
   rTheme.createDefaultLegend(null);
// OR, Create a theme legend instance using theme and
   setting hashtable
   // Add theme settings to hashtable
      Hashtable ht = new Hashtable();
      ht.put("geomtype",
   RangedThemeLegend.REGION_GEOMETRY);
      ht.put("lableorder",
   RangedThemeLegend.ORDER_ASCENDING);
      RangedThemeLegend rThmLeg = new
   RangedThemeLegend(rTheme, ht);
// Set legend title
rThmLeg.setTitle("Ranged Theme legend");
// send legend to image file
rThmLeg.toFile("c:\\temp\\rangeLeg.gif", "image/gif");
```

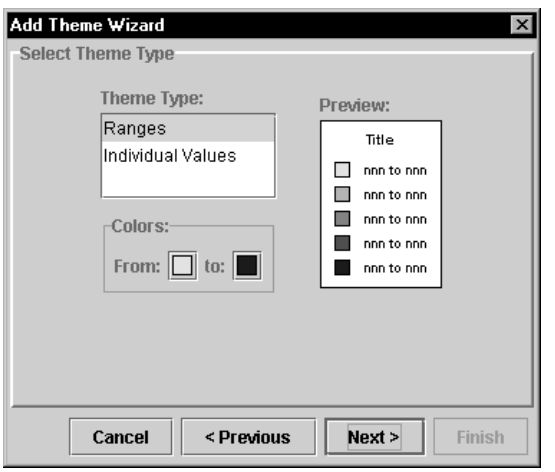
The LegendContainerBean manages how legends are laid out and displayed. It can be dropped in along-side a VisualMapJ object and display any legends that are added to the VisualMapJ maps. VisualMapJ notifies the LegendContainerBean when a theme changes and the LegendContainerBean updates its display accordingly.

The LegendContainerBean is demonstrated in the sample applet, SimpleMap. See Chapter 6: MapXtreme JavaBeans for more information.

AddTheme Wizard Bean

The AddTheme Wizard Bean is a guided tool to allow you to easily add a RangedTheme or IndividualValueTheme to your map. The theme can be based on any supported column and layer in the current map. Currently there is support for creating a theme based on numeric, string and date column data, and on point, line and region layers. Part of the operation will be to create a default theme legend which is associated with the new theme.

Through the Wizard you can choose a name for the theme. For RangedThemes, you can also choose the number of ranges (bins), distribution method (equal count, equal ranges, etc.), and the value that all break-points (bin upper-bound values) should be rounded to. You can also specify the rendition (style) of the points, lines, or regions that will be thematically shaded by setting the rendition for the first and last ranges. Appropriate renditions will then be computed for the ranges in between.



For IndividualValueThemes, you can choose which values in a layer should be given a distinct shading to distinguish them from other values.

At run-time the AddThemeWizard Bean hooks up to an existing VisualMapJ instance. In order for this to work, VisualMapJ must be a child of the component that the AddTheme Bean was added to. For example, if the AddTheme Bean is added to a JPanel to which a VisualMapJ instance was already added, then the VisualMapJ instance will be found and hooked up. If the automated hookup cannot occur, then you must use the **setVisualMapJ**(VisualMapJ) method of the AddTheme Bean to seed the Bean with a working instance of VisualMapJ.

The AddThemeWizard Bean extends AbstractAction, so it can be added to a JMenu or JToolbar. This keeps the menu and toolbar in synch. When the menu item or toolbar button is clicked, the AddTheme Wizard displays.

The AddTheme Bean is demonstrated in the sample applet, SimpleMap. See page 83 for more information.