CS3101-2 Scala, Fall 2014: Problem Set 2

Daniel Bauer

Total points: 20
Due date: Nov 4, 11:59pm EST

Submission instructions:
Place the files for all problems in a directory named `[your_uni]_week[`X`]`, where X is the number of the problem set. For instance if your uni is xy1234 and you are submitting the problem set for the first week, the directory should be called `xy1234_week1`. Either zip or tar and gzip the directory (using `tar -c xy1234_week1 | gzip > xy1234_week1.tgz`) and upload it to your directory in the drop box for this class on Courseworks.

The Aquarium Simulation

In this problem set we will simulate an aquarium containing different types of `AquariumElements`: `Rocks`, `Plants`, and different `Fish`. The aquarium is a 2D grid. Each cell is either empty or contains an `AquariumElement`.

The file `AquariumSimulator.scala` defines an `Aquarium` object, which maintains an array of AquariumElements. Each `AquariumElement` comes with a `locX` and `locY` attribute to indicate its position. Elements can be added to the aquarium using the `add` method of `Aquarium`. The `draw` method visualizes the state of the aquarium as an ASCII graphics map on which each `AquariumElement` is represented as a single character (# for rocks, $ for plants).

For instance, the main method of the `AquariumSimulator` adds a few `Rocks` and a `Plant`.

```
Aquarium.add(new Rock(1,1))
Aquarium.add(new Rock(1,2))
Aquarium.add(new Plant(2,3))
```

The `Aquarium.draw` method will print the following map:

```
......
.#....
.#....
..$...
......
......
```

Compile the source file (`fsc AquariumSimulator.scala`) and then run the simulator (`scala AquariumSimulator`). The `main` method in the object AquariumSimulator calls `Aquarium.update` repeatedly until the user enters 'quit'. For now, `update` does nothing (since all objects in the aquarium are stationary).

Familiarize yourself with the code in `AquariumSimulator.scala`. We did not discuss some of the concepts used in the implementation of Aquarium yet (specifically pattern matching and options), but you should get an overall idea of how the simulator works.

Part 1 (3 points)

`Aquarium` is implemented as a singleton object. Change it into an `Aquarium` class. The primary constructor of this class should accept a width and a height parameter. Change the main method in the `AquariumSimulator` object to create and use a 10x10 `Aquarium`.

Part 2 (3 points)

The `update` method of the `Aquarium` iterates through every cell. If it encounters an `AquariumElement` in this cell it calls the `move` method on this object. The `move` method returns one of the following `Symbols`, `'N` for north, `'S` for south, `'E` for east, `'W` for west, or `'Stay` for no movement at all. The `Aquarium` will try to move the `AquariumElement` according to the return value. The move is ignored if it would make the `AquariumElement` leave the 2D grid. `Plants` and `Rocks` don't move and return 'Stay.

Create a class `Fish` that extends the abstract class `BaseFish`. Implement the `move` method (overriding the abstract definition in `AquariumElement`) to move the fish in a random direction (north, south, east, or west). You can use `scala.util.Random.nextInt(4)` to produce a random number between 0 and 3 (inclusive). The character to represent a `Fish` on the map can be `f`.

In the `main` method of `AquariumSimulator` add a few `Fish` objects to the aquarium and run the simulation.

Part3 (3 points)

When a `Fish` f attempts to move to a cell that is already occupied by an `AquariumElement` e, the collision is handled in the following way. If f`.eat(`e`)` returns true, e is replaced with f on the grid. If e`.eat(`f`)` is true, f disappears. Otherwise the `Fish` can't move and stays in its original location.

Create a new class `HungryFish` to extend `Fish`. Override the `eat` method to return `true` if the `AquariumElement` passed to it is edible (its `edible` attribute is true). The character used to represent a `HungryFish` on the map can be `F`.

Add some `HungryFish` instannces to the aquarium and run the simulation.

Part4 (4 points)

Note that `HungryFish` are cannibals and eat other fish (since all `LifeForms` are edible by default). Modify your classes so that fish have a `size` attribute and only larger fish eat smaller fish. Make `size` a parameter of the default constructor.

Part 5 (4 points)

Typing '`new HungryFish(locX, locY, size)`' to create a new instance is cumbersome. Create companion objects and apply methods for all `AquariumElements` that make it possible to create instances without `new`.

Part 6 (3 points)

Draw a class diagram showing the inheritance relation between `AquariumElements` (submit a .pdf or .png file).