P2: Refining Your OO Design (41 marks)

Focus: *polymorphism, abstract classes, interfaces*

In this part, you will refine the design you created in P1 of the project. You will add more code and change some of the existing code. Start by downloading the starter_code that comes with this assignment then do as required below.

[6 marks] **(A) the `Animal` class**:

- [+1] Change `Animal` to an abstract class.
  [+2] Explain in a comment in your program why it makes sense for this class to be abstract.
- [+1] Change `sound()` to an abstract method.
  [+2] Explain in a comment why it makes sense for this method to be abstract. Why don't we just remove it from `Animal` and keep it in the subclasses?

[34 marks] **(B) the `Farm` class**:

- [+1] change your code so that the `animals` array will have enough space for a 100 animals. (No need to add a 100 animals in the `Farm`'s constructor – see the next two points).
- [+4] Add a method `boolean add(Animal anim)` that adds an animal to the next empty spot in the animals array. For example, when you first create `animals`, it will have a 100 empty spots (i.e. all `null`). The code below adds a new chicken at index 0 and a cow at index 1. The remaining spots will remain empty (i.e. equal to `null`).
  ```
          myFarm.add(new Chicken());
          myFarm.add(new Cow());
  ```
  The `add` method should return `true` when the given animal is added successfully, and `false` when the farm is full (i.e. no empty spots in `animals` array).
- [+1] modify the `Farm`'s constructor to use the above `add` method in order to add a chicken, a cow, and two llamas. The `animals` array should now have exactly 4 animals and 96 empty spots.
- [+3] modify the `getAnimals` method so that it returns an array with existing animals only (i.e. don't return the full `animals` array if it has empty spots). For example, if you have only 4 animals in your farm, `getAnimals` will return a new array of the size 4 with these 4 animals.
- [+4] Add a method `void animSort()` that sorts the animals in the `animals` array based on their `energy`. You must use the `Arrays.sort` method in your implementation. For example, suppose that
  ```
    animals = [Cow1, Chicken1, Llama1, null, null, …, null]
  ```
  and assume that the order based on the energy is `Llama1 < Chicken1 < Cow1`. Then, `animSort` would change `animals` to:
  ```
    animals = [Llama1, Chicken1, Cow1, null, null, …, null]
  ```
  Note that `Arrays.sort` cannot sort `null` values.

  [+4] Implement any changes outside the `Farm` class in order for `animSort()` to work properly.

- [+2] Add a method `boolean addClone(Animal anim)` that clones the given animal, `anim`, and adds it to the `animals` array. For example, if `animals` has 4 animals and we call `myFarm.addClone(animals[2])`, a clone of `animals[2]` would be created and added to `animals` at the next available spot. The method `addClone` returns `true` if the animal is added successfully to `animals` and `false` if the farm is full.

  [+4] Implement any changes outside the `Farm` class in order for `addClone` to work properly.

- [+3] Add a method `printAnimals` that will print the list of animals currently living in the farm. For example, if `myFarm` has only two animals, then `myFarm.printAnimals()` may print this output:

```
Chicken1: alive at (0.0,0.0) Energy=28.9
Cow1    : alive at (0.0,0.0) Energy=87.5
```

- [+4] Add a method `int getNumChicken()` that returns the number of chicken in the farm.

  [+2] Add two more methods, `getNumCows()` and `getNumLlamas()` that return the number of cows and llamas on the farm. (hint: use `instanceof`).

- [+2] Add a method void `printSummary()` that prints the total number of animals, the number of each animal type, and the amount of available food (see sample output in part C below).

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

```
public class FarmTest {
   public static void main(String[] args) throws CloneNotSupportedException {
      Farm myFarm = new Farm();
      for(Animal a: myFarm.getAnimals())
         a.setEnergy(Math.random()*100);
      System.out.println("\nInitial list of animals:\n-------------------------");
      myFarm.printAnimals();
      System.out.println("\nAdding a clone of the second animal\n---------------------
--------------");
      myFarm.addClone(myFarm.getAnimals()[1]);
      myFarm.printAnimals();
      System.out.println("\nAfter SORTING:\n--------------");
      myFarm.animSort();
      myFarm.printAnimals();
      System.out.println("\nFarm summary:\n--------------");
      myFarm.printSummary();
   }
}
```

Above code should print an output similar to the one below. Note that this output may be slightly different based on several factors such as your animals' energy and the available food in the farm.

```
Chicken1 says: I'm hungry

Initial list of animals:
-------------------------
Chicken1: alive at (0.0,0.0) Energy=44.5
Cow1    : alive at (0.0,0.0) Energy=98.2
Llama1  : alive at (0.0,0.0) Energy=72.4
Llama2  : alive at (0.0,0.0) Energy=94.3

Adding a clone of the second animal
-----------------------------------
Chicken1: alive at (0.0,0.0) Energy=44.5
Cow1    : alive at (0.0,0.0) Energy=98.2
Llama1  : alive at (0.0,0.0) Energy=72.4
Llama2  : alive at (0.0,0.0) Energy=94.3
Cow1    : alive at (0.0,0.0) Energy=98.2

After SORTING:
--------------
Chicken1: alive at (0.0,0.0) Energy=44.5
Llama1  : alive at (0.0,0.0) Energy=72.4
Llama2  : alive at (0.0,0.0) Energy=94.3
Cow1    : alive at (0.0,0.0) Energy=98.2
Cow1    : alive at (0.0,0.0) Energy=98.2

Farm summary:
--------------
The farm has:
- 5 animals (1 Chicken, 2 Cows, and 2 Llamas)
- 1000 units of available food
```

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 P2 and write your code within this package.

3- Zip the package P2 and name the zipped file as: **YourStudentID_P2**. e.g., “1234567_P2”.

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.