✔ Done

Java Fundamentals 2 - Arrays and Collections

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.

Question 1

1. Create a class named *DogBreeds*
2. Create a *main* method
3. Within the *main* method, declare and assign the following array:
   String[ ] dogBreeds = {“bulldog”, “dalmatian”, “jack Russell terrier”, “moodle”, “Yorkshire terrier”, “golden retriever”, “BULL terrier”, “pomeranian”, “pit bull terrier”, “Siberian husky”}
4. Use a for loop to iterate through the array, and change all dog breed names to lowercase (e.g., Siberian husky should be siberian husky)
5. Verify your changes by printing to standard output
6. Use a for each loop to iterate through the array and print the available terrier breeds (hint: use the method contains() to see if a breed is a type of terrier)
7. If the dog breed is a pit bull terrier, don’t print it, just skip to the next breed.
8. Your output should be as follows:

```
********ALL BREEDS*********
bulldog
```

```
dalmatian
jack russell terrier
moodle
yorkshire terrier
golden retriever
bull terrier
pomeranian
pit bull terrier
siberian husky

*****TERRIERS*******
jack russell terrier
yorkshire terrier
bull terrier

Process finished with exit code 0
```

Question 2

1. Create a class named *AdoptionRequest.*
2. Create a *main* method.
3. Within the *main* method, use dialog boxes to request user information for:
    1. Desired dog breed
    2. Are they only willing to adopt a purebred (yes/no)
    3. Desired dog sex (male, female or either)
    4. Maximum age (in years) they're willing to adopt (max input = 20 years)
    5. Minimum age (in years) they're willing to adopt (min input = 0 years)

6. Contact details including name and phone number (10 digits/chars)

4. For each of the above, use an appropriate loop to validate user input, requesting repeated user input until it matches your requirements. Ensure you parse numerical input appropriately.
5. Ensure you handle termination of the program (when the user clicks cancel or close).
6. Use exception handling to account for any exceptions, e.g., *NullPointerException* or *NumberFormatException*.

Question 3

1. Create a class name *BreedAvailability*.
2. Create a *main* method.
3. Inside the *main* method, create a *String* array. It must contain the following dog breeds: *Dalmatian, Rottweiler, German Shephard, Jack Russell terrier, Wolfhound, Siberian Husky, Chihuahua, Moodle, Poodle, Labrador, and Maremma.*
4. Create an empty *String* array to contain a user's 5 favourite dog breeds.
5. Use a *Scanner* object to request that the user input their 5 favourite dog breeds, adding them to the array created in step 4.
6. Create a nested for loop to iterate through both arrays, comparing values to determine if any of the 5 favourite breeds are available (Note: ensure that the comparison is not case-sensitive). If a breed is available, print an appropriate message to standard output.
7. Next, sort the array of available dog breeds in alphabetical order (**Hint**: use the *Arrays* class)
8. Print the entire list of available breeds (in one line) to standard output (**Hint**: use the *toString* method from the *Arrays* class).

Question 4

1. Create a class named *BreedAvailabilityV2*.
2. Create a *main* method (throws *IOException*). Alternatively, use a *try-catch* statement to handle the *IOException*.

3. Within the *main* method, load the contents of the file *availableBreeds.txt* (on Moodle under Workshop heading), using the *split* method to separate breed based on comma (,) changing all text to lowercase and saving the result in a *String* array.
4. Create an empty *ArrayList* as of *Strings* follows: *List<String> availableBreeds = new ArrayList<>();*
5. Add the contents of the *String* array into this new *ArrayList*. Next, print the contents of the *ArrayList* to standard output to visualise its contents.
6. Since the file was written, another dalmatian and greyhound have become available. Add them to the *ArrayList,* using the *add* method then print the *ArrayList* to standard output to visualise its contents.
7. Notice that rottweiler was spelled incorrectly in the file – change it to the correct spelling, i.e. from Rotweiler to Rottweiler using the *set* method. Visualise your changes.
8. Use *Collections* to sort the *List* (default is in alphabetical order) then print the *ArrayList* to standard output.
9. Create a *HashSet*, adding the *ArrayList* into the *HashSet* upon declaration, i.e., *Set<String> availableBreedsHashSet = new HashSet<>(availableBreeds);*
10. Next, print the contents of the *HashSet* to standard output. What do you notice?
11. Change the *HashSet* to a *TreeSet* and print the contents of the *TreeSet* to standard output. What do you notice?

Question 5

1. Create a class named *BreedAvailabilityV3*.
2. Create a *main* method (throws *IOException*). Alternatively, use a *try-catch* statement to handle the *IOException*.
3. Within the *main* method, load the contents of the file *availableBreeds.txt* (on Moodle under Workshop heading), splitting on comma (,) changing all text to lowercase and saving the result in a *String* array.
4. Create a *Map* with key type *String* and value type int to store the name of the breed (key) and the number of dogs of that breed that are available (value).
5. Iterate through the breeds stored in the *String* array. On each iteration, use the *containsKey* method to check if the *Map* contains the breed. If it does, increment the value of that breed by 1. If it doesn't, add the breed to the *Map*, with a corresponding value of 1.
6. Print all the key value pairs to standard output.

7. Since the file was written, a greyhound and another dalmatian have become available. Alter the *Map* accordingly. (**HINT**: you'll have to create a new key/value pair for greyhound)
8. Notice that rottweiler was spelled incorrectly in the file – change the key to the correct spelling.
9. Print all the key value pairs to standard output to visualise changes.
10. Next, use a *JOptionPane* input dialog to request that the user input the breed of a dog they wish to adopt.
11. Check if the *Map* contains a key that matches the user input.
12. If it does, use a *JOptionPane* input dialog to ask the user whether they would like to adopt a dog of this breed.
13. If the user would like to adopt the dog, decrement the value (number of dogs) associated with that breed. If the value is now zero, remove the breed from the Map.
14. If the *Map* doesn't contain the user's requested breed, use a *JOptionPane* message dialog to let the user know that their requested breed is not available.
15. Print all the key value pairs to standard output. In the event that the user chooses to adopt a dog, what happens to the relevant key/value pair?

Last modified: Friday, 1 July 2022, 11:58 AM