pet

```
public class Pet {
    protected String name;
    protected int age;
    protected String color;
    protected double weight;
    // Define type 1 to be a dog, 2 to be a cat
    protected int type;
    protected String breed;

    public Pet() {
    }
    public Pet(String name, int age, String color, double weight) {
        this.name = name;
        this.age = age;
        this.color = color;
        this.weight = weight;
    }
    public void speak() {
        System.out.println("I am pet");
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double weight) {
        this.weight = weight;
    }
```

```
    public int getType() {
        return type;
    }

    public void setType(int type) {
        this.type = type;
    }
    public String getBreed() {
        return breed;
    }
    public void setBreed(String breed) {
        this.breed = breed;
    }

    @Override
    public String toString() {
        return getName() + " " + getAge() + " " + getColor() + " " +
getWeight();
    }
    public String toWriteInfo() {
        return "";
    }


    @Override
    public boolean equals(Object obj) {
        if(this == obj) {
            return true;
        }
        if( !(obj  instanceof Pet)) {
            System.out.println("Type Error");
            return false;
        }
        Pet p = (Pet)obj;
        if(name.equals(p.getName()) && age == ((Pet) obj).getAge()
            && color.equals(((Pet) obj).getColor())
                && weight == ((Pet) obj).weight
                && type == ((Pet) obj).getType()
        ) {
            return true;
        }
        return false;
    }
}
```

dog

```
public class Dog extends Pet{

    public Dog(String name, int age, String color, double weight, String breed)
{
        super(name, age, color, weight);
        this.breed = breed;
```

```
        this.type = 1;
    }

    @Override
    public void speak() {
         System.out.println("Woof~ " + "I'm " + name + " , a " + age + "-year-
old " + breed + " dog");
    }


    @Override
    public String toString() {
        return "Type：  dog  " + "Name：" +  name + "  Age：" + age + "  Color："
+ color + "  Weight：" + weight + "  Breed：" + breed;
    }
    @Override
    public String toWriteInfo() {
        // Formatting strings
        return String.format("%-30d | %-15s | %-10d | %-10s | %-10.2f | %-10s",
1, name, age, color, weight, breed);
    }

    @Override
    public boolean equals(Object obj) {
        if(this == obj) {
            return true;
        }
        if( !(obj instanceof Dog)) {
            return false;
//            System.out.println("Type Error");
        }
        if(super.equals(obj) && breed.equals( ((Dog)obj).breed)) {
            return true;
        }
        return false;
    }

}
```

cat

```
public class Cat extends Pet{
    public Cat(String name, int age, String color, double weight, String breed)
{
        super(name, age, color, weight);
        this.breed = breed;
        this.type = 2;
    }

    @Override
    public void speak() {
        System.out.println("Miaow~ " + "I'm " + name + " , a " + age + "-year-
old " + breed + " cat");
    }
```

```
    @Override
    public String toString() {
        return "Type:  Cat  " + "Name: " +  name + "  Age: " + age + "  Color: "
+ color + "  Weight: " + weight + "  Breed: " + breed;
    }

    @Override
    public String toWriteInfo() {
//        return "2 " + getName() + " " + getAge() + " " + getColor() + " " +
getWeight() + " " + getBreed();
        return String.format("%-30d | %-15s | %-10d | %-10s | %-10.2f | %-10s",
2, name, age, color, weight, breed);
    }

    @Override
    public boolean equals(Object obj) {
        if(this == obj) {
            return true;
        }
        if( !(obj instanceof Pet)) {
            return false;
//            System.out.println("Type Error");
        }
        if(super.equals(obj) && breed.equals(((Cat)obj).breed)) {
            return true;
        }
        return false;
    }

}
```

clinic

```
import java.io.*;
import java.util.ArrayList;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;

public class Clinic {
    private ArrayList<Pet> pets;
    private String name = "YH";

    public Clinic() {
        pets = new ArrayList<>();
        // Importing Clinic Information and Pet Information
        readFromClinicsDetails();
        readFromPetDetails();
    }

    public String getName() {
        return name;
```

```
    }

    /**
     * Write the program's data about the clinic into the file
ClinicsDetails.txt
     */
    private void writeToClinicsDetails() {
        // Processing of current clinic information
        // deposit a document
        try (FileWriter writeFile = new FileWriter("ClinicsDetails.txt")) {
            // Writing intuitive content to a file
            writeFile.write("Clinic Information: \n");
            writeFile.write("Clinic Name: " + name + '\n');
            writeFile.write("Number of cats: " + findCountOfCat() + '\n');
            writeFile.write("Number of dogs: " + findCountOfDog() + '\n');
            writeFile.write("Colors that make up the largest percentage of pets:
" + findMoreColor() + '\n');
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    /**
     * Importing information from the file ClinicsDetails.txt
     */
    private void readFromClinicsDetails() {
        try (BufferedReader br = new BufferedReader(new
FileReader("ClinicsDetails.txt"))) {
            String line = br.readLine(); // Read the first line, "Clinic
Information."
            line = br.readLine(); // Read the clinic name line.
            // If it is empty, set the name
            if(line == null) {
                name = "YH";
                return ;
            }
            String[] strArr = line.split(":");
            name = strArr[1].trim();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    /**
     * Write the program pet's data into the PetDetails.txt file
     */
    private void writePetDetails() {
        try(FileWriter writeFile = new FileWriter("PetDetails.txt")) {
            writeFile.write("Pet Information\n");
            // Formatting strings and printing them
            writeFile.write(String.format("%-30s  %-16s  %-11s  %-11s  %-11s
%-11s\n", "Pet type (1 dog, 2 cats)", "Name", "Age", "Color", "Weight", "Breed")
);
            for(Pet pet: pets) {
                writeFile.write(pet.toWriteInfo() + "\n");
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
```

```
        }
    }

    /**
     * Importing information from within the PetDetails.txt file
     */
    private void readFromPetDetails() {
        try (BufferedReader br = new BufferedReader(new
FileReader("PetDetails.txt"))) {
            String line;
            // Read the first line, the pet message line, which is irrelevant
for program reads
            line = br.readLine();
            // Read the second line, which is the content of the intuitive
description of the data written, independent of the program read-in
            line = br.readLine();
            while( (line = br.readLine()) != null) {
                // Reading method
//                "getType() " + getName() + " " + getAge() + " " + getColor() +
" " + getWeight() + " " + getBreed();

                // Cutting according to `|`
                // Remove spaces and perform a type conversion to get the
corresponding type of information
                String[] strArr = line.split("\\|");
                int petType = Integer.parseInt(strArr[0].trim());
                String petName = strArr[1].trim();
                int petAge = Integer.parseInt(strArr[2].trim());
                String petColor = strArr[3].trim();
                double petWeight = Double.parseDouble(strArr[4].trim());
                String petBreed = strArr[5].trim();

                // Instantiate objects based on different pet types and add them
to the clinic
                if(petType == 1) {
                    Dog dog = new Dog(petName, petAge, petColor, petWeight,
petBreed);
                    addPet(dog);
                } else {
                    Cat cat = new Cat(petName, petAge, petColor, petWeight,
petBreed);
                    addPet(cat);
                }
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    /**
     * When the program is about to exit, the information is stored in a file.
     */
    public void fromExeAppForWrite() {
        writePetDetails();
        writeToClinicsDetails();
    }
```

```
    /**
     * Get the number of cats
     * @return count of cats
     */
    private int findCountOfCat() {
        int countOfCat = 0;
        for(Pet pet: pets) {
            if(pet.getType() == 2) ++countOfCat;
        }
        return countOfCat;
    }

    /**
     * Get the number of dogs
     * @return count of dogs
     */
    private int findCountOfDog() {
        int countOfDog = 0;
        for(Pet pet: pets) {
            if(pet.getType() == 1) ++countOfDog;
        }
        return countOfDog;
    }

    /**
     * Get the color that has the most pets in it
     * @return broadest color
     */
    private String findMoreColor() {
        // Mapping colors and quantities with Map
        Map<String, Integer> mapColorToCounts = new TreeMap<>();
        // Define the initial value of the color
        String moreColor = "None";

        // Iterate over pets and remap the number of current pet colors
        for(Pet pet: pets) {
            int nowColorCount = mapColorToCounts.getOrDefault(pet.getColor(),
0);
            mapColorToCounts.put(pet.getColor(), ++nowColorCount);
        }

        // Define the number of pets corresponding to the color
        int maxCount = -1;
        // Record the number of times the maximum value occurs
        int cnt = 0;
        // After converting to a Set, it is traversed with an iterator to update
the color with the maximum number of pets and the corresponding number of pets
        Set<Map.Entry<String,Integer>> entrySet = mapColorToCounts.entrySet();
        for(Map.Entry<String, Integer> iterator: entrySet) {
            if(iterator.getValue() > maxCount) {
                maxCount = iterator.getValue();
                cnt = 1;
            } else if(iterator.getValue() == maxCount) ++cnt;
        }
        // If the maximum value is -1, it means there is no pet, just return it
directly
        if(maxCount == -1) return moreColor;
```

```
        // Record the most occurring color
            // Depending on the number of maxima, the spacers are set to reach
the final output as [ color1, color2, ... color_n]
        moreColor = " [ ";
        for(Map.Entry<String,Integer> iterator: entrySet) {
            if(iterator.getValue() == maxCount) {
                moreColor += iterator.getKey();
                --cnt;
                if(cnt > 0) {
                    moreColor += " , ";
                } else moreColor += " ]";
            }
        }
        return moreColor;
    }

    /**
     * Displaying clinic information
     */
    public void displayClinicsDetails() {
        System.out.println("The name of this clinic:  " + name);
        System.out.println("Number of cat pet registrations:  " +
findCountOfCat());
        System.out.println("Number of dog pet registrations:  " +
findCountOfDog());
        System.out.println("The main colors in pets are:  " + findMoreColor());
    }

    /**
     * Specific Pets
     * @param other Pets to Judge
     * @return Returns true if present, otherwise returns
     */
    public boolean notEmpty(Pet other) {
        for(Pet pet: pets) {
            if(pet.equals(other)) return true;
        }
        return false;
    }

    /**
     * Add Pet
     * @param pet Pets to add
     */
    public void addPet(Pet pet) {
        pets.add(pet);
    }

    /**
     * Delete Pets
     * @param pet Pets to be deleted
     * @return Returns whether the deletion was successful, true - success,
false - failure
     */
    public boolean removePet(Pet pet) {
        return pets.remove(pet);
    }
```

```
    /**
     * Display by pet color
     */
    public void displayPetByColor() {
        // Mapping a color to a list of pets of that color with Map
        Map<String, ArrayList<Pet>> mapPetToColor = new TreeMap<>();
        for(Pet pet: pets) {
            // If the list of pets of the current color is empty, allocate
space.
            ArrayList<Pet> temp = mapPetToColor.getOrDefault(pet.getColor(),
null);
            if(temp == null) {
                temp = new ArrayList<>();
            }
            // Add this pet to the pet list
            temp.add(pet);
            // Updates to the list of pets of that color
            mapPetToColor.put(pet.getColor(), temp);
        }
        // Print out the color and the corresponding color of the pet
        mapPetToColor.forEach( (key, value) -> {
            System.out.println("Color: " + key);
            for(Pet pet: value) {
                System.out.println(pet);
            }
        });
    }

    /**
     * Display by pet type
     * 同理
     */
    public void displayPetByBreed() {
        // Mapping pet types and corresponding pet lists with Map
        Map<Integer, ArrayList<Pet>> mapPetToColor = new TreeMap<>();

        // Remapping the pet list for pet types
        for(Pet pet: pets) {
            ArrayList<Pet> temp = mapPetToColor.getOrDefault(pet.getType(),
null);
            // If null, create a pet list instance
            if(temp == null) {
                temp = new ArrayList<>();
            }
            // Adding pets to the pet list
            temp.add(pet);
            // carry out remapping
            mapPetToColor.put(pet.getType(), temp);
        }
        // Output with foreach statements
        mapPetToColor.forEach( (key, value) -> {
            // Determine if you are currently a dog or cat based on the key
value.
            System.out.println("Pet Category: " + (key == 1 ? "Dog" : "Cat"));
            for(Pet pet: value) {
                System.out.println(pet);
            }
        });
```

```
    }

    /**
     * Default Display
     */
    public void displayDefaultAllPets() {
        for(Pet pet: pets) {
            System.out.println(pet);
        }
    }

    /**
     * Search by name
     * @param name Name to search for
     */
    public void searchPetByName(String name) {
        // Define a boolean value that determines if the pet has already been
found
        boolean ok = false;
        for(Pet pet: pets) {
            // If there is no match, continue
            if( !name.equalsIgnoreCase(pet.getName())) continue;
            // Otherwise, set the boolean ok to true and output speak
            ok = true;
            pet.speak();
        }
        // If the boolean ok is true, it means that the pet in question was
found, making an early return
        if(ok) return ;
        System.out.println("None to be found.");
    }

    /**
     * Search by color
     * @param color Colors to search for
     */
    public void searchPetByColor(String color) {
        // Define a boolean value that determines if the pet has already been
found
        boolean ok = false;
        for(Pet pet: pets) {
            // If there is no match, continue
            if( !color.equalsIgnoreCase(pet.getColor())) continue;
            // Otherwise, set the boolean ok to true and output pet of
information
            ok = true;
            pet.speak();
        }
        // If the boolean ok is true, it means that the pet in question was
found, making an early return
        if(ok) return ;
        System.out.println("None to be found.");
    }
}
```

petmanage

```
import com.sun.xml.internal.bind.v2.TODO;

import java.io.IOException;
import java.util.Scanner;

public class PetManage {
    private Clinic clinic;
    private Scanner sc = new Scanner(System.in);

    // Test
    public static void main(String[] args) throws IOException {
        PetManage pm = new PetManage();
        pm.MainMenu();
    }

    public PetManage() {
        clinic = new Clinic();
    }

    /**
     * The main interface of the pet management system
     */
    public void MainMenu() throws  IOException {
        clinic.displayClinicsDetails();
        int choice = -1;
        do {
            System.out.println("Welcome to " + clinic.getName() + " Veterinary
Clinic");
            System.out.println("1, Add Pet");
            System.out.println("2, Remove Pet");
            System.out.println("3, Showcase");
            System.out.println("4, Search");
            System.out.println("5, Displaying clinic information");
            System.out.println("6, Exit");
            System.out.println("Please enter the option you want to select: ");

            choice = sc.nextInt();
            sc.nextLine();

            switch (choice) {
                case 1:
                    addPetToClinic();
                    break;
                case 2:
                    removePetFromClinic();
                    break;
                case 3:
                    displayMenu();
                    break;
                case 4:
                    searchPetMenu();
                    break;
                case 5:
                    displayClinicsDetails();
```

```
                case 6:
                    clinic.fromExeAppForWrite();
                    break;
                default:
                    System.out.println("Illegal option, please re-enter");
                    break;
            }
        } while(choice != 6);
    }

    /**
     * Get a valid age value
     * @return age of value
     */
    private int getValidAge() {
        System.out.println("Please enter its age: ");
        int age = sc.nextInt();
        sc.nextLine();
        while(age < 0 || age > 100) {
            System.out.println("Please enter a valid age value: ");
            age = sc.nextInt();
            sc.nextLine();
        }
        return age;
    }

    /**
     * Get a valid weight value
     * @return weight of valid
     */
    private double getValidWeight() {
        System.out.println("Please enter its weight: ");
        double weight = sc.nextDouble();
        sc.nextLine();
        while(weight < 0) {
            System.out.println("Please enter a valid weight value: ");
            weight = sc.nextDouble();
            sc.nextLine();
        }
        return weight;
    }
    /**
     *  Get information about the specific pet to be operated on
     * @param operationName Name of the operation
     * @return Specific pets
     */
    private Pet getPet(String operationName) {
        Pet pet = new Pet();
        // Define a Boolean variable to determine if the input is legitimate:
the task requires that there can be no null values
        boolean notHasEmptyValue = true;

        // Read in the pet's details in turn
        System.out.println("We're going to " + operationName + " pet now.");
        System.out.println("Please enter its name: ");
        String name = sc.nextLine();
        // If the read-in is null, notHasEmptyValue set false
        if(name.length() == 0) notHasEmptyValue = false;
```

```
        // Get age value
        int age = getValidAge();

        System.out.println("Please enter its color");
        String color = sc.nextLine();
        // If the read-in is null, notHasEmptyValue set false
        if(color.length() == 0) notHasEmptyValue = false;

        // Get weight of pet
        double weight = getValidWeight();

        // Get pet of type
        System.out.println("There are several types of pets: ");
        System.out.println("1, Dog");
        System.out.println("2, Cat");
        System.out.println("Please enter its type");
        int petType = sc.nextInt();
        sc.nextLine();

        // Get pet of type
        System.out.println("Please enter a species: ");
        String breed = sc.nextLine();
        // If the read-in is null, notHasEmptyValue set false
        if(breed.length() == 0) notHasEmptyValue = false;
        // If the boolean value is false, it means there is a null value and the
pet read in is not legal
        if(!notHasEmptyValue) {
            System.out.println("Null value, end!");
            return null;
        }

        // Instance creation based on reading in pet types
        if(petType == 1) {
            Dog dog = new Dog(name, age, color, weight, breed);
            pet = dog;
        } else if(petType == 2) {
            Cat cat = new Cat(name, age, color, weight, breed);
            pet = cat;
        } else {
            System.out.println("Error");
        }
        return pet;
    }

    /**
     * Adding pet
     */
    private void addPetToClinic() {
        // Get information about the pet you want to add
        Pet pet = getPet("add");
        // If the value is null, it means, there is a null value or an error in
the read.
        if(pet == null) {
            System.out.println("Please enter valid pet information");
            return ;
        }
        // If it already exists
```

```
        if(clinic.notEmpty(pet)) {
            System.out.println("It's been there before.");
            return ;
        }
        clinic.addPet(pet);
        System.out.println("Pet added successfully");
    }
    /**
     * Removing pet
     */
    private void removePetFromClinic() {
        // Get information about the pet to be deleted
        Pet pet = getPet("remove");
        // If the value is null, it means, there is a null value or an error in
the read.
        if(pet == null) {
            System.out.println("Deletion failed, please check input of pet
information, etc.");
            return ;
        }
        // Delete successfully, return to this statement
        if(clinic.removePet(pet)) {
            System.out.println("Deleted successfully");
            return ;
        }
        System.out.println("The pet was not found in the in-clinic pets");

    }

    /**
     * Displaying pet interface information
     */
    private void displayMenu() {
        int choice = -1;
        do {
            System.out.println("Make a selection to display by that type");
            System.out.println("1, Show by color");
            System.out.println("2, Show by type");
            System.out.println("3, Default Display");
            System.out.println("4, Exit");
            System.out.println("Please export the option you want to select: ");
            choice = sc.nextInt();
            sc.nextLine();

            switch (choice) {
                case 1:
                    displayStatisticsByColor();
                    break;
                case 2:
                    displayStatisticsByBreed();
                    break;
                case 3:
                    displayDefaultAllPets();
                    break;
                case 4:
                    break;
                default:
                    System.out.println("Please enter a valid option number");
```

```
                    break;
            }
        } while(choice != 4);
    }

    /**
     * Show by Pet Color
     */
    private void displayStatisticsByColor() {
        clinic.displayPetByColor();
    }

    /**
     * Show by Pet Type
     */
    private void displayStatisticsByBreed() {
        clinic.displayPetByBreed();
    }

    /**
     * Show by default
     */
    private void displayDefaultAllPets() {
        clinic.displayDefaultAllPets();
    }

    /**
     * Search for a specific pet
     */
    private void searchPetMenu() {
        int choice = -1;
        do {
            System.out.println("Options available: ");
            System.out.println("1, Search by name");
            System.out.println("2, Search by color");
            System.out.println("3, Exit");
            System.out.println("Please enter the options to be selected: ");
            choice = sc.nextInt();
            sc.nextLine();

            if(choice == 3) continue;

            if(choice <= 2 && choice >= 1) {
                String str;
                switch (choice) {
                    case 1:
                        System.out.println("Please output the name of the query:
");
                        str = sc.nextLine();
                        clinic.searchPetByName(str);
                        break;
                    case 2:
                        System.out.println("Please output the color to be
queried: ");
                        str = sc.nextLine();
                        clinic.searchPetByColor(str);
                        break;
                }
```

```
            } else {
                System.out.println("Error!");
            }
        } while(choice != 3);
    }

    // Displaying clinic information
    private void displayClinicsDetails() {
        clinic.displayClinicsDetails();
    }
}
```

ClinicTest

```
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class ClinicTest {

    @Test
    public void testInputDogInformation() {
        Clinic myclinic = new Clinic();
        // Max | 4 years | Brown | 50 | Labrador Retriever
        Pet dog = new Dog("Max", 4,"Brown", 50, "Labrador Retriever");
        myclinic.addPet(dog);

        // The delete function returns true if the deletion is successful,
otherwise it returns false.
        // Since the dog is inserted, it is possible to delete it for the first
time, i.e., it returns true
        assertEquals(true,myclinic.removePet(dog), "The Test didn't Pass." );

        // After deleting, deleting again will report an error
        assertEquals(false,myclinic.removePet(dog), "The Test didn't Pass." );

    }

    @Test
    public void testInputCatInformation() {
        Clinic myclinic = new Clinic();
        // Whiskers | 3 years | Gray | 8B | ritish Shorthair
        Pet cat = new Cat("Whiskers", 3, "Gray", 88, "ritish Shorthair");
        myclinic.addPet(cat);

        // The delete function returns true if the deletion is successful,
otherwise it returns false.
            // Since the dog is inserted, it is possible to delete it for the
first time, i.e., it returns true
        assertEquals(true, myclinic.removePet(cat), "The Test didn't Pass.");

        // After deleting, deleting again will report an error
        assertEquals(false, myclinic.removePet(cat), "The Test didn't Pass.");
    }
```

```
}
```