✔ Done

In this week's tutorial, you will apply the knowledge and skills you have gained to improve the design of Pinkman's Pet Finder. You will first analyse the design of the program in light of further functionality changes, and use further encapsulation and data abstraction to improve the flexibility and reusability of your code.

Pinkman's Pets has requested the following changes to their Pet Finder:

- Make sex optional (user should be able to select *NA*, and if they do, results should display pets of both sexes).
- Enable users to filter results based on adoption fee (min to max price).
- Enable users to search for guinea pigs too (like cats, hair type should be a filter).
- Display the breeds in the dropdown list in alphabetical order

Part 1

Analyse the code base, considering how you would implement the changes requested by Greek Geek, taking into account the following criteria:

- How would you add in the *NA* functionality for *Sex*? Does the current design make it easy?
- Is it necessary to compare criteria for which the user has selected *NA*?
- Maybe a data structure to contain only those criteria relevant to the user would be a good idea? What kind of data structure would be appropriate?
- How easy would it be to add functionality enabling the user to specify a price range for the adoption fee?
- Would adding the price range code result in duplicate code? Would it be possible to delegate the task to a method (and use it for the age range too)?
- How would you add functionality for choosing guinea pigs? Is it straightforward? Is creating a *DreamGuineaPig* class a good design choice? Is there another way?

Part 2

Using the insights gained from your analysis, let's redesign the program. In part 3, we'll add in the new functionality.

1. Before we commence, review the *allPets* file. Notice the addition of short and long hair (not just hairless). As such, revise the *Hair* enum, by adding in *SHORT* and *LONG* to the options. Notice the addition of guinea pig too. Keep these changes in mind as you edit the *FindAPet loadPets* method.
2. Next, create an enum for type of pet (including *CAT* and *DOG*).
3. To enable all level of training data (including *NA*) to be read from the file in the same way, add the constant *NA* to the *LevelOfTraining* enum.
4. Next, go to *FindAPet loadPets*, and use data abstraction to improve the flexibility of your code.
    1. Identify the criteria that are directly compared, in the search for compatible pets (e.g. pet type, purebred, breed, etc.)
    2. Encapsulate these criteria, by creating a *Map* in *loadPets* in which you store key, value pairs, e.g. *Breed: Chihuahua, Sex: Male.*
       HINT 1: Create an *Enum* to contain all the criteria labels (these will be used as keys in the *Map).*
       HINT 2: The *Map* value datatype should be *Object*, allowing you to store any type in the *Map*.
    3. You won't need to separate pets based on type anymore, simply add the type data to the *Map*, e.g.; *Type: Cat.*
       HINT: use a *try-catch* to parse the pet type data as enum, allowing your program to skip reading guinea pigs (because we haven't added *GUINEA_PIG* to the *PetType* enum yet). Although we could simply just add it now, it is important to realise how powerful try-catch blocks are - they allow you to catch an exception, but keep running the program any way.
    4. You won't need the references to *DreamCat* or *DreamDog* anymore...
    5. Once you've added the pet criteria to the *Map*, create a *DreamPet* object, but only pass in the criteria *Map* as the argument. We'll be changing up *DreamPet*, so don't worry about the red line!

5. Next, delete the *DreamDog* and *DreamCat* classes

6. Then open *DreamPet*, and complete the following steps:
    1. Delete all the fields (except for *minAge* and *maxAge)*, along with their getters, and remove all references to the deleted fields from the constructors
    2. Create a new field to represent the *Map* of criteria. Initialise this *Map* in both constructors and create a getter for it. Also create a getter for an individual value (given the key).
    3. Change the *getDreamPetDescription* method so that it:
        1. has a parameter (criteria *Map* representing the user's choices)
        2. iterates through the user's *Map*, using its keys to get the values of *this.DreamPet,* concatenating values together to create a description *String*.
    4. Change *compareDreamPets*, so that it simply iterates through the petCriteria *Map*, checking values for equality.

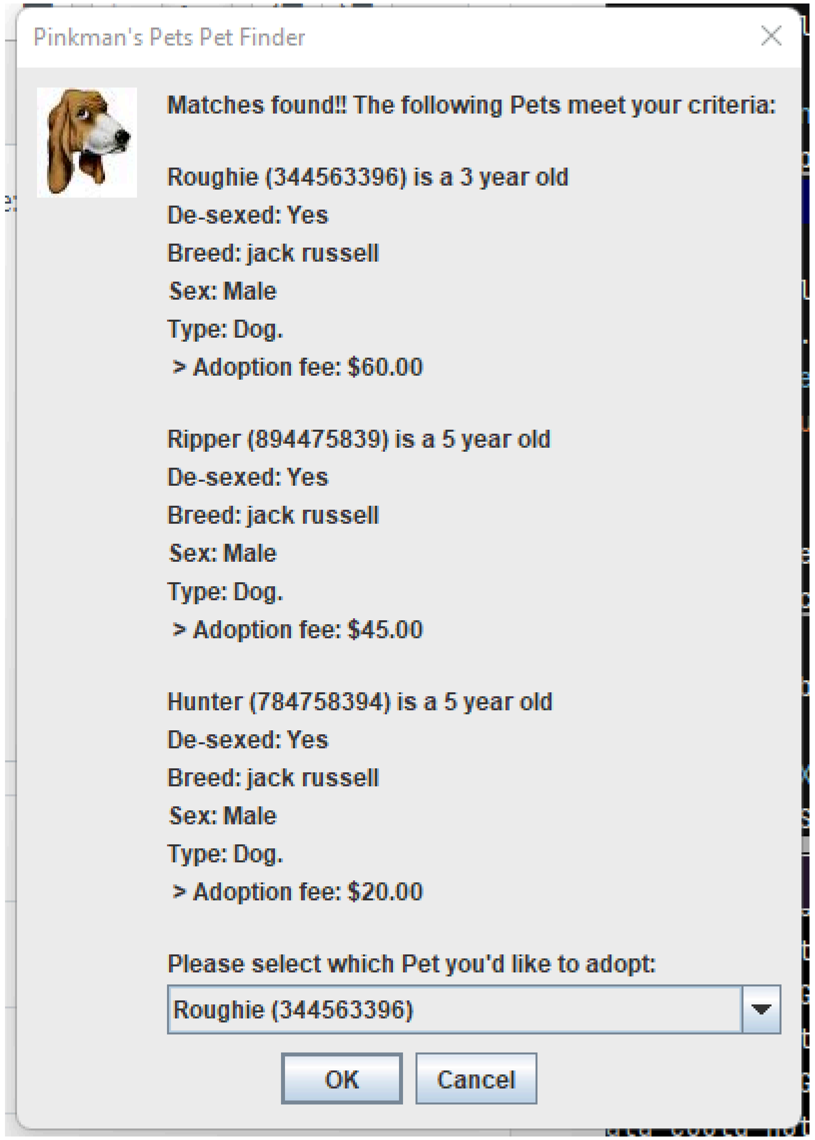
7. Now go back to *FindAPet getUserCriteria:*
    1. Create a new criteria *Map* to contain the features of the user's desired pet (Criteria: Object).
    2. Add the user's selections of type, breed, purebred, sex and desexed to the *Map* **HINT**: only add breed and purebred status to the *Map* if the user doesn't select *NA*.
    3. Remove the references to *DreamCat* and *DreamDog.* Instead, request that the user choose what hair type they want <u>if they select *Cat*</u> as the type of pet they want. Remember to add their choice to the *Map* if it isn't *NA*.
    4. Create and return a *DreamPet* object, passing in the min and max age, as well as the *Map*.
    5. Move the type of pet selection from *main* to *getUserCriteria*, allowing the user to select from the range of pets in the *PetType* enum, rather than the two hardcoded values.
       HINT: you have to remove the parameter from *getUserCriteria*.

8. Next, in *FindAPet*, create a method named *minMaxValues* that has 2 *String* parameters; the first representing a message informing the user to input the lowest value in their desired range, and the second representing a message requesting input for the greatest value in their range.
    1. This method should return an *int* or *double* array containing 2 values, and use *while/try-catch* as before to ensure correct user input.
    2. You'll be using this method to obtain both age and adoption fee ranges.

   3. Call this new method to get the user to input an age range.

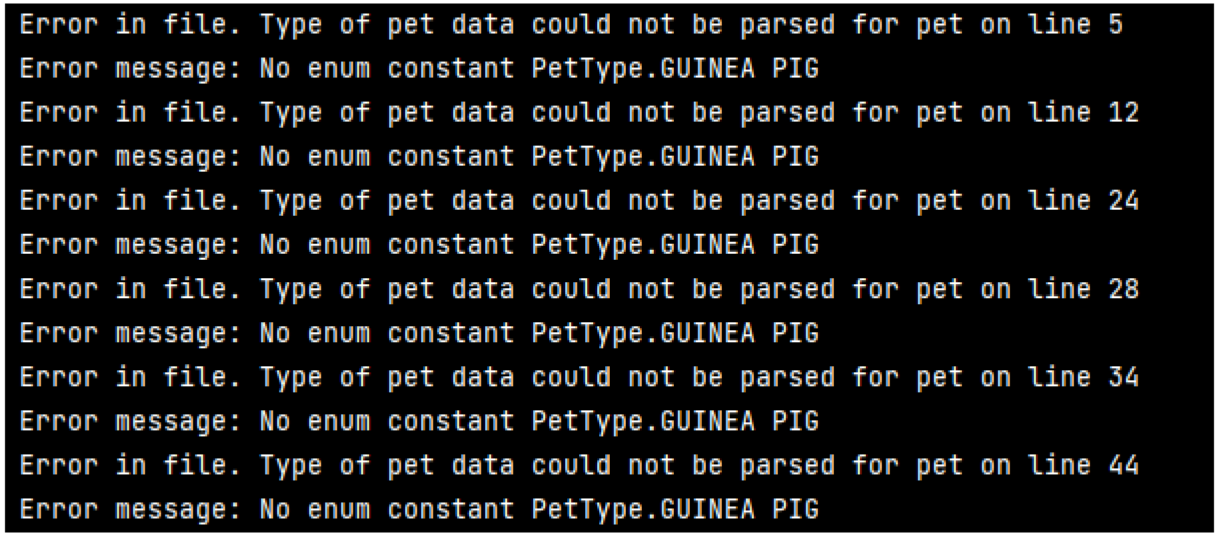
9. Now go to *AllPets getAllBreeds* and remove references to *DreamCat* and *DreamDog* (you'll only need one *if* statement now) - simply compare if the user's chosen type is the same as the *Pet p*'s type, and if it is, add it to the *allBreeds Set*.
10. Finally, in *Pet toString*, add a *Map* parameter, then pass it into the *getDreamPetDescription* method call.
    HINT: also ensure you pass it into the *toString* method call in *FindAPet processSearchResults* too.
11. Now that your program is ready, run the following example:
    Dog
    jack russell
    Male
    Yes (desexed)
    Not applicable/NA (purebred)
    1 (min age)
    7 (max age)

    You should get the following output:



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

Roughie (344563396) is a 3 year old
De-sexed: Yes
Breed: jack russell
Sex: Male
Type: Dog.
 > Adoption fee: $60.00

Ripper (894475839) is a 5 year old
De-sexed: Yes
Breed: jack russell
Sex: Male
Type: Dog.
 > Adoption fee: $45.00

Hunter (784758394) is a 5 year old
De-sexed: Yes
Breed: jack russell
Sex: Male
Type: Dog.
 > Adoption fee: $20.00

Please select which Pet you'd like to adopt:
Roughie (344563396)
```

Your console should appear as follows:



```
Error in file. Type of pet data could not be parsed for pet on line 5
Error message: No enum constant PetType.GUINEA PIG
Error in file. Type of pet data could not be parsed for pet on line 12
Error message: No enum constant PetType.GUINEA PIG
Error in file. Type of pet data could not be parsed for pet on line 24
Error message: No enum constant PetType.GUINEA PIG
Error in file. Type of pet data could not be parsed for pet on line 28
Error message: No enum constant PetType.GUINEA PIG
Error in file. Type of pet data could not be parsed for pet on line 34
Error message: No enum constant PetType.GUINEA PIG
Error in file. Type of pet data could not be parsed for pet on line 44
Error message: No enum constant PetType.GUINEA PIG
```

Part 3

Now let's add the new changes!

1. To make *NA* available for sex:
    1. Add it as an enum constant in *Sex*
    2. In *FindAPet getUserCriteria*, add an *if* statement so that *Sex* is only added to the *Map* if user hasn't selected *NA* (similar to breed and purebred).

2. Next, to enable users to filter based on adoption fee range:
    1. Create two new fields in *DreamPet* called *minFee* and *maxFee*, as well as getters, and constructor initialisation. **ANALYSIS POINT:** Are you noticing code duplication? Think about ways this could be reduced.
    2. In *FindAPet getUserCriteria*, use the new *minMaxValues* method to get the fee range.
    3. Pass the result into the *DreamPet* initialisation.
    4. Next, in *AllPets findMatch*, exclude pets that have adoption fees above or below the user's desired range from the results.

3. Finally, add *GUINEA_PIG* to the *Type* enum (you will also have to go to *FindAPet loadPets*, and ensure that when you read the file, you also replace whitespace with underscore).
4. Ensure that you give users the choice to select hair-type for guinea pigs too!
   That's all you need to do to add a new type of pet! Now, that is FLEXIBLE code!
5. To display the breeds in alphabetical order, simply change the type of set in *AllPets getAllBreeds* to a *TreeSet*.
6. Now test your program by running it on different combinations of input and analysing your results to ensure they are accurate.
   ANALYSIS POINT: run your program, and select *NA* for breed (any pet type will do). Input values for the other filters, and then visualise your results. Notice how the results don't show the pet breed? Is this optimal? Compare how this is done in *SeekAGeek* (where all the real geek data is displayed). Which is better? Is there a way we can achieve a better balance between showing enough/too much information?

Well done! You have completed Tutorial 7!

Last modified: Tuesday, 23 April 2024, 3:10 PM