CSE 1142 - COMPUTER PROGRAMMING II
Programming Assignment # 1
DUE DATE: 14/03/2018 - 23:59 (No extension)

In this homework, you will implement an animal farm simulator program with an object-oriented approach.

1. Implement an `Animal` class with the following UML diagram.

Animal	
-	name : String
-	legNumber: int
-	age: int
-	pregnancyPerYear: int
-	numberOfOffsprings: int
+	Animal (name: String, age: int)
+	sayGreeting() : void
+	reproduce() : void
+	toString() : String

- An Animal object represents an animal with a unique name, leg number, age, pregnancy per year and number of offsprings per pregnancy.
- `sayGreeting` method should print "`Have nothing to say!`" on the screen.
- `reproduce` method should print "`None of your business!!`" on the screen.
- `toString` method returns a string containing name, age and leg number of the Animal. An example string as the following:

```
My name is Cinnamon!
I am 3 years old!
I have 4 legs!
```

- Access and modifier methods are not shown in the UML diagram for simplicity. However you are supposed to implement these methods as well.

2. Implement a `Bird` class with the following UML diagram.

Bird	
+	Bird (name:String, age:int)
+	fly(): void
+	reproduce(): void
+	omnivore(): void

- Bird class extends Animal class. In Bird class's constructor, you are supposed to call the Animal class's constructor.

- `fly` method prints “`I can fly to the endless skies!`” on the screen.
- You should override `reproduce` method to print “`I lay eggs!`” on the screen.
- `omnivore` method prints “`I can eat everything!`” on the screen.

3. Implement a `Mammal` class with the following UML diagram.

Mammal	
+	Mammal (name:String, age:int)
+	walk(): void
+	reproduce(): void
+	herbivore(): void

- Mammal class extends Animal class. In Mammal class's constructor, you are supposed to call the Animal class's constructor.
- `walk` method prints “`I can walk to the far away lands!`” on the screen.
- You should override `reproduce` method to print “`I give birth!`” on the screen.
- `herbivore` method prints “`I eat plants only!`” on the screen.

4. Implement a `Chicken` class with the following UML diagram.

Chicken	
-	count : int
+	Chicken(name:String, age:int)
+	sayGreeting(): void
+	getCount():int
+	decrementCount(): void

- A `Chicken` object represents a real-life chicken. It extends Bird class. In Chicken class's constructor, you are supposed to call the super class's constructor, set leg number to be 2, number of offsprings to be 1 and pregnancy per year to be 200.
- `count` static data field keeps the number of chickens created. You should change it appropriately.
- You should override `sayGreetings` method to print “`I have nothing to say other than I am against Pigs!`” on the screen.
- `getCount` method is the accessor method for the count data field.
- `decrementCount` method decrements the count data field.

5. Implement a `Donkey` class with the following UML diagram.

	Donkey
-	count : int = 0
+	Donkey(name:String, age:int)
+	sayGreeting(): void
+	getCount():int
+	decrementCount(): void

- A `Donkey` object represents a real-life donkey. It extends Mammal class. In Donkey class's constructor, you are supposed to call the super class's constructor, set leg number to be 4, number of offsprings to be 1 and pregnancy per year to be 1.
- `count` static data field keeps the number of donkeys created. You should change it appropriately.
- You should override `sayGreetings` method to print "`Life will go on as it has always gone –that is, badly!`" on the screen.
- `getCount` method is the accessor method for the count data field.
- `decrementCount` method decrements the count data field.

6. Implement a `Horse` class with the following UML diagram.

	Horse
-	count : int = 0
+	Horse(name:String, age:int)
+	sayGreeting(): void
+	getCount():int
+	decrementCount(): void

- A `Horse` object represents a real-life horse. It extends Mammal class. In Horse class's constructor, you are supposed to call the super class's constructor, set leg number to be 4, number of offsprings to be 1 and pregnancy per year to be 1.
- `count` static data field keeps the number of horses created. You should change it appropriately.
- You should override `sayGreetings` method to print "`I will work harder!`" on the screen.
- `getCount` method is the accessor method for the count data field.
- `decrementCount` method decrements the count data field.

7. Implement a `Pig` class with the following UML diagram.

Pig	
-	count : int = 0
+	Pig(name:String, age:int)
+	sayGreeting(): void
+	getCount():int
+	decrementCount(): void

- A `Pig` object represents a real-life pig. It extends Mammal class. In Pig class's constructor, you are supposed to call the super class's constructor, set leg number to be 4, number of offsprings to be 12 and pregnancy per year to be 2.
- `count` static data field keeps the number of pigs created. You should change it appropriately.
- You should override `sayGreetings` method to print "`All animals are equal, but some animals are more equal than others!`" on the screen.
- `getCount` method is the accessor method for the count data field.
- `decrementCount` method decrements the count data field.

8. Implement a `Raven` class with the following UML diagram.

Raven	
-	count : int = 0
+	Raven(name:String, age:int)
+	sayGreeting(): void
+	getCount():int
+	decrementCount(): void

- A `Raven` object represents a real-life raven. It extends Bird class. In Raven class's constructor, you are supposed to call the super class's constructor, set leg number to be 2, number of offsprings to be 5 and pregnancy per year to be 1.
- `count` static data field keeps the number of ravens created. You should change it appropriately.
- You should override `sayGreetings` method to print "`A happy country where we poor animals shall rest forever!`" on the screen.
- `getCount` method is the accessor method for the count data field.
- `decrementCount` method decrements the count data field.

9. Implement a `Sheep` class with the following UML diagram.

	Sheep
-	count : int = 0
+	Sheep(name:String, age:int)
+	sayGreeting(): void
+	getCount():int
+	decrementCount(): void

- A `Sheep` object represents a real-life sheep. It extends Mammal class. In Sheep class's constructor, you are supposed to call the super class's constructor, set leg number to be 4, number of offsprings to be 1 and pregnancy per year to be 1.
- `count` static data field keeps the number of sheep created. You should change it appropriately.
- You should override `sayGreetings` method to print "`Four legs good, two legs better!`" on the screen.
- `getCount` method is the accessor method for the count data field.
- `decrementCount` method decrements the count data field.

10. Implement an `AnimalFarm` class with the following UML diagram.

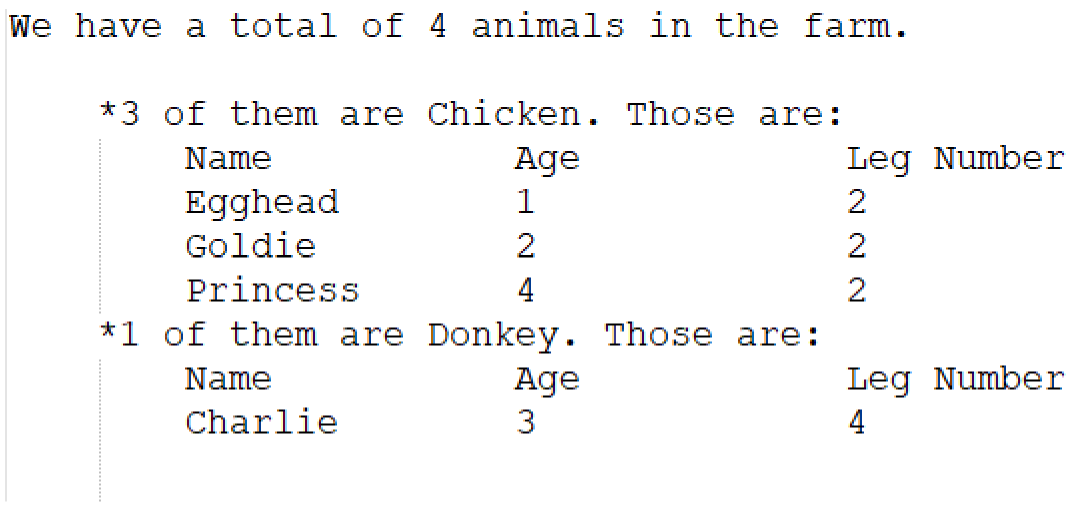
	AnimalFarm
-	animalList : Arraylist<Animal>
-	animalNames : ArrayList<String>
-	CAPACITY: int
-	numberOfAnimals : int = 0
+	AnimalFarm(capacity:int)
+	getNumberOfAnimals():int
+	addAnimal(animal:Animal):boolean
+	removeAnimal(name:String):Boolean
+	printAllAnimalGreetings():void
+	printOneAnimalGreeting(animal:Animal):void
+	printAllAnimalNames():void
+	printOneAnimalName(animal:Animal):void
+	printAllAnimals():void
+	nextYearPopulationForecast():int
+	animalMovements():void
+	eatingHabits():void
+	sortAlphabetically():void
+	sortBasedOnLegNumber():void
+	sortBasedOnAge():void
+	searchBasedOnName(name: String):void
+	searchBasedOnAge(age:int):void
+	printReport(filename:String)

- An `AnimalFarm` object represents an animal farm containing animals.
- `animalList` is an ArrayList of Animal objects. It contains all animals in the animal farm.
- `animalName` is an ArrayList of String objects. It contains all animals' names in the animal farm.
- `CAPACITY` represents the capacity of the animal farm.
- `numberOfAnimals` represent the number of animals the farm has.
- In the constructor of the `AnimalFarm` class, an animal farm with the given capacity has to be created.
- `getNumberOfAnimals` is the accessor for the `numberOfAnimals` field.
- `addAnimal` method adds the given animal to the `animalList` only if
    - there is enough room in the farm, and
    - there is no animal with the same name.

It also adds the animal name to the `animalName` for further check. It returns true if the animal and its name are successfully added to the ArrayLists, false otherwise. If the user tries to enter an animal with the same name, then `addAnimal` method throws an `IllegalNameException` you will implement.

- `removeAnimal` method removes an animal from the `animalList` and `animalName`. It also decrements the counter of either Chicken, Donkey, Horse, Pig, Raven, or Sheep classes depending on the type of the removed animal. It returns true if the animal and its name are successfully removed from the ArrayLists, false otherwise.
- `printAllAnimalGreetings` method iterates on the `animalList` and calls `printOneAnimalGreeting` method with each animal on the list.
- `printOneAnimalGreeting` method calls the `sayGreetings` method of the animal.
- `printAllAnimalNames` method iterates on the `animalList` and calls `printOneAnimalName` method with each animal on the list.
- `printOneAnimalName` method prints the `name` of the animal.
- `printAllAnimals` method iterates on the `animalList` and calls `toString` method of each animal.
- `nextYearPopulationForecast` method calculates the expected population for the next year. For this purpose, it has to use the `numberOfOffsprings` and `pregnancyPerYear` for every animal in the animal farm.
- `animalMovements` method iterates on the `animalList` and if the animal is of type `Bird`, calls `fly` method of the animal. If the animal is of type `Mammal`, however, it calls the `walk` method.
- `eatingHabits` method iterates on the `animalList` and if the animal is of type `Bird`, calls `omnivore` method of the animal. If the animal is of type `Mammal`, however, it calls the `herbivore` method.
- `sortAlphabetically` method sorts the animals alphabetically based on their names **without** changing the original `animalList` and prints the sorted version on the screen.
- `sortBasedOnLegNumber` method sorts the animals increasingly based on their leg numbers **without** changing the original `animalList` and prints the sorted version on the screen.

- `sortBasedOnAge` method sorts the animals increasingly based on their ages **without** changing the original `animalList` and prints the sorted version on the screen.
- `searchBasedOnName` method finds and prints the animal with the given name.
- `searchBasedOnAge` method finds and prints the animals with the given age.
- `printReport` method prints the information about the animal farm on a file in the following format:



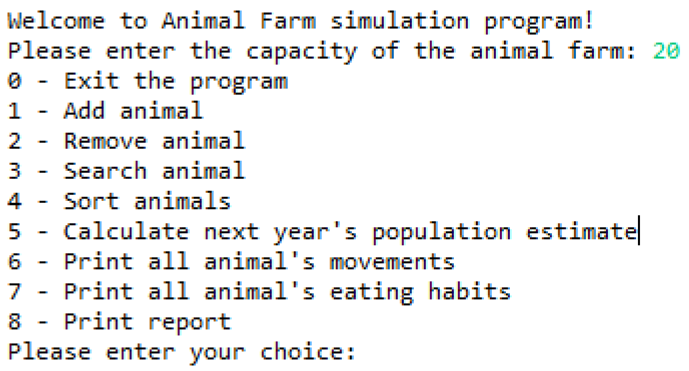
```
We have a total of 4 animals in the farm.

    *3 of them are Chicken. Those are:
        Name           Age            Leg Number
        Egghead        1              2
        Goldie         2              2
        Princess       4              2
    *1 of them are Donkey. Those are:
        Name           Age            Leg Number
        Charlie        3              4
```

First, you will write the total number of animals in the animal farm. Then you will print every animal type separately, and print name, age and leg number attributes of each animal. You should follow the format above.

11. Implement an `IllegalNameException` class that extends Exception class. This exception will be thrown when the user tries to add an animal with an existing name.

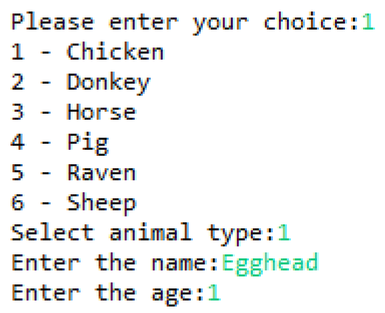
12. Write a test program with the following specifications:

- First, it has to ask for the capacity of the animal farm and create an animal farm object with given value as capacity.
- Then it will print a menu as follows to perform operations:



```
Welcome to Animal Farm simulation program!
Please enter the capacity of the animal farm: 20
0 - Exit the program
1 - Add animal
2 - Remove animal
3 - Search animal
4 - Sort animals
5 - Calculate next year's population estimate
6 - Print all animal's movements
7 - Print all animal's eating habits
8 - Print report
Please enter your choice:
```

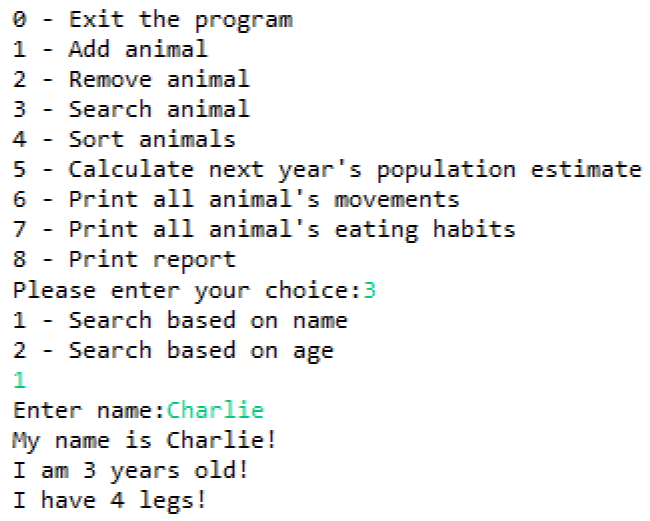
- If the user enters 0, the program will terminate.
- If the user enters 1, you will print another menu for user to choose animal type (Chicken,Pig, Donkey, etc.). After user chooses the animal type, you will ask for name and age of the animal and create an appropriate type animal with specified information and call `addAnimal` method of the `AnimalFarm` class. This is also where you will handle a possible exception. For example:



```
Please enter your choice:1
1 - Chicken
2 - Donkey
3 - Horse
4 - Pig
5 - Raven
6 - Sheep
Select animal type:1
Enter the name:Egghead
Enter the age:1
```

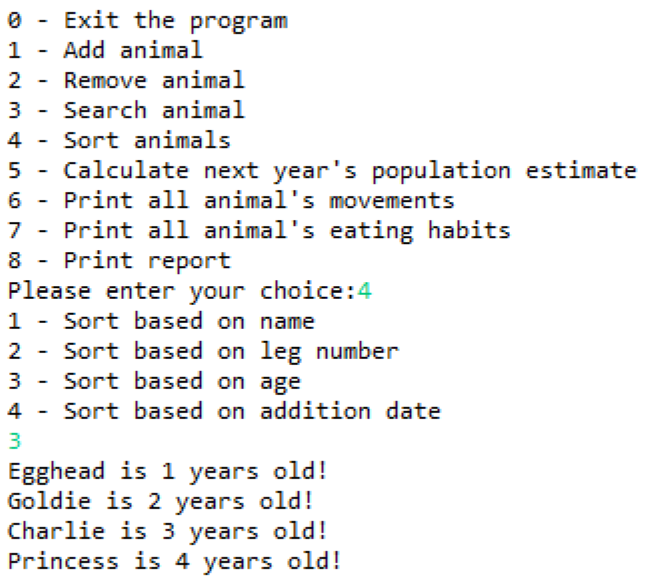
- If user enters 2, you will ask the user the name of the animal and call the `removeAnimal` method of the `AnimalFarm` class.

- If the user enters 3, you will print another menu for user to choose in between a search based on name or age. According to the user's choice, you will either ask for name or age then call either `searchBasedOnName` or `searchBasedOnAge` method of the `AnimalFarm` class. For example:

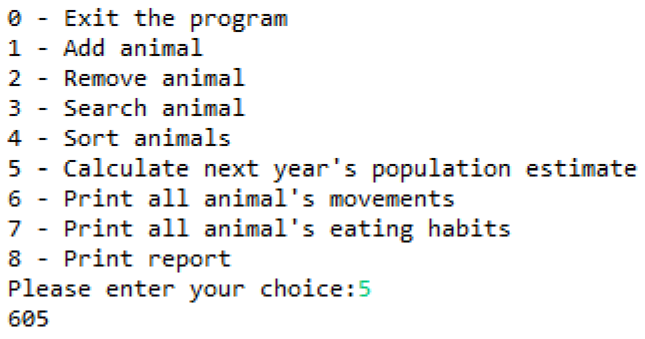


```
0 - Exit the program
1 - Add animal
2 - Remove animal
3 - Search animal
4 - Sort animals
5 - Calculate next year's population estimate
6 - Print all animal's movements
7 - Print all animal's eating habits
8 - Print report
Please enter your choice:3
1 - Search based on name
2 - Search based on age
1
Enter name:Charlie
My name is Charlie!
I am 3 years old!
I have 4 legs!
```

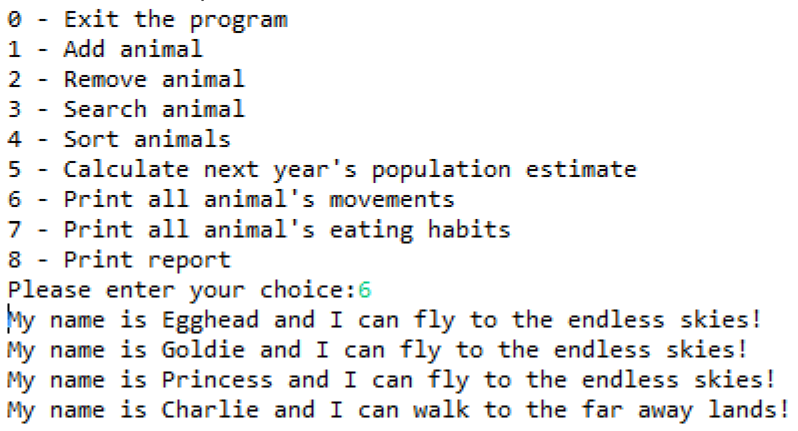
- If the user enters 4, you will print another menu for user to choose in between a sort based on name, leg number, age or addition date. According to the user's choice, you will call either `sortAlphabetically`, `sortBasedOnLegNumber`, `searchBasedOnAge` or `printAllAnimals` method of the `AnimalFarm` class. For example:


```
0 - Exit the program
1 - Add animal
2 - Remove animal
3 - Search animal
4 - Sort animals
5 - Calculate next year's population estimate
6 - Print all animal's movements
7 - Print all animal's eating habits
8 - Print report
Please enter your choice:4
1 - Sort based on name
2 - Sort based on leg number
3 - Sort based on age
4 - Sort based on addition date
3
Egghead is 1 years old!
Goldie is 2 years old!
Charlie is 3 years old!
Princess is 4 years old!
```

- If the user enters 5, you will call the `nextYearPopulationForecast` method of the `AnimalFarm` class. For example:


```
0 - Exit the program
1 - Add animal
2 - Remove animal
3 - Search animal
4 - Sort animals
5 - Calculate next year's population estimate
6 - Print all animal's movements
7 - Print all animal's eating habits
8 - Print report
Please enter your choice:5
605
```

- If the user enters 6, you will call the `animalMovements` method of the `AnimalFarm` class. For example:


```
0 - Exit the program
1 - Add animal
2 - Remove animal
3 - Search animal
4 - Sort animals
5 - Calculate next year's population estimate
6 - Print all animal's movements
7 - Print all animal's eating habits
8 - Print report
Please enter your choice:6
My name is Egghead and I can fly to the endless skies!
My name is Goldie and I can fly to the endless skies!
My name is Princess and I can fly to the endless skies!
My name is Charlie and I can walk to the far away lands!
```

- If the user enters 7, you will call the `eatingHabits` method of the `AnimalFarm` class. For example:

```
0 - Exit the program
1 - Add animal
2 - Remove animal
3 - Search animal
4 - Sort animals
5 - Calculate next year's population estimate
6 - Print all animal's movements
7 - Print all animal's eating habits
8 - Print report
Please enter your choice:7
My name is Egghead and I can eat everything!
My name is Goldie and I can eat everything!
My name is Princess and I can eat everything!
My name is Charlie and I eat plants only!
```

- If the user enters 8, you will ask the user for a filename and call the `printReport` method of the `AnimalFarm` class. For example:

```
0 - Exit the program
1 - Add animal
2 - Remove animal
3 - Search animal
4 - Sort animals
5 - Calculate next year's population estimate
6 - Print all animal's movements
7 - Print all animal's eating habits
8 - Print report
Please enter your choice:8
Enter filename:animalFarmOut.txt
```

After each operation, this menu will be printed on the screen over again, until the user enters 0 to exit the program.

You are expected to use the same class and method names in your implementation.

Submission Instructions

Please zip and submit your files using filename YourNumberHW1.zip
(ex: 150713852HW1.zip) to Canvas system (under Assignments tab).
Your zip file should contain the following files:

1. 12 Java source files: Animal.java, Mammal.java, Bird.Java, Chicken.java, Donkey.java, Horse.java, Pig.java, Raven.java, Sheep.java, AnimalFarm.java, IllegalNameExeption.java, Test.java
2. 12 Java class files: Animal.class, Mammal.class, Bird.Java, Chicken.class, Donkey.class, Horse.class, Pig.class, Raven.class, Sheep.class, AnimalFarm.class, IllegalNameExeption.class, Test.class

Notes:

1. Write a comment at the beginning of each program to explain the purpose of the program.

2. Write your name and student ID as a comment.
3. Include necessary comments to explain your actions.
4. Select meaningful names for your variables and class names.
5. You are allowed to use the materials that you have learned in lectures & labs.
6. Do not use things that you did not learn in the course.
7. In case of any form of **copying and cheating** on solutions, you will get **FF** grade from the course! You should submit your own work. In case of any forms of cheating or copying, both giver and receiver are equally culpable and suffer equal penalties.
   All types of plagiarism will result in FF grade from the course.
8. No late submission will be accepted.

<u>Grading:</u>

- Animal Class → 5 points
- Mammal Class → 5 points
- Bird Class → 5 points
- Chicken Class → 5 points
- Donkey Class → 5 Points
- Horse Class → 5 points
- Pig Class → 5 points
- Raven Class → 5 points
- Sheep Class → 5 points
- AnimalFarm Class → 20 Points
- IllegalNameException Class → 5 points
- Test Class → 5 points
- The execution of test cases → 25 points