Tentamen på
Kurs:
DA339A, Objektorienterad programmering

Provkod: 2002 Tentamen 2, 2,5 hp

210809 kl 8.15 – 13.15

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 så 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.

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 dynamiskt diagram.
B	En klass kan ha en association till sig själv.
C	Vid generalisering så ska multiplicitet alltid sättas ut.
D	Navigeringsriktning på en association visar vilken klass som anropar den andra klassen i associationen.
E	En klass B kan inte vara superklass till en annan klass C och subklass till en tredje klass A samtidigt.
F	En klass av typen controller i MVC-mönstret hanterar interaktionen med användaren via ett grafiskt gränssnitt.
G	Subklassens egen konstruktor anropas och exekveras alltid först och sen exekveras superklassens konstruktor.
H	Alla abstrakta metoder i en abstrakt klass måste implementeras, dvs. skall kodas, i klassens subklass. Om subklassen inte kan implementera en abstrakt metod, skall subklassen definieras också som `abstract` för att subklassen en nivå lägre att skriva kod till abstrakta metoden.
I	Ett interface är en typ (datatyp) som innehåller en uppsättning av abstrakta metoder, variabler, konstanter och enum.
J	En abstrakt klass är en klass som innehåller abstrakta metoder, icke-abstrakta metoder, dvs. metoder med implementation, variabler och konstanter.
K	Ett interface kan inte implementera ett annat interface.
L	Ett `try-` block måste följas av antingen ett finally eller ett catch block, eller båda.
M	Överlagring (overloading) innebär att man kan ha flera metoder samma namn men med olika uppsättning av metod parametrar. Vid överskuggning (overriding) har man en och samma metod i superklassen med olika implementation i olika subklasser i en klasshierarki.
N	Anta att metoden **`public void setNames (String [] names){/*kod*/ }`** finns i en klass. Metoden kan anropas från en annan metod så här: `setNames(studNames[]);`

O	En array som skickas som argument till en metod fungerar precis som att primitiva typer, int, double, etc. vilket innebär att ev. ändringar i metoden kommer inte att påverka den ursprungliga arrayen i den anropande metoden.
P	Klassvariabler och klassmetoder finns i endast en uppsättning som delas mellan samtliga instanser av klassen. Modifieraren `static` används för att ange att en variabel eller en metod är en klassvariabel resp klassmetod
Q	Om vi inte definierar metoden `toString` finns ändå metoden tillgänglig, eftersom den ärvs av klassen `Object` som är superklass till alla klasser.
R	Nyckelordet `extends` används för att implementera ett interface och nyckelordet `implements` används för att ärva en superklass.
S	Undantag (exceptions) är fel som bara inträffar när programmet körs.
T	Nyckelordet `throws` används i en metods signatur för att tala om att ett undantag kan inträffa i metoden.

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

Uppgift 2 2p

Du har följande förslag på en superklass och tre subklasser till denna i ett system hos ett företag som hanterar, säljer och hyr ut bostäder:

Super-klass	Sub-klasser
Bostad	Hyresgäst, Storlek, Radhus

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

Svar:
Hyresgäst – inte generalisering, är lämpligtvis en annan klass associerad med Bostad
Storlek – inte generalisering, är lämpligast ett attribut av någon inbyggd datatyp hos Bostad
Radhus – kan vara en subklass till bostad då det kan betraktas som en speciell typ av bostad som exempelvis har en tomt till skillnad från en lägenhet

Uppgift 3 2p

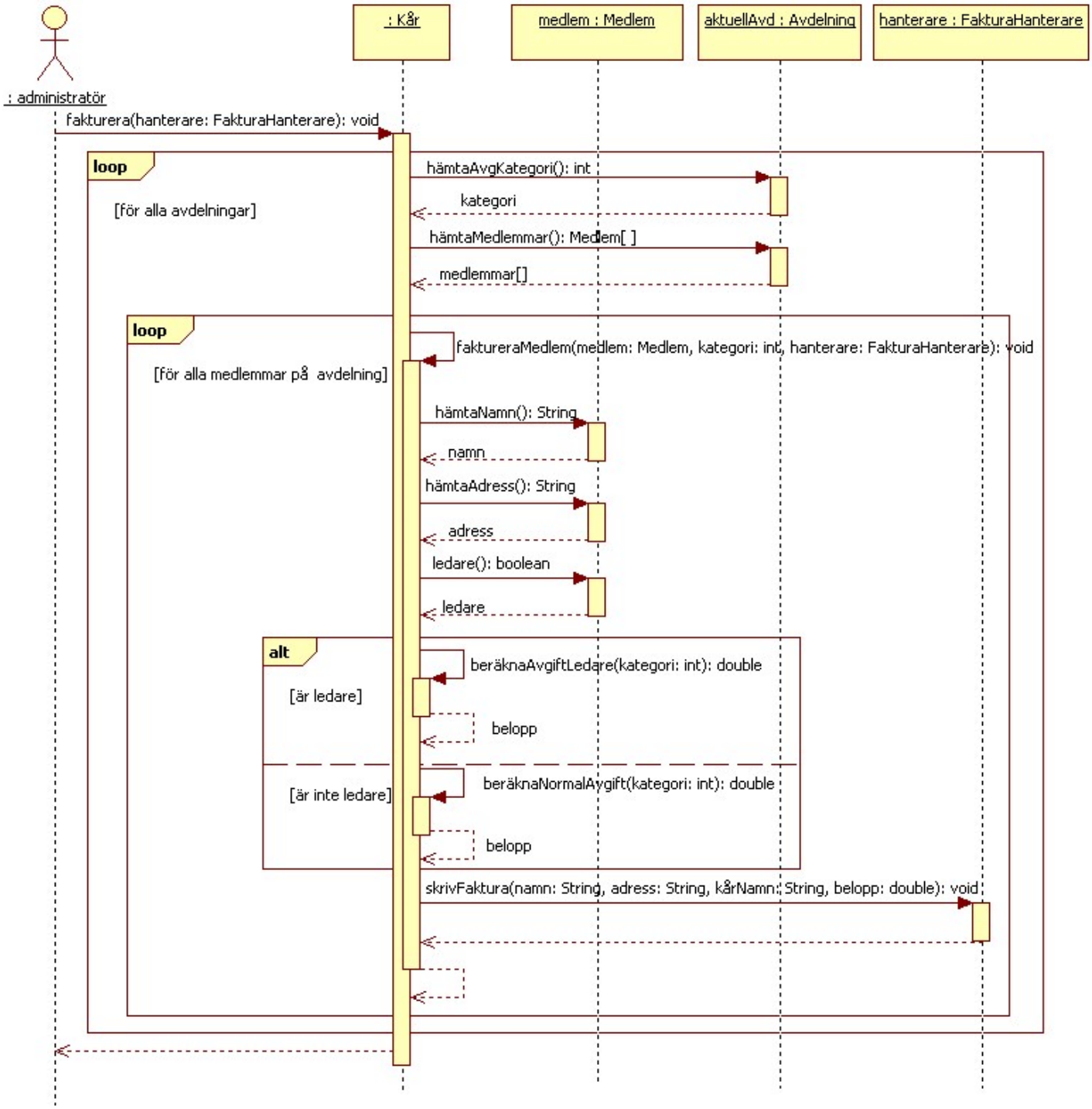
Ge ett exempel på en lämplig komposition och ett exempel på en lämplig aggregation och använd dessa exempel för att kortfattat förklara skillnaden mellan en komposition och en aggregation.

Kommentar runt svar:
Exempel som ges behöver vara rimliga/bra exempel på helhet som byggs upp av delar för att illustrera grunden i vad aggregation/komposition innebär. Förklaringen i skillnad behöver innehålla att en komposition illustrerar ett starkare förhållande mellan helheten och delarna där delarna endast kan ingå i en helhet än i en aggregation där delarna kan ingå i flera olika helheter.

Uppgift 4 4p

Sekvensdiagrammet nedan visar vad som sker när en scoutkår ska skicka fakturor på medlemsavgift till sina medlemmar.

Skriv kod för klassen Kår som finns i diagrammet. Koden skall visa vilka metoder klassen har samt den kod i metoderna som går att utläsa från diagrammet. Vissa antaganden kan göras om villkor i iterationer och selektioner samt villkor i dessa. Programspråket förutsätts vara Java.



Lösning

```
public class Kår {
  String kårNamn;
  ...
  public double beräknaNormalAvgift(int kategori) {...}
  public double beräknaAvgiftLedare(int kategori) {...}
  public void fakturera(FakturaHanterare hanterare) {
    for(alla avdelningar){
      int kategori = akutellAvd.hämtaAvgkategori();
      Medlem[] medlemmar = aktuellAvd.hämtaMedlemmar();
      for(int i=0; i<medlemmar.length; i++){
        faktureraMedlem(medlemmar[i], kategori);
      }
    }
  }

  public void faktureraMedlem(Medlem medlem, int kategori,
FakturaHanterare hanterare) {
    String namn = medlem.hämtaNamn();
    String adress = medlem.hämtaAdress();
    boolean ledare = medlem.ledare();
    double belopp;
    if(ledare){
      belopp = beräknaAvgiftLedare(kategori);
    }
    else{
      belopp = beräknaNormalAvgift(kategori);
    }
    hanterare.skrivFaktura(namn, adress, kårNamn, belopp);
  }
}
```

Uppgift 5 4p

5a 1,5p

Du ska skriva en enkel klass som representerar en bil och döpa klassen till `Car`. Klassen ska innehålla data om bilens modell, pris och antal dörrar (instansvariabler). Använd lämpliga typer och variabelnamn till variablerna.

5b 1,5p

Skapa en enda konstruktor för klassen `Car`. I konstruktorn ska alla instansvariabler få värden tilldelade till sig. Instansvariabler skall vara deklarerade med modifieraren private.

5c 1p

Skriv en toString-metod som returnerar en string med informationen om objektets data.

Krav: Alla instansvariabler skall vara deklarerade med modifieraren private.

Du kan svara på deluppgifterna i samma svar – det vill säga skriva allt som efterfrågas för klassen Car i ett svar för uppgift 5.

Obs. Denna klass används i uppgift 6.

Lösningsföslag

```
public class Car
{
    private int numOfDoors;
    private double price;
    private String model;

    public Car(int doors, double price, String model)
    {
        numOfDoors = doors;
        this.price = price;
        this.model = model;
    }

    @Override
    public String toString()
    {
        String strOut = String.format("Model: %s Dörrar: %d Pris:
%.2f", model, numOfDoors, 2500000.0);

        return strOut;
    }
}
```

Uppgift 6 4p

6a 1p

Skapa en klass Taxi. Denna klass skall ha en instansvariabel taxiID (String) och skall ärva klassen Car som du skrivit i uppgift 5.

6b 2p

Skapa en enda konstruktor för klassen `Taxi` med nödvändiga parametrar för att initiera alla instansvariabler.

6c 1p

Skriv en toString-metod i klassen `Taxi` så att resultatet nedan erhålls.

En testkörning av metoden Main (nedan) ger resultatet som följer.

```
public class Main
{
    public static void main(String[] args)
    {
        Taxi car = new Taxi(5, 250000, "Tesla", "ABC-0001S");
        System.out.println(car);
    }
}
```

Output:

```
Taxi ID: ABC-0001S
Modell: Tesla Dörrar: 5 Pris: 2500000,00
```

Du kan svara på deluppgifterna i samma svar – det vill säga skriva allt som efterfrågas för klassen Taxi i ett svar för uppgift 6.

Lösningsförslag

```
public class Taxi extends Car
{
    private String taxiID;

    public Taxi(int doors, double price, String model, String id)
    {
        super(doors,price,model);
        taxiID = id;

    }

    @Override
    public String toString()
    {
        return "Taxi ID: " + taxiID +"\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. Skriv ut namn på alla associationer för att förtydliga dessa. Om det går att ange en multiplicitet för en association så skall 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:
Man vill skapa ett program som stödjer skrivandet av böcker, vilket är en stor, mer eller mindre seriös, hobbyverksamhet för många. Detta är den inledande beskrivningen man har efter några första intervjuer med presumtiva användare av systemet.

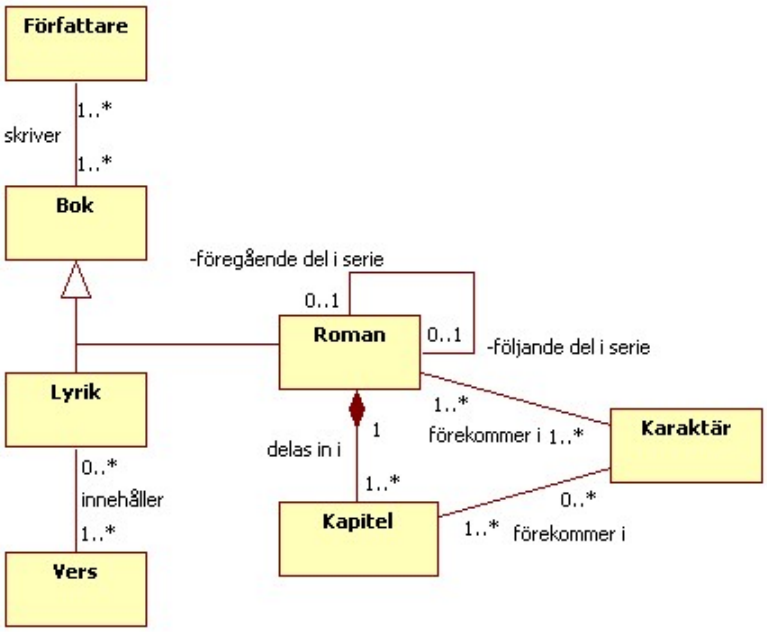
Man hittade två tydliga typer av böcker som skrevs - romaner och lyrik. Dessa är böcker men som skiljer sig åt väsentligt i hur detaljerna ser ut för deras uppbyggnad. Flera av författarna man intervjuat arbetar med andra författare på samma bok. Så en bok kan ha flera författare till den innehållande texten.

Lyrik innehåller verser. De som författade lyrik ville kunna hantera verser för sig och kunna sätta i hop verser till böcker med lyrik. En vers kan kanske komma att ingå i olika böcker i slutänden.

De författare som skrev romaner vill kunna hantera flera romaner som följer på varandra i en serie. Ingen har dock gett utryck för att det behövs en trädstruktur utan det räcker att kunna hantera en kedja av romaner. Så man behöver veta vilken roman som utspelar sig före en annan och vilken som kommer efter. Givetvis så förekommer böcker som inte ingår i någon serie.

Det finns en mängd karaktärer som behöver hanteras gällande romanerna. Man behöver hålla reda på vilka karaktärer som förekommer i vilka romaner men också på en mer detaljerad nivå i vissa fall. I de mer detaljerade fallen så behöver man veta i vilka kapitel som en karaktär förekommer. Romaner utan direkt kapitelindelning beräknas ha endast ett kapitel. Man ansåg också att det kunde finnas kapitel som inte innehåll någon väsentlig karaktär.

Lösningsförslag

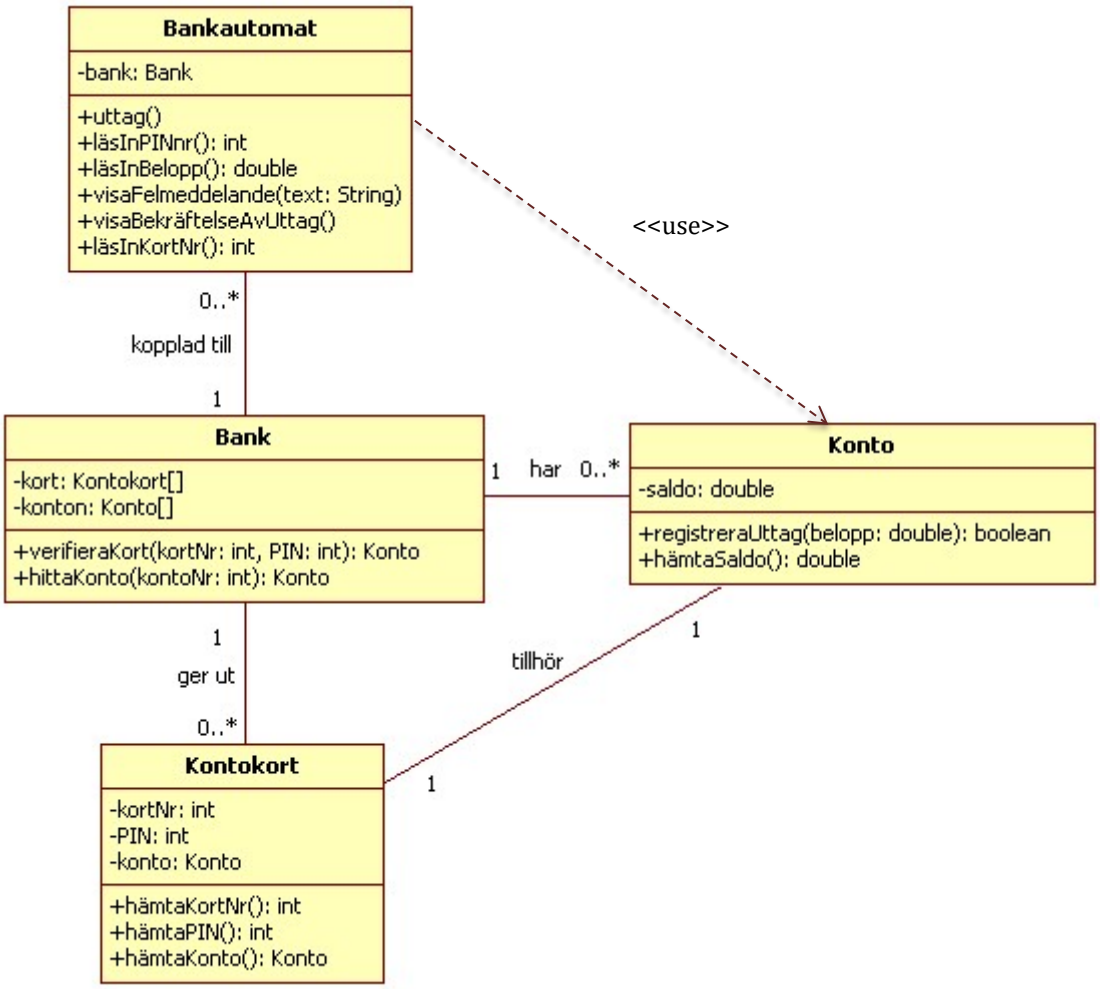


Uppgift 8 7p

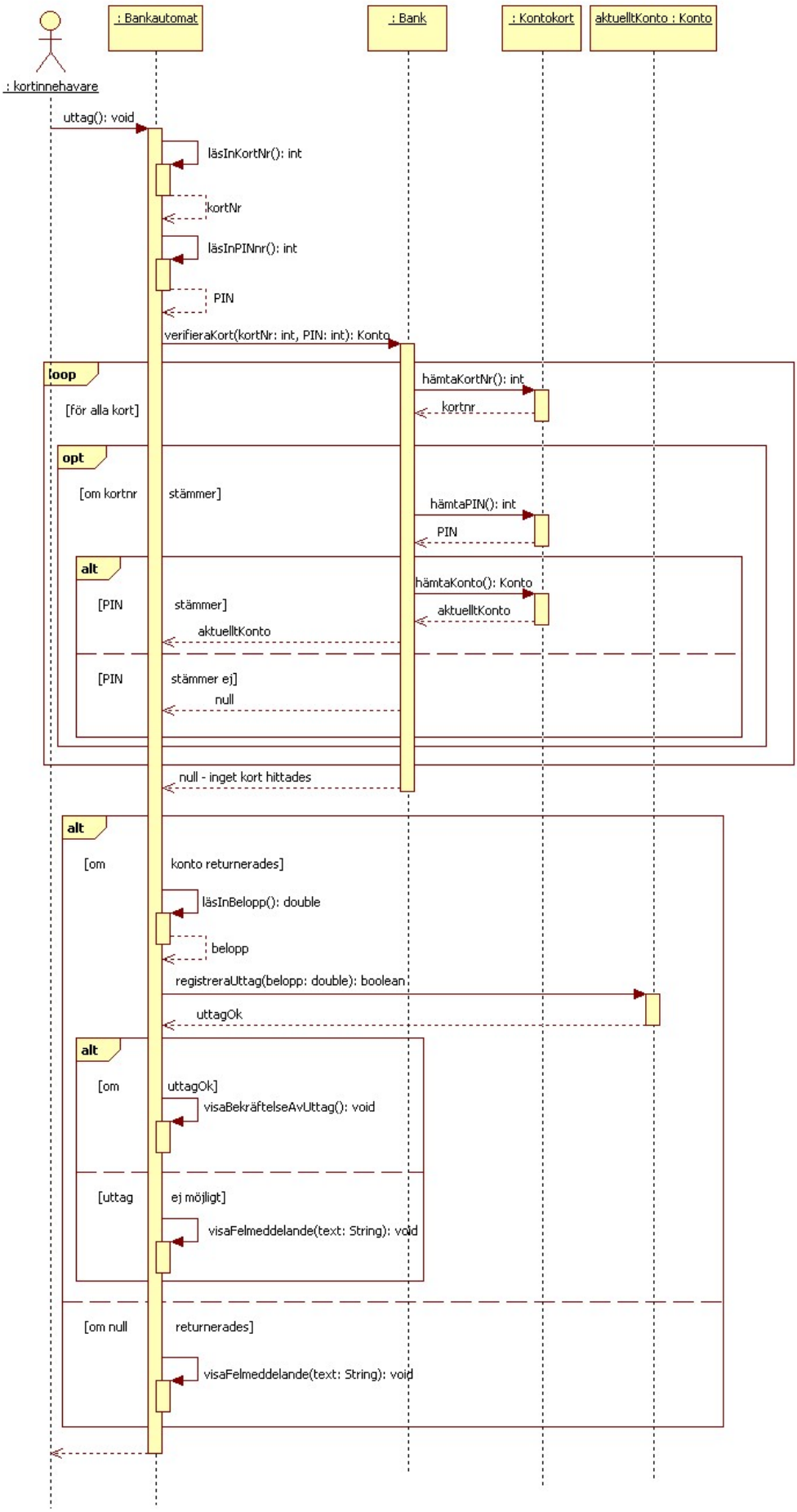
Klassdiagrammet nedan beskriver ett system som hanterar ett banksystem.

Givet klassdiagrammet nedan rita ett sekvensdiagram som visar vad som sker när ett uttag sker enligt följande modell: Bankautomaten som är representerad av ett Bankautomat-objekt känner endast till ett Bank-objekt och måste hämta all sin information och övriga objekt via detta. Bankomaten läser in PIN-kod från kortinnehavaren och läser av kortnumret från plastkortet. Plastkortet kan dock inte i sig bekräfta något om PIN-nummer utan detta måste bekräftas via banken. Lösningen ska innebära att bankomaten visar felmeddelanden för om kontot inte kan hittas i banken, om PIN inte stämmer eller saldot är för litet för uttaget. Lösningen ska visa vad som sker för att hitt ett konto och verifiera PIN.

Du får inte lägga till attribut eller operationer till klasserna. Du får däremot skapa en aktör som representerar kortinnehavaren som interagerar med Bankomat-objektet.



Lösningsförslag



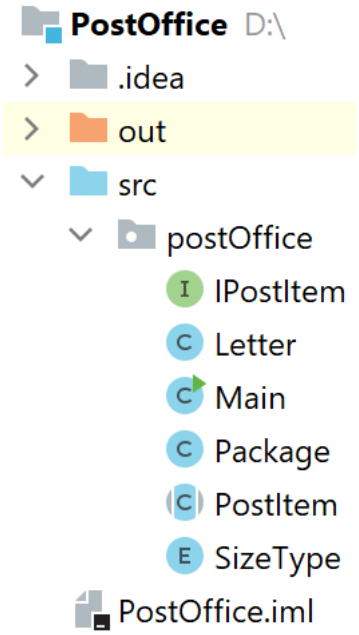
Uppgifter för betyg VG

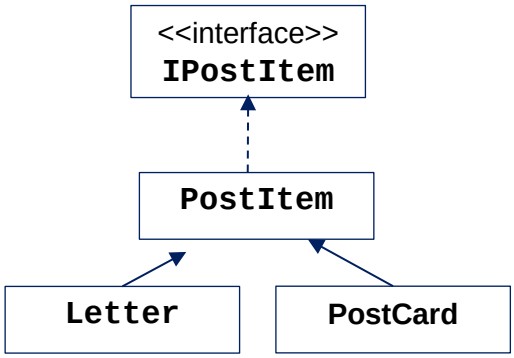
Uppgift 9

Ett företag har öppnat ett litet postkontor. Ditt uppdrag är att skriva en applikation som kan ta hand om företagets försändelser. Applikationen är stor men här löser vi en liten del av den.

Uppgiften består av en **enum** (storlekar), ett interface **`IPostItem`**, och ett antal klasser nämligen **`PostItem`**, **`Letter`** (brev) och **`Package`** (paket) samt en startklass (**`Main`**).

Klassen **`PostItem`** är en abstrakt klass som implementerar **`IPostItem`**. Klassen **`Letter`** och **`Package`** är två subklasser till klassen **`PostItem`**..






```
public enum SizeType
{
    Normal,
    Large,
    Heavy,
    Long
}
```

```
public interface IPostItem
{
    public void setSize(SizeType size);
    public SizeType getSize();
    public void setWeight(double weight);
    public double getWeight();
    public double calcCost();
}
```

Utgående från de ovanstående bilderna och skriv kod enligt följande anvisningar.

Krav:

- Alla instansvariabler skall deklareras private.
- Koden skall vara välstrukturerad och lätt att läsa.
- Variabler och metoder skall ha lämpliga namn.
- För att få VG måste du använda arv och polymorfism genom dynamisk bindning.
- Minst en try-catch (Exception e) sats användas inne i någon metod som du bedömer vara lämplig.
- Alla klasser (förutom Main) skall bara ha en default-konstruktor. Konstruktorer med parametrar skall inte förekomma.
- Alla klasser (förutom Main) skall ha en toString-metod. Subklasserna skall anropa superklassens toString-metod.

9a

Skriv klassen **`MailItem`** (försändelse) som skall implementera interfacet `IPostOffice`. Metoden `calcCost` skall vara en **`abstract`** metod. Skriv en toString-metod som ger info om storleken och vikten av post-objektet.

Tips: Du behöver några instansvariabler för att kunna implementera interfacets metoder. Bestäm själv dessa och deras typer.

9b

Skriv klassen **`Letter`** (brev). Deklarera en private boolesk variabel `isRegistered` (rekommenderad post). Skriv metoden `calcCost` enligt följande värden.

Vikt	Kostnad
mindre än 20	10.50
mellan 20 och 99	56.90
mellan 100 och 499	150.00
500 eller mer	-1.0

Om brevet är rekommenderat (om isRegistered är true) ska ett belopp på 0.1 * vikten läggas till på kostnaden. Skriv också en toString-metod.

9c

Skriv klassen **`Package`** (paket) som skall ha en instansvariabel double `maxDimension`, med set- och get-metoder, och en toString-metod. Dessutom skall metoden `calcCost` räkna beloppen enligt följande tabell:

Size	Kostnad
Normal	35.0
Large	54.0
Alla andra storlekar	130.0

9d

För att testa programmet utgå ifrån metoden main och skriv färdigt metoden `sendPostItem` så att applikationen ska ge följande output:

Följande output erhålls när main-metoden körs (utskriftsraderna motsvarar //1, //2 och //3 i koden).

```
Registered letter, Size: Normal, 28,0 g, cost = 59,70
Package, Size: Large, 67,0 g, max dim. 43,5 cm, cost = 54,00
Not a valid post item!
```

```
public class Main
{
    static enum ItemType{Letter, Package, Other};
    static ItemType item;

    public static void main(String[] args)
    {
        //Testa appen:
        sendPostItem(ItemType.Letter, 28.0, SizeType.Normal); //1
        sendPostItem(ItemType.Package, 67.0, SizeType.Large); //2
        sendPostItem(ItemType.Other, 0, null);                //3
    }
    static void sendPostItem(ItemType item, double weight,
                             SizeType size)
    {
        PostItem postItem = null;
        boolean ok = true;

        switch (item)
        {
```

//Koda färdigt switchen och resten av metoden!
//Använd dynamisk bindning – mkt viktigt!

Lösningsförslag:

```
public abstract class MailItem implements IPostItem
{
    private SizeType size;
    private double weight;

    public MailItem()
    {
        size = SizeType.Normal;
        weight = 0.0;
    }

    //implementation of the interface methods.
    @Override
    public void setSize(SizeType size)
    {
        this.size = size;
    }
    @Override
```

```
    public SizeType getSize()
    {
        return size;
    }
    @Override
    public void setWeight(double weight)
    {
        this.weight = weight;
    }
    public double getWeight()
    {
        return weight;
    }
    //different for different type of post item
    public abstract double calcCost();
     @Override
    public String toString(){
        return String.format("Size: %s, %.1f g", size.toString(),weight);
    }
}

public class Letter extends MailItem
{
    private boolean isRegistered;

    public Letter ()
    {
        isRegistered = false;
    }

     public void setRegistered(boolean registered)
     {
        isRegistered = registered;
     }

    @Override
    public double calcCost()
    {
        double cost = 0.0;
        double weight = getWeight();

        if(weight < 20){
            cost = 10.5;
        }
        else if(weight > 20 && weight <= 99){
            cost = 56.90;
        }
        else if(weight > 99 && weight <= 499){
            cost = 150;
        }
        else if (weight > 499){
            cost = -1.0;
        }
        if (isRegistered)
            cost += 0.1*weight;

        return cost;
    }

    public String toString()
    {
```

```
        String isReg = isRegistered ? "Registered letter" : "Standard
letter";
        String strOut = String.format("%s, %s, cost = %.2f",
                isReg, super.toString(), calcCost());
        return strOut;
    }
}
```

```
public class Package extends MailItem
{

    private double maxDimension;
    public Package()
    {
    }

    public double getMaxDimension()
    {
        return maxDimension;
    }

    public void setMaxDimension(double maxDimension)
    {
        this.maxDimension = maxDimension;
    }

    @Override
    public double calcCost()
    {
        double cost = 0.0;
        SizeType size = getSize();
        if (size == SizeType.Normal)
            cost = 35.0;
        else if (size == SizeType.Large)
            cost = 54.0;
        else
            cost = 130.0;

        return cost;
    }
    public String toString()
    {
        String strOut = String.format("Package, %s, max dim. %.1f cm, cost
= %.2f",
                super.toString(),maxDimension, calcCost());
        return strOut;
    }
}


public class Main
{
    static enum ItemType{Letter, Package, Other};
    static ItemType item;

    public static void main(String[] args)
    {
        //Testa appen:
        sendPostItem(ItemType.Letter, 28.0, SizeType.Normal);
        sendPostItem(ItemType.Package, 67.0, SizeType.Large);
        sendPostItem(ItemType.Other, 0, null);
    }
    static void sendPostItem(ItemType item, double weight, SizeType size)
    {
        MailItem mailItem = null;
```

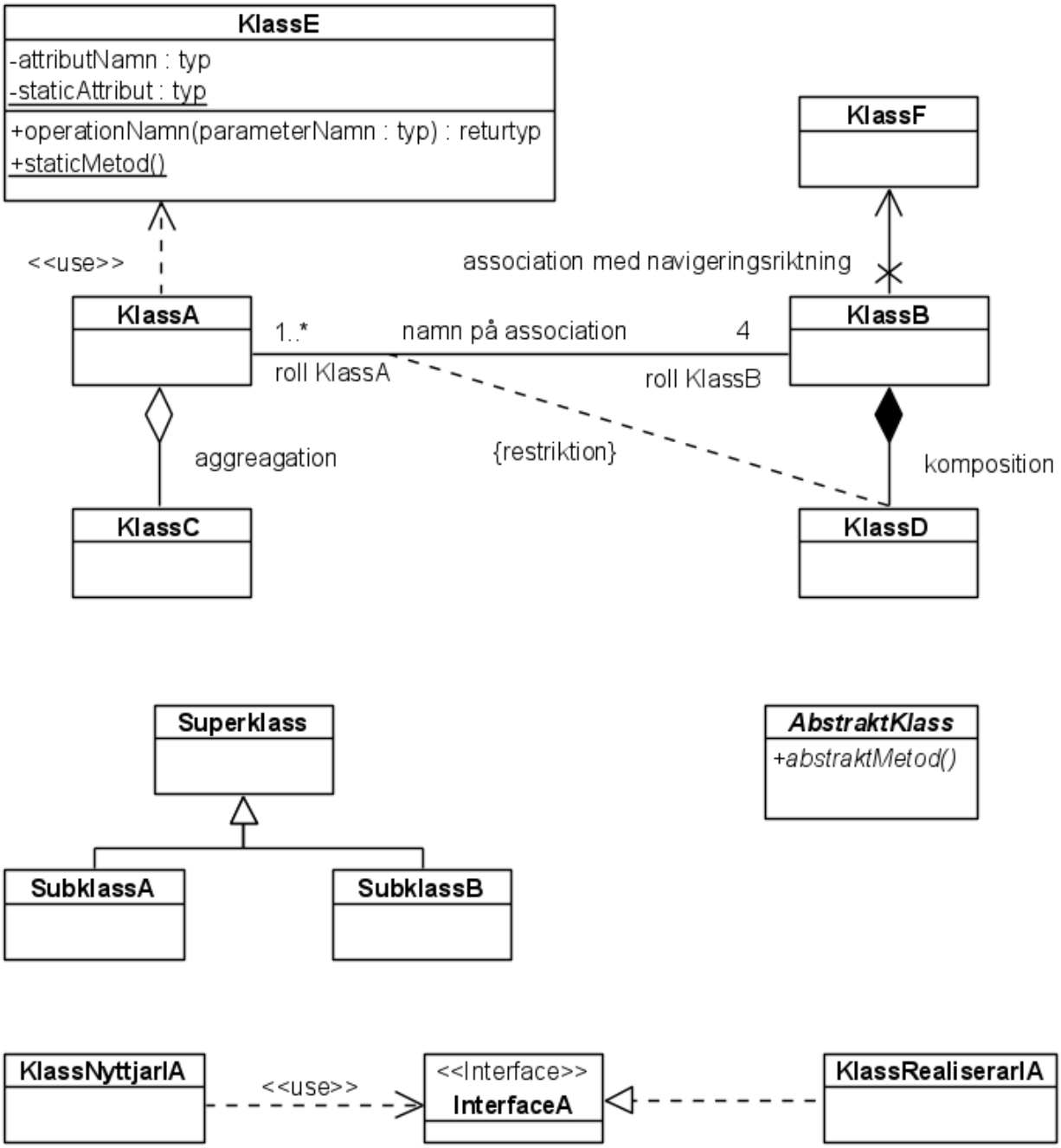
```
        boolean ok = true;

        switch (item)
        {
            case Letter:
                mailItem = new Letter();
                //Registered
                ((Letter) mailItem).setRegistered(true);
                break;
            case Package:
                mailItem = new Package();
                ((Package) mailItem).setMaxDimension(43.5);
                break;
            default:
                ok = false;
                break;
        }
        if (ok)
        {
            mailItem.setWeight(weight);
            mailItem.setSize(size);
            System.out.println(mailItem.toString());
        }
        else
            System.out.println("Not a valid post item!");

    }

}
```

Bilaga 1 UML-notation klassdiagram



Bilaga 2 UML-notation sekvensdiagram