Tentamen på
Kurs:
DA339A, Objektorienterad programmering

Provkod: 2002 Tentamen 2, 2,5 hp

220218 kl 14.15 – 19.15

Tillåtna hjälpmedel:

- **Inga tillåtna hjälpmedel utöver det som finns i den trycka tentamen och eventuell medtagen engelsk-svensk ordbok eller annan språkkombination (anteckningar får inte förekomma i medtagna ordböcker)**

Betygsgränser

- Del med uppgifter för G: 40 poäng
- Del med uppgifter för VG: 1 uppgift med deluppgifter, bedöms i sin helhet som godkänd eller inte godkänd

För betyg G Godkänt krävs 24 poäng på del med uppgifter för betyg G.

För betyg VG Väl godkänd krävs betyg G på del med uppgifter för betyg G och att uppgiften på del med uppgifter för betyg VG bedöms som godkänd i sin helhet.

Övriga tentamensanvisningar

- Följ instruktionerna som anges på tentamensomslaget (det som signeras av tentavakten och innehåller antal inlämnade papper med mera) - det är huvudförutsättningarna för att uppgifterna ska kunna rättas.
- Skriv läsligt och använd gärna stödlinjerna i tentamensomslaget. Svar som inte kan läsas på grund av otydlig handstil erhåller inte poäng.
- Röd penna får ej användas för svar (reserverad för rättning).
- Besvara alla frågor med en beskrivning som visar din förståelse och kunskap i frågan. Använd fullständiga meningar som gör det tydligt vad du svarar på och vad ditt svar är.
- För svar som ges i form av kod eller pseudokod krävs att denna är lämpligt formaterad så man tydligt kan se vilka blocka av kod som hänger samman. Indentera väl!
- Häfte med frågor och anteckningspapper/kladdpapper får tas med från tentamenssalen när tentan lämnats in.

Lärare besöker tentan ungefär vid följande tider: 15.00 och 17.00

Uppgifter för betyg G 40 p

Uppgift 1 10p 0,5/påstående

Vilka påståenden A-T nedan är korrekta/sanna och vilka är felaktiga/falska? Alla påståenden kan förutsättas vara i förhållande till Java. Svara genom att lista alla bokstäver som gäller för frågorna med ett kommatecken emellan, enligt modellen:

Korrekta/sanna: X, Y, Z (obs. X,Y,Z skall ersättas av delfrågans bokstav)
Felaktiga/falska: Å, Ä (obs. Å,Ä skall ersättas av delfrågans bokstav)

Skriv texten ”Korrekta/sanna” och ”felaktiga/falska” innan listan med bokstäver. Om det bara är två listor med bokstäver vet vi inte vilka svar som du anser är vad.

Fråga	**Påstående**
A	Ett klassdiagram är ett statiskt diagram.
B	En use-association är en svagare association mellan två klasser än en vanlig enkel association.
C	Enligt en strikt Model-View-Controller-arkitektur kommunicerar klasser av typen Model och View inte med varandra.
D	Det kan finnas flera olika associationer mellan två klasser men dessa måste då vara olika typer av associationer (exempelvis en komposition och en generalisering).
E	Ett sekvensdiagram visar vilka anrop som sker eller kan ske mellan olika objekt under exekvering.
F	Om en klass är abstrakt kan man inte skapa objekt av klassen.
G	Vid hantering av undantag, kan det finnas flera `try` och `finally` block men bara ett `catch` block.
H	En klass kan ha endast en (1) superklass och kan implementera flera interface.
I	Undantag (exceptions) är fel som bara inträffar när programmet körs.
J	Vid överlagring (overloading) sker bindning (dvs. vilken metod skall anropas) statiskt vid kompilering.
K	Nyckelordet **throw** används inne i en metod och följs av en undantagstyp och kastar alltid undantaget.
L	Nyckelordet **throws** används tillsammans en metods signatur och tvingar metoden att implementera **try-catch** inne i metodkroppen.
M	Skillnaden mellan en abstract-metod och en överskuggad metod är en abstrakt metod har implementation (kropp) i superklassen, men en överskuggad metod behöver inte ha en kropp i subklassen.
N	En abstrakt metod måste implementeras av alla subklasser till superklassen som innehåller den abstrakta metoden.
O	Ett interface kan inte ärva (extends) ett annat interface.
P	När typen av objekt är bestämt vid kompilering, sker en dynamisk bindning.
Q	När en array eller ett annat objekt skickas som argument till en metod, tillämpas en mekanism som heter **pass-by-reference**, vilket betyder att objektets data skickas till metoden.

R	Nyckelordet **extends** används för att implementera ett interface och nyckelordet **implements** används för att ärva en superklass.
S	Vid överskuggning har man samma metod med olika implementation i subklasser i en klasshierarki.
T	Överskuggning (overriding) innebär att en metod i en subklass ska användas i stället för den metod som subklassen har ärvt från en överordnad klass. En överskuggande metod har samma namn och samma argument som metoden som den överskuggar.

Svar
Korrekta/sanna: A, B, C, E, F, H, I, J, K, N, Q, R, S, T
Felaktiga/falska: D, G, L, M, O, P, R,

Uppgift 2 2p

Förklara hur du kan resonera för att avgöra om en association mellan två klasser lämpligtvis kan vara en komposition eller inte.

Svar: Se steg för att avgöra om något är en komposition eller aggregation i slides till föreläsning F14:

1. Är förhållandet något som kan beskrivas semantiskt som en helhet som byggs upp av delar? Helheten behöver delarna för att ”bli något meningsfullt”.
2. Är svaret **nej** eller tveksamt på frågan i 1 – använd en vanlig enkel association och gå inte vidare.
3. Är svaret **ja** på frågan i 1 behöver du avgöra om det är en aggregation eller en komposition.
4. Fråga dig om helheter kan dela delar med varandra?
5. Om svaret är ja på frågan i 4 – sannolikt bäst att använda en aggregation. Är svaret nej – sannolikt bäst att använda en komposition.

Uppgift 3 2p

Du har följande förslag på en superklass och tre subklasser till denna i ett system som hanterar biljetter för ett tågbolag:

Super-klass	Sub-klasser
Biljett	Passagerare, Klass1Biljett, Plats

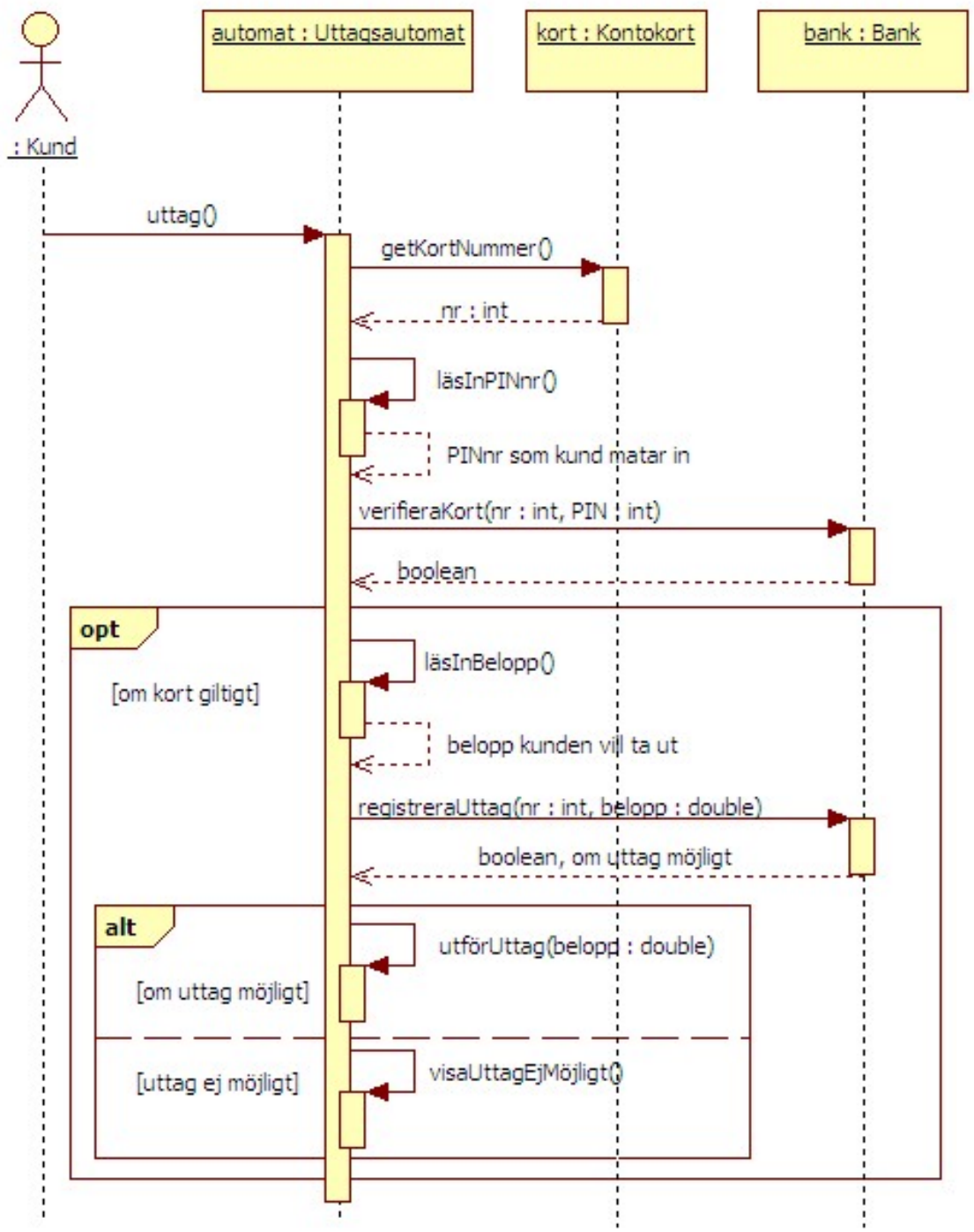
Motivera varför detta är en lämplig generalisering **eller inte** är en lämplig generalisering. Ditt svar ska motivera för respektive sub-klass huruvida det är en lämplig generalisering eller inte i förhållande till den givna super-klassen.

Svar:
Passagerare: Är inte en lämplig sub-klass till Biljett. Är rimligtvis ett attribut som är en referens till en klass Passagerare som representerar den som köpt biljetten och ska sitta på platsen.
Klass1Biljett: Är sannolikt en lämplig sub-klass till Biljett som tillför information som är specifik för biljetter i 1:a klass.
Plats: Är inte en lämplig subklass till Biljett utan snarare ett attribut i klassen biljett som representerar vilken plats i en vang en biljett gäller.

Uppgift 4 4p

I sekvensdiagrammet nedan finns information om klassnamn och metoder i dessa klasser samt viss information om vad som sker i metoderna. Skriv kod för alla de klasser som finns i diagrammet.

Koden skall endast visa vilka metoder klasserna har samt den kod i metoderna som går att utläsa från diagrammet. Vissa antaganden kan göras om eventuella iterationer och selektioner samt villkor i dessa om det är iterationer eller selektioner som representeras i diagrammet.



Lösning

```
class Uttagsautomat {
  public uttag(){
    int nr = kort.getKortNummer();
    int PIN = läsInPINnr();
    boolean kortOk = bank.verifieraKort(nr, PIN);
    if(kortOk){
      double belopp = läsInBelopp();
      boolean uttagOk = registreraUttag(nr, belopp);
      if(uttagOk){
        utförUttag(belopp);
      }
      else{
        visaUttagEjMöjligt();
      }
    }
  }

  public läsInPINnr(){…} //möjligt anta returnerar tal eller sträng
  public läsInBelopp(){…} //möjligt anta returnerar tal
  public utförUtag(double belopp){…}
  public visa UttagEjMöjligt(){…}
}

class Kontokort {
  public int getKortNummer(){…}
}

class Bank {
  public boolean verifieraKort(int nr, int PIN){…}
  public boolean registreraUttag(int nr, double belopp){…}
}
```

Uppgift 5 4p

5a 1,5p

Du skall skriva en klass som representerar en hund och döpa klassen till `Dog`. Klassen skall innehålla data om hundens kön, ålder och mankhöjd (instansvariabler). Använd lämpliga typer och variabelnamn till variablerna. Instansvariabler skall vara deklarerade med modifieraren `private`.

5b 1,5p

Skapa en enda konstruktor för klassen `Dog`. I konstruktorn, som skall ha nödvändiga parametrar, skall alla instansvariabler få värden tilldelade till sig via parametrarna.

5c 1p

Skriv en `toString`-metod som returnerar en sträng med information om objektets data.

Du kan svara på deluppgifterna i samma svar, det vill säga skriva allt som efterfrågas för klassen `Dog` i ett svar för uppgift 5.
Observera: Denna klass används i uppgift 6.

Lösningsförslag:

```
public class Dog {
  private String sex;
  private int age;
  private double height;

  public Dog(String sex, int age, double height)
  {
    this.sex=sex;
    this.age=age;
    this.height=height;
  }


  public String toString() {
   return
     String.format("Sex: %s, Age: %s, Height: %s", sex, age, height);
  }
}
```

Uppgift 6 4p

6a 1p

Skapa en klass `Labrador`. Denna klass skall ha en privat instansvariabel *`name`* som är en textsträng och ska ärva från klassen `Dog` som du skrivit i uppgift 5.

6b 2p

Skapa en konstruktor för klassen `Labrador` med nödvändiga parametrar för att initiera alla instansvariabler. Lösningen ska nyttja att `Labrador` ärver `Dog`.

6c 1p

Skriv en `toString`-metod i klassen Labrador så att resultatet nedan erhålls. Lösningen ska nyttja att `Labrador` ärver `Dog`. En testkörning av metoden Main (nedan) ger resultat som följer.

```
public class MainProgram {
    public static void main(String[] args) {
      Labrador labrador = new Labrador("female", 7, 59.4, "Lufsen");
      System.out.println(Labrador.toString());
    }
}
```

Output:

```
Dog name: Lufsen
Sex: female, Age: 7, Height: 59.4
```

Lösningsförslag:
Kommentar: Det hade smugit sig in ett fel i koden ovan:
`System.out.println(Labrador.toString());`
Borde vara:
`System.out.println(labrador.toString());`
Skulle någon skrivit en lösning som utgår från Labrador istället för labrador och det är rimligt så rättas uppgiften utifrån det.

```
public class Labrador extends Dog{
   private String name;
   public Labrador(String sex, int age, double height, String name) {
      super(sex, age, height);
      this.name = name;
   }


   public String toString() {
     return String.format("Dog name: %s ",name)
       + "\n" + super.toString();
   }
}
```

Uppgift 7 7p

Konstruera ett lämpligt klassdiagram utifrån systembeskrivningen nedan. Identifiera klasser för systemet. Använd dig av så lämpliga associationstyper som möjligt för att beskriva hur klasserna relaterar till varandra. Använd generalisering, aggregation och/eller komposition där detta är möjligt och lämpligt – du förväntas använda minst en generalisering och minst en komposition eller aggregation i din lösning.

Skriv ut namn på alla associationer för att förtydliga dessa. Om det går att ange en multiplicitet för en association ska detta finnas – gör eventuella lämpliga antaganden där detta kan vara rimligt. Gör inga tillägg med information som inte finns i systembeskrivningen.

Det är inte nödvändigt att visa attribut eller operationer i klassdiagrammet. Det räcker att skriva ut klassnamn och namn på associationer och multiplicitet. Korrekt UML-notation ska användas (se Bilaga 1).

Systembeskrivning
I kommunen Alleboda vill man hålla reda på de idrottsklubbar som finns och deras medlemmar. Systemet behövs för att rättvist fördela de skattepengar som används för att stöda denna typ av verksamhet.

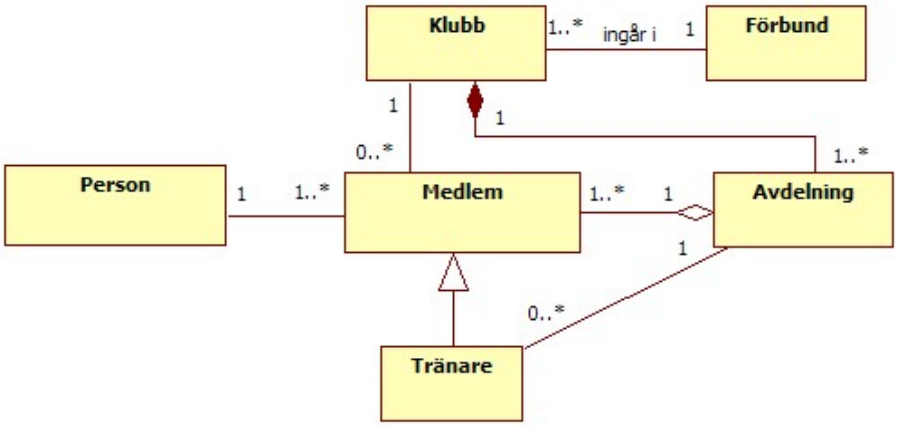
I systemet behöver man veta vilka idrottsklubbar som finns. För varje klubb behöver man också hålla reda på avdelningar inom klubben då olika åldersgrupper sponsras på olika sätt. De flesta klubbar har något rikstäckande förbund som klubben är knuten till. Dessa förbund och vilka klubbar som är knutna till dem vill man också hålla reda på i systemet.

Varje klubb har ett antal medlemmar. Medlemmar kan vara knutna till en avdelning av klubben. Medlemmar kan inte vara knutna till mer än en avdelning som medlem. Om en avdelning ändras i systemet är det dock viktigt att medlemmarna i denna avdelning finns kvar knutna till klubben och kan flytas över till andra avdelningar om det behövs.

Det finns en speciell form av medlemmar som man vill föra statistik över och detta är medlemmar som fungerar som tränare på någon avdelning. Man vill veta vilka medlemmar som är tränare och vilken avdelning de tränar. En avdelning behöver dock inte ha några tränare men kan även ha flera olika tränare. En tränare är också medlem i någon avdelning i egenskap av vanlig medlem.

För varje medlem vill man hålla reda på personen bakom för att veta hur många egentliga personer som är engagerade i klubbverksamheten i kommunen. En person kan dock vara medlem i flera klubbar.

Lösningsförslag

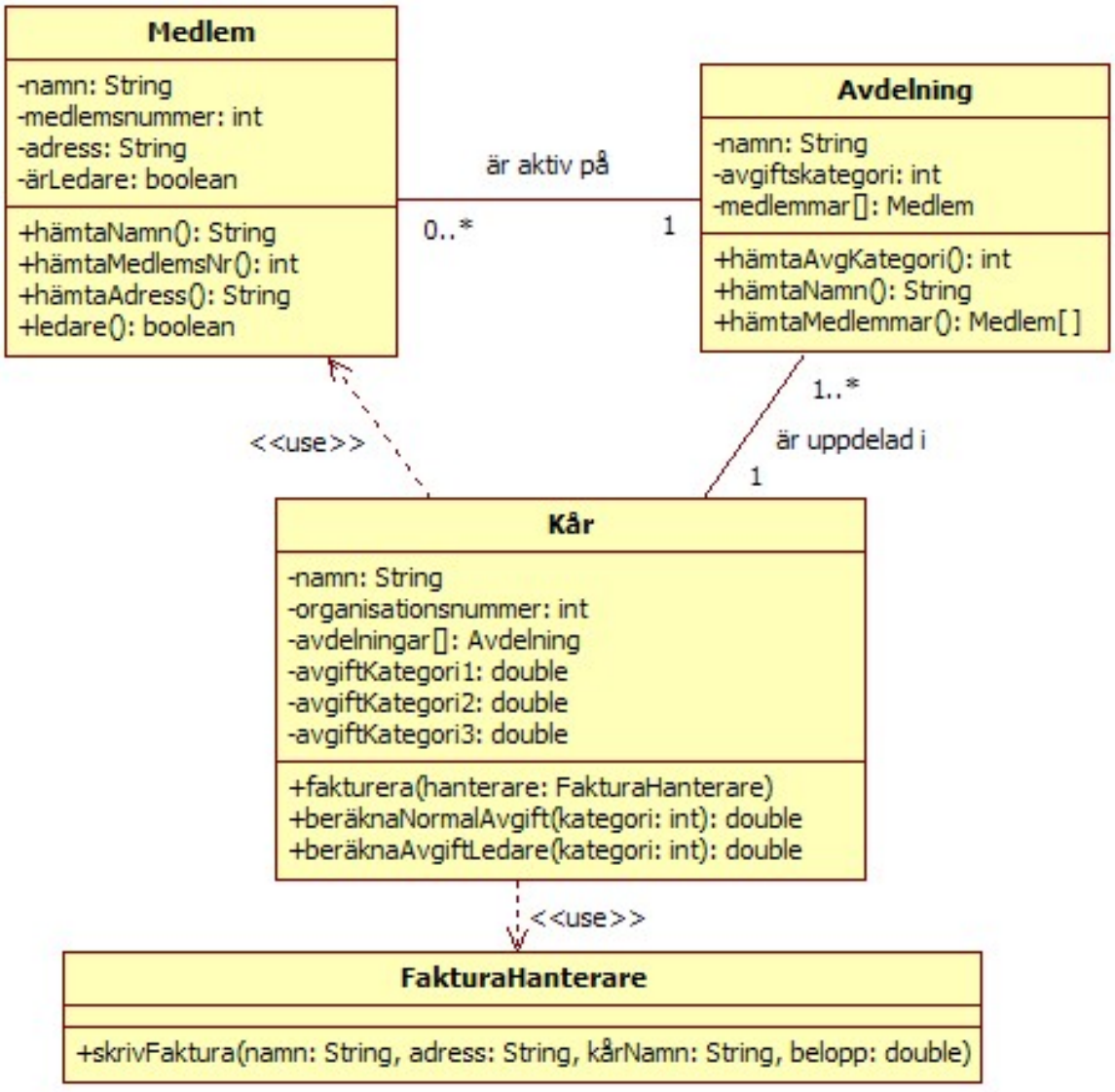


Uppgift 8 7p

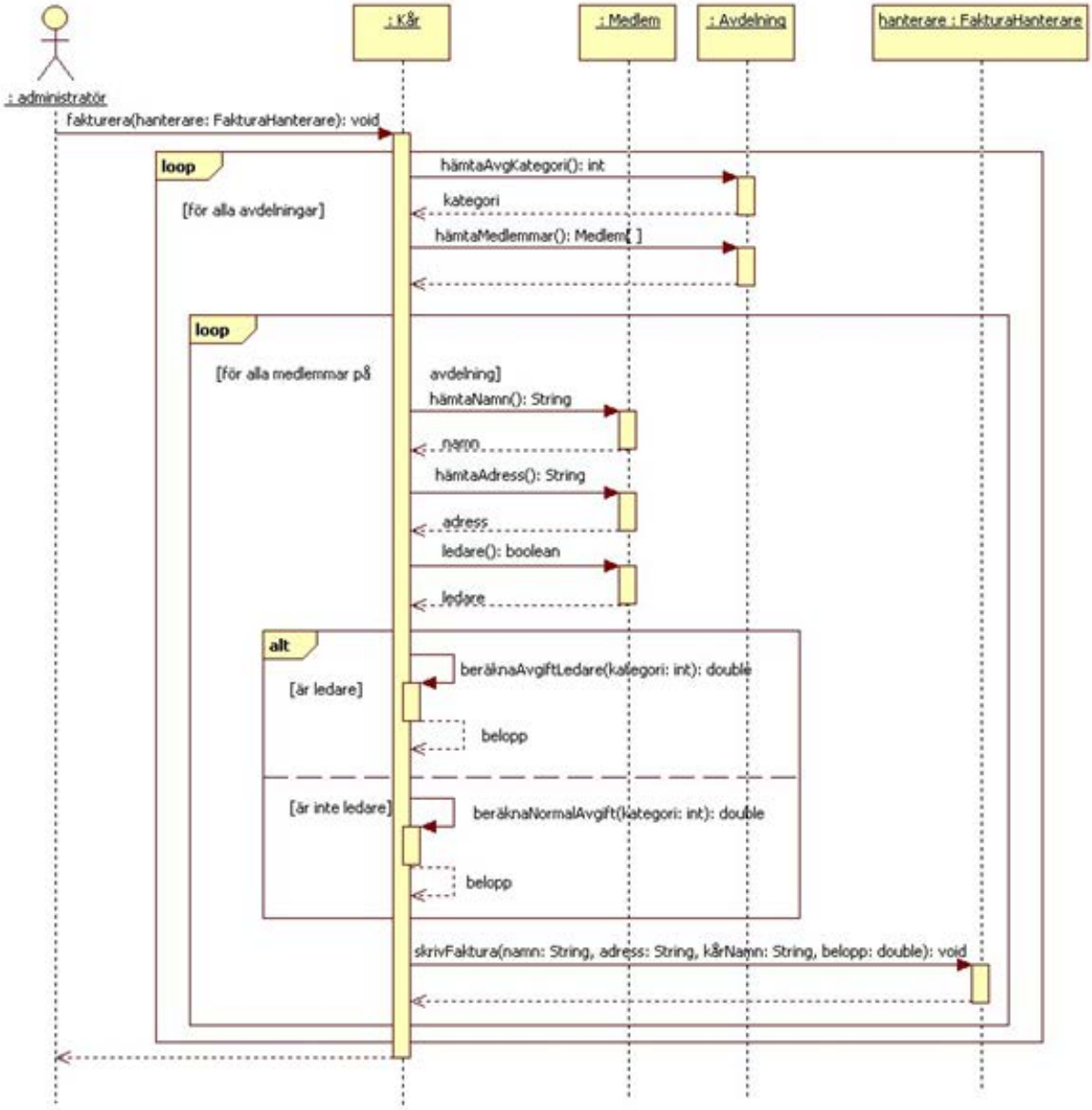
Klassdiagrammet nedan beskriver ett system som hanterar ett scoutförbunds kårer och medlemmar. Systemet används bland annat för att skriva ut fakturor på medlemsavgift för alla medlemmar två gånger om året. En administratör från varje kår har till gång till systemet och går in och anger att fakturor skall skrivas ut för kåren. De medlemmar som är ledare på en avdelning får rabatt på medlemsavgiften som en uppskattning för det arbete de gör. Systemet har tre kategorier av avgifter som kan användas för olika avdelningar med olika mycket verksamhet. Så när fakturorna skrivs ut så måste man veta vilken avgiftskategori avdelningen som medlemmen tillhör har och om medlemmen är ledare eller ej.

Rita ett sekvensdiagram som visar vad som sker när en administratör vill skriva ut fakturor för medlemsavgiften för en kårs medlemmar.

Du skall inte skapa några nya klasser, lägga till operationer eller attribut i befintliga klasser utan skall använda dig av det som visas i klassdiagrammet. Du får dock skapa en aktör som representerar administratören och som interagerar med klassen Kår genom att initiera metoden fakturera(hanterare: FakturaHanterare).



Lösningsförslag



Uppgifter för betyg VG

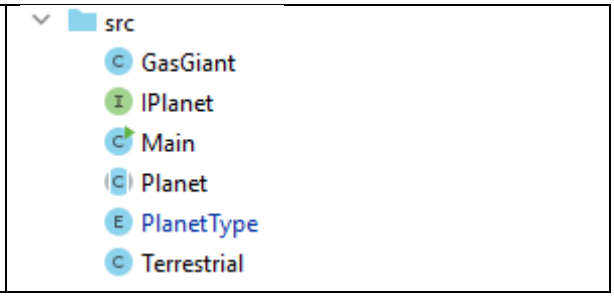
Uppgift 9

Enumerationen `PlanetType` är given. Du skall skriva en klass `Planet`, som använder enumerationen och också implementerar interfacet `IPlanet`. Du skall också skriva ett par konkreta klasser. Tänk på att klasserna inte är stora, det vill säga du skall skriva så mycket kod som frågorna nedan kräver – inte mer:

Obs! det är inte obligatoriskt att skapa och använda konstruktorer i denna och de andra klasser som du skriver i denna uppgift. Setter och getter metoder skriver du efter behov.

Glöm inte att deklarera de variabler du använder, strukturera metoderna på rätt sätt och använder rätt syntax. Till höger i tabellen nedan visas de källfiler som skall kompletteras i uppgiften.

```
public enum PlanetType {
    Planetoid,      // Class D
    Gas_Giant,      // Class J or T
    Terrestrial,    // Class M
    Rogue           // Class R
}
```



9a: Interface

Komplettera interfacet `IPlanet` enligt nedan, punkt 1 till 3.

```
public interface IPlanet {
        // En planet (Planet) måste ha en typ enligt enum-typen PlanetType
        // 1. Komplettera med en getter- och en setter-metod,
        // som klasserna (Planet) måste implementera

        // En planet skall räkna ut tyngd för en kropp baserat på dess massa.
        // weight = gravity * mass
        // 2. Komplettera med en metod som returnerar weight som
        // ett double tal och tar massan som argument,
        // exempel på metodnamn: calculateWeight

        // 3. Skriv en metod getPlanetInfo som skall returnera en String.
}
```

9b: Abstrakt klass och abstrakt metod

Skriv en klass Planet som implementerar interfacet i 9a. Klassen skall innehålla all kod som implementation av interfacet kräver, med undantag av metoden `getPlanetInfo`, som skall definieras som en abstrakt metod i klassen. Du får gärna deklarera variabler och skriva flera metoder ifall du behöver dem (men endast sådant som behövs för lösningen). En egenskap är nödvändig: planetens gravitation. Deklarera en variabel för denna.

Definiera `getPlanetInfo` i klassen `Planet`, som skall returnera en String. I denna del, skriv bara de ändringar som du behöver göra i klassen `Planet`.

9c: Konkreta klasser

Skriv två klasser, `GasGiant` och `Terrestrial`, som subklasser till `Planet`. Deklarera en instansvariabel i varje klass. enligt följande:

- Klassen `GasGiant`: `hydrogenPercentage` (typ double), som innehåller ett värde på andelen väte i planetens komposition.
- Klassen `Terrestrial`: `noOfSpecies` (typ int), som innehåller antalet unika arter på en planet.

Båda klasser skall överskugga (override) den abstrakta metoden `getPlanetInfo`. Metoden kan returnera en text-sträng som: "Jupiter" respektive "Jorden", alternativt skriva ut all information om planeten, se testutskrift i slutet på uppgiften (efter 9d).

Klasserna behöver inte vara stora utan de skall vara så pass kompletta att de tillsammans med `Planet` och `IPlanet` kan kompileras. Små syntax-fel är tillåtna.

9d: Dynamisk bindning

Skriv färdigt metoden `createPlanet` (se nedan) så main-metodens anrop fungerar för följande testvärden:

GasGiant:

- hydrogenPercentage = 90 %
- gravity = 24,79 N/kg
- namn = Jupiter

Terrestrial:

- noOfSpecies = 8 700 000
- gravity = 9,82 N/kg
- namn = Terra

```
public class Main {
    public static void main(String[] args) {
        createPlanet(PlanetType.Gas_Giant);
        createPlanet(PlanetType.Terrestrial);
    }

    private static void createPlanet(PlanetType planetType) {
        // Komplettera
        // Krav:
        //   dynamisk bindning
        //   metoderna getPlanetInfo och calculateWeight måste anropas här
        //   eller i klasserna.
    }
}
```

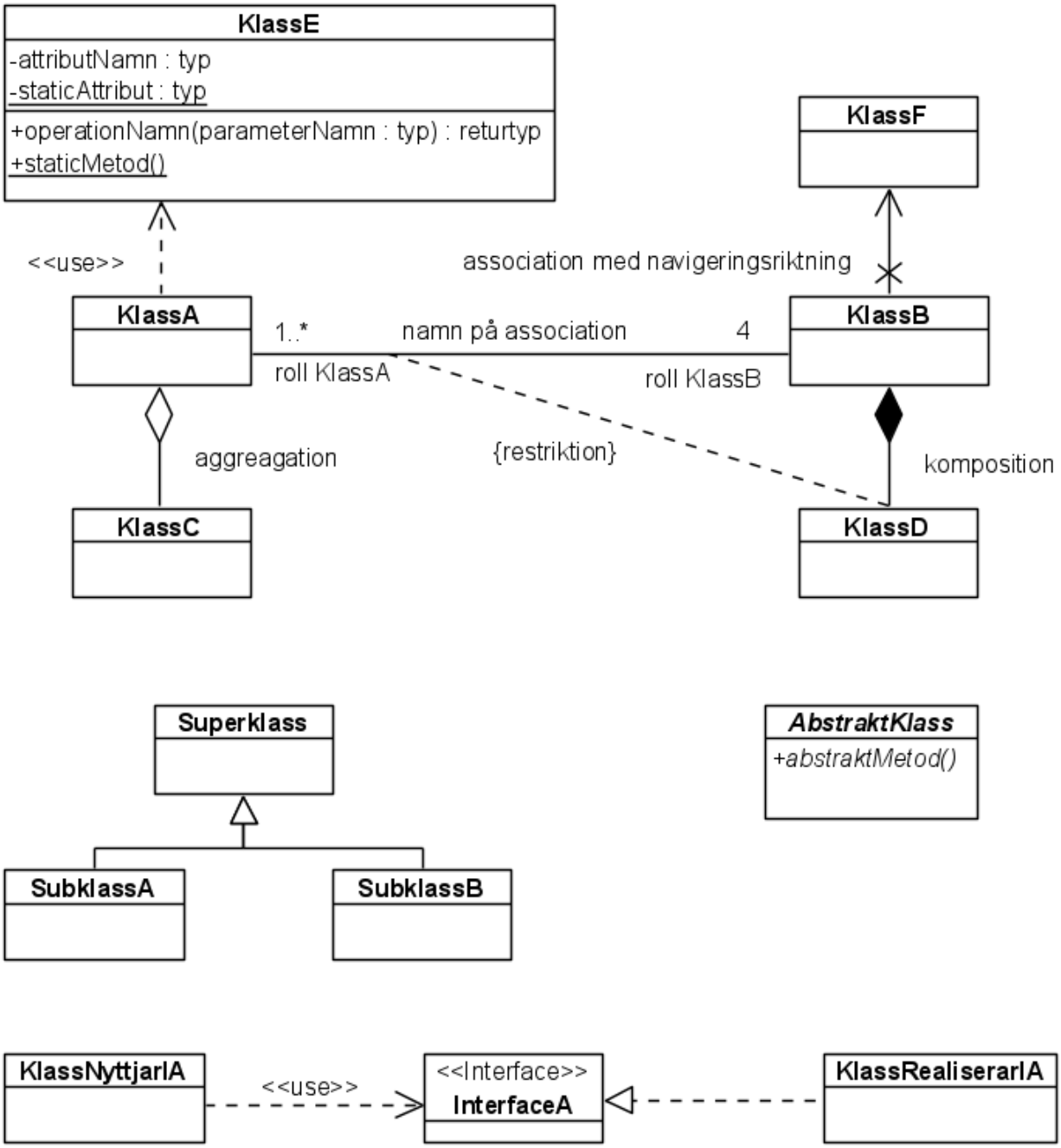
```
Utdata från main-metoden:
Jupiter:
 gravitation = 24.79 N/kg
 väte = 90.0 %
 100 kg på Jupiter har en tyngd av 2479.0 N

Terra:
 gravitation = 9.82 N/kg
 antal arter = 8700000
 100 kg på Terra har en tyngd av 982.0 N
```

Lösningsförslag, OBS! Innehåller konstruktorer, men inget krav i uppgift. Används setters som lösning istället är det ok:

```
public enum PlanetType {
    Planetoid,      // Class D
    Gas_Giant,      // Class J or T
    Terrestrial,    // Class M
    Rogue           // Class R
}


public class Main {
    public static void main(String[] args) {
        createPlanet(PlanetType.Gas_Giant);
        createPlanet(PlanetType.Terrestrial);
    }

    private static void createPlanet(PlanetType planetType) {
        Planet planet=null;
        switch (planetType)
        {
            case Gas_Giant -> planet = new GasGiant("Jupiter", 24.79, 90);
            case Terrestrial -> planet = new Terrestrial("Terra", 9.82, 8700000);
            default -> System.out.println("Ingen " + planetType + " implementerad");
        }

        if (planet != null)
        {
            System.out.println(planet.getPlanetInfo());
        }
    }
}


public interface IPlanet {
    void setPlanetType(PlanetType pType);
    PlanetType getPlanetType();
    String getPlanetInfo();
    double calculateWeight(double mass);
}

public abstract class Planet implements  IPlanet{
    private String name;
    private double gravity;
    private PlanetType pType;

    Planet(PlanetType pType, String name, double gravity)
    {
        this.pType=pType;
        this.name=name;
        this.gravity=gravity;
    }

    @Override
    public void setPlanetType(PlanetType pType) {
        this.pType=pType;
    }

    @Override
    public PlanetType getPlanetType() {
        return pType;
    }

    @Override
    public abstract String getPlanetInfo();


    @Override
    public double calculateWeight(double mass) {
        return gravity * mass;
    }

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

```
    public double getGravity()
    {
        return gravity;
    }
}

public class GasGiant extends Planet{
    double hydrogen_percentage;

    GasGiant(String name, double gravity, double hyd)
    {
        super(PlanetType.Gas_Giant, name, gravity );
        hydrogen_percentage=hyd;
    }

    @Override
    public String getPlanetInfo() {
        return getName() + ":\n" +
                " gravitation = " + getGravity() + " N/kg \n" +
                " väte = " + hydrogen_percentage + " % \n" +
                " 100 kg på " + getName() + " har en tyngd av " + calculateWeight(100) + " N
\n";
    }
}


public class Terrestrial extends Planet{

    int noOfSpecies;

    Terrestrial(String name, double gravity, int noOfSpecies)
    {
        super(PlanetType.Terrestrial, name, gravity );
        this.noOfSpecies=noOfSpecies;
    }

    @Override
    public String getPlanetInfo() {
        return getName() + ":\n" +
                " gravitation = " + getGravity() + " N/kg \n" +
                " antal arter = " + noOfSpecies + " \n" +
                " 100 kg på " + getName() + " har en tyngd av " + calculateWeight(100) + " N
\n";
    }
}
```

Bilaga 1 UML-notation klassdiagram



Bilaga 2 UML-notation sekvensdiagram