COMP2396 - Assignment 2(a)

Due: 28 Feb, 2019 23:55

Introduction

This assignment tests your understanding of **inheritance** and **polymorphism** in Java.

This assignment consists of two parts. You need to implement the **basic requirements** of the program in part (a). In part(b), you have to **extend the program** by submitting **additional** files. Part (b) will be released on 1 March, 2019. **Files submitted in part (b) must work with the files in part (a) submitted on or before 28 Feb 2019. You are not allowed to modify the class hierarchy that you have defined and implemented in part (a).**

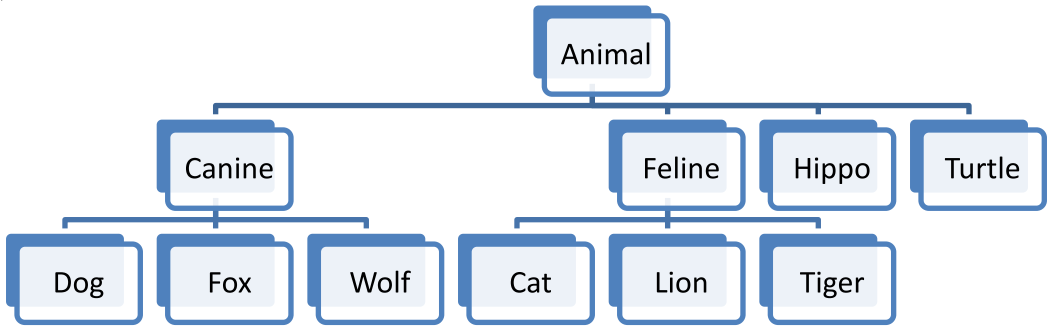
In this assignment, you are required to simulate the wild life in a forest. Program design **will be evaluated** in this assignment. You must make good use of **inheritance** and **polymorphism** to get all marks for this assignment.

You are also required to write **JavaDoc** for all non-private classes and non-private class members. **Programs without JavaDoc will not be marked.**

Task

You need to implement the main program, `Forest.java`, which simulate a forest. A forest is represented by a 2D array of cells. Each cell can be used to hold **one animal** (an object of a sub-class of **Animal** class).

Similar to the **Animal** class hierarchy in the lecture, you are required to implement a hierarchy of animals, as shown below:



When the program starts (i.e., the `main()` method inside `Forest.java`), it will create a forest of size **15x15**.

Once the forest is generated, print the layout of the forest as follow:

```
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
```

The program then displays the following menu:

```
1. Dog (d)
2. Fox (f)
3. Wolf (w)
4. Cat (c)
5. Lion (l)
6. Tiger (t)
7. Hippo (h)
8. Turtle (u)
What would you like to add to the Forest?

Please enter your choice (1-8, or 0 to finish the animal input):
```

When the user enters option 1-8, the program will generate a random location for the **animal** and show the **animal's** moving behavior. The moving behavior of each element will be defined later. E.g. if the user enters 1, the program will show the following:

```
Added Dog at (2,0): Dog is Canine, Canine moves in four directions,
one or two steps a time.

...............
...............
d..............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
```

```
...............
...............
```

The process should be repeated until the user enters `0` to finish the input. The user may choose to add the **same kind of animal** again. Please make sure no two animals on the same location.

Ask for input again if the user enters any other input. You can assume that the user will always input an integral value. Your program does not need to handle non-integer input.

A **dot** indicates an empty space, and a letter indicate an animal at that location. The **first letter** of each of the 8 animals is used to label them except **Turtle** is labeled with a '**u**'.

Consider the case that the user select one for each type of animal, the Forest may look like follow after finishing input:

```
..c............
...t...........
...............
.........d.....
....w..........
...............
..f............
.l.............
h..............
................
...............
...............
...........u...
...............
...............
```

The program then ask the user **press enter** to run a cycle of simulation, type "**print**" to print the Forest or type "**exit**" to leave:

```
Press enter to iterate, type 'print' to print the Forest or 'exit' to
quit:
```

Refer to the tutorial slides on how it can be done.

In every cycle of simulation, all animals in the forest will take turns to move. If the target location is already occupied by another animal, the moving animal will perform an attack. For an animal moves 2 steps and if another animal exists on the path of movement, i.e. on the first step of the moving animal, they will fight. Either one of the animal involved will die at the end of the attack. The survived animal will take up the target location. The dead animal body will be stored in a "dead animal list" together with the location that the animal got killed.

The program should ask for user input after every cycle.

When user types “print”, the program will print the Forest and the list of dead animals and their locations.

```
..c...........
...t..........
..............
.........f....
.....u........
..............
..............
.l............
h.............
.....w........
..............
..............
..............
..............
..............

Dog died at location (3,9)
```

The program terminates only when user types “**exit**”. When the program terminates, it will print the Forest, followed by the list of living animals and their locations, and then followed by the list of dead animals and their locations.

Animal moving

In every cycle of simulation, animals will be moved in the following order: **Cat**, **Dog**, **Fox**, **Hippo**, **Lion**, **Tiger**, **Turtle**, **Wolf**.

Different animals move **randomly** in a different manner according to the following rules:

- **Feline** moves in all **eight directions**, **one** step a time.
- **Canine** moves in **four directions**, **one or two** steps a time.
- **Turtle** has 50% chance stay in the **same position**, and 50% chance move in **four directions**, one step at a time.
- All other animals move in **four directions**, **one** step a time.
- If the animal is located at the corner or on the boundary, it will move to the “movable” positions follow the above rules.

Note that the animals should **not** move out of the forest. When an animal moves from one location to another, your program should print the following information:

```
Animal_type moved from (?, ?) to (?, ?)
```

For example:

```
Fox moved from (2, 0) to (0, 0)
Hippo moved from (3, 0) to (4, 0)
...
```

Animal attacking

When an animal moves to a location that is occupied by another animal, the moving animal will **attack** the occupying animal **before moving**. The result of an attack follows the following rules:

- If a **Feline** attacks a **Canine**, **Feline** wins and **Canine** dies.
- If a **Canine** attacks a **Feline**, there is a 50% chance that one wins and the other dies.
- If a **Lion** attacks a **Hippo**, **Lion** wins and **Hippo** dies.
- If a **Fox** attacks a **Cat**, **Fox** wins and **Cat** dies.
- If any **Animal** attacks a **Turtle**, there is a 20% chance that the **Animal** wins and the **Turtle** dies.
- If a **Turtle** attacks an **Animal**, there is a 50% chance that the **Turtle** wins and the **Animal** dies.
- If two **same** animals meet and fight (Eg. Dog vs Dog), there is a 50% chance that one wins and the other dies.
- For all other cases, the **attacker loses and dies**.
- The dead location is the fight place, and should be stored in a "dead animal list".
- If the animal dies, it disappear and does not occupied the position.
- The animal may attack multiple times if multiple animal blocked its path.

When an animal attacks another animal, your program should print the following information:

```
Attacker_type from (?, ?) attacks occupant_type at (?, ?) and
wins/loses
The Loser dies at (?, ?)
```

The location of the dead animal should be the location of the dead body.

For example:

```
Tiger from (2, 1) attacks Cat at (2, 2) and loses
Tiger dies at (2, 1)
Lion from (4, 6) attacks Hippo at (3, 5) and wins
Hippo dies at (3, 5)
Lion moved from (4, 6) to (3, 5)
```

Note that an animal will move if the attack is successful.

Sample run

A sample run is provided as `sampleRun_revised.txt` and is available on Moodle.

Part (a) requirements

Your program must fulfill the following requirements:

- Define the animal class hierarchy by using **inheritance** and **polymorphism** in the program design. You should use abstract class whenever necessary. Allow **easy extension** to the program by adding more **animals.**
- All **instant variables** must be **private**. Define **getters** to access these variables only if necessary.
- `Forest.java` is the main program that controls the **program flow** and **user interaction**

only. You are required to define the program logic in other classes.

Part (b) requirements

- Part (b) of the assignment will be released **after** you have submitted part (a).
- You will be asked to define some **new animals**.
- You may also be asked to **remove an animal** from the forest.
- You will have to submit the **new classes** for the new animals, as well as a **new main program** (in place of `Forest.java`). These files must work with your files submitted in part (a).
- You are **not allowed** to modify the files submitted in part (a) when you submit part (b).

Marking (65% for part a)

- **35% marks** are given to the **program design.**
  - ➢ You will be awarded all the marks if you are implementing the **move** and **attack** of the animals by making use of **inheritance** and **polymorphism**.
  - ➢ You can check it by avoiding code duplication as much as possible.
  - ➢ **Economy is valuable in coding: the easiest way to ensure a bug-free line of code is not to write the line of code at all.**
- **15% marks** are given to the **functionality** of your program.
  - ➢ You may add **additional classes, instant variables and methods** to the class.
  - ➢ Your program output must be **identical** to what is described in this document, with the exception of the trailing spaces at the end of each line of output.
- **15% marks** are given to your **JavaDoc**. A complete JavaDoc includes documentation of every classes, member fields and methods that are not private. JavaDoc for the main method may be omitted.

Submission:

Please submit all source files (`*.java`) in a single compressed file (in `.zip` or `.7z`) to Moodle. **Late submission is not allowed**.

Do not submit *.class* file.

-- END --