DA339A Laboration L13

Syfte

Den här laborationens syfte är att du ska öva med flera klasser i samma applikation, skapa en/flera av samma objekt samt att jobba med metoder med returtyp och parametrar.

Redovisning

Denna laboration behöver inte redovisas

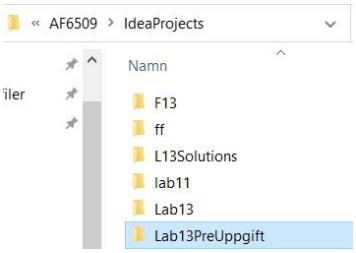
Förberedelser

Den här laborationen är direkt kopplad till det innehåll som finns i föreläsning F13 och tidigare laborationer. Laborationen kommer att börja med ett antal små uppgifter och med hjälp av erfarenheten samt kunskapen vi får från dessa, göra en större uppgift som skall göras i flera steg.

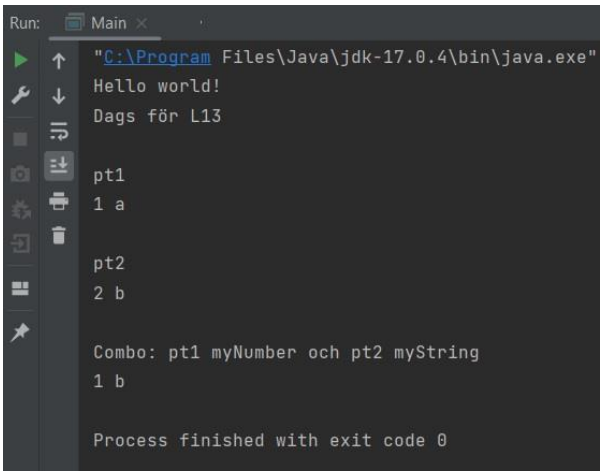
Uppgiften är tänkt att göras som ett IntelliJ projekt och därmed förutsätts det att du har IntelliJ installerat på din dator och att det fungerar bra. Ifall du vill fortsätta att jobba i en vanlig textbehandlare som Atom, kompilera och köra i Git Bash går det också bra. All beskrivning i detta dokument är dock anpassad till IntelliJ.

Uppgift 0

I denna uppgift ska du granska ett färdigt projekt som innehåller två klasser, **Main** och **PrintTest**. Börja med att ladda ner projektet, **Lab13PreUppgift.zip**, från Canvas, packa upp den och lägg den på en plats där du lätt kan hitta den samt köra den i IntelliJ. Som default brukar det skapas en folder som har namnet IdeaProjects i din Användare/User folder där nya projekt sparas om man inte valt någon annan plats.



När du öppnat projektet i IntelliJ, försök köra koden och se vad resultatet visar (se bild nedan).



När du sätt resultatet av programmet, granska koden i klassen **PrintTest**, variablerna (både private och public) de två konstruktors, getter- och setter-metoderna samt public och private metoderna som finns.

Därefter granska koden i **Main** klassen. Med hjälp av kommentarerna i båda klasserna samt resultatet när du kör projektet, skapa ett klassdiagram samt ett sekvensdiagram.

Lösningsförslag finns i slutet av laborationen.

Uppgift 1

I denna uppgift ska du utveckla ett litet Java program. Programmet ska vara en representation av när en hundägare registrera sig som en ny medlem på ett hunddagis med en eller två hundar. Ett körningsexempel presenteras nedan:

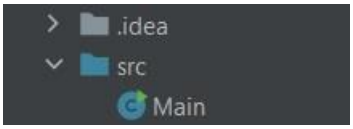
```
Your name please:
Sebastian Wöhrman
Welcome Sebastian Wöhrman!
Your address:
Sandvägen 7
Do you have 1 or 2 dogs:
2
Name of dog 1:
Alfa
Gender of dog 1 M or F?
F
Age of dog 1:
10
Name of dog 2:
Beta
Gender of dog 2 M or F?
M
Age of dog 2:
5
You are now a member!
Name: Sebastian Wöhrman
Address: Sandvägen 7
Number of dogs you own: 2
Dog info: (Name, Gender, Age)
Alfa, F, 10
Beta, M, 5

Total cost: 1000
```

I uppgift ska du skriva klasser, deras metoder och skapa objekt (en instans av en klass) av dessa klasser. Klasserna kommer att innehålla ett antal olika metoder och attribut (getter- och settermetoder) som du skall använda för att få vissa utskrifter.

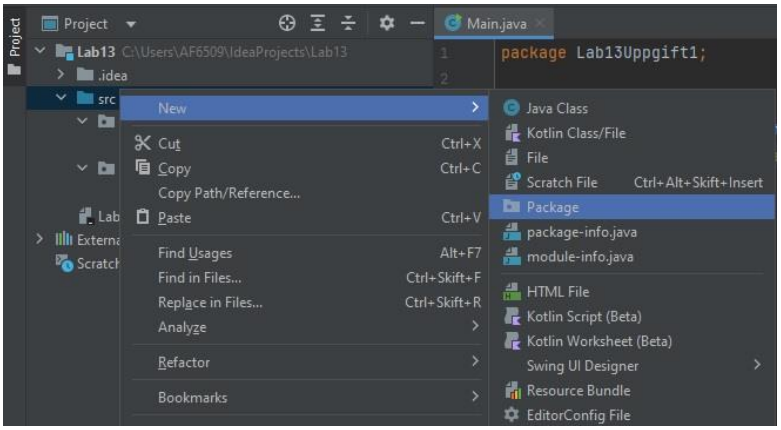
Uppgift 1.1: Skapa ett nytt projekt i IntelliJ

Öppna IntelliJ, skapa ett nytt projekt och ge projektet ett namn (t.ex. **Lab13**). När projektet skapats finns det automatiskt en klass i *src* foldern som har namnet **Main**.

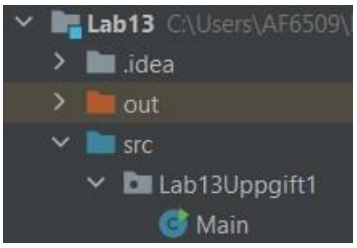


Uppgift 1.2: Skapa ett paket

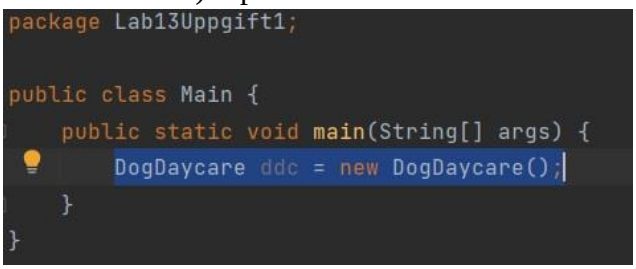
Nu ska du skapa ett paket (package) och ge paketet ett namn (t.ex. **L13Uppgift1**). Du skapar ett paket genom att höger klicka på *src* under projektmenyn, därefter New – Package (se bild nedan).



Efter du skapat paketet ska du dra/flytta din **Main** klass så att den läggs i paketet.



I din **Main** klass komplettera med följande kod (**DogDayCare** är en klass du kommer skapa senare i labben). Spara filen.

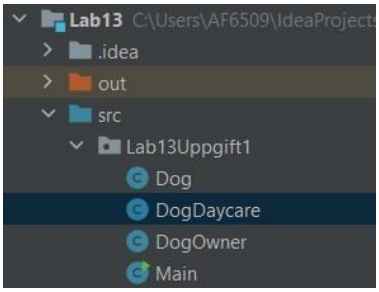


```
package Lab13Uppgift1;

public class Main {
    public static void main(String[] args) {
        DogDaycare ddc = new DogDaycare();
    }
}
```

Uppgift 1.3: Skapa en klass

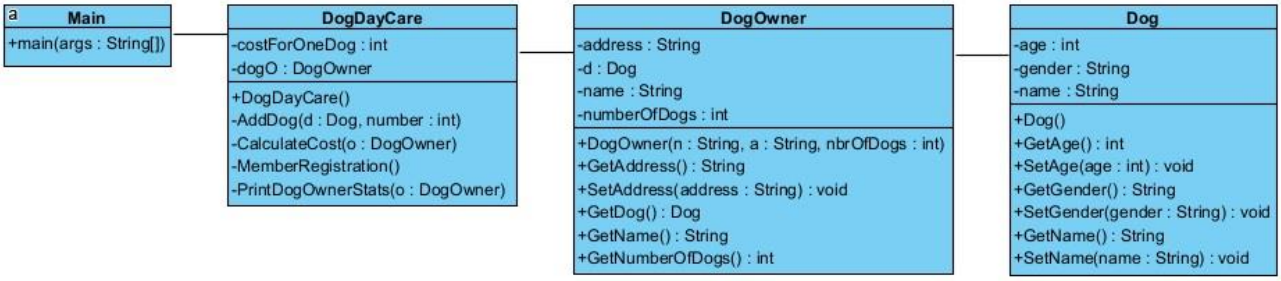
Nu är det dags att skapa din först klass i det packet som du nyss skapade. Högerklicka på paketets namn (**L13Uppgift1** enligt exemplet innan), därefter New – Java Class. Ge klassen namnet **DogDayCare** och klicka sen på Create.



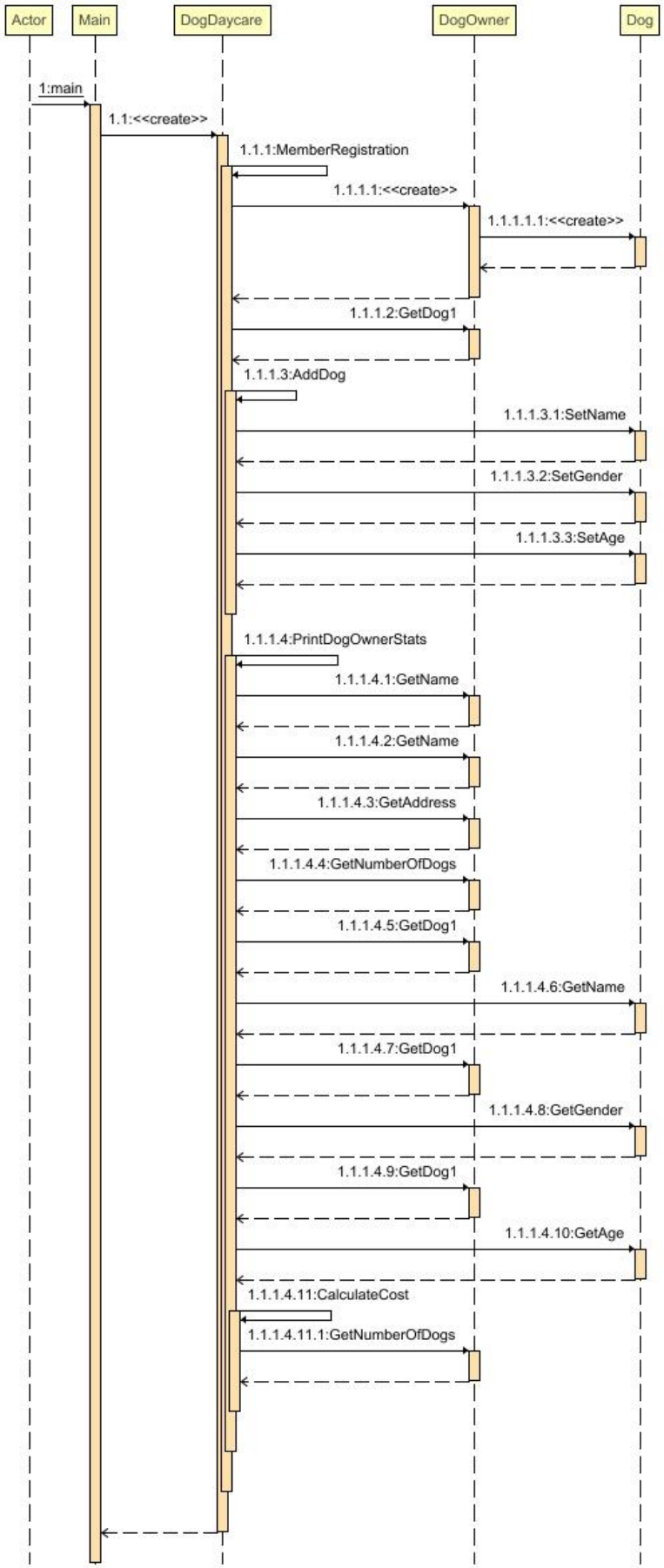
Uppgift 1.4: Studera klassdiagram och sekvensdiagram

Nedan följer två diagram som bör ge dig en idé om hur du ska lösa uppgiften. Diagrammen behöver inte följas exakt men borde kunna fungera som ett litet lösningsförslag. Studera diagrammen och jämför med diagrammen du skapade i uppgift 0.

Klassdiagram:

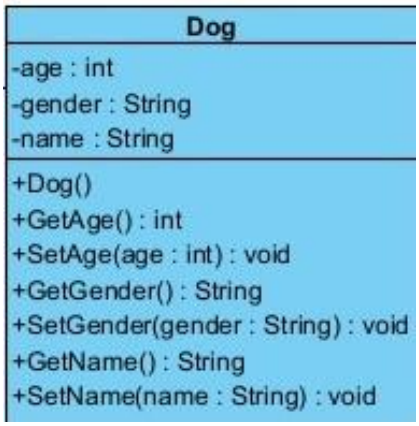


Sekvensdiagram:



Uppgift 1.5: Skriv klassen Dog

Det är alltid bra att börja med de klasser som inte alls eller minst är beroende av de andra klasserna i systemet. Här är klassen **Dog** en sådan klass. Titta i klassdiagrammet och skriv klassen färdigt.



Följande framgår av diagrammet:

- Klassen har tre privata instansvariabler
- Klassen har en konstruktor
- Getter- och settermetoder till instansvariablerna

Att tänka på:

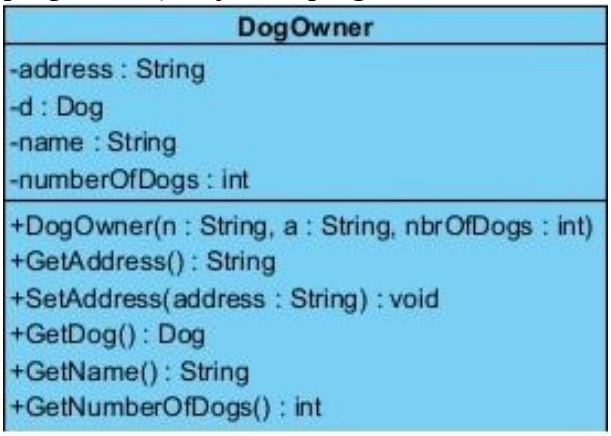
- Inga **static**-metoder.
- Alla instansvariabler skall deklareras som **private**.
- Du ska inte behöva lägga till flera instansvariabler än vad som är angivna i klassdiagrammet.

Skapa en ny klass i paket och nämna den **Dog**.

- Deklarera instansvariabler.
- Skriv en default-konstruktor (som kan vara tom)
- Skriv en getter- och en settermetod till varje instansvariabel.
- I setter-metoderna, kolla så att värdet som kommer till metoden (som parameter) inte är negativt eller inte är en string om det är ett tal som ska vara inparameter, innan du sparar det i instansvariabeln.
- När du är färdig med klassen, kompilera programmet (menyn Build – Build Project eller Rebuild Projekt i IntelliJ). Rebuild rekommenderas alltid.
- Om kompilering går bra, fortsätt till nästa steg.

Uppgift 1.6: Skriv klassen DogOwner

Denna klass skapar en eller två hundar beroende på vad användaren (användare = den som kör programmet) väljer när programmet körs.



Följande framgår av diagrammet:

- Klassen har fyra (fem om man använder två hundar) privata instansvariabler.
- Klassen har en konstruktor med tre inparametrar (två string och en int).
- Getter- och settermetoder till instansvariablerna (name, address, dog1, dog2, numberOfDogs).

Att tänka på:

- Inga **static**-metoder.
- Alla instansvariabler skall deklareras som **private**.
- Du ska inte behöva lägga till flera instansvariabler än vad som är angivna i klassdiagrammet.

Skapa en ny klass i paket och nämna den **DogOwner**.

- Deklarera instansvariabler.
- Skriv en konstruktor (som ska sätta värdena på name och address, samt skapa hund objekt beroende på om det är en eller två som ägaren har).
- Skriv en getter- och en settermetod till varje instansvariabel.
- När du är färdig med klassen, kompilera programmet (menyn Build – Build Project eller Rebuild Projekt i IntelliJ). Rebuild rekommenderas alltid.
- Om kompilering går bra, fortsätt till nästa steg.

Uppgift 1.7: Skriv klassen DogDayCare

Denna klass skall hantera all kommunikation med användaren. Klassen använder sig av ett objekt av DogOwner för att skapa en hundägare, hundarna som tillhör ägaren och deras värden.

Följande framgår av diagrammet:

- Klassen har två privata instansvariabler.
- Klassen har en konstruktor, där metoden **MemberRegistration** anropas.
- **MemberRegistration** metoden hanterar all input (med hjälp av en scanner) som behövs för **DogOwner** och **Dog** objekten.
- **AddDog** metoden anropas i **MemberRegistration** när en hund ska skapas.
- **CalculateCost** metoden beräknar totala kostnaden för ägaren och anropas i metoden **PrintDogOwnerStats**.
- **PrintDogOwnerStats** metoden skriver ut ett ”kvitto” med all information om ägaren och hens hundar samt den totala kostnaden.

Att tänka på:

- Inga **static**-metoder.
- Alla instansvariabler skall deklareras som **private**.
- Du ska inte behöva lägga till flera instansvariabler än vad som är angivna i klassdiagrammet.

Skapa en ny klass, **DogDayCare**. Denna klass har två instansvariabler:

- En referensvariabel som kan heta **dogO** (**DogOwner** objekt, du kan döpa den till vad du vill) och en int för att beräkna totala kostnaden (exempel **costForOneDog**, värdet får du själv bestämma).

- Skapa en default-konstruktor. Konstruktorn ska anropa **MemberRegistration** metoden senare när vi skapat den, så konstruktorn kan vara tom för tillfället.

- Skapa metoden **MemberRegistration**. Metoden kommer att behöva en Scanner för att kunna ta emot input från användaren. Använd bilden på körningsexemplet i början av uppgift 1 för att ta reda på vilka inputs som du kommer behöva. Kom ihåg att använda dig av **DogOwner** och **Dog** getter- och settermetoder för att tilldela värdena som behövs. Exemplet nedan frågar vad hundens namn ska vara och den sparas i en variabel och därefter anropas hundens **SetName** metod där namnet skickas in.

```
Scanner scanner2 = new Scanner(System.in);
System.out.println("Name of dog " + number + ": ");
String dogName = scanner2.nextLine();
d.SetName(dogName);
```

- Skapa metoden **AddDog**. Metoden kommer att anropas i **MemberRegistration** metoden när en hund ska skapas (metoden skapar inte en Dog men den sätter dess värden). Du kommer behöva en Scanner för att ta emot användarens input. Som ovan glöm inte att använda **Dog** klassens setter metoder.

- Skapa metoden **CalculateCost**. Metodens syfte är att beräkna totala kostnaden ägaren behöver betala. Metoden kommer använda ett **DogOwner** objekt som inparameter för att använda metoden **GetNumberOfDogs** vilket retunerar ett int värde som multipliceras med variabeln **CostForOneDog** som skapades tidigare i klassen.

- Skriv metoden PrintDogOwnerStats. Denna metod kommer att skriva ut ”kvittot” och anropas i slutet av **MemberRegistration** metoden. Utskriften ska ha liknande utseende som bilden nedan:

```
You are now a member!
Name: Sebastian Wöhrman
Address: Sandvägen 7
Number of dogs you own: 2
Dog info: (Name, Gender, Age)
Alfa, F, 10
Beta, M, 5

Total cost: 1000
```

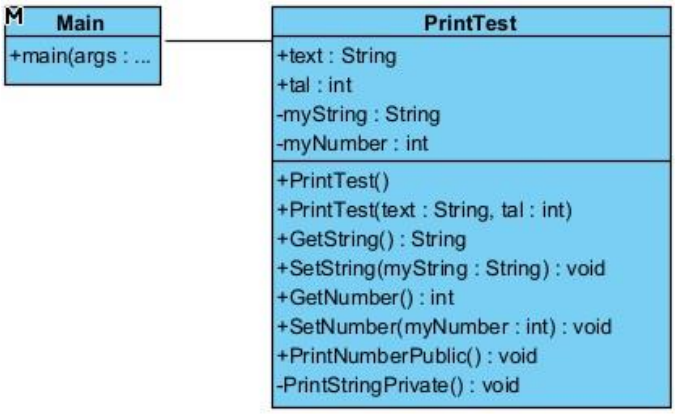
Kompilera och rätta alla kompileringsfel. Läs vad kompilatorn klagar över och försök att hitta felet och åtgärda det. Alla vänster- och högerparenteser skall matcha, annars kan många fel uppstå på grund av en felplacerad eller saknad parentes. Satser skall avslutas med ett semikolon, osv.

Det finns lösningsförslag på alla klasser och deras metoder i slutet av dokumentet.

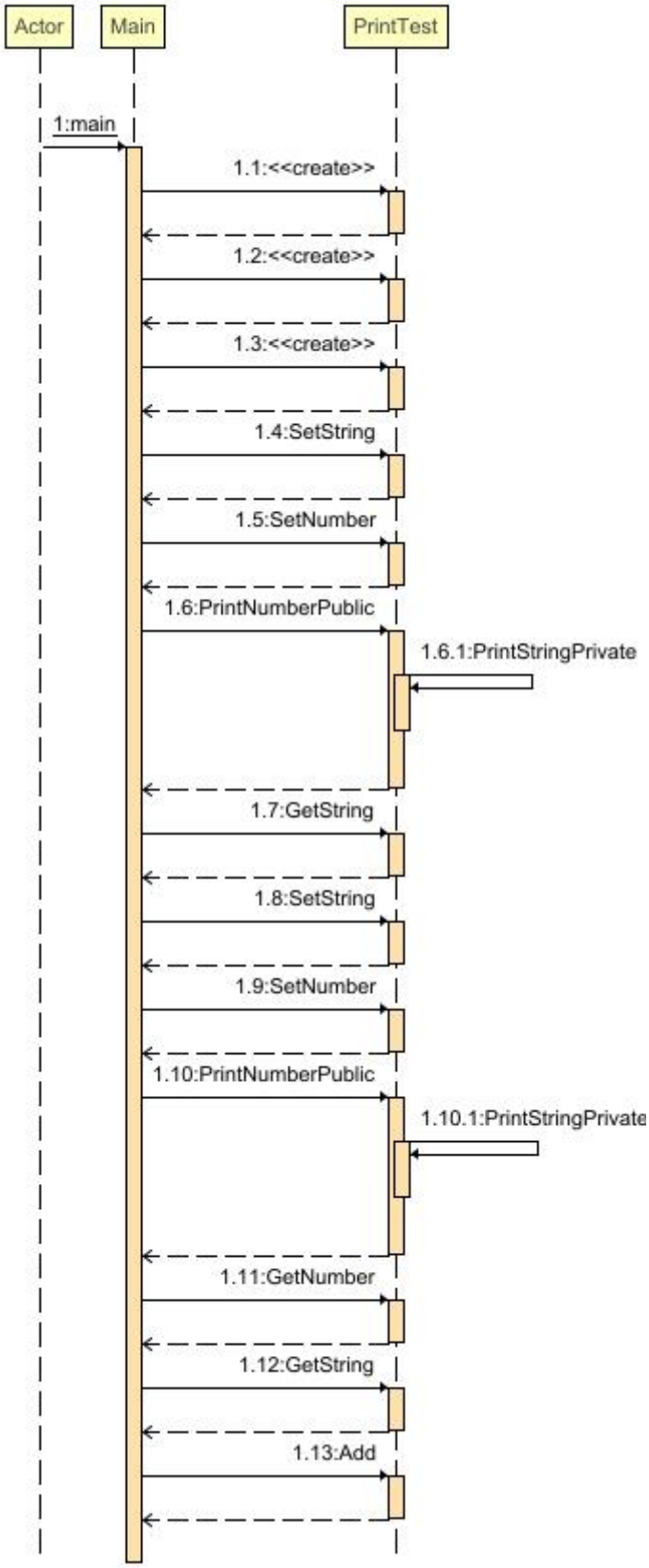
Lösningsförslag

Uppgift 0

Klassdiagram:



Sekvensdiagram:



Uppgift 1

Klass DogDayCare

MemberRegistration metod:

```
private void MemberRegistration() {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Your name please: ");
    String name = scanner.nextLine();

    System.out.println("Welcome " + name + "!");
    System.out.println("Your address: ");
    String address = scanner.nextLine();

    int numberOfDogs = 0;

    do {
        System.out.println("Do you have 1 or 2 dogs: ");
        numberOfDogs = scanner.nextInt();
        if (numberOfDogs > 0 && numberOfDogs <= 2) {
            dog0 = new DogOwner(name, address, numberOfDogs);
        }
    } while (numberOfDogs != 1 && numberOfDogs != 2);

    AddDog(dog0.GetDog1(), number: 1);
    if (numberOfDogs == 2) {
        AddDog(dog0.GetDog2(), number: 2);
    }
    PrintDogOwnerStats(dog0);
}
```

AddDog metod:

```
private void AddDog(Dog d, int number) {
    Scanner scanner2 = new Scanner(System.in);
    System.out.println("Name of dog " + number + ": ");
    String dogName = scanner2.nextLine();
    d.SetName(dogName);

    System.out.println("Gender of dog " + number + " M or F?");
    String dogGender = scanner2.nextLine();
    d.SetGender(dogGender);

    System.out.println("Age of dog " + number + ": ");
    int dogAge = scanner2.nextInt();
    d.SetAge(dogAge);
}
```

CalculateCost och PrintDogOwnerStats metod:

```
private void CalculateCost(DogOwner o) {
    System.out.println("Total cost: " + costForOneDog * o.GetNumberOfDogs());
}
1 usage
private void PrintDogOwnerStats(DogOwner o){
    System.out.println("You are now a member!");
    System.out.println("Name: " + o.GetName());
    System.out.println("Address: " + o.GetAddress());
    System.out.println("Number of dogs you own: " + o.GetNumberOfDogs());
    System.out.println("Dog info: (Name, Gender, Age)");
    System.out.println(o.GetDog1().GetName() + ", " + o.GetDog1().GetGender() + ", " + o.GetDog1().GetAge());
    if (o.GetNumberOfDogs() == 2){
        System.out.println(o.GetDog2().GetName() + ", " + o.GetDog2().GetGender() + ", " + o.GetDog2().GetAge());
    }
    System.out.println();
    CalculateCost(o);
}
```

Klass DogOwner

Konstruktor:

```
public DogOwner(String n, String a, int nbrOfDogs){
    name = n;
    address = a;
    numberOfDogs = nbrOfDogs;
    if (nbrOfDogs == 1){
        d1 = new Dog();
        //d2 = null;
    }
    else if (nbrOfDogs == 2){
        d1 = new Dog();
        d2 = new Dog();
    }
}
```

Getter- och settermetoder:

```
public String GetName(){
    return name;
}
1 usage
public String GetAddress(){
    return address;
}
public void SetAddress(String s){
    address = s;
}
4 usages
public Dog GetDog1(){
    return d1;
}
4 usages
public Dog GetDog2(){
    return d2;
}
3 usages
public int GetNumberOfDogs(){
    return numberOfDogs;
}
```

Klass Dog

```
package Lab13Uppgift1;

public class Dog {
    2 usages
    private String name;
    2 usages
    private String gender;
    2 usages
    private int age;

    3 usages
    public Dog(){

    }
    1 usage
    public void SetName(String n) { name = n; }
    2 usages
    public String GetName() { return name; }
    1 usage
    public void SetGender(String g) { gender = g; }
    2 usages
    public String GetGender() { return gender; }
    1 usage
    public void SetAge(int a) { age = a; }
    2 usages
    public int GetAge() { return age; }
}
```