CSC207 Lab 4 — Java Inheritance

To earn lab marks, you must arrive on time, actively participate for the entire session, and make good progress.

1 Overview

This week, you are going to implement two Java classes. You'll also document the classes by providing doc comments for the Javadoc tool.

2 Log in and get things set up

`s1` drives and `s2` navigates.

1. Update your subversion repository to get the newly created `lab4` directory. This directory contains directories `src` and `week4lab`, and a file `Organism.java`.
2. Start Eclipse and select `lab4` as a workspace to work in today.
3. Create a new Java Project called `Week4Lab`. You should now be able to view the file `Organism.java` in Eclipse, in package `week4lab`.

3 Class `Organism`

Suppose we are trying to simulate the ecosystem of a tide pool. The pool contains all sorts of living things that float around and interact with their neighbours. Each critter moves around in its own special way.

Please, consult the UML class diagram as you study and implement the two classes.

`Organism` is an `abstract` class because plain `Organism`s should not be instantiated; instead, you will instantiate subclasses of `Organism`. You'll write a subclass in a bit.

Every `Organism` has a name, an `(x, y)` coordinate within the tide pool, a movement speed, and a current direction, which (to keep things simple) we will represent as one of "north", "south", "east", or "west". We store the valid directions in a static instance variable `VALID_DIRECTIONS`. Notice that once this variable is defined, it is a good idea to use it in `Organism`, and its subclasses, instead of hard-coding "north", "south", etc. Also note that this instance variable is `final`: it means once created, it cannot be modified later (it is a constant).

We start you off with a constructor that accepts the name, coordinates, movement speed, and direction as parameters.

Each `Organism` can move; we also have provided a method `move` that makes the `Organism` change location. It changes the `(x, y)` values according to the movement speed and direction.

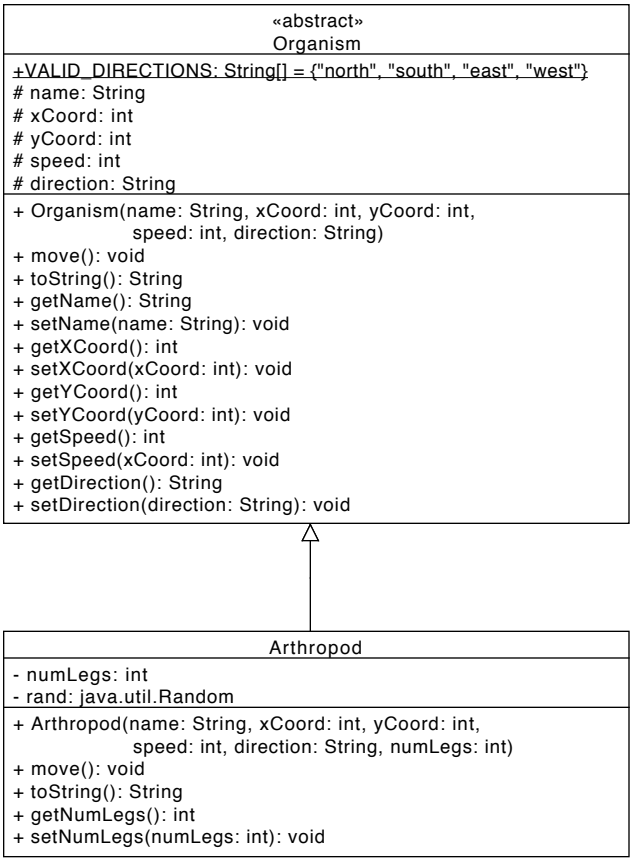
- Write a `toString` method that returns the name and coordinates as a `String`.
- Have the IDE generate getters and setters for the instance variables.

4 A subclass of `Organism`

Switch roles!

Not all real organisms behave like our generic `Organism`. Crabs walk, green algae float, and mussels swim — though not the same way fish do. Insects, spiders, and crabs are all *arthropods*, which are segmented creatures that have an external skeleton and jointed limbs.

Write an `Arthropod` class that is a subclass of `Organism`.



- Each `Arthropod` has a different number of legs. Add an instance variable to keep track of the number of legs.
- Add a constructor to `Arthropod` that takes the name, coordinates, movement speed, direction, and number of legs as parameters. Recall that the first thing you should do in this constructor is call the constructor you inherited from `Organism`:

  ```
  super(name, xCoord, yCoord, direction, speed);
  ```

  Then have your constructor set the number of legs.
- Generate a getter and a setter for the number of legs.

  Switch roles!
- Override the `toString` method so that, in addition to the name and coordinates, it also reports the number of legs. To do this, you should call the inherited `toString` method and then append the leg information. Recall that to call an inherited method, we use the keyword `super` and the inherited method name:

  ```
  super.toString()
  ```

5 Try it!

Create a class `W4Lab` with a `main` method. Try this:

```
Organism lobster = new Arthropod("Homarus gammarus", 0, 0, 2, "north", 10);
System.out.println(lobster);
lobster.move();
System.out.println(lobster);
```

Discuss with your partner what the output should be and then run it.

6 Random movement

Switch roles!

So far, `Organism`s always move in a pre-determined direction. In class `Arthropod`, override the `move` method. Select a random direction, set the direction, and then call the inherited `move` method to move in that direction. (Do you see why it is a good idea to design your method this way?)

Class `java.util.Random` will help you here. Add a `Random` instance variable to `Arthropod` and, whenever `move` is called, ask the `Random` object for another `int` in the range [0..4) in order to pick a new direction. **Think carefully** about how you are going to use the variable `VALID_DIRECTIONS` of class `Organism` here.

Now modify your `main` method to have `lobster` move 20 times, each time printing the lobster information. The `x` and `y` coordinates should both change.

Show your work (including your Javadocs!) to your TA.

Using a text editor (e.g., `nedit`, `kate`, or `pico`), in your `lab4` directory create a file named `partner.txt`. In that file there should be exactly one line and that line should contain only the CDF username of `s2` (assuming you are using `s1`'s repository during this lab). Add and commit that file to the repository.

Add all new Java files to the repository. Commit all changes. **Do not include the `bin` folder or Eclipse's "hidden" files!**

---

7 Tide Pools — Extra material

Switch roles!

If you get this far, create a class `TidePool` that keeps track of `Organism`s. You'll need a way to add creatures to the pool and have them move. Discuss with your partner a choice of data structure you will use to store the `Organism`s. Write an `add` method that takes an `Organism` as a parameter and adds it to the `TidePool`. Also write a `move` method that tells **all** the creatures inside the `TidePool` to move.

Switch roles!

Write a `toString` method that returns a `String` representation of all creatures in the `TidePool`. (If you're using any kind of `List`, you can just call its `toString` method.)

Now modify your main method to create a `TidePool` object, add several Homarus gammari, and then tell the `TidePool` to move several times, printing the state at the end of every iteration.