```
1  // Fig. 20.1: TemperatureServer.java
2  // TemperatureServer interface definition
3  import java.rmi.*;
4
5  public interface TemperatureServer extends Remote {
6     public WeatherInfo[] getWeatherInfo()
7        throws RemoteException;
8  }
```

Fig. 20.1 `TemperatureServer` interface.

```
1  // Fig. 20.2: TemperatureServerImpl.java
2  // TemperatureServerImpl definition
3  import java.rmi.*;
4  import java.rmi.server.*;
5  import java.util.*;
6  import java.io.*;
7  import java.net.*;
8
9  public class TemperatureServerImpl extends UnicastRemoteObject
10                                 implements TemperatureServer {
11    private WeatherInfo weatherInformation[];
12
13    public TemperatureServerImpl() throws RemoteException
14    {
15       super();
16       updateWeatherConditions();
17    }
18
19    // get weather information from NWS
20    private void updateWeatherConditions()
21       throws RemoteException
22    {
23       try {
24          System.err.println(
25             "Updating weather information..." );
26
27          // Traveler's Forecast Web Page
28          URL url = new URL(
29             "http://iwin.nws.noaa.gov/iwin/us/traveler.html" );
30
31          BufferedReader in =
32             new BufferedReader(
33                new InputStreamReader( url.openStream() ) );
34
35          String separator = "</PRE><HR> <BR><PRE>";
36
37          // locate first horizontal line on Web page
38          while ( !in.readLine().startsWith( separator ) )
39             ;    // do nothing
40
41          // s1 is the day format and s2 is the night format
42          String s1 =
43             "CITY            WEA     HI/LO   WEA     HI/LO";
44          String s2 =
45             "CITY            WEA     LO/HI   WEA     LO/HI";
46          String inputLine = "";
47
48          // locate header that begins weather information
49          do {
50             inputLine = in.readLine();
51          } while ( !inputLine.equals( s1 ) &&
52                    !inputLine.equals( s2 ) );
53
```

Fig. 20.2 Class **`TemperatureServerImpl`** (part 1 of 3).

```
54          Vector cityVector = new Vector();
55
56          inputLine = in.readLine();  // get first city's info
57
58          while ( !inputLine.equals( "" ) ) {
59             // create WeatherInfo object for city
60             WeatherInfo w = new WeatherInfo(
61                inputLine.substring( 0, 16 ),
62                inputLine.substring( 16, 22 ),
63                inputLine.substring( 23, 29 ) );
64
65             cityVector.addElement( w ); // add to Vector
66             inputLine = in.readLine();  // get next city's info
67          }
68
69          // create array to return to client
70          weatherInformation =
71             new WeatherInfo[ cityVector.size() ];
72
73          for ( int i = 0; i < weatherInformation.length; i++ )
74             weatherInformation[ i ] =
75                ( WeatherInfo ) cityVector.elementAt( i );
76
77          System.err.println( "Finished Processing Data." );
78          in.close();  // close connection to NWS server
79       }
80       catch( java.net.ConnectException ce ) {
81          System.err.println( "Connection failed." );
82          System.exit( 1 );
83       }
84       catch( Exception e ) {
85          e.printStackTrace();
86          System.exit( 1 );
87       }
88    }
89
90    // implementation for TemperatureServer interface method
91    public WeatherInfo[] getWeatherInfo()
92    {
93       return weatherInformation;
94    }
95
96    public static void main( String args[] ) throws Exception
97    {
98       System.err.println(
99          "Initializing server: please wait." );
100
101      // create server object
102      TemperatureServerImpl temp =
103         new TemperatureServerImpl();
104
105      // bind TemperatureServerImpl object to the rmiregistry
106      String serverObjectName = "//localhost/TempServer";
```

Fig. 20.2 Class **`TemperatureServerImpl`** (part 2 of 3).

```
107         Naming.rebind( serverObjectName, temp );
108         System.err.println(
109            "The Temperature Server is up and running." );
110      }
111   }
```

Fig. 20.2 Class **`TemperatureServerImpl`** (part 3 of 3).

```
1  // Fig. 20.3: WeatherInfo.java
2  // WeatherInfo class definition
3  import java.rmi.*;
4  import java.io.Serializable;
5
6  public class WeatherInfo implements Serializable {
7     private String cityName;
8     private String temperature;
9     private String description;
10
11    public WeatherInfo( String city, String desc, String temp )
12    {
13       cityName = city;
14       temperature = temp;
15       description = desc;
16    }
17
18    public String getCityName() { return cityName; }
19
20    public String getTemperature() { return temperature; }
21
22    public String getDescription() { return description; }
23 }
```

Fig. 20.3 **`WeatherInfo`** class definition.

```
1  // Fig. 20.4: TemperatureClient.java
2  // TemperatureClient definition
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import java.rmi.*;
7
8  public class TemperatureClient extends JFrame
9  {
10    public TemperatureClient( String ip )
11    {
12       super( "RMI TemperatureClient..." );
13       getRemoteTemp( ip );
14
15       setSize( 625, 567 );
16       setResizable( false );
17       show();
18    }
19
20    // obtain weather information from TemperatureServerImpl
21    // remote object
22    private void getRemoteTemp( String ip )
23    {
24       try {
25          // name of remote server object bound to rmi registry
26          String serverObjectName = "//" + ip + "/TempServer";
27
28          // lookup TemperatureServerImpl remote object
29          // in rmiregistry
30          TemperatureServer mytemp = ( TemperatureServer )
31             Naming.lookup( serverObjectName );
32
33          // get weather information from server
34          WeatherInfo weatherInfo[] = mytemp.getWeatherInfo();
35          WeatherItem w[] =
36             new WeatherItem[ weatherInfo.length ];
37          ImageIcon headerImage =
38             new ImageIcon( "images/header.jpg" );
39
40          JPanel p = new JPanel();
41
```

Fig. 20.4 **`TemperatureClient`** class definition (part 1 of 2).

```
42          // determine number of rows for the GridLayout;
43          // add 3 to accommodate the two header JLabels
44          // and balance the columns
45          p.setLayout(
46             new GridLayout( ( w.length + 3 ) / 2, 2 ) );
47          p.add( new JLabel( headerImage ) ); // header 1
48          p.add( new JLabel( headerImage ) ); // header 2
49
50          for ( int i = 0; i < w.length; i++ ) {
51             w[ i ] = new WeatherItem( weatherInfo[ i ] );
52             p.add( w[ i ] );
53          }
54
55          getContentPane().add( new JScrollPane( p ),
56                                BorderLayout.CENTER );
57       }
58       catch ( java.rmi.ConnectException ce ) {
59          System.err.println( "Connection to server failed. " +
60             "Server may be temporarily unavailable." );
61       }
62       catch ( Exception e ) {
63          e.printStackTrace();
64          System.exit( 1 );
65       }
66    }
67
68    public static void main( String args[] )
69    {
70       TemperatureClient gt = null;
71
72       // if no sever IP address or host name specified,
73       // use "localhost"; otherwise use specified host
74       if ( args.length == 0 )
75          gt = new TemperatureClient( "localhost" );
76       else
77          gt = new TemperatureClient( args[ 0 ] );
78
79       gt.addWindowListener(
80          new WindowAdapter() {
81             public void windowClosing( WindowEvent e )
82             {
83                System.exit( 0 );
84             }
85          }
86       );
87    }
88 }
```

Fig. 20.4 **`TemperatureClient`** class definition (part 2 of 2).

```
1  // Fig. 20.5: WeatherItem.java
2  // WeatherItem definition
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class WeatherItem extends JLabel {
7     private static ImageIcon weatherImages[], backgroundImage;
8     private final static String weatherConditions[] =
9        { "SUNNY", "PTCLDY", "CLOUDY", "MOCLDY", "TSTRMS",
10         "RAIN", "SNOW", "VRYHOT", "FAIR", "RNSNOW",
11         "SHWRS", "WINDY", "NOINFO", "MISG" };
12    private final static String weatherImageNames[] =
13       { "sunny", "pcloudy", "mcloudy", "mcloudy", "rain",
14         "rain", "snow", "vryhot", "fair", "rnsnow",
15         "showers", "windy", "noinfo", "noinfo" };
16
17    // static initializer block to load weather images
18    static {
19       backgroundImage = new ImageIcon( "images/back.jpg" );
20       weatherImages =
21          new ImageIcon[ weatherImageNames.length ];
22
23       for ( int i = 0; i < weatherImageNames.length; ++i )
24          weatherImages[ i ] = new ImageIcon(
25             "images/" + weatherImageNames[ i ] + ".jpg" );
26    }
27
28    // instance variables
29    private ImageIcon weather;
30    private WeatherInfo weatherInfo;
31
32    public WeatherItem( WeatherInfo w )
33    {
34       weather = null;
35       weatherInfo = w;
36
37       // locate image for city's weather condition
38       for ( int i = 0; i < weatherConditions.length; ++i )
39          if ( weatherConditions[ i ].equals(
40               weatherInfo.getDescription().trim() ) ) {
41             weather = weatherImages[ i ];
42             break;
43          }
44
45       // pick the "no info" image if either there is no
46       // weather info or no image for the current
47       // weather condition
48       if ( weather == null ) {
49          weather = weatherImages[ weatherImages.length - 1 ];
50          System.err.println( "No info for: " +
51                              weatherInfo.getDescription() );
52       }
53    }
```

Fig. 20.5 **`WeatherItem`** class definition (part 1 of 2).

```
54
55    public void paintComponent( Graphics g )
56    {
57       super.paintComponent( g );
58       backgroundImage.paintIcon( this, g, 0, 0 );
59
60       Font f = new Font( "SansSerif", Font.BOLD, 12 );
61       g.setFont( f );
62       g.setColor( Color.white );
63       g.drawString( weatherInfo.getCityName(), 10, 19 );
64       g.drawString( weatherInfo.getTemperature(), 130, 19 );
65
66       weather.paintIcon( this, g, 253, 1 );
67    }
68
69    // make WeatherItem's preferred size the width and height of
70    // the background image
71    public Dimension getPreferredSize()
72    {
73       return new Dimension( backgroundImage.getIconWidth(),
74                             backgroundImage.getIconHeight() );
75    }
76 }
```

Fig. 20.5 **`WeatherItem`** class definition (part 2 of 2).

Fig. 20.6 The **`rmiregistry`** running.

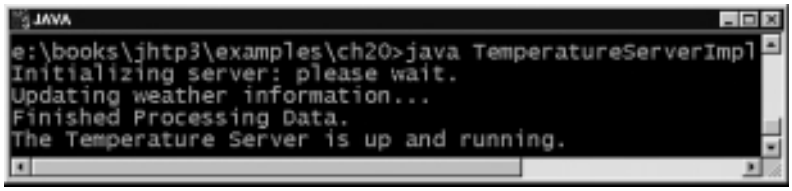

Fig. 20.7 The **`TemperatureServerImpl`** remote object executing.

Fig. 20.8 **`TemperatureClient`** running.