KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN

UNIVERSITAS BRAWIJAYA

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

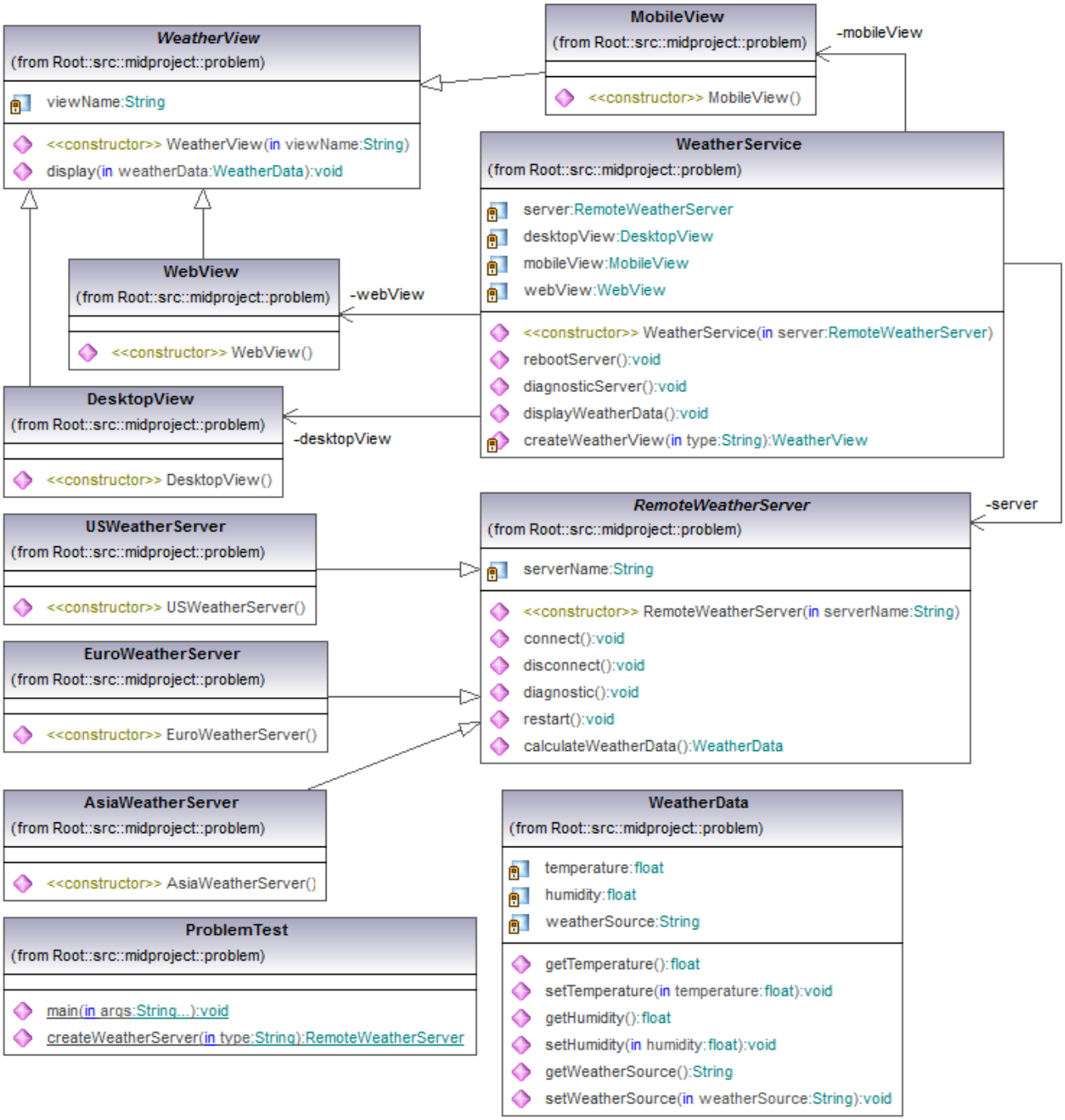
UJIAN TENGAH SEMESTER GENAP TAHUN AKADEMIK 2011 / 2012

Mata Kuliah	:	Pola-Pola Perancangan	**Hari, tanggal**	:	Selasa, 10 April 2012
Dosen	:	Eriq Muhammad Adams J., ST., M.Kom	**Waktu**	:	10.00 – 12.00
			Sifat ujian / Jumlah Soal	:	Tutup Buku / 1

Weather Station, Inc Problem :

A weather company named Weather Station, Inc have remote weather servers located in US, Europe, and Asia which provide weather data *(please read : WheaterData, RemoteWheaterServer class and its sub classes)*. Wheater Service (*please read : WheaterService class*) can perform remote operation such reboot, diagnostic, and get wheather data from remote weather servers and display to different platforms : web, mobile, desktop platform *(please read : WheaterView class and its sub classes)*. **Please improve following code and class diagram using three appropriate design patterns !**

Wheater Station, Inc 's Class Diagram



Wheater Station, Inc 's Code

RemoteWheaterServer.java

```
public abstract class RemoteWeatherServer {
  private String serverName;
  public RemoteWeatherServer(String serverName) { this.serverName = serverName;}
  public void connect() { System.out.println("connect to " + serverName);}
  public void disconnect() { System.out.println("disconnect from " + serverName);}
  public void diagnostic() { System.out.println("run diagnostic on " + serverName);}
  public void restart() { System.out.println("restarting " + serverName);}
  public WeatherData calculateWeatherData() {
      System.out.println(serverName + " calculating weather data ...");
      WeatherData weatherData = new WeatherData();
      weatherData.setWeatherSource(serverName);
      weatherData.setHumidity((float) Math.random() * 100f);
      weatherData.setTemperature((float) Math.random() * 50f);
      return weatherData;
  }
}
```

AsiaWheaterServer.java

```
public class AsiaWeatherServer extends RemoteWeatherServer {
  public AsiaWeatherServer() { super("Asia Weather Server");}
}
```

USWheaterServer.java

```
public class USWeatherServer extends RemoteWeatherServer {
  public USWeatherServer() { super("United States Weather Server");}
}
```

EuropeWheaterServer.java

```
public class EuropeWeatherServer extends RemoteWeatherServer {
  public EuropeWeatherServer() { super("Europe Weather Server");}
}
```

WeatherView.java

```
public abstract class WeatherView {
  private String viewName;
  public WeatherView(String viewName) { this.viewName = viewName;}
  public void display(WeatherData weatherData) {
      System.out.println(viewName +" gets data from " + weatherData.getWeatherSource() + " : ");
      System.out.println("Temperature : " + weatherData.getTemperature());
      System.out.println("Humidity : " + weatherData.getHumidity());
  }
}
```

WebView.java

```
public class WebView extends WeatherView {
  public WebView() { super("Web Weather View");}
}
```

DesktopView.java

```
public class DesktopView extends WeatherView {
  public DesktopView() { super("Desktop Weather View");}
}
```

MobileView.java

```
public class MobileView extends WeatherView {
  public MobileView() { super("Mobile Weather View");}
}
```

WeatherData.java

```
public class WeatherData {
  private float temperature;
  private float humidity;
  private String weatherSource;
  public float getTemperature() { return temperature;}
  public void setTemperature(float temperature) { this.temperature = temperature;}
```

```
  public float getHumidity() { return humidity;}
  public void setHumidity(float humidity) { this.humidity = humidity;}
  public String getWeatherSource() { return weatherSource;}
  public void setWeatherSource(String weatherSource) { this.weatherSource = weatherSource;}
}
```

WeatherService.java

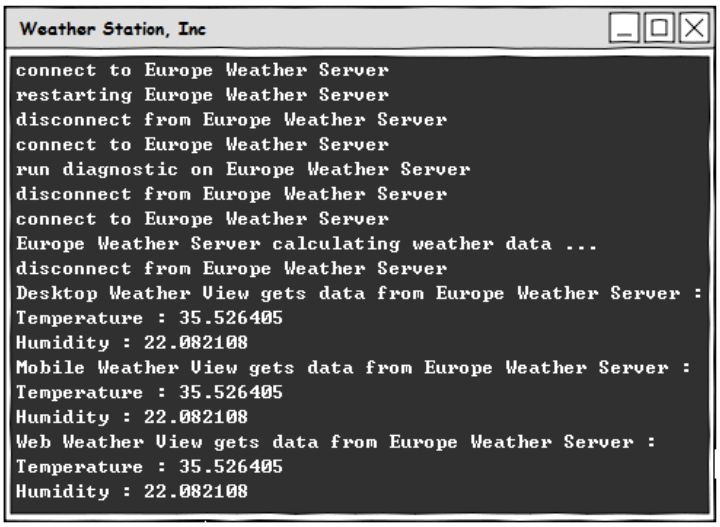
```
public class WeatherService {
  private RemoteWeatherServer server;
  private DesktopView desktopView;
  private MobileView mobileView;
  private WebView webView;
  public WeatherService(RemoteWeatherServer server) {
      this.server = server;
      desktopView = (DesktopView) createWeatherView("desktop");
      webView = (WebView) createWeatherView("web");
      mobileView = (MobileView) createWeatherView("mobile");
  }
  public void rebootServer() {
      server.connect();  server.restart();  server.disconnect();
  }
  public void diagnosticServer() {
      server.connect();  server.diagnostic();  server.disconnect();
  }
  public void displayWeatherData() {
      server.connect();
      WeatherData data = server.calculateWeatherData();
      server.disconnect();
      desktopView.display(data);
      mobileView.display(data);
      webView.display(data);
  }
  private WeatherView createWeatherView(String type) {
      if ("web".equals(type)) { return new WebView();}
      if ("mobile".equals(type)) { return new MobileView();}
      return new DesktopView();
  }
}
```

Main Class (reboot, diagnostic, and display weather data (to all platforms) from europe weather server):

ProblemTest.java

```
public class ProblemTest {
  public static void main(String... args) {
      RemoteWeatherServer server = createWeatherServer("eu");
      WeatherService usWeatherService = new WeatherService(server);
      usWeatherService.rebootServer();
      usWeatherService.diagnosticServer();
      usWeatherService.displayWeatherData();
  }
  public static RemoteWeatherServer createWeatherServer(String type) {
      if ("us".equals(type)) { return new USWeatherServer();}
      if ("eu".equals(type)) { return new EuroWeatherServer();}
      return new AsiaWeatherServer();
  }
}
```

Main Program Output :



SOLUTION :

PROBLEM ANALYSIS :

We're going to improve codes and class diagram using Observer, Factory, and Command pattern. Please look detail at `WeatherService` class, this class is not open for change (closed for change). We have three big problems :

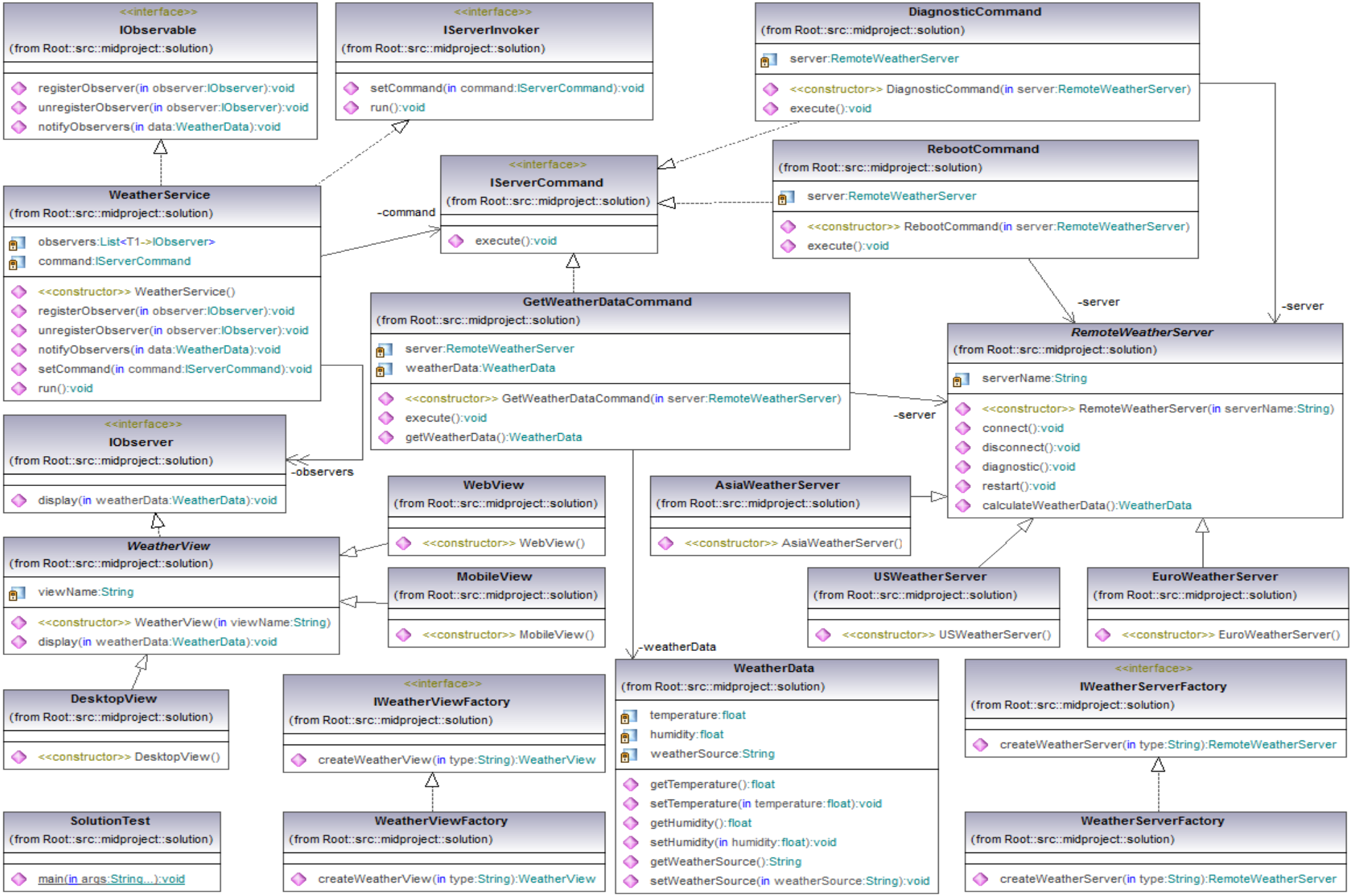
- First problem arise if we want to add new operations then we will change the existing class (in this case is `WheaterService` class).
- Second problem arise when we add new platform for displaying wheater data then we will also change the `displayWheaterData()` and we have to add new class properties (subclass of `WheaterView` class), then it's means that existing class have to be changed .
- ThirdProblem is in the main class, in `ProblemTest` class there is static method : `createWeatherServer()` to manage object creation. This is not good to have main class too much responsibility, main class should be decoupled from core/main procedures.

SOLUTION TO PROBLEMS :

- Firstly, we have to move out the (server) operations in `WheaterService` class and encapsulates the operations to be a command object using Command pattern. In detail, we add a command interface (`IServerCommand` interface), an invoker (`IServerInvoker` interface) to invoke command, and encapsultes `rebootServer()`, `diagnosticServer()`, `displayWheaterData()` in a form of `IServerCommand` , thus we have three command objects : `RebootCommand` class, `DiagnosticCommand` class, and `GetWheaterDataCommand` class. In this case we may write `WheaterService` class to be an implementation or realization of `IServerInvoker` interface.
- Secondly, we have to move `createWeatherView()` out, and place it in a `WheaterViewFactory` class (which implements `IWheaterViewFactory` interface) to handle object creation using factory pattern to reduce `WheaterService` class 's responsibilities. Then, write `WheaterService` class to make observable by convert it to be a form of `IObservable` (subject that notify to all observers) and make `WheaterView` class and its subclasses to be a form of `IObserver` (object that receive notification from subject) using Observer pattern.
- Thirdly, we have to move `createWeatherServer()` out, and place it in a `WheaterServerFactory` class (which implements `IWheaterServerFactory` interface) to handle object creation using factory pattern to reduce main class 's responsibilities.

IMPROVEMENT CODES : Please read details improvement codes in given source codes (*MidExamOfDesignPattern.zip*).

IMPROVEMENT CLASS DIAGRAM : Look at next page …


<<interface>>
IObservable
(from Root::src::midproject::solution)
registerObserver(in observer:IObserver):void
unregisterObserver(in observer:IObserver):void
notifyObservers(in data:WeatherData):void
<<interface>>
IServerInvoker
(from Root::src::midproject::solution)
setCommand(in command:IServerCommand):void
run():void
DiagnosticCommand
(from Root::src::midproject::solution)
server:RemoteWeatherServer
<<constructor>> DiagnosticCommand(in server:RemoteWeatherServer)
execute():void
WeatherService
(from Root::src::midproject::solution)
observers:List<T1->IObserver>
command:IServerCommand
<<constructor>> WeatherService()
registerObserver(in observer:IObserver):void
unregisterObserver(in observer:IObserver):void
notifyObservers(in data:WeatherData):void
setCommand(in command:IServerCommand):void
run():void
-command
<<interface>>
IServerCommand
(from Root::src::midproject::solution)
execute():void
RebootCommand
(from Root::src::midproject::solution)
server:RemoteWeatherServer
<<constructor>> RebootCommand(in server:RemoteWeatherServer)
execute():void
GetWeatherDataCommand
(from Root::src::midproject::solution)
server:RemoteWeatherServer
weatherData:WeatherData
<<constructor>> GetWeatherDataCommand(in server:RemoteWeatherServer)
execute():void
getWeatherData():WeatherData
-server
-server
-server
RemoteWeatherServer
(from Root::src::midproject::solution)
serverName:String
<<constructor>> RemoteWeatherServer(in serverName:String)
connect():void
disconnect():void
diagnostic():void
restart():void
calculateWeatherData():WeatherData
<<interface>>
IObserver
(from Root::src::midproject::solution)
display(in weatherData:WeatherData):void
-observers
WebView
(from Root::src::midproject::solution)
<<constructor>> WebView()
AsiaWeatherServer
(from Root::src::midproject::solution)
<<constructor>> AsiaWeatherServer()
WeatherView
(from Root::src::midproject::solution)
viewName:String
<<constructor>> WeatherView(in viewName:String)
display(in weatherData:WeatherData):void
MobileView
(from Root::src::midproject::solution)
<<constructor>> MobileView()
USWeatherServer
(from Root::src::midproject::solution)
<<constructor>> USWeatherServer()
EuroWeatherServer
(from Root::src::midproject::solution)
<<constructor>> EuroWeatherServer()
-weatherData
DesktopView
(from Root::src::midproject::solution)
<<constructor>> DesktopView()
<<interface>>
IWeatherViewFactory
(from Root::src::midproject::solution)
createWeatherView(in type:String):WeatherView
WeatherData
(from Root::src::midproject::solution)
temperature:float
humidity:float
weatherSource:String
getTemperature():float
setTemperature(in temperature:float):void
getHumidity():float
setHumidity(in humidity:float):void
getWeatherSource():String
setWeatherSource(in weatherSource:String):void
<<interface>>
IWeatherServerFactory
(from Root::src::midproject::solution)
createWeatherServer(in type:String):RemoteWeatherServer
SolutionTest
(from Root::src::midproject::solution)
main(in args:String...):void
WeatherViewFactory
(from Root::src::midproject::solution)
createWeatherView(in type:String):WeatherView
WeatherServerFactory
(from Root::src::midproject::solution)
createWeatherServer(in type:String):RemoteWeatherServer