✔ Done

The Design and Development Process

The easiest way to complete this tutorial is by using the IntelliJ Community Edition IDE. It is available on turing, or you can download and install it onto your own device.

This week you will be introduced to the OO design and development process. You will interpret a request to create a program, composing a use case, a requirements list, and UML diagrams. You will then use your design to code a functional, robust program from scratch. Next, you will revise your program design, improving its functionality and robustness using Java/OO concepts we have learned this week,.

Part 1

1. You have received the following message from Pinkman's Pets:
   "Hi, I'm after someone who can create an app for me. I run a pet adoption centre, and dogs are the main type of pet I adopt out. I'd like for the users of the app to be able to search my database of dogs for a dog that meets their criteria. Criteria should include breed, age, sex, and desexed status. If a dog in my database meets the user's criteria, their info, e.g. name and microchip number should be shown to the user. The user should then be able to choose whether or not to adopt the dog. If they do want to adopt the dog, they must provide their name, phone number and email address (write the adoption request info to a file) so that I can get in touch with them."

2. Compose a use case, starting with the user (person looking for a pet) opening the app and searching for a dog. Ensure you consider alternate pathways, i.e. if no match is found OR match is found but user doesn't want to adopt OR match is found and user wants to adopt.

3. Using your use case, write up a requirements list outlining all functionality that the program MUST have to operate as Pinkman requested.
4. Next, design appropriate UML diagrams to represent the classes in your program.
   HINT: you should have a *Dog* class, a *Person* record (for the person seeking to adopt a dog), an *AllDogs* class and a *FindADog* class. Consider carefully which fields and methods your classes will need (but don't worry about implementation just yet!)

Part 2

1. Use your UML diagrams to help you code the necessary classes.
2. Remember to create a constructor for the *Dog* class. It will be appropriate to use a default constructor for *AllDogs*.
3. You should have 6 fields for the *Dog* class (re-read Pinkman's message if you have anything less!) and getters for each field.
4. Your *Person* record should just contain a header.
5. *AllDogs* should have 1 field (to store all the *Dog* objects), a method to add *Dog* objects to the field, and a method to compare an individual *Dog* against all the *Dogs* in its field, returning a *Dog* that meets their criteria (or *null* if none meet the criteria).
6. *FindADog* should contain:
    1. A method to load the dog data from the *allDogs.txt* file (provided), returning an *AllDogs* object, containing one instance of the *Dog* class per line in the file.
    2. A method to get user search criteria input, returning a *Dog* object that represents the user's 'dream' dog.
    3. A method to get the user to enter their contact details, returning a *Person* object.
    4. A method to write the adoption request info to a file AND
    5. A *main* method that calls the above methods to enable the user to place an adoption request. It should use the *Dog* generated by the method in 6.2 to search the *AllDogs* database (*findMatch*). If a match is returned, the user should be asked if they want to adopt the dog.
    6. If they wish to adopt the dog, call the method in 6.3 and write the contact info (and relevant dog info) to a file named *firstname_lastname_adoption_request.txt* for Pinkman.
    7. Remember to output an appropriate message if no matching dog is found, or the user doesn't want to adopt.

7. Remember to document your code carefully using *@param* and *@return*
8. Remember to use appropriate exception handling.
9. To test whether your program works correctly, input the following:

```
chihuahua
female
yes
2
```

You should obtain the following output:

```
Would you like to adopt Pixy (778435462)?
```

Next, enter the following:

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

You should obtain the following message:

```
"Thank you! Your adoption request has been submitted.  One of our friendly staff will be in touch shortly."
```

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 Pixy (778435462). Their phone number is 0475757575 and their email address is
jmorty@geekmail.com
```

Part 3

You've sent the new program off to Pinkman, but he has come back with the following problem:

"Hi, someone looking for a dog has just done a search and entered the following data: Labrador, Male, Yes, 4, and nothing has come up. However, I know that Rocket meets this criteria. Could you please fix this ASAP? Someone else also accidently bumped 'g' when entering the preferred age, and the whole program crashed. Could you please change it to just ask them to re-enter the age correctly?"

1. Identify what is wrong with the program and fix it, focusing on improving user input and *String* comparison.
   HINT: Maybe creating a couple of *Enums* (for *sex* and *desexed* status), with drop-down lists might help?
2. Did you know how to fix the age problem? Handling a *NumberFormatException* and using an appropriate regex would have helped.
3. Now, analyse the *Person* record. Do you see potential for another number-related exception crashing the program? Handle it in *FindADog*.
4. Perhaps taking advantage of regex to ensure that the phone number and email are valid would be a good idea too?
5. Re-run your program, entering the same information to ensure it still works.

Part 4

Your program is now a lot more robust, however, you've just received the following message from Pinkman:

Hi, great changes! The program is a lot more stable now. I do have a couple of things I want changed though. At the moment, when a user puts details in to adopt a dog, the program only ever returns one dog even if there is more than one dog that meets the criteria. For example, both Roughie and Ripper are 3 year old desexed male Jack Russells - they should both come up in a search.

Also, I'd like for users to be able to specify an age range rather than a specific age. Most people don't really care if a dog is 2 or 3 years old, so this would really maximise the chances of a dog finding their forever home.

And... one more thing. Would it be possible for you to create a drop-down list of the dog breeds I have available, rather than having the user type in the breed they want? This would prevent spelling errors, etc. and allow the user to see the range of breeds I have available.

You should have identified 3 changes: create a dropdown list of available breeds, allow users to search using an age range, and return more than one dog in the results list. Perhaps make relevant changes to your use case, requirements list and UML diagrams.

1. To create the dropdown list of available breeds, add a method in the *AllDogs* class that returns a set (no duplicates) of breeds. Next, call it in a *JOptionPane*. (Remember to check if it works before moving on to the next changes!)
2. Next, implement the age range change. First, create min and max age fields, as well as getters and setters in *Dog*.
3. Next, in *FindADog* remove the code requesting the dog's age, and replace it with code request min and max age. (HINT: set the age argument to -1 in the *Dog* object declaration, and use the setters to set the min and max age)
4. Finally, in *AllDogs*, change *findMatch* to check whether a *Dog*'s age fits within an age range, rather than whether it has the same specific age.
5. Address the problem of the program only returning one option. This will require you to:
    1. Edit *findMatch* in *AllDogs* to return a collection of compatible *Dogs*.
    2. Alter the *main* method in *FindADog* to display all the compatible *Dogs* returned by *findMatch*
    3. Allow the user to select one of them (HINT: use a *JOptionPane* populated with the *Dog*s' names/microchip numbers)
    4. If the user selects a *Dog*, write the adoption request to the file as was done previously.
       HINT: use a *Map* to link the user's choice to the corresponding *Dog* object so that you can pass the *Dog* object into the *writeAdoptionRequestToFile* method.
6. Remember to output an appropriate message if the user chooses not to adopt any of the dogs.
7. Run your completed program. When requested, input the following dog information:

```
jack russell
Male
Yes
1
7
```

You should obtain the following output:

```
Matches found!! The following dogs meet your criteria:
Roughie (344563396) is a 3 year old Male jack russell. Desexed: Yes
Ripper (894475839) is a 3 year old Male jack russell. Desexed: Yes
Hunter (784758394) is a 5 year old Male jack russell. Desexed: Yes
```

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 Ripper (894475839). Their phone number is 0475757575 and their email address is jmorty@geekmail.com
```

Well Done! You have created a functional, robust program tailored to the needs of your customer!

Last modified: Monday, 4 March 2024, 6:31 AM