Beatriz Aarão
A71015

Trabalho3 Arquitetura de Sistemas de Software MIEI

O propósito deste trabalho, passa por desenvolver uma API/serviço para que se possa aceder às leituras de vários dispositivos XDK, independentemente do tipo de plataforma usada pelos clientes. Espera-se que estes clientes possam **mostrar** e **gerir** a informação dos valores da **temperatura**, **humidade**, **pressão atmosférica**, **áudio** e **luminosidade**.

PERGUNTA1

Na pergunta um deste trabalho, foi pedido que se atualizado o código fornecido de forma a obter uma melhor distribuição das responsabilidades/funcionalidade implementadas. Era ainda pedido que identificasse e removesse eventuais *code smells* presentes no código fornecido.

Ao analisar o código fornecido, percebi de imediato que existiam dois tipos de métodos, sendo estes relativos à interface com o utilizador e à gestão dos valores dos sensores. Deste modo, de forma a separar as responsabilidades, foi criada uma classe ***View*** responsável por gerir a interface com o utilizador. A classe ***WeatherStation*** ficou assim responsável pela gestão dos valores dos sensores.

Se seguida, o código foi analisado método a método de forma a remover eventuais *code smells.* Tendo em conta a classe ***WeatherStation*** foi analisado o método *update*, o qual pude concluir ser maior do que o devido. De forma a resolver este problema, foi utilizado o procedimento *"Extract"*[1], o qual consistiu na criação de um método *update* para cada um dos sensores, métodos esses invocados no *update* inicial.

Tendo por ultimo em conta a classe ***View*** pude verificar que os vários métodos faziam um uso excessivo de *Switch Statements.* De forma a resolver isto, foi usado o procedimento "*polymorphism*"[2]. Desta forma, foi criada a interface Sensor, a qual passou a implementar todos os métodos anteriormente existentes na ***View***, com a diferença que ao invés de estes receberem um int de forma a identificar o sensor, passaram a receber um Sensor. Esta interface é implementada por todos os sensores, nomeadamente os sensores Temperatura, Humidade, Pressão, Áudio e Luminosidade.

PERGUNTA2

Na pergunta dois, era pedido que se introduzisse o *design pattern* **MVC** (*Model-View-Controller*) no código implementado na pergunta 1. Esse padrão separa as responsabilidades da aplicação em três diferentes camadas sendo estas:

- **Model (Modelo)** -> Representa os seus dados, provendo meios de acesso (leitura e escrita) a esses dados.

- **View (Visão)** -> Manipula os dados para - e apenas para – exibição. Exibe os dados.

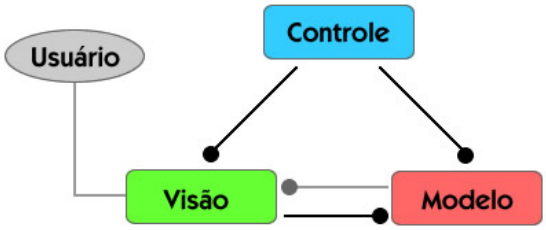


Figura 1 - MVC Design Pattern

- **Controller (Controle)** -> Responsável por decidir que *Model* usar, quais os pedidos a efetuar ao *Model*, qual combinação de *views* será usada para exibir os dados retornados pelo *Model*.

[1] (https://sourcemaking.com/refactoring/extract-method)
[2] (https://sourcemaking.com/refactoring/replace-conditional-with-polymorphism)

Implementação do Exercício

De forma a implementar o padrão pedido, a classe ***WeatherStation*** foi definido como um ***Model***, passando a chamar-se ***WeatherModel***. Esta classe é assim responsável pelo método *update* responsável por guardar os valores atualizados do XDK.

A classe ***View*** passou a chamar-se ***WeatherView***, ficando responsável por todos os métodos referentes à interface com o utilizador.

Por ultimo, foi criado um ***Controller,*** o qual possui uma instância da classe ***View*** e uma da classe ***Model***. Todos os métodos desta classe dizem respeito ao chamamento de métodos das classes ***View*** e ***Model***.

Foi ainda criada uma classe de nome ***MVCPatternDemo***, a qual possui os métodos responsáveis por gerar valores aleatórios para o XDK. Possui ainda o método Main, o qual possui exemplos de chamadas dos métodos sugeridos no código do professor.

Output

De seguida, é demostrado uma sequência de 3 outputs gerados:

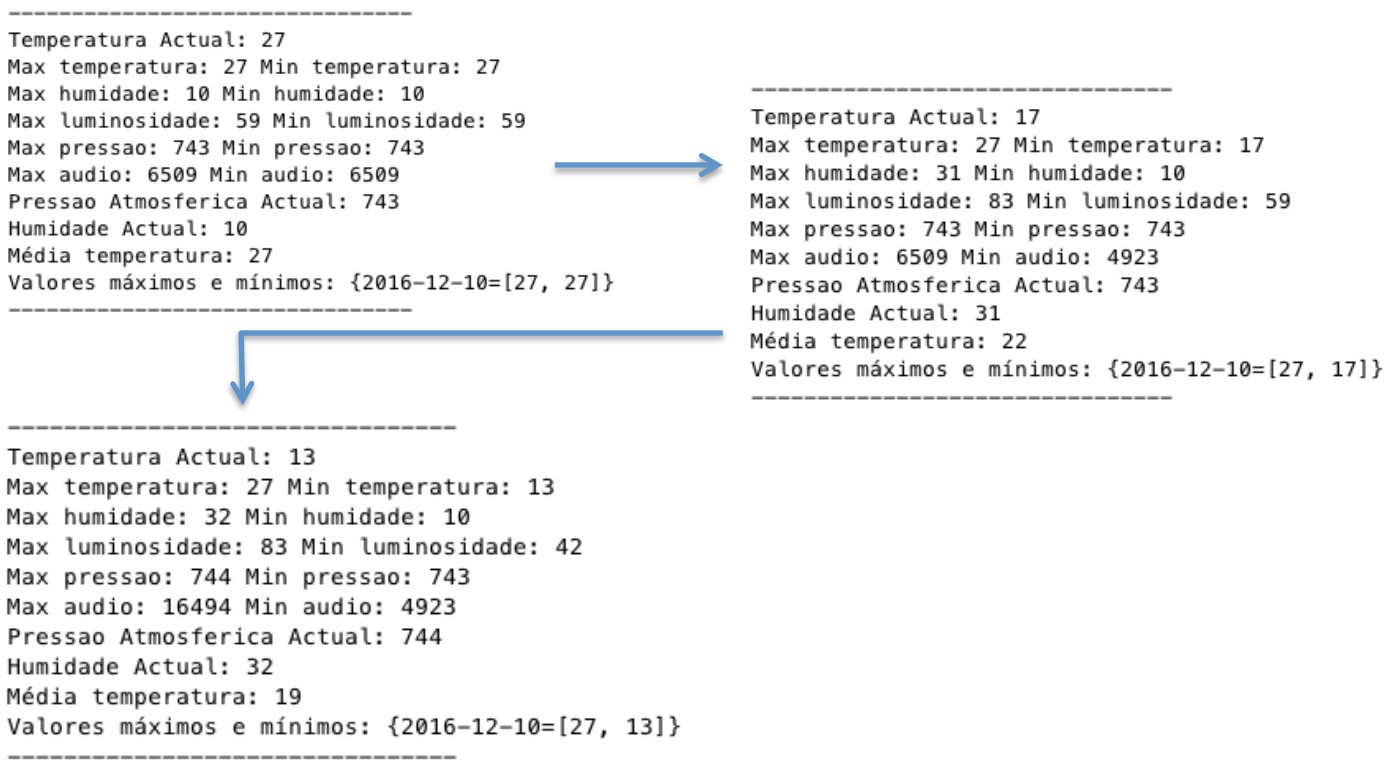


Figura 2 - Output gerado pelo programa

PERGUNTA3

Na pergunta três, era pedido que se introduzisse o *design pattern* **Client-Server** no código implementado na pergunta 1. Este padrão, permite que um objeto em execução numa máquina virtual Java invoque métodos num outro objeto em execução numa outra máquina virtual Java.

Este padrão compreende dois programas separados sendo estes:

- **Servidor** -> Cria objetos remotos, faz referencias para estes objetos acessíveis e aguarda que os clientes invoquem métodos nestes objetos.

- **Cliente** -> Obtém uma referencia remota para um dos objetos remotos, invocando métodos nos mesmos.

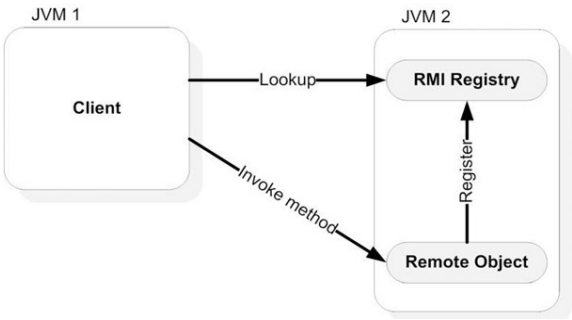


Figura 3 - Comunicação entre o RMI cliente e o servidor (objeto remoto).

Implementação do Exercício

De forma a implementar o padrão pedido, foi implementada a interface ***InterfaceServer***, a qual é implementada pela classe ***ServerImpl***. Esta classe, é responsável pela declaração de todos os métodos que irão ser invocados pelo cliente. De forma a garantir a divisão das responsabilidades, foram mantidas as duas classes criadas na questão anterior, ou seja, a classe ***DataC*** fica responsável pelo método *update*, e a classe ***View*** fica responsável por todos os métodos de interface com o utilizador. Todos os métodos da classe ***ServerImpl*** possuem a cláusula "*throws RemoteException*".

De seguida, foi criada a classe ***RMIServer***, a qual é responsável pela criação e registo do objecto remoto de dados que irá ser usado pelo Cliente.

Por ultimo, foi criada a classe ***RMIClient***, a qual é responsável pela obtenção de uma referencia para o objeto remoto criado na classe ***RMIServer*** e pela invocação dos métodos usando esse objeto.

Output

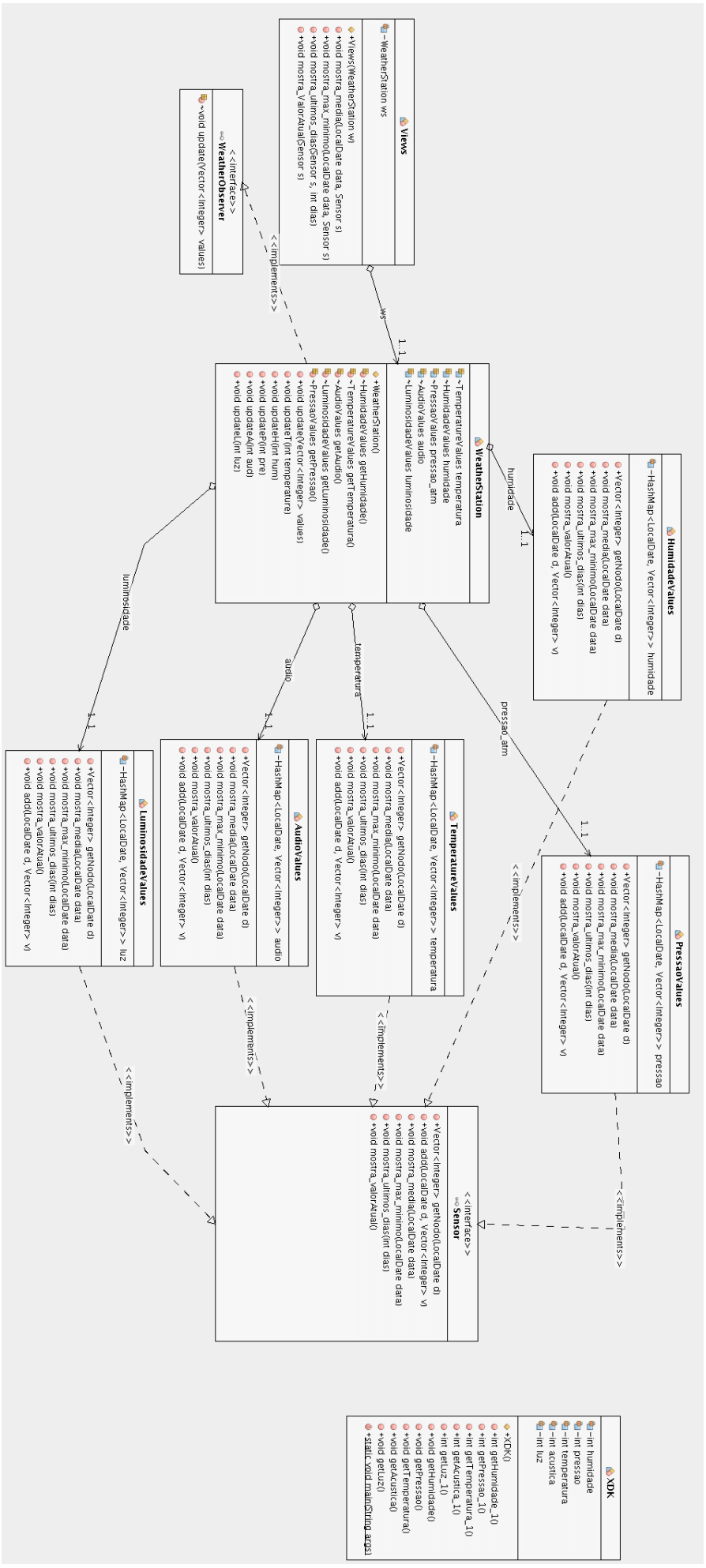
De seguida, é demostrado o Output gerado pelo Servidor e pelo Cliente:

```
Last login: Sun Dec 11 20:50:44 on ttys001
MBPdeBeatriz4:~ beatrizaarao$ cd NetBeansProjects/Client-Server/src/
MBPdeBeatriz4:src beatrizaarao$ rmiregistry &
[1] 3560
MBPdeBeatriz4:src beatrizaarao$ java -Djava.security.policy=policy client.server
.RMIServer &
[2] 3563
MBPdeBeatriz4:src beatrizaarao$ Servidor preparado
```

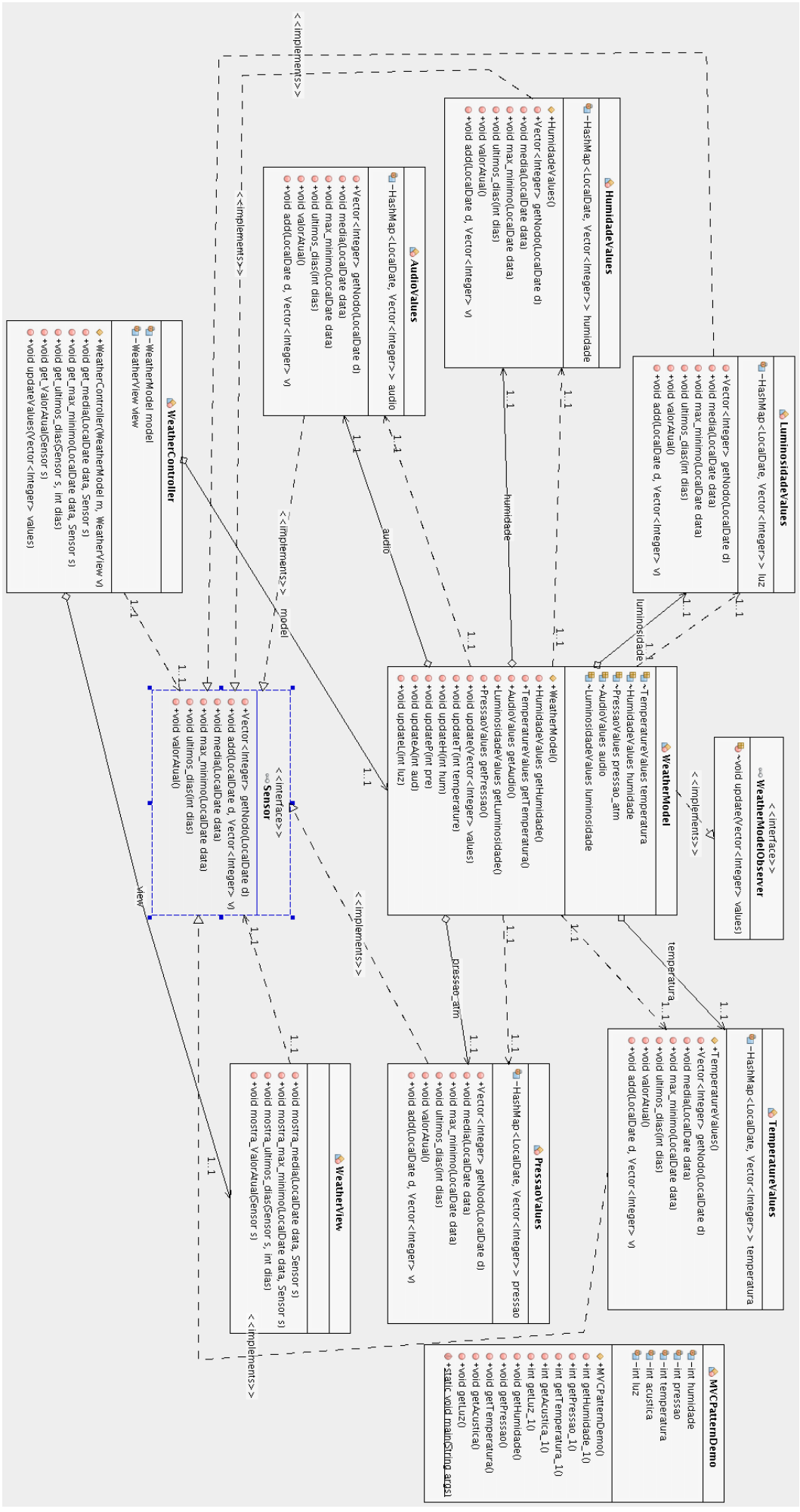
```
-bash: cd: NetBeansProjects/Client-Server/src/: No such file or directory
MBPdeBeatriz4:src beatrizaarao$ java -Djava.security.policy=policy client.server.RMIClient &
[1] 3576
MBPdeBeatriz4:src beatrizaarao$ Cliente Preparado
---------------------------------
Temperatura Actual: 19
Max temperatura: 19 Min temperatura: 19
Max humidade: 35 Min humidade: 35
Max luminosidade: 60 Min luminosidade: 60
Max pressao: 746 Min pressao: 746
Max audio: 4399 Min audio: 4399
Pressao Atmosferica Actual: 746
Humidade Actual: 35
Média temperatura: 19
Valores máximos e mínimos: {2016-12-11=[19, 19]}
---------------------------------
---------------------------------
Temperatura Actual: 15
Max temperatura: 19 Min temperatura: 15
Max humidade: 49 Min humidade: 35
Max luminosidade: 83 Min luminosidade: 60
Max pressao: 753 Min pressao: 746
Max audio: 15445 Min audio: 4399
Pressao Atmosferica Actual: 753
Humidade Actual: 49
Média temperatura: 17
Valores máximos e mínimos: {2016-12-11=[19, 15]}
---------------------------------
---------------------------------
```

Anexos

Pergunta1



Pergunta2


LuminosidadeValues
-HashMap<LocalDate, Vector<Integer>> luz
+Vector<Integer> getNodo(LocalDate d)
+void media(LocalDate data)
+void max_minimo(LocalDate data)
+void ultimos_dias(int dias)
+void valorAtual()
+void add(LocalDate d, Vector<Integer> v)
HumidadeValues
-HashMap<LocalDate, Vector<Integer>> humidade
+HumidadeValues()
+Vector<Integer> getNodo(LocalDate d)
+void media(LocalDate data)
+void max_minimo(LocalDate data)
+void ultimos_dias(int dias)
+void valorAtual()
+void add(LocalDate d, Vector<Integer> v)
AudioValues
-HashMap<LocalDate, Vector<Integer>> audio
+Vector<Integer> getNodo(LocalDate d)
+void media(LocalDate data)
+void max_minimo(LocalDate data)
+void ultimos_dias(int dias)
+void valorAtual()
+void add(LocalDate d, Vector<Integer> v)
<<interface>>
WeatherModelObserver
~void update(Vector<Integer> values)
WeatherModel
+WeatherModel()
+HumidadeValues getHumidade()
+TemperatureValues getTemperatura()
+AudioValues getAudio()
+LuminosidadeValues getLuminosidade()
+PressaoValues getPressao()
+void update(Vector<Integer> values)
+void updateT(int temperature)
+void updateH(int hum)
+void updateP(int pre)
+void updateA(int aud)
+void updateL(int luz)
~TemperatureValues temperatura
~HumidadeValues humidade
~PressaoValues pressao_atm
~AudioValues audio
~LuminosidadeValues luminosidade
TemperatureValues
-HashMap<LocalDate, Vector<Integer>> temperatura
+TemperatureValues()
+Vector<Integer> getNodo(LocalDate d)
+void media(LocalDate data)
+void max_minimo(LocalDate data)
+void ultimos_dias(int dias)
+void valorAtual()
+void add(LocalDate d, Vector<Integer> v)
PressaoValues
-HashMap<LocalDate, Vector<Integer>> pressao
+Vector<Integer> getNodo(LocalDate d)
+void media(LocalDate data)
+void max_minimo(LocalDate data)
+void ultimos_dias(int dias)
+void valorAtual()
+void add(LocalDate d, Vector<Integer> v)
WeatherController
-WeatherModel model
-WeatherView view
+WeatherController(WeatherModel m, WeatherView v)
+void get_media(LocalDate data, Sensor s)
+void get_max_minimo(LocalDate data, Sensor s)
+void get_ultimos_dias(Sensor s, int dias)
+void get_valorAtual(Sensor s)
+void updateValues(Vector<Integer> values)
<<interface>>
Sensor
+Vector<Integer> getNodo(LocalDate d)
+void add(LocalDate d, Vector<Integer> v)
+void media(LocalDate data)
+void max_minimo(LocalDate data)
+void ultimos_dias(int dias)
+void valorAtual()
WeatherView
+void mostra_media(LocalDate data, Sensor s)
+void mostra_max_minimo(LocalDate data, Sensor s)
+void mostra_ultimos_dias(Sensor s, int dias)
+void mostra_ValorAtual(Sensor s)
MVCPatternDemo
-int humidade
-int pressao
-int temperatura
-int acustica
-int luz
+MVCPatternDemo()
+int getHumidade_1()
+int getPressao_1()
+int getTemperatura_1()
+int getAcustica_1()
+int getLuz_1()
+void getHumidade()
+void getPressao()
+void getTemperatura()
+void getAcustica()
+void getLuz()
+static void main(String args)
<<implements>>
luminosidade
humidade
audio
temperatura
pressao_atm
model
view
1..1

Anexos

Pergunta3