✔ Done

In this tutorial, you will be altering the functionality of Pinkman's app, extending it to allow users to submit an adoption request for cats.

Pinkman is loving your new changes, and has sent you following message:

"Hi, I can't get over how awesome the app is! It is saving me so much time chasing people up for exactly what they want, and I'm getting so many requests out of work hours, my business has doubled! Given this blazing success, I'd be stoked if you could extend it to cats too! Pretty much all the same requirements apply, i.e. breed, age, sex, desexed, etc. but could you add a hairless/not hairless criteria for cats?

While you're at it, could you please add in 'level of training' for dogs? I had someone bring back a dog because he wouldn't sit, or stop barking and wasn't house trained! I'd like to prevent that, so at the point where the user can choose which dog they want to adopt, could you output the level of training too? I have 4 levels as follows:
NONE: no training at all
BASIC: sit, stay, lay down on instruction
INTERMEDIATE: play fetch, and will stop barking if instructed
ADVANCED: perform tricks - handshake, play dead, jump over a bar

Also output the recommended exercise/walking time for the dog too...and one more thing. Could you make breed selection optional?"

Upon carefully reading his message, you may have noticed that we probably need to create a general pet class, maybe *DreamPet*, and then use inheritance to create a *Dog* class and a *Cat* class. It's probably a good idea, because there's a good chance that he'll want you to add other pets later too! Once we've done that, we can add the dog specific stuff to the *DreamDog* class, and the cat specific stuff to the *DreamCat* class. So, let's get started!

Part 1

1. Using the tutorial solutions from last week as inspiration, create a class named *Pet* to represent the individual, real, pets. This will be almost identical to the old *Dog* class, however, the *DreamDog* field should be changed to *DreamPet*, and remember to change all dog references to pet within the class, e.g. in the constructor and getter, as well as the method that describes the pet.
   NOTE: you may even want to change the *Dog* class to a *Pet* record rather than *Pet* class, as it is no function other than to represent a pet's info, and describe it (maybe even change *getPetDescription* to *toString*?). If you do this, remember to fix any resulting errors in *FindAPet*.
2. Next, using the tutorial solutions from last week, create a class named *DreamPet* to represent the conceptual, or imaginary pet. It should include the following:
   1. Fields common to both cats and dogs (breed, sex, desexed, age range, purebred).
   2. A constructor allowing the fields to be initialised.
   3. Getters for accessing the fields.
   4. A method to describe the pet's generic features.
   5. A method to match the generic pet features against a real pet's generic features.

3. Was there much difference between the old *DreamDog* class and the new *DreamPet* class?

Part 2

1. Next, create a new *DreamDog* class. It should extend *DreamPet*, and include the following:
   1. Two fields - one for level of training, and one for amount of exercise.
      1. HINT: look at Pinkman's new *allPets.txt* file - would the level of training data be suitable for a new *Enum*, e.g., *LevelOfTraining?* (For now, just create an empty *Enum*. Write it up after you've created the *DreamDog* class).
      2. The amount of exercise should probably be a *double* or an *int*

   2. A constructor that initialises both *DreamDog* fields and the superclass fields.
   3. Getters for the two *DreamDog* fields.
   4. A *getDreamPetDescription* method that overrides the superclass method, adding in the level of training and amount of exercise info (in addition to the superclass description).
   5. A *compareDreamPets* method to override the superclass method. Although we haven't added in any other filters (only 2 descriptors, as per Pinkman's instructions) we have to ensure that the *DreamPet* is a *DreamDog* (HINT: use *instanceOf*).

2. Next, analyse the *allPets.txt* file, and fill out the level of training *Enum*. It should contain a *toString* method, that prints a meaningful, 'prettified' message to the user, based on the *Enum* constant relevant to the dog.

Part 3

Next, create a new *DreamCat* class. It should also extend *DreamPet*, and include the following:

1. One field representing whether the cat is hairless or not. This involves consideration of future design changes - maybe he'll ask for a short-haired or long-haired option in the future too, so instead of creating a *boolean isHairless*, create an *Enum* called *Hair* containing *HAIRLESS*, and *NA*.
2. A constructor that initialises both the *DreamCat* and superclass fields.
3. A getter for the field.
4. A *getDreamPetDescription* method that overrides the superclass method, adding the *hair* criteria into the description (in addition to the superclass description).
5. A *compareDreamPets* method to override the superclass method, allowing cats to be compared based on the *hair* field.

   Well done! You have applied inheritance to create 2 new classes, and have applied the polymorphic technique of method overriding to extend the functionality of the *DreamPet* class.

Part 4

Before we can test our new changes, we have to create another 2 classes that will be very similar to *FindADog* and *AllDogs* from last week (call them *FindAPet* and *AllPets* instead). First, create the *AllPets* class. It should contain:

1. a field (an *ArrayList* of *Pet* objects)
2. an *addPet* method, to add a Pet to the ArrayList
3. a *getAllBreeds* method to return a set of all available breeds
    1. this method will be slightly different to that used last week - it should take in a *String* parameter (type, which could equal "*Cat*" or "*Dog*"). If the user chooses *Cat*, then only add the breeds of the *Pet* objects that have a *DreamCat* attributes (HINT: use *instanceOf*). Do the same for *Dog*.
    2. Pinkman requested that breed be optional, so add "*NA*" to the set (we will also have to change the *compareDreamPets* method in *DreamPet* to account for this change - just add *!petCriteria.getBreed().equals("NA")* to the if statement that relates to breed. This will mean that the breed condition is only tested if the user hasn't selected *NA*.
4. a *findMatch* method, to compare each *Pet* in the *ArrayList* to a *DreamPet* object (passed in as an argument).

Part 5

1. Create the *FindAPet* class. It should contain 6 methods:
    1. A *main* method - interacts with the user, and calls methods
    2. A *static* method called *loadPets* - reads the file and returns an *AllPets* database object.
    3. A *static* method called *getUserDetails* - returns *Person* object representing the user.
    4. A *static* method called *getUserCriteria* - returns a *DreamPet* representing the user's desired pet.
    5. A *static* method called *processSearchResults* - allows user to visualise search results and select chosen pet. Handles situation where there are no matches.
    6. A *static* method called *writeAdoptionRequestToFile* to place the user's adoption request

2. First, create the *loadPets* method to load the contents of the *allPets.txt* file.
    1. Create an *AllPets* object, in which *Pet* objects will be stored.
    2. Load the file, and iterate through its contents, splitting the *String*s on comma as was done previously. You will find that most elements are the same as when you were only loading dogs. The elements that are different include type, hairless, training level and exercise minutes.
    3. Create a *DreamPet* object (*DreamCat* if type is cat, otherwise, create a *DreamDog*) initialising it with the extracted data.
    4. Create a *Pet* object, and initialise it with the *Pet* data, as well as the *DreamPet* object.
    5. Ensure that *loadPets* returns the *AllPets* object.

3. Next, copy the *getUserDetails* method over from last week's tutorial solution to enable user input of name, phone number and email.

4. Next, create the *getUserCriteria* method. It should have a *String* parameter that represents the pet type.
    1. Request that the user select a breed (or *NA*), remembering to pass the *type* into *getAllBreeds.*
    2. Request user input for purebred, sex and desexed status as well as *minAge* and *maxAge*, just as you did last week.
    3. Next create an uninitialised *DreamPet* object, outside an *if else* statement
        1. If the type is "*Cat*", request that the user choose whether or not they want a hairless cat, then use the variables containing user input to create a *DreamCat* object, assigning it to the *DreamPet* object.
        2. If the type is "*Dog*", use the variables containing user input to create a *DreamDog* object, assigning it to the *DreamPet* object (HINT: the level of training should be initialised to *null*, and the amount of exercise to *0* - this is poor design we'll address with constructor overloading soon)

5. Next, create the *processSearchResults* method. It should have a *DreamPet* parameter that represents the user's 'dream' cat or dog.
    1. Next, create a *List* of type *Pet* and assign to it the return value of the *allPets.findMatch* method.

   2. As was done last week, iterate over the compatible pets, creating a *String* containing their info. Use a *Map* containing *String* keys (pet name and microchip number) and *Pet* values to store the a mapping of the choices a user has to the actual *Pet* objects.
   3. Use the map keys to populate a drop down list, storing the user's choice in a *String* then send the chosen *Pet* to the *writeAdoptionRequestToFile* method to log the user's adoption request.
   4. Remember to handle the full range of user selections/criteria, i.e. if no pets meet their criteria, or if they choose not to adopt any pets.

6. Next, create the *writeAdoptionRequestToFile* method, which should take in a *Person* object and a *Pet* object. It should be identical to the previous *writeAdoptionRequestToFile* method, except that you should replace *Dog dog* with *Pet pet*.

7. Finally, at the top of the class, create the *main* method:
   1. Use the *loadPets* method to initialise the *AllPets* field (same as last week).
   2. Welcome the user, and ask them whether they want to adopt a cat or a dog, storing their choice in a *String* variable (*type*).
   3. Call the *getUserCriteria* method to request the user enter their search criteria, saving the return value as a *DreamPet* object. Remember to pass in the user's selected type.
   4. Pass the *DreamPet* object into the *processSearchResults* method to enable the user to visualise search results and place an adoption request.

Part 6

Finally, let's improve our *DreamCat* and *DreamDog* classes, by creating overloaded constructors.

1. Let's first look at the *FindAPet getUserCriteria* method, where we create the *DreamPet* object based on user input. In the case that the user picks *Dog*, the *DreamDog* object is created using *null* and *0* values for level of training and amount of exercise. To address this, let's go into *DreamDog*, and create a constructor that doesn't need those arguments. and call that constructor instead.

2. Next, in *FindAPet loadPets*, when we create the *DreamPet* object, again we have *0* arguments (for *minAge* and *maxAge*). This affects both *DreamCat* and *DreamDog*. As such, let's create two new constructors (in *DreamDog* and *DreamCat*) to accommodate creation of a *DreamPet* object without min and max age parameters.
   HINT: you'll need a new constructor for *DreamPet* that doesn't take min and max age too!
3. Notice how the original *DreamDog* constructor has now become redundant, as it is not used anywhere in the program. You can leave it there (just in case Pinkman wants to add level of training and/or amount of exercise as user search criteria) or you can simply delete it to clean up the code.

Part 7

1. Finally, let's test our new app, by entering the following:

```
Cat
NA
Male
Yes
Yes
0
6
Hairless

Matches found!! The following pets meet your criteria:

Yoda (307986756) is a 6 year old Male purebred donskoy.
> Desexed: Yes.
> Hair: Hairless.
> Adoption fee: $180.00

Hipchee (543654675) is a 2 year old Male purebred sphynx.
> Desexed: Yes.
> Hair: Hairless.
> Adoption fee: $120.00

Tigger (147876365) is a 2 year old Male purebred sphynx.
> Desexed: Yes.
> Hair: Hairless.
> Adoption fee: $120.00
```

Try testing other cat preference combinations and options too, to make sure it works in all scenarios!

2. Test the dog option too....

```
Dog
jack russell
Male
Yes
Not applicable
1
7
```

You should get the following output:

```
Matches found!! The following pets meet your criteria:

Roughie (344563396) is a 3 year old Male purebred jack russell.
> Desexed: Yes.
> Level of Training: Play fetch, and will stop barking if instructed.
> Exercise per day: 45 minutes.
> Adoption fee: $60.00

Ripper (894475839) is a 5 year old Male purebred jack russell.
> Desexed: Yes.
> Level of Training: Play fetch, and will stop barking if instructed.
> Exercise per day: 45 minutes.
> Adoption fee: $45.00

Hunter (784758394) is a 3 year old Male purebred jack russell.
> Desexed: Yes.
> Level of Training: Sit, stay, lay down on instruction.
> Exercise per day: 35 minutes.
> Adoption fee: $20.00
```

Select Ripper as your chosen dog and click OK. Next, enter the following:

```
Jesse Morty
0475757575
jmorty@geekmail.com
```

Check that the file *Jesse_Morty_adoption_request.txt* exists in your project, and open it. It should contain the following information:

```
Jesse Morty wishes to adopt Roughie (344563396). Their phone number is 0475757575 and their email address
is jmorty@geekmail.com
```

Try testing other dog preferences too, to make sure all scenarios work.

Well done! You have completed a challenging revamp to Pinkman's new Pet Finder app and added a very useful OO principle to your toolkit.

Last modified: Wednesday, 13 March 2024, 7:21 AM