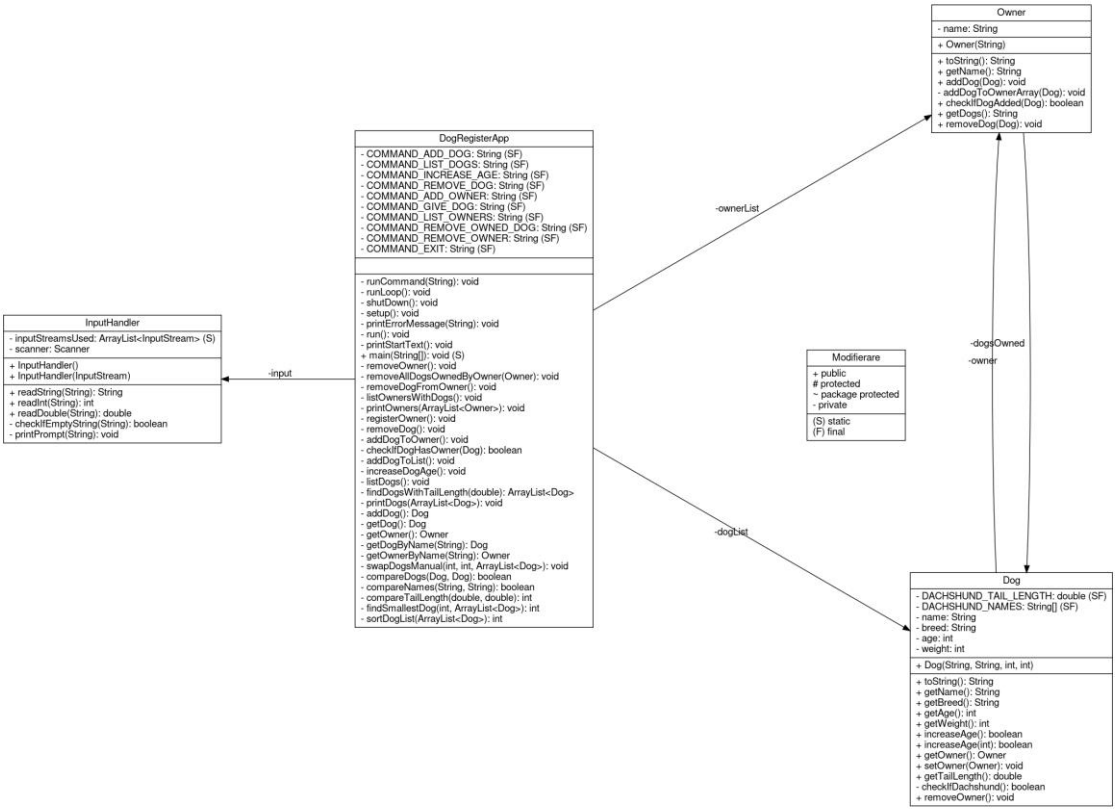
1 INLÄMNINGSHISTORIK

Datum	Beskrivning av vad som är nytt/förändrat i denna inlämning
22-01-16	Första inlämningen

2 KLASSDIAGRAM FÖR KODEN TILL U11.1



3 VAD SOM ÄNDRATS EFTER U9.1 OCH U12.1

Efter U9.1 tyckte jag att min kod såg bra ut i helhet. Jag var osäker på några namn som jag använde för att jag tyckte att det var svårt att få ett bra perspektiv på vilka namn som fungerade bra och vilka som kunde varit lite svåra att förstå. Därför valde jag att inte ändra namnet för att sedan fråga om det på granskningen av koden i U12.1.

De enda ändringar som jag gjorde efter U9.1 var att försöka få bort repetition av kod. Jag skrev en hjälpmetod i huvudklassen DogRegisterApp för att skriva ut fel meddelanden (printErrorMessage) då jag ansåg att det förtydligade koden avsevärt istället för att ha olika System.out.println anrop i mina andra metoder som skrev ut olika felmeddelanden. Först tänkte jag att denna printErrorMessage metod skulle kunna haft skyddsnivån public och varit static men efter att jag skrivit U11.1 märkte jag att alla felmeddelanded rapporterades i min huvudklass och bestämmde därför att den skulle vara privat. Om jag hade haft en annan klassuppdelning tror jag att mina första lösning hade varit bättre.

Under granskningen (U12.1) fick jag endast några anmärkningar på min kod. Detta gällde några av de namn på variabler. Till exempel fick jag feedbacken att döpa om min huvudklass ”App” till något i stilen av det som den nu heter (”DogRegisterApp”) och ägarens hund array som jag hade döpt till ”dogArray” fick det nya namnet ”dogsOwned”. I samband med denna feedback döpte jag också om metoden för att lägga till en hund i ägarens array från ”addDogToArray” till ”addDogToOwnerArray” för att försöka öka tydligheten lite mer.

Jag hade under inlämningarnas gång ändrat metoden i hundklassen för att få svanslängden till ”calculateTailLength” men ändrade tillbaka den till ”getTailLength” för att stämma överrens med klassdiagrammet i U6.2. Jag flyttade också listan med orden för tax på andra språk till att vara en klassvariabel istället för att den ska ligga på metod nivå.

Den sista ändring jag gjorde efter U12.1 var att jag ändrade metoden för att ta bort alla hundar som ägdes av en ägare som skulle tas bort från registret. Detta gjorde jag för att inte behöva returnera arrayen för ägda hundar från owner klassen till huvudklassen för programmet.

4 EXEMPEL PÅ ANVÄNDNING AV HJÄLPMETODER

```
private void removeDog(){
    Dog dogToRemove = getDog();
    if(dogToRemove == null) return;

    if (checkIfDogHasOwner(dogToRemove)){
        Owner ownerOfDog = dogToRemove.getOwner();
        ownerOfDog.removeDog(dogToRemove);
    }
```

```
private void removeDogFromOwner(){
    Dog dogToRemove = getDog();
    if (dogToRemove == null ||!checkIfDogHasOwner(dogToRemove)) return;
```

```
public double getTailLength(){
    return (checkIfDachshund(this.breed)) ? DACHSHUND_TAIL_LENGTH : this.age*this.weight/10.0;
}
```

```
private void addDogToOwner(){
    Dog dog = this.getDog();
    if (dog == null) return;
    if (checkIfDogHasOwner(dog)){
        printErrorMessage("dog already has an owner");
        return;
    }
```

```
private void listOwnersWithDogs(){
    if (ownerList.size() == 0){
        printErrorMessage("No owners registered");
    }
    else{
        printOwners(ownerList);
    }
}
```

```
private void removeOwner(){
    Owner ownerToRemove = getOwner();
    if (ownerToRemove!= null){
        removeAllDogsOwnedByOwner(ownerToRemove);
        ownerList.remove(ownerToRemove);
        System.out.println(ownerToRemove.getName() + " removed");
    }
}
```

5 SORTERINGSKODEN U11.1

```
private void swapDogsManual(int i, int k, ArrayList<Dog> dogs){
    Dog tmpDog = dogs.get(i);
    dogs.set(i, dogs.get(k));
    dogs.set(k, tmpDog);
}
private boolean compareDogs(Dog first, Dog second){
    int tailComparisonResult = compareTailLength(first.getTailLength(), second.getTailLength());
    if (tailComparisonResult == 1) return true;
    else if (tailComparisonResult == -1) return false;

    if (compareNames(first.getName(), second.getName())){
        return true;
    }
    return false;
}
private boolean compareNames(String s, String d){
    if (s.toLowerCase().compareTo(d.toLowerCase()) > 0){
        return true;
    }
    return false;
}
private int compareTailLength(double a, double b){
    if (a > b) return 1;
    if (a < b) return -1;
    return 0;
}
private int findSmallestDog(int startIndex, ArrayList<Dog> dogs){
    int smallestIndex = startIndex;
    for (int i = startIndex+1; i < dogs.size(); i++) {
        if(compareDogs(dogs.get(smallestIndex), dogs.get(i))){
            smallestIndex = i;
        }
    }
    return smallestIndex;
}
private int sortDogList(ArrayList<Dog> dogsToSort){
    int swaps = 0;
    for (int i = 0; i < dogsToSort.size(); i++) {
        int smallestIndex = findSmallestDog(i, dogsToSort);
        if(smallestIndex != i){
            swapDogsManual(i, smallestIndex, dogsToSort);
            swaps++;
        }
    }
    return swaps;
}
```

6 KODEN TILL KLASSEN OWNER FRÅN U11.1

```
import java.util.Arrays;

public class Owner {

    private String name;

    private Dog[] dogsOwned = new Dog[0];


    public Owner(String name){

        this.name = name;

    }


    public String toString(){

        return String.format("%s", this.name);

    }


    public String getName(){

        return this.name;

    }


    public Dog[] getOwnedDogs(){

        return Arrays.copyOf(dogsOwned, dogsOwned.length);

    }


    public void addDog(Dog dog){

        Owner currentDogOwner = dog.getOwner();


        if (currentDogOwner != this && currentDogOwner != null) return;

        if (checkIfDogAdded(dog)) return;


        this.addDogToOwnerArray(dog);

        if (currentDogOwner == this) return;


        dog.setOwner(this);

        return;

    }


    private void addDogToOwnerArray(Dog dogToAdd){

        dogsOwned = Arrays.copyOf(dogsOwned, dogsOwned.length + 1);

        dogsOwned[dogsOwned.length-1] = dogToAdd;

    }


    public boolean checkIfDogAdded(Dog dog){

        if (dogsOwned.length > 0){

            for (Dog ownedDog : dogsOwned) {

                if (dog == ownedDog) return true;

            }

        }

        return false;
```

```
    }

    public String getDogs(){
        String output = "[";
        int arrayLength = dogsOwned.length;
        for (int i = 0; i < arrayLength; i++) {
            output += dogsOwned[i].getName();
            if (i != arrayLength - 1) output += ", ";
        }
        return output + "]";
    }

    public void removeDog(Dog dogToRemove){
        if(!checkIfDogAdded(dogToRemove)) return;

        Dog[] newArray = new Dog[dogsOwned.length-1];

        int newIndex = 0;
        for (int i = 0; i < dogsOwned.length; i++) {
            if (dogsOwned[i].equals(dogToRemove)) continue;
            newArray[newIndex++] = dogsOwned[i];
        }

        dogsOwned = newArray;

        if (dogToRemove.getOwner() != null)
            dogToRemove.removeOwner();
    }
}
```

7 TESTPROGRAMMETS FEEDBACK PÅ U11.1 OCH DINA KOMMENTARER TILL DENNA FEEDBACK

Information
Testprogrammet ska utföra 43 steg totalt, varav 23 är funktionella tester.
Endast fel och informativa meddelanden visas.

Stilkontroll: Checkstyle
Checkstyle (https://checkstyle.sourceforge.io/) är ett verktyg som kontrollerar om man följer vanliga kodkonventioner, alltså saker man ska göra på ett visst sätt, men som inte kontrolleras av kompilatorn. Exakt vilka konventioner som gäller varierar från arbetsplats till arbetsplats, men de flesta är ganska generella. Några av de saker som checkstyle kontrollerar är direkta fel, som till exempel namn som skrivs på fel sätt. Dessa måste rättas till innan du kan bli godkänd. Andra är varningar för saker som kan vara fel, men som inte behöver vara det. Dessa måste du själv kontrollera och bedöma om de ska rättas eller inte.
Checkstyle gav varningar:
Checkstyle: Mer än programsats per rad (WARN)
* [WARN] DogRegisterApp.java:24 (Only one statement per line allowed.)
* [WARN] DogRegisterApp.java:25 (Only one statement per line allowed.)
* [WARN] DogRegisterApp.java:26 (Only one statement per line allowed.)
* [WARN] DogRegisterApp.java:27 (Only one statement per line allowed.)
* [WARN] DogRegisterApp.java:28 (Only one statement per line allowed.)
* [WARN] DogRegisterApp.java:29 (Only one statement per line allowed.)
* [WARN] DogRegisterApp.java:30 (Only one statement per line allowed.)
* [WARN] DogRegisterApp.java:31 (Only one statement per line allowed.)
* [WARN] DogRegisterApp.java:32 (Only one statement per line allowed.)

Det bör (oftast) inte vara mer än en programsats på varje rad. Anledningen är att det kan göra det svårare att läsa och felsöka koden om det händer flera saker på samma rad.

Sammanfattning
Lyckade utförda steg: 42/43. 1 steg misslyckades.

Lyckade funktionella tester: 23/23. 0 tester misslyckades.
Lyckade enhetstester (JUnit): 0/0. 0 tester misslyckades.

Den enda checkstyle varning testprogrammet ger tycker jag är fel i detta fall. ”Only one statement per line allowed” kommer ifrån min switch i kommando-loopen. Testprogrammet säger att om man har flera programsatser per rad kan leda till att koden blir svårare att läsa. I fallet av en swtich med bara ett metodanrop i varje case tycker jag att koden blir mer lättläst på detta sätt då varje case och metod nu har en egen rad och cases kommer precis efter varandra utan blankrader. Om break ska vara på en egen rad blir switchen större och klumpigare att läsa.