COMP202 SPRING 2017

PS

Date: **Thursday, February 16**

***Note that following project will be covered in the PS by the corresponding TA.*

Corresponding TA: Devriş İşler

1 DESCRIPTION

The goal of this PS is for you to refresh your knowledge about Java OOP. The simple examples are given to help you remember important concepts of Java and OOP such as ***Inheritance, Interfaces, Exceptions, Abstract Class*** and so on. In order facilitate this, a toy problem is proposed. The corresponding TA will solve the given problem **with you** in the PS so, don't hesitate to attend ☺, especially if you are not up to speed on OOP.

Suppose we want to keep track of information of cats and dogs which are pets. Consider the following properties for your implementation;

1. There are three classes called ***Cat***, ***Dog*** and ***Pet***. You should decide whether these should be abstract or concrete. (Even though it is easy to see which one it is after reading all the requirements, try to understand why we need an abstract class here.)
2. Every pet has a name, an age, an owner, and a species. Every pet has following functionalities (which corresponds to methods in your implementation)

Methods	***Description***
`eat(String food)`	Pet eats the served food. (You can simply print something on console when it is called depending on the pet type (cat/dog))
`sound()`	Returns sound of the pet

3. A pet has a constructor which takes a name, a species, an owner name, and an age as parameters.
4. A cat has variables as period of vet checkups (such as once in 6 months) and first date of seeing the vet in addition to pet variables. It also has the following additional functionalities;

Methods	***Description***
`ageCheck()`	Checks if the age of the cat is above 2 or not. If it is, throws an age exception which is a custom exception you are required to implement.
`talk()`	Cat talks to its owner. (You can simple output some sentences as your cat would say to you if it can talk as humans do use your imagination ☺)

5. A dog has variables which represents dog's puppy-name in addition to a pet variables
6. Every cat and dog has a friendly-look. So, all these pets should act friendly. Being friendly is described as follows;

Methods	***Description***
`friend(String species)`	Returns an opinion about given species of their kind.
`talk()`	Cat/Dog talks to its owner. (You can simple output some sentences as your cat/dog would say to you if it can talk as humans do ☺ use your imagination ☺)

2 IMPORTANT!! (INPUT/OUTPUT)

- You will be given a text file (named as *Pets.txt*) from which you base your cat and dog objects on. Assume there is a fixed number of cats and dogs which is 4 for this PS. You can create an array size of 4 for each dog and cat.
- The text file is shown in Figure 1.

```
Cat,Boncuk,Margay,David,1,6,01.02.2015
Cat,Bella,Chinese Mountain,Kamil,2,6,07.05.2016
Dog,Oreo,Bush,Karolina,1,Park
Cat,Luna,Pampas,Aisha,3,6,07.07.2015
Dog,Oliver,New Guinea Singing,Alex,2,Chong
Cat,Kitty,Sand,Paulina,4,6,04.06.2015
Dog,Molly,Raccoon Dog,Bartek,3,Lommy
Dog,Simba,Painted,Felix,4,Twitty
```

Figure 1: Pets.txt file

- The structure of the file differs depending on the pet type which is specified (what is the type of pet) in the first word (it is either a cat or dog). You can simply check the first word. For example, if it is a cat, you can proceed based on the cat structure, otherwise proceed based on the dog structure.
  - For a cat: $name, species, ownerName, age, vetDateInterval, firstVetDate$
  - For a dog: $name, species, ownerName, age, nameOfItsPuppy$
- **Do not forget** to create necessary constructors for the ***Cat*** and ***Dog*** classes.
- Remember to implement the necessary methods from the abstract class and the interface.
- Be careful about variables' privacy, and remember how to reach/update the private variables.
- Create a ***Test.java*** file to test your implementation. (reading the text file, creating the objects, handling the exceptions, calling the methods for test)
- Remember to catch the exception in the Test.java file and handle the exception depending on how you want to do it. (For example, you may ask to change the age of the cat).
- Being friendly is an interface. (Try to understand the reasoning)
- The output could be something similar to figure in Figure 2. Please notice it does not call all the method of cat and dog objects. This is just a simple output which gives detailed information about pets and catches the exception of a cat object. You can play with output on you desire to refresh

your memory about Java OOP. It does help you to remember, so play with your code ☺ which we'll do in PS all together.

```
CAT 1: I'm cat Boncuk,1.0 old, and I am living with David I am special because I'm Margay
CAT 2: I'm cat Bella,2.0 old, and I am living with Kamil I am special because I'm Chinese Mountain
CAT 3: I'm cat Luna,3.0 old, and I am living with Aisha I am special because I'm Pampas
CAT 4: I'm cat Kitty,4.0 old, and I am living with Paulina I am special because I'm Sand
DOG 1: I'm dog Oreo,1.0 old, and I am living with Karolina I am a friendly Bush
DOG 2: I'm dog Oliver,2.0 old, and I am living with Alex I am a friendly New Guinea Singing
DOG 3: I'm dog Molly,3.0 old, and I am living with Bartek I am a friendly Raccoon Dog
DOG 4: I'm dog Simba,4.0 old, and I am living with Felix I am a friendly Painted
I see an exception because 3.0 is an old age
```

Figure 2: Visualization of Test.java output.

Have fun implementing the project ☺