P4: Improving Design Using Collections (44 marks + 4 BONUS)

***Focus:** custom linked lists*

In this part, you will create a custom linked list, called `AnimalList`, and use it instead of standard arrays in `Farm`. Start by downloading the starter_code that comes with this assignment then do as required below.

[30 marks + 4 Bonus] **(A) `AnimalList` class**:

Create this class which represents a custom ***singly*** linked-list where each node contains an element of the type `Animal` and a reference to the ***next*** node. Your implementation should be similar to what we did in the lecture except that this class should **not** be generic (i.e. it can only contain animals).

`AnimalList` should:

- [+0.5] Implement `Iterable<Animal>` interface (to allow iterating over the nodes)
- [+0.5] Implement `Serializable` (to allow saving to binary file)
- **NOT** extend `LinkedList` class or implement `List` interface
- Include an implementation of the following standard linkedlist methods:
  - [+1] `int size()`: returns the number of nodes in the list.
  - [+1] `boolean isEmpty()`: returns `true` if the list is empty, and `false` otherwise.
  - [+1] `void addFirst(Animal animal)`: adds an animal to the beginning of the list.
  - [+1] `void addLast (Animal animal)`: adds an animal to the end of the list.
  - [+2] `void add(int index, Animal animal)`: adds an animal at the given index. If `index` is invalid, throw an `IndexOutOfBoundsException`.
  - [+1] `Animal removeFirst()`: removes the first animal from the list and returns it. If the list is empty, return `null`.
  - [+1] `Animal removeLast()`: removes the last animal from the list and returns it. If the list is empty, return `null`.
  - [+2] `Animal remove(int index)`: removes the animal at the given index and returns it. If `index` is invalid, throw an `IndexOutOfBoundsException`.
  - [+1] `Animal getFirst()`: return first animal in the list. If list is empty, return `null`.
  - [+1] `Animal getLast()`: return last animal in the list. If list is empty, return `null`.
  - [+2] `Animal get(int index)`: return the animal at the given index. If `index` is invalid, throw an `IndexOutOfBoundsException`.
  - [+2] `Animal set(int index, Animal animal)`: replaces the animal at the given index with `animal`. If `index` is invalid, throw an `IndexOutOfBoundsException`.
  - [+2] `String toString()`: returns a string representation of all animals in the list, each animal on one line (see output from `FarmTest` class at the end of this assignment).
  - [+1] `Iterator<Animal> iterator()`: returns an iterator that can be used to iterate over the animal list.
- Include an implementation of the following custom methods:
  - [+2] `AnimalList getHungryAnimals()`: returns a list with references to animals whose energy is < 50. If the farm has no hungry animals, return `null`.

    - [+2] `AnimalList getStarvingAnimals()`: returns a list with references to animals whose energy is < 17. If the farm has no starving animals, return `null`.
    - [+2] `AnimalList getAnimalsInBarn()`:returns a list with references to animals who are located in a the barn. Assume the barn is a rectangular area defined by the two points (450,50) and (550,150) which represent the (x,y) coordinates of two corners in the rectangular area. If no animals are located in the barn, return `null`.
    - [+2] `double getRequiredFood()`: returns the amount of food required to feed all animals until they are full.
- include two inner classes:
    - [+1] **`AnimalNode`** class which defines the nodes of `AnimalList`.
    - [+1] **`MyIterator`** class which is used as part of the implementation of the `iterator()` method above, and it should include one attribute of the `animalNode` type and two methods: `boolean hasNext()` and `Animal next()`.

- [+4 BONUS]: include a method that returns a list of all animals of a specific type. For example, `list.getByType(Cow.class)` will return a list that only contains the farm cows. Note that the argument is a class name, not a string. This method can then be used to get the number of any animal type in the farm. For example: instead of using `myFarm.getNumChicken()`, you can write `myFarm.getAnimals().getByType(Cow.class).size()`.

[11 marks] **(B) `Farm` class**: change the `animals` attribute to the type `AnimalList` (instead of standard Java arrays). Make all necessary changes including the following:

- Modify statements as needed or even completely remove them if they are no longer needed.
    - Example1: use `size()` instead of `length`.
    - Example2: `printAnimals()` should now use the `toString` method of the `AnimalList` class.
    - Example3: remove attributes that are no longer needed.
- Assume there is no limit to the number of animals in the farm.
- Change `add` and `addClone` methods to return `void` instead of `boolean` (since we can add infinite number of animals).
- Don't worry about how to sort animals in `animSort()`. For now, replace the body of this method with a simple statement that prints on the console "not supported yet". We will learn about sorting algorithms in a future lecture.

[1 mark] **(B) Update the `FarmTest` class** with the code given below.

```
Farm myFarm = new Farm("stat.dat");
myFarm.getAnimals().addFirst(new Cow());
myFarm.getAnimals().addLast(new Cow());
myFarm.getAnimals().add(2, new Cow());
Animal temp = myFarm.getAnimals().removeFirst();
System.out.println("removed " + temp.getName());
myFarm.getAnimals().set(0, new Llama());
//you may want to test other linked list methods here

for(Animal a: myFarm.getAnimals())
   a.setEnergy(10+Math.random()*90);
System.out.printf("\nAvailable food before feeding: %.2f\n",
     myFarm.getAvailableFood());
System.out.println("\nInitial list of animals:\n------------------------");
myFarm.printAnimals();
System.out.println("Adding a clone of the second animal\n----------------");
myFarm.addClone(myFarm.getAnimals().get(1));      //change
myFarm.printAnimals();
System.out.println("List of starving animals:\n------------");
System.out.println(myFarm.getAnimals().getStarvingAnimals());
System.out.println("List of hungry animals:\n------------");
System.out.println(myFarm.getAnimals().getHungryAnimals());
System.out.printf("Amount of food needed to feed all animals:
     %.2f\n",myFarm.getAnimals().getRequiredFood());
System.out.println("\nFeeding animals:\n--------------");
myFarm.feedAnimals();
System.out.printf("\nAvailable food after feeding: %.2f\n",
     myFarm.getAvailableFood());
System.out.println("\nFarm summary:\n--------------");
myFarm.printSummary();
myFarm.exit("stat.dat");
```

Above code should print an output similar to the one on the next page. Note that the sample output below may be different from yours based on several (e.g. different energy values).

[2 marks] You will need to delete the stat.dat file that was previously created by your code from P3 (i.e. the one that uses the standard Java arrays), otherwise you will get an error. **Can you explain why? (add your answer as a comment in FarmTest class.**

Submission Instructions

For this part of the project, you need to do the following:

1- Create a Java project of with any name of your choice.
2- Create a package with the name P4 and write your code within this package.
3- Zip the package P4 and name the zipped file as: YourStudentID_P4. e.g., "1234567_P4".
4- Submit the zipped file to Canvas.

Note that you can resubmit an assignment one more time, but the new submission overwrites the old submission and receives a new timestamp.

```
Cannot open file. Using default values!
```

```
removed Cow2
Cow4 says: I'm hungry
Cow3 says: I'm STARVING

Available food before feeding: 1000.00

Initial list of animals:
-------------------------
Llama4  : alive at (0.0,0.0) Energy=80.4
Cow4    : alive at (0.0,0.0) Energy=35.1
Cow1    : alive at (0.0,0.0) Energy=75.7
Llama1  : alive at (0.0,0.0) Energy=50.6
Llama2  : alive at (0.0,0.0) Energy=55.4
Llama3  : alive at (0.0,0.0) Energy=85.3
Cow3    : alive at (0.0,0.0) Energy=15.0

Adding a clone of the second animal
---------------------------------
Llama4  : alive at (0.0,0.0) Energy=80.4
Cow4    : alive at (0.0,0.0) Energy=35.1
Cow1    : alive at (0.0,0.0) Energy=75.7
Llama1  : alive at (0.0,0.0) Energy=50.6
Llama2  : alive at (0.0,0.0) Energy=55.4
Llama3  : alive at (0.0,0.0) Energy=85.3
Cow3    : alive at (0.0,0.0) Energy=15.0
Cow4    : alive at (0.0,0.0) Energy=35.1

List of starving animals:
------------
Cow3    : alive at (0.0,0.0) Energy=15.0

List of hungry animals:
------------
Cow4    : alive at (0.0,0.0) Energy=35.1
Cow3    : alive at (0.0,0.0) Energy=15.0
Cow4    : alive at (0.0,0.0) Energy=35.1

Amount of food needed to feed all animals: 367.52

Feeding animals:
--------------
Llama4 ate 9.0 units
Cow4 ate 20.0 units
Cow1 ate 20.0 units
Llama1 ate 9.0 units
Llama2 ate 9.0 units
Llama3 ate 9.0 units
Cow3 ate 20.0 units
Cow3 says: I'm hungry
Cow4 ate 20.0 units

Available food after feeding: 884.00

Farm summary:
--------------
The farm has:
- 8 animals (0 Chicken, 4 Cows, and 4 Llamas)
- 884.00 units of available food
Data saved successfully to stat.dat.
```