Assignment 2 - Real Time Weather Map

In this assignment, we create a multi-threaded application to show live weather data (temperatures) for Cork, Ireland, on a map. For the purposes of the experiment, simulated data is used, with each second representing an hour.

A class library is provided in .jar format, with javadocs in HTML format. This includes the map GUI, and implements much of the basic functionality.

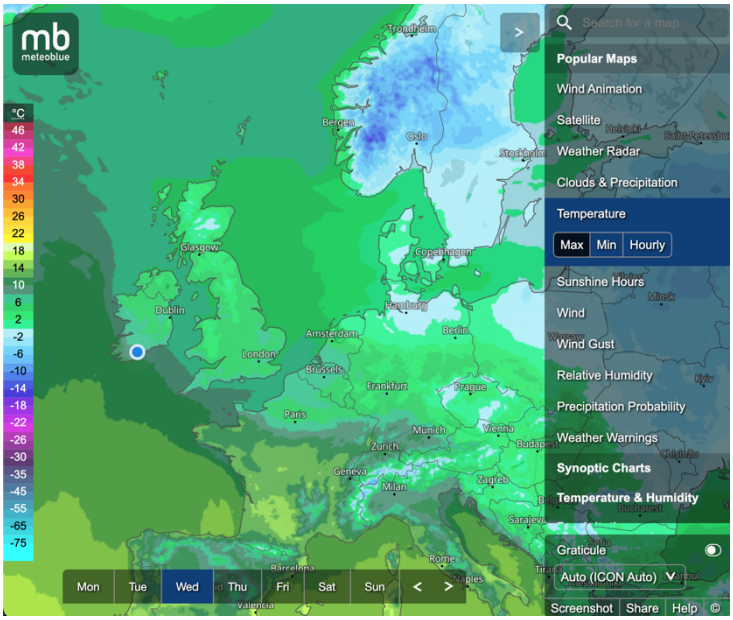

Figure 1: An example of a weather map.
(Your solution does not look like this)
Source: http://www.meteoblue.com

Basic Functionality

By extending the class GUI, and using the other classes provided in the JAR, implement an application which shows live temperature data from the weather sensors on the map.

If you implement everything correctly, you should see regularly-spaced coloured squares over the land area, in colours which change.

You are strongly recommended to begin with a thorough reading of the javadocs, and a paper sketch of the classes. Illustrate the worker threads on your sketch.

Weather Data Sensors

Access to the weather sensors is via the Weather class, which exposes two streams (for sending and recieving data respectively), acting as a gateway server for communication with all the sensors. Only instantiate the Weather class once.

The weather sensor server uses messages with a binary serialization format (i.e. byte-based serialization), the format for which are listed in the appendix. The Weather class needs to run on its own worker thread.

Initially, the Weather class is queried for the available sensors (their IDs as integers). This is done by writing a QUERY_SENSOR_LIST message into the stream (only do this once). Read the response message (in format RESPONSE_SENSOR_LIST), containing the list of available sensors. The list of sensor IDs is used until the application is closed. The sensor names in the response are not used.

Use a java.util.Timer class to periodically fetch new weather data, by sending a QUERY_SENSOR message for every weather sensor, using its specified ID. Read the responses (in format RESPONSE_SENSOR). Recommended interval is 0.5 seconds. Update the GUI with the new temperatures. Close the weather class as the application exits.

Tip: There are no extra bytes between the messages. Only the data for the fields are sent, not the field names. Sending corrupt data to the Weather class will cause it to terminate.
Tip: Validate the version and message type fields you receive. It will make the debugging easier.
Tip: Leave the temperatures for the other tiles unchanged.
Tip: Whilst it can be tempting to represent the messages as separate classes, this can increase the complexity, and is not mandatory.

GUI

The GUI shows the weather data. Additionally, it has a start/stop button, which controls whether the weather data is being queried. Implement this with a callback, using the interface provided. Other than callbacks, which design pattern is relevant here? (Write a comment in your code)

Tip: leave the timer running all the time, and use a boolean to control if the weather is queried.

Logger & Queue

Each time new weather data is received from the simulator, write a line to a log file, via Buffer<T>. The logger should have its own worker thread. This worker thread receives items from a queue, and writes each item to a file on disk.

Derive from the provided Logger class for your file-based logger. For the Buffer, use the provided class, or implement your own (it needs to be thread safe). It is OK if the log file is overwritten each time the application starts. If it is helpful for debugging, you can add other messages from other threads too. Remember, the application needs to work on other peoples PCs (Windows or Mac), so don't use (e.g.) absolute paths.

Tip: Ensure the worker thread does not prevent the application from closing. If you decide to make it a daemon thread, it will need an exit flag (or similar).

Coding Style

- Try to use English as much as possible for the code (e.g. class and method names) - this is standard practice. It is OK to use Swedish in the comments.
- To follow good OO practice, your solution must not use the static keyword, except for main method(s), and constants of basic datatypes (i.e. int, double, string). Static methods, and mutable static attributes must not be used.
- Apart from the Java standard libraries, and the provided JAR file, no additional JAR files and libraries may not be used.
- Vanilla exceptions (i.e. "Exception" or "Throwable") must not be caught.
- Generative AI tools (like ChatGPT) must not be used. Instead, take this oppurtunity to learn to programme for yourself, gain confidence as a programmer, and get feedback on your own work. Save the generative AI to boost your productivity as a developer.
- Do not use advanced techniques like reflection to modify the behaviour of the provided classes, as this will disrupt the oral exam.
- It is OK to "comment out" code as you experiment with different implementations, but before you submit, try to clean up as much as possible, and remove any "commented out" or otherwise unused code. Keep refining your code until it is as simple as it can be, whilst still fulfilling the requirements.

Diagram

For this assignment, you need to create a *sequence diagram* representing all your communication with the weather class. (Ignore any classes which are not visible to you). Use the class names as labels. Use fragments appropriately. Don't forget to include a check for the "pause" button. Represent the responses from the weather class as being asynchronous when you draw your diagram.

Include a PNG image in the root of your ZIP file. Use a tool like Visual Paradigm. A scanned/photo copy is not acceptable.

Submission

Upload a .zip file containing (a) your solution, preferrably in IntelliJ project structure, and (b) your diagram, in PNG format. Do not include any nested .zip files within this zip file. For this assignment, there is no requirement to use javadocs. However, it may be helpful during the redovisning if you write comments to help you explain your code! You are also required to submit a review for the "kamratgranskning" (see below).

Kamratgranskning & Redovisning

For this submission, you need to do a 'kamratgranskning' (peer-review), this is required for a passing grade. When the deadline is passed, you will be randomly matched to another student.

You need to download their submission and review it. You need to upload a short document. Write a sentence or two for each of these questions (you will present this feedback in the redovisning).

1. Does their code build? run? is there exception messages visible in the output?
2. Are messages added to the log file?
3. Does the application implement the required functionality?
4. How are worker threads used in the code?
5. Are the coding standards followed? The use of statics?
6. Is the diagram correct?
7. How does the solution differ from yours? How easy is to read?
8. Any other suggestions for improvements.

(Note: If your paired student's submission is empty, corrupt, or incomplete, you are not penalized) Ensure you have uploaded your response into Canvas in good time for your redovisning time slot.

As before, you will be required to undertake a short, mandatory, oral exam ("redovisning"), where your code will be reviewed and discussed with a teacher. The teacher will ask you to, for example:

- demonstrate functionality of your application.
- explain functionality in your code.
- make small modifications to demonstrate your understanding.
- briefly justify implementation choices you have made.

This time, we do it in the randomly-allocated peer-review pairs.

When it is time for the redovisning, prepare by looking at the code again to refresh your memory!

Plagarism

This is an individual assignment. Attend the scheduled lab sessions to get help from the teachers. Do not share your code with other students. Doing so deprives them of the oppurtunity to learn for themselves. Remember, you are still guilty of plagarism if another student copies your work. It is your responsibility to ensure this does not happen.

Work is routinely checked for plagarism. In some cases, checking for plagarism may delay you getting your grade on a particular assignment. Plagarism may result in a disruption to your studies, which may affect your entry requirements for other courses in your degree programme.

Please do not post your solution online (e.g. GitHub), even after your exam. This tempts other students to plagarise your work, depriving them of the oppurtunity for learning.

Appendix – Message Formats

QUERY_SENSOR_LIST

Query the server for the list of available sensors.

Data type	Name	Value	Comment
Int32	Version	1	
Int32	MessageType	1	

RESPONSE_SENSOR_LIST

Server's response to QUERY_SENSOR_LIST message.

Data type	Name	Value	Comment
Int32	Version	1	
Int32	Message Type	2	
Int32	Number of Sensors		i.e. how many of the following blocks.
Int32	Sensor ID		This block is written for each sensor. Sensor name is UTF8 Encoded.
String	Sensor Name		

QUERY_SENSOR

Query the server for the current temperature for a (single) sensor.

Data type	Name	Value	Comment
Int32	Version	1	
Int32	MessageType	3	
Int32	Sensor ID		The ID of the Sensor to Query

RESPONSE_SENSOR

Server's response to the QUERY_SENSOR message. Temperature for a single sensor.

Data type	Name	Value	Comment
Int32	Version	1	
Int32	MessageType	4	
Int32	Sensor ID		The ID of the Sensor to Query
Int32	Row	0..99	The row on the map.
Int32	Col	0..99	The col on the map.
Float64	Temperature		Double-precision floating point ("double"). Degrees C.