Kursus 02101
Paul Fischer, DTU Compute

Indledende Programmering – Efterår 2021
Hjemmeopgave 3

Dato: 9. 11. 2021, **Slut dato**: 20. 11. 2021, kl.23:59 på DTULearn

Dette er den tredje afleveringsopgave i kurset 02101. Husk at disse opgaver er *obligatoriske* og indgår i kursets samlede bedømmelse. Procenttallet for hver opgave indikerer den estimerede tid.

Læs den generele information om afleveringsopgaverne på Learn:
`VejledningAfleveringsopgaver.pdf` og `Samarbejdspolitik.pdf`.
Husk, at programmerne skal køre i *CodeJudge*, hvor der også ligger et par testfiler.
Husk desuden: Ingen mellemrumsymboler før eller efter outputtet.
Der skal laves en rapport kun for opgaver 3.2.

In general, make your programs robust, for example ensure that they can handle illegal inputs.

Problem 1 [10% + 15% + 15%]: [Inheritance]
On CodeJudge you find a skeleton of a class `Plane`. A plane is described by its manufacturer and type (both strings) and an id-number an integer. The default id-number is 0.

a) Complete the method `equals` such that it returns true if and only if the fields `id` of the two planes are identical. Also override the method `toString` such that it returns the word “Plane” followed by the id-number, a blank, the manufacturer, a blank and finally the type. For example
`Plane4 Boing 737-800`.

b) Design subclasses `PassengerPlane` and `FreightPlane` of the class `Plane`. Passenger planes have an additional integer field for the number of seats, while freight planes have an additional integer field for the payload in tons. The `toString` of the two classes should provide the relevant information as

```
Plane2 Boing 737 seats:241
Plane3 Antonov 124 payload:110
```

c) Design a class `Airport` which allows planes to land and start. The airport keeps track of the planes currently on the airport. The class has to define the following methods:

```
public void land(Plane plane)
public void start(int id)
public String toString()
```

The first method adds a plane to the those on the air port, and an assigns an a new id-number to that plane. The id-numbers start with 1 and are then increased by 1. If a plane wants to land which already has the id-number of plane on the airport, than is is not added.

The second method removes the plane plane from those on the airport if it was actually there.

The last method list all planes currently on the airport, one per line in order of the landings.

End of Problem 1

Problem 2 [60%]: [A simulation]
Peberholm is an artificial island and part of the Øresund-link between Copenhagen and Malmö. It is made of rocks and sand. No vegetation has been planted, the island serves as a test area for studying how initially barren land becomes inhabited by pants and animals in a natural way. We simulate the occupation of the island by plants.
The area is a square grid of size $n \times m$, where $n, m > 0$. Plants can only be located at grid positions and each grid position is occupied at most one plant. There are four kinds of plants: trees, bushes, flowers, and mosses. They differ in the way they look, how large they are and how they spread. Trees are black, bushed are green, flowers are red, and mosses are blue. Planting (and drawing) is done by the given program.
The simulation proceeds in steps. In each step each plant spreads its seeds as follows. Each plant generates s seeds. Each seed is placed by adding random numbers between $-r$ and r to the coordinates of the mother plant. The seeds which are not fallen into the water immediately grow to full size plants which have a size of $b \times b$. The parameters s, r, and b are individual for each plant type. They are located in the file `PeberholmConstantsAndUtilities.java` and are named `<TYPE>_SEED_NO` and `<TYPE>_RANGE`, respectively, where `<TYPE>` is one of `TREE`, `BUSH`, `FLOWER`, or `MOSS`. Also the colors in which the plants are shown is defined there.
Initially there is one plant of every type, placed at a random position of the grid, which is provided by `PeberholmSimulation`. We assume that the plants grow really fast. That is, a seed immediately becomes the full size plant. Seeds falling into Øresund are lost.
On CodeJudge you find tree Java files. File `PeberholmSimulation.java` defines a class which runs the simulation including all the graphic. File `PeberholmConstantsAndUtilities.java` defines some constants determining the island size etc. It also contains some "utility" methods that you might find helpful. File `PeberholmDriver.java` contains the `main` method where a simulation is created and started. Class `Plant` has to be completed. In `PeberholmConstantsAndUtilities.java` one can set a constant `SHOW` which controlls whether the proces is shown as graphic.
What you have to do:

1) Look at the files `PeberholmSimulation.java`, `PeberholmConstantsAndUtilities.java`, `PeberholmDriver`, and `Plant` and understand what the classes do and how they interact.

2) Complete `Plant` and define the missing classes to make the simulation run. Ensure that the programmer is forced to define all methods necessary to run the simulation. This can be done in two ways, both are appropriate.

3) Ensure that the plants behave as described.

4) Make sure that each plant, in its `toString` method, also reports the type (tree, bush, etc).

- Be aware of that the parametres defined in `PeberholmConstantsAndUtilities.java` might be different in different tests.

The file `PeberholmSimulation.java` and `PeberholmDriver` must not be changed.

What you do not have to do:
The planting is taken care of in `PeberholmSimulation.java`. Also, there will never be two plants at the same location, you do not have to anything to ensure this.

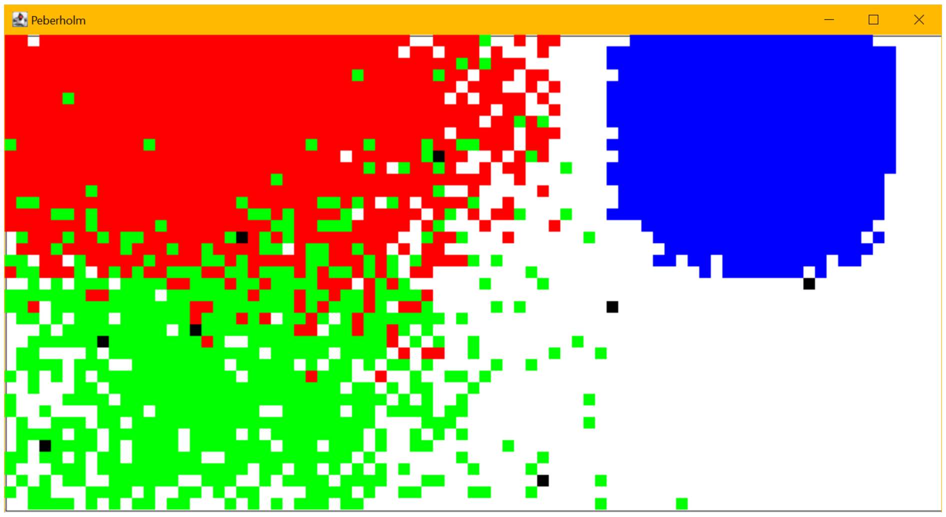

Figure 1: Example of a runs of the simulation showing a state after 7 steps.

End of Problem 2