Air Quality Simulator

Figure 1: The Millenium Development Goals

Introduction

Air pollution is a problem, especially in the Öresund region, which can sometimes have the worst air quality in the country. Air quality and pollution relate to several of the millenium development goals.

In this assignment, we develop a basic interactive simulator for air quality, intended for use by students and teachers in educational settings.

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

Your solution must fulfill all the requirments in this document for a passing grade. Pay particular attention to all mentions of "must" or "must not", "not permitted" etc.

Air Quality

Air quality is modelled as a grid of values over the Öresund region. Higher positive values indicate worse air quality (pollution). Negative air quality values are not used. Air quality is modelled as a floating-point number (double).

Basic Functionality

By extending the abstract class AirQualityApp, the following behaviour must be implemented:

- The map image, provided on the ImageResources class, must be used.
- When clicking on the map, one element of the specified type of item is created, and shown at the grid location of the mouse click with the correct icon.
- The type of element (bus, car, etc.) is determined by the combo box (i.e. dropdown list) shown in the GUI.
- When the button is clicked, the simulator runs one "time step" (analagous to a turn-based game). Otherwise the simulator is paused.
- During each time step, the air quality values are updated in the grid, and some of the items move around the grid, according to the rules outlined below.

Elements

These can only be created on (and remain on) land grid squares:
Car: Emits +5 pollution at its grid location, then moves 1 square in a random compass direction.
Bike: Moves 1 square in a random compass direction. Emits no pollution.
Bus: Choose yourself how to implement.

Can be created and move on all tiles:
Airplane: When created, a compass direction is chosen at random. Emits +10 pollution at its grid location, then moves 5 squares in its predetermined direction.

More than one moving element is able to occupy the same square.
Elements which move off the map must be removed from the simulation.

This element does not move, and can only be created on land:

Woodland: Reduces air quality by 5 at its grid location (with a lower bound of zero). Does not move.

It must not be possible to place non-moving elements (e.g. woodlands, or your own non-moving element types) on the same tile.

You can add more element types of your own if you wish, but all those above must be implemented according to the requirements.

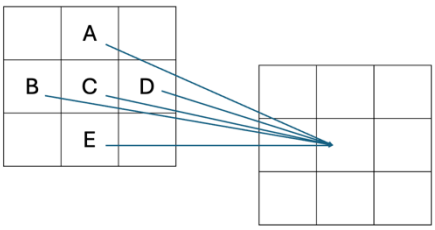
Diffusion

The grid of pollution is initially zero everywhere. To model diffusion of air pollution, at each time step (after the emission and item movement), air pollution is spreads out to neighbouring squares.

At each time step, for each cell in the grid, the pollution value is the average of (the current values in the cell) + (the sum of the pollution values in the adjacent 4 cells). At the edges, neighbouring cells outside the boundary are considered to have zero pollution.
i.e. Cell 'C' is not at a boundary, so the new value for the cell 'C' after the time step will be equal to:
(A + B + C + D + E) / 5

To implement the diffusion process, for each time step:
1. Create a new grid of values with all cells zero.
2. For each cell in the new grid, compute the cell value based on the values of neighouring tiles in the old grid, according the description and diagram.
3. Replace the old grid of values with the new one.

(You can also achieve the same result swapping two arrays).



Apart from the boundaries, the total amount of air pollution must be conserved during the diffusion process. Diffussion is not affected by land/sea.

Visualization

Pollution values are visualized on the map, using the appropriate method(s) on the provided GUI class. At each time step, all the pollution values must be updated. Your implementation must invoke the .repaint() method to ensure all the updated pollution values are shown.

Your Task

Your task is to model, and then implement, a hierarchy of classes to represent the elements, the grid of pollution values, and any additional classes.

The following classes are provided in the library and must be used:
- AirQualityApp. Must be used to display the GUI.
- Direction. Compass directions.
- IelementIcon. represents icons on the map.
- ImageResources. Must be used for icons for the required icons. (You can use your own icons for any additional elements.)
- IsLand (and IIsLand). Must be used to check movement/placement of elements. You can choose to use IsLand, or implement IIsLand yourself. There is a "land" strip of tiles for approximately on the Öresund bridge.
- MovedOutOfGridException. Must be throw by any element if it moves outside of the grid, and handled to implement the required behaviour.

Create a libs directory in your project root, and put the JAR file in it. Follow the instructions here to add the JAR file as a module dependency: https://www.jetbrains.com/help/idea/working-with-module-dependencies.html Otherwise, you must not use any 3rd party code, libraries except those in the standard Java libraries. Include this when you upload the project. No other JAR files are allowed.

Tips
- Begin by reading all the documentation for the provided classes (start with index.html).
- This assignment is about modelling! Before you start coding, begin your modelling process by sketching a class diagram for the provided classes, and add more classes with your own ideas.
- Likewise, for the array operations – sketch your solution first!
- Study this document, and check through the document carefully to ensure you have implemented everything correctly. Experiment with the provided classes, run the GUI, and let your model evolve. Bring a pencil and paper to the labs.
- The debugger is there to help you!.... Use it to step through each new part of the code as you write it.
- Leave plenty of time to test, write documentation, diagram, zip your project and upload it in good time for the deadline.
- It is normal to feel anxious about the programming, avoid procrastination by doing a little bit every day. Sketching your solution on paper can also help.
- Attend the lab help sessions from the beginning, don’t wait to fall behind!
- To begin writing code, extend the AirQualityApp class... good luck!

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.
- Except when dealing with the text for the combo box, you must not use switch statements,

instanceof, or similar to implement element-specific behaviour ("element" here means bike, bus, woodland, etc.), instead, apply OO principles, and encapsulate logic inside the classes. Try to create a hierarchy of classes to re-use your code as much as possible. (It is of course OK to use switch statements in other ways, like enum values).
- 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.

Submission
You must write Javadoc comments for all public members (constructor(s), methods, attributes), for a minimum of one of your elements (car, bike, etc.), and generate javadoc HTML documentation in a "docs" directory within your project. Include this in your zip file.
See: https://www.jetbrains.com/help/idea/javadocs.html#generate-javadoc

You must also include a class diagram in PNG format, in the root of your Zip file. It must include all your classes, and all those you have used in the provided library code. You can generate it using the IDE, or create it using e.g. Visual Paradigm (if you want to practice using it for the exam!). It only needs to show the inheritance relationships, not the associations (but be ready to discuss this in the redovisning). Ensure the text is clearly legible. Do not scan/photo a sketch.

It must be possible to build and run the project without modifying the code. Be careful with file paths, so that the application runs on Mac and Windows operating systems.

Upload a .zip file containing your solution, preferrably in IntelliJ project structure. Do not include any nested .zip files within this zip file.

Redovisning
For a passing grade on this assignment, you will be required to undertake a short, mandatory, 1-on-1 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.
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. This document and the provided code (including the JAR file, its contents, and documentation) is copyright, Ben Blamey /