

Tentamen på

Kurs:

DA339A, Objektorienterad programmering

Provkod: 2002 Tentamen 2, 2,5 hp

220105 kl 8.15 – 13.15

Tillåtna hjälpmedel:

- Inga tillåtna hjälpmedel utöver det som finns i den tryckta 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 stöddlinjerna 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.

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 sekvensdiagram är ett statiskt diagram.
B	En klass kan ha en association till sig själv.
C	Ett objekt har alltid en unik identitet.
D	Abstrakta klasser används för att skapa objekt av när man är osäker på vilken klass det kommer att vara under exekvering.
E	Navigeringsriktning på en association anger vilken klass som anropar den andra klassen i associationen
F	En klass kan vara både en superklass och en subklass.
G	Vid hantering av undantag, kan det finnas endast ett try och ett finally block men flera catch block.
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 klass kan ha endast en superklass och kan implementera endast ett interface.
K	En superklass konstruktor anropas och exekveras alltid först och sen exekveras subklassens egen konstruktor.
L	Ett interface kan implementera ett annat interface.
M	Vid överlagring (overloading) sker bindning(dvs. vilken metod skall anropas)statiskt vid kompilering.
N	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.
O	En klass som ärver från en annan klass kallas för en superklass eller basklass.
P	Nyckelordet throws används tillsammans en metods signatur och därmed slipper man att använda try-catch inne i metoden.
Q	När typen av objekt är bestämt vid kompilering, sker en dynamisk bindning.

R	Alla metoder i ett interface är implicit public och abstract .
S	När en array eller ett annat objekt skickas som argument till en metod, tillämpas en mekanism som heter pass-by-value , vilket betyder att en kopia av objektets data skickas till metoden.
T	Nyckelordet throw används inne i en metod och följs av en undantagstyp och kastar alltid undantaget.

Svar:

Korrekta/sanna: B, C, E, F, G, H, K, M, P, R, T

Felaktiga/falska: A, D, I, J, L, N, O, Q, S

Uppgift 2 2p

Redogör kortfattat för någon fördel och någon nackdel med att använda generalisering/arv i programkod.

Svar: Se exempel på fördelar och nackdelar med generalisering i slides till föreläsning F17.

Uppgift 3 2p

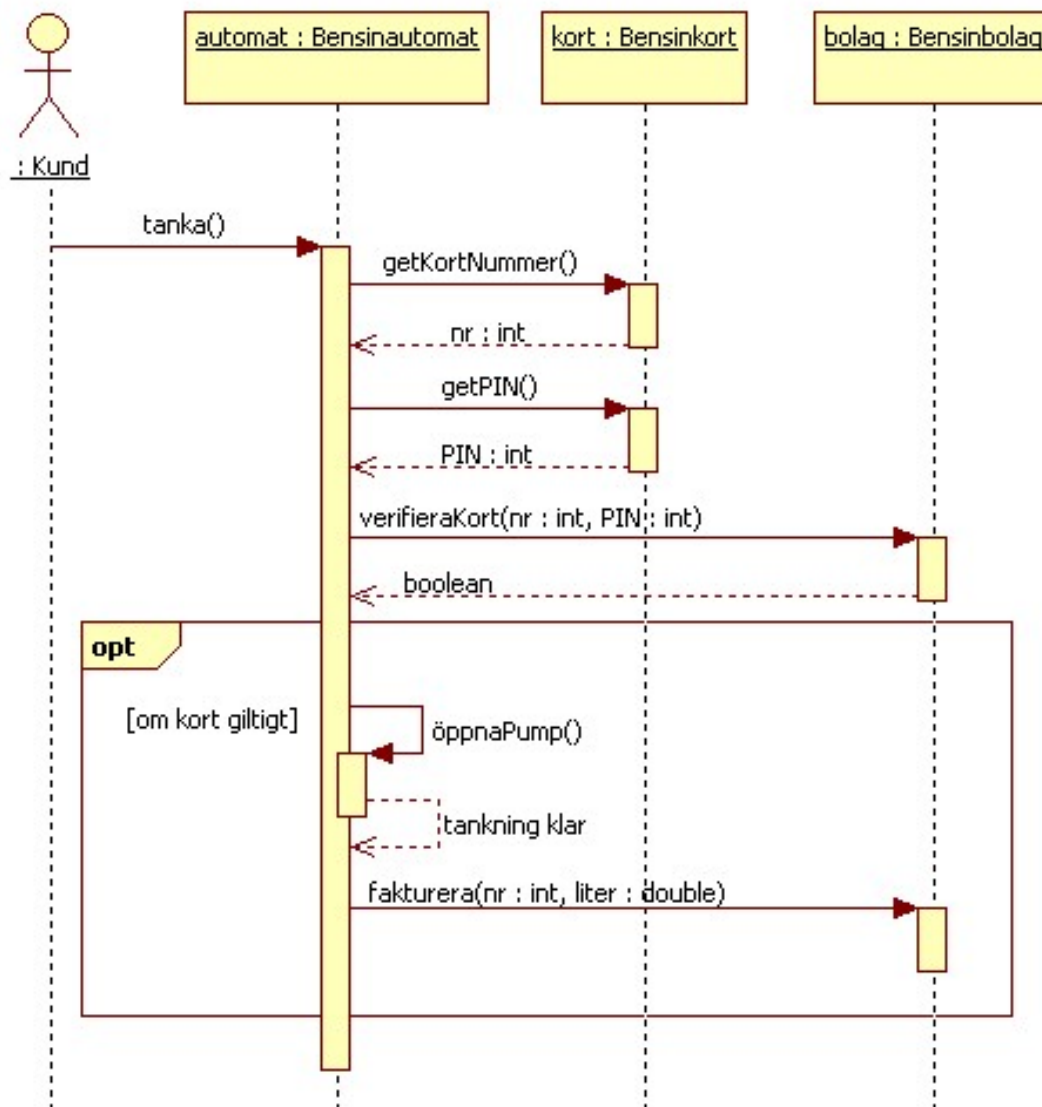
Förklara kortfattat varför vi i en strikt Model-View-Controller-arkitektur inte vill att Boundary-klasser och Entity-klasser ska kommunicera direkt med varandra.

Svar: Se förklaringar i slides och litteratur till föreläsning F16.

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 de klasser som finns i diagrammet.

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



Lösning

```
class Bensinautomat{
    public void tanka(){
        int nr = kort.getKortNummer();
        int PIN = kort.getPIN();
        boolean okKort = bolag.verifieraKort(nr, PIN);
        if(okKort){
            öppnaPump();
            bolag.fakturera(nr, literTankat);
        }
    }

    public void öppnaPump(){...}
}

class Bensinkort{
    public int getKortNummer(){...}
    public int getPIN(){...}
}

class Bensinbolag{
    public boolean verifieraKort(int nr, int PIN){...}
    public void fakturera(int nr, double liter){...}
}
```

Uppgift 5 4p

5a 1,5p

Du ska skriva en enkel klass som representerar en cykel och döpa klassen till *Bike*. Klassen ska innehålla data om cykelns modell, pris och antal växlar(instansvariabler). Använd lämpliga typer och variabelnamn till variablerna.

5b 1,5p

Skapa en enda konstruktor för klassen *Bike*. I konstruktorn, som skall ha nödvändiga parametrar, ska alla instansvariabler få värden tilldelade till sig via parametrarna. Instansvariabler skall vara deklarerade med modifieraren `private`.

5c 1p

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

Du kan svara på deluppgifterna i samma svar. Det vill säga skriva allt som efterfrågas för klassen *Bike* i ett svar för uppgift 5.

Obs. Denna klass används i uppgift 6.

Lösningsförslag

```
public class Bike
{
    private int numOfGears;
    private double price;
    private String model;

    public Bike(int gears, double price, String model)
    {
        numOfGears = gears;
        this.price = price;
        this.model = model;
    }

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

        return strOut;
    }
}
```

Uppgift 6 4p

6a 1p

Skapa en klass `MountainBike`. Denna klass skall ha en instansvariabel `bromsTyp` (String) och skall ärvä klassen `Bike` som du skrivit i uppgift 5.

6b 2p

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

6c 1p

Skriv en `toString`-metod i klassen `MountainBike` så att resultatet nedan erhålls. En testkörning av metoden `Main` (nedan) ger resultatet som följer.

```
public class Main
{
    public static void main(String[] args)
    {
        MountainBike mBike = new MountainBike(21, 15000,
"Scott", "Hydrauliska bromsar");
        System.out.println(mBike);
    }
}
```

Output:

```
Bromstyp: Hydrauliska bromsar
Modell: Scott Växlar: 21 Pris: 15000,00
```

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

De 8p som finns på fråga 5 och 6 kan fördelas olika över fler frågor än 2 eller med delfrågor på en fråga om man vill.

Lösningförslag

```
public class MountainBike extends Bike
{
    private String bromsTyp; // brakeType

    public MountainBike(int gears, double price, String model,
        String bt)
    {
        super(gears, price, model);
        bromsTyp = bt;
    }

    @Override
    public String toString()
    {
        return "Bromstyp: " + bromsTyp + "\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.

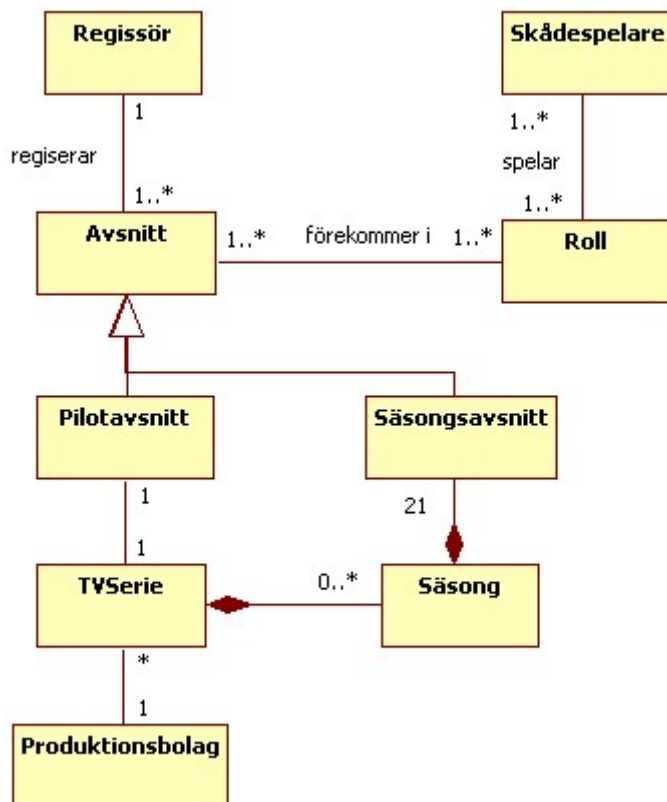
Det är inte nödvändigt att visa attribut eller operationer i klassdiagrammet. Det räcker att skriva ut klassnamn och namn på associationer och multiplicitet. Korrekt UML-notation ska användas (se Bilaga 1).

Systembeskrivning

En TV-serie hos ett produktionsbolag är uppbyggd av olika avsnitt. Det produceras alltid ett pilotavsnitt som testas innan man drar igång produktionen av en hel säsong med 21 avsnitt. Pilotavsnittet tillhör inte någon säsong utan är en speciell typ av avsnitt som står vid sidan av säsongerna. Då det inte är säkert att en series pilotavsnitt tas emot väl av publiken så är det möjligt att serien läggs ner utan att någon säsong startar.

Om en säsong startar så skrivs alla 21 säsongsavsnitt direkt och rollbesättning och val av regissör för varje avsnitt påbörjas. Man har ingen övre gräns för hur länge en serie pågår utan så länge publiken är nöjd så fortsätter man skriva nya säsonger.

Då vissa serier pågår länge så händer det att en roll i en serie byter skådespelare. Det händer också att skådespelare som inte spelar några av de större rollerna dyker upp i flera avsnitt i olika mindre roller. Man har därför ett behov av att hålla reda på vilka roller som förekommer i vilka avsnitt så man inte skriver in två roller som spelas av samma skådespelare i samma avsnitt.



Uppgift 8 7p

Klassdiagrammet nedan beskriver ett system som hanterar journaler på en veterinärklinik.

Rita ett sekvensdiagram som visar vad som sker när man ska skriva ut vad som står i journalen för ett djur enligt följande mall:

Journal

Namn: <djurets namn>

Ägare: <namn ägare 1>, <namn ägare 2>, ...

Journalanteckningar av: <namn på personal>

Journalanteckningar:

<id>, <datum>

<text>

<id>, <datum>

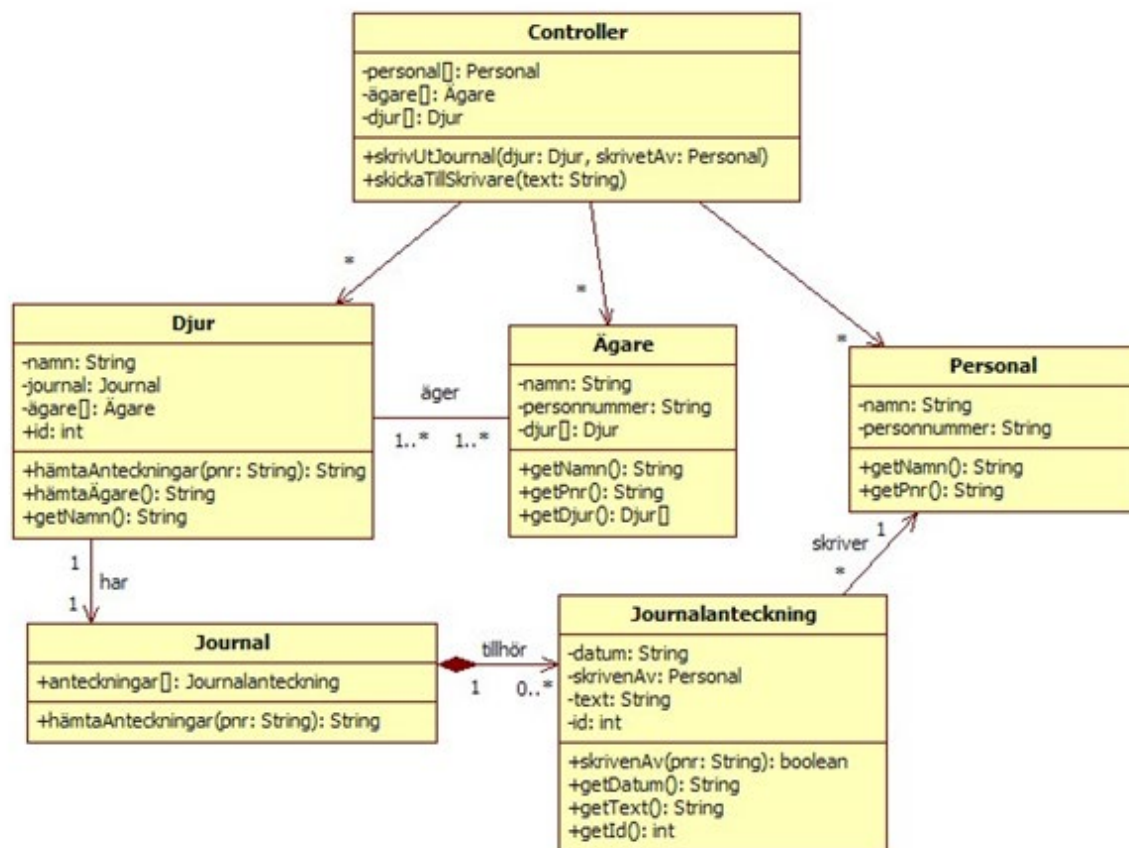
<text>

...

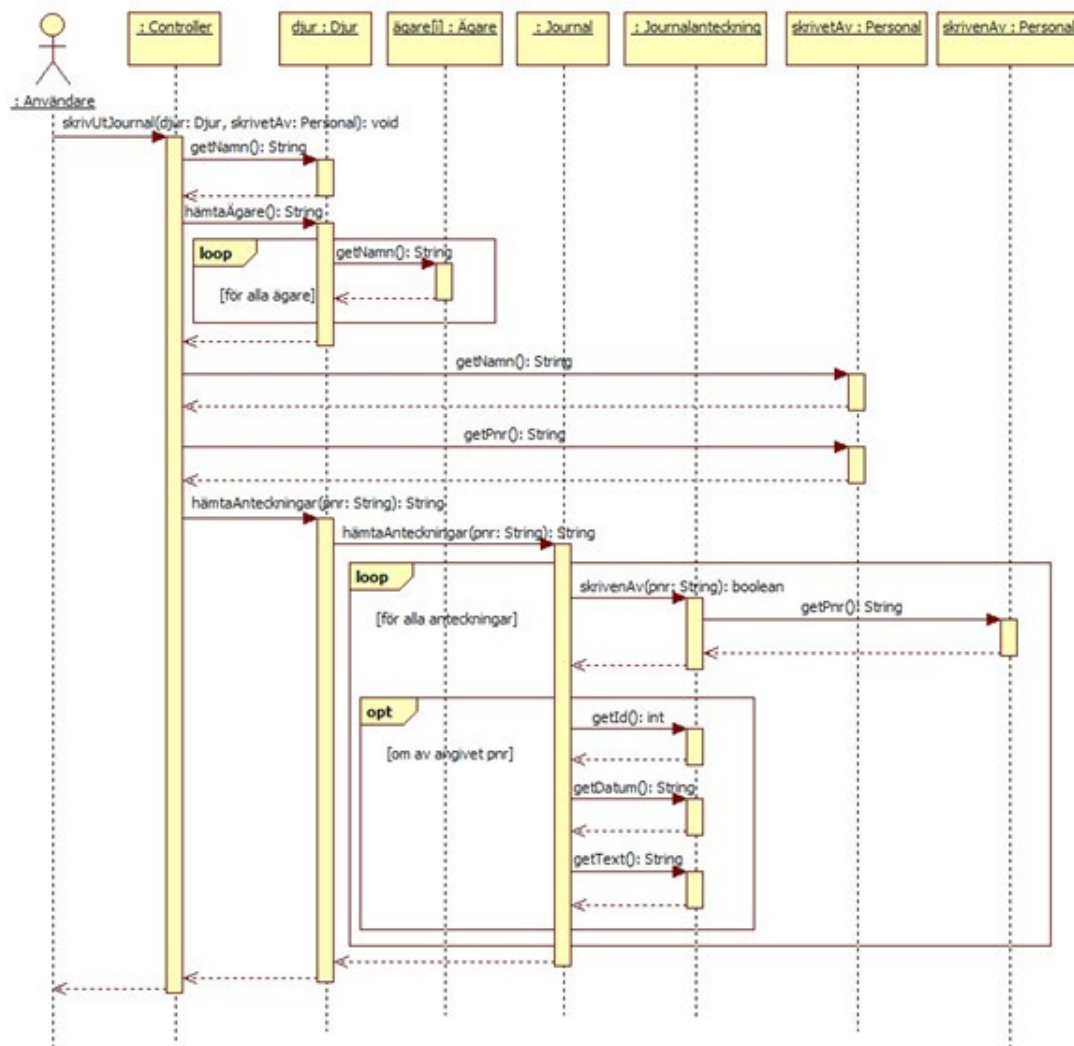
Du skall inte skapa några nya klasser, lägga till operationer eller attribut i befintliga klasser utan skall använda dig av det som visas i klassdiagrammet. Du får dock skapa en aktör som initierar händelsen genom interaktion med ett objekt av klassen Controller via meddelandet: skrivUtJournal(djur: Djur, skrivetAv: Personal). Du kan anta att parametrarna finns givna via den klass där anropet till Controller-objektet sker.

Meddelandet hämtaAnteckningar(pnr: String):String i klassen Journal levererar en textsträng med id, datum och text enligt kursiv text i mallen ovan.

Meddelandet hämtaÄgare():String i klassen Djur levererar en stäng med alla ägares namn.



Lösningsförslag

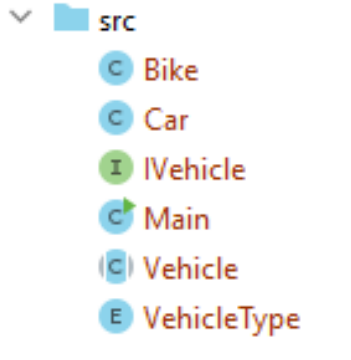


Uppgifter för betyg VG

Uppgift 9

Enumerationen **VehicleType** är given. Du skall skriva en klass **Vehicle**, som använder enumerationen och också implementerar interfacet **IVehicle**. Du skall också skriva ett par konkreta klasser. Tänk på att klasserna inte är stora, dvs du skall skriva så mycket kod som frågorna nedan kräver.

Glöm inte att deklarerar de variabler du använder, strukturerar metoderna på rätt sätt och använda rätt syntax. Figuren till höger visar de källfiler som skall kompletteras i uppgiften.

<pre>public enum VehicleType { Car, // Bil Bike, // Cykel Boat, // Båt Airplane // Flygplan }</pre>	
---	---

9a: interface

Komplettera interfacet enligt nedan (punkt 1 till 3)

```
public interface IVehicle {  
    // Ett fordon (Vehicle) måste ha en typ enligt enum-typen VehicleType  
    // 1. Komplettera med en getter- och en setter-metod  
    // som klasserna (Vehicle) skall implementera.  
  
    // Ett fordon skall räkna ut och returnera power-to-weight baserat på  
    // effekt och vikt. powerToWeight = power / weight  
    // 2. Komplettera med en metod som returnerar power-to-weight som  
    // ett double tal.  
  
    // 3. Skriv en metod getVehicleInfo som skall returnera en String.  
}
```

9b: abstrakt klass och abstrakt metod

Skriv en klass **Vehicle** som implementerar interfacet i 9a. Klassen skall innehålla all kod som implementation av interfacet kräver, med undantag av metoden **getVehicleInfo**, som skall definieras som en abstrakt metod i klassen. Du får gärna deklarerar variabler och skriva flera metoder ifall du behöver dem.

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

Definiera **getVehicleInfo** i klassen **Vehicle**, som skall returnera en string. I denna del, skriv bara de ändringar som du behöver göra i klassen **Vehicle**.

9c: konkreta klasser

Skriv två klasser, **Car** och **Bike**, som subklasser till **Vehicle**. Deklarera en valfri instansvariabel i varje klass. Ifall du inte kommer på något bra attribut, använd följande:

- Klassen **Car**: numOfSeats(int)
- Klassen **Bike**: numOfGears(int)

Båda klasser skall överskugga (override) den abstrakta metoden **getVehicleInfo**. Metoden kan returnera en test sträng som: "Fast Car" respektive "Nice Bike".

Klasserna behöver inte vara stora utan de skall vara så pass komplett så de tillsammans med **Vehicle** och **IVehicle** kan kompileras. Små syntax-fel är tillåtna.

9d: dynamisk bindning

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

Car:

- num of seats = 5 (eller det attribut du har enligt 9c)
- power = 200
- weight = 2000

Bike:

- num of gears = 21
- power = 5
- weight = 5

```
public class Main {  
  
    public static void main(String[] args) {  
        createVehicle(VehicleType.Car);  
        createVehicle(VehicleType.Bike);  
    }  
  
    public static void createVehicle(VehicleType vehicleType)  
    {  
        //Komplettera  
        //Krav:  
        //  dynamisk bindning  
        //  metoderna getVehicleInfo och calculatePowerToWeight()  
        //  måste anropas.  
    }  
}
```

Utdata från main-metoden:

```
Fast Car  
Vehicle has a power-to-weight ratio of 0.1 hp/kg  
Nice Bike  
Vehicle has a power-to-weight ratio of 1.0 hp/kg
```

Lösningsförslag:

```
public enum VehicleType {
    Car,          // Bil
    Bike,         // Cykel
    Boat,         // Båt
    Airplane      // Flygplan
}

public interface IVehicle {
    void setVehicleType(VehicleType vType);
    VehicleType getVehicleType();
    String getVehicleInfo();
    double calculatePowerToWeight();
}

public abstract class Vehicle implements IVehicle{

    private VehicleType vehicleType;
    private double power;
    private double weight;

    public double getPower()
    {
        return power;
    }

    public void setPower(double power)
    {
        this.power=power;
    }

    public double getWeight() {
        return weight;
    }

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

    @Override
    public void setVehicleType(VehicleType vType) {
        vehicleType = vType;
    }

    @Override
    public VehicleType getVehicleType() {
        return vehicleType;
    }

    @Override
    public abstract String getVehicleInfo();

    @Override
    public double calculatePowerToWeight() {
        return power / weight;
    }

}

public class Car extends Vehicle{

    private int numOfSeats;
```

```
@Override
public String getVehicleInfo() {
    return "Fast Car";
}

public int getNumOfSeats() {
    return numOfSeats;
}

public void setNumOfSeats(int numOfSeats) {
    this.numOfSeats = numOfSeats;
}
}

public class Bike extends Vehicle {

    private int numOfGears;

    @Override
    public String getVehicleInfo() {
        return "Nice Bike";
    }

    public int getNumOfGears() {
        return numOfGears;
    }

    public void setNumOfGears(int numOfGears) {
        this.numOfGears = numOfGears;
    }
}

public class Main {

    public static void main(String[] args) {
        createVehicle(VehicleType.Car);
        createVehicle(VehicleType.Bike);
    }

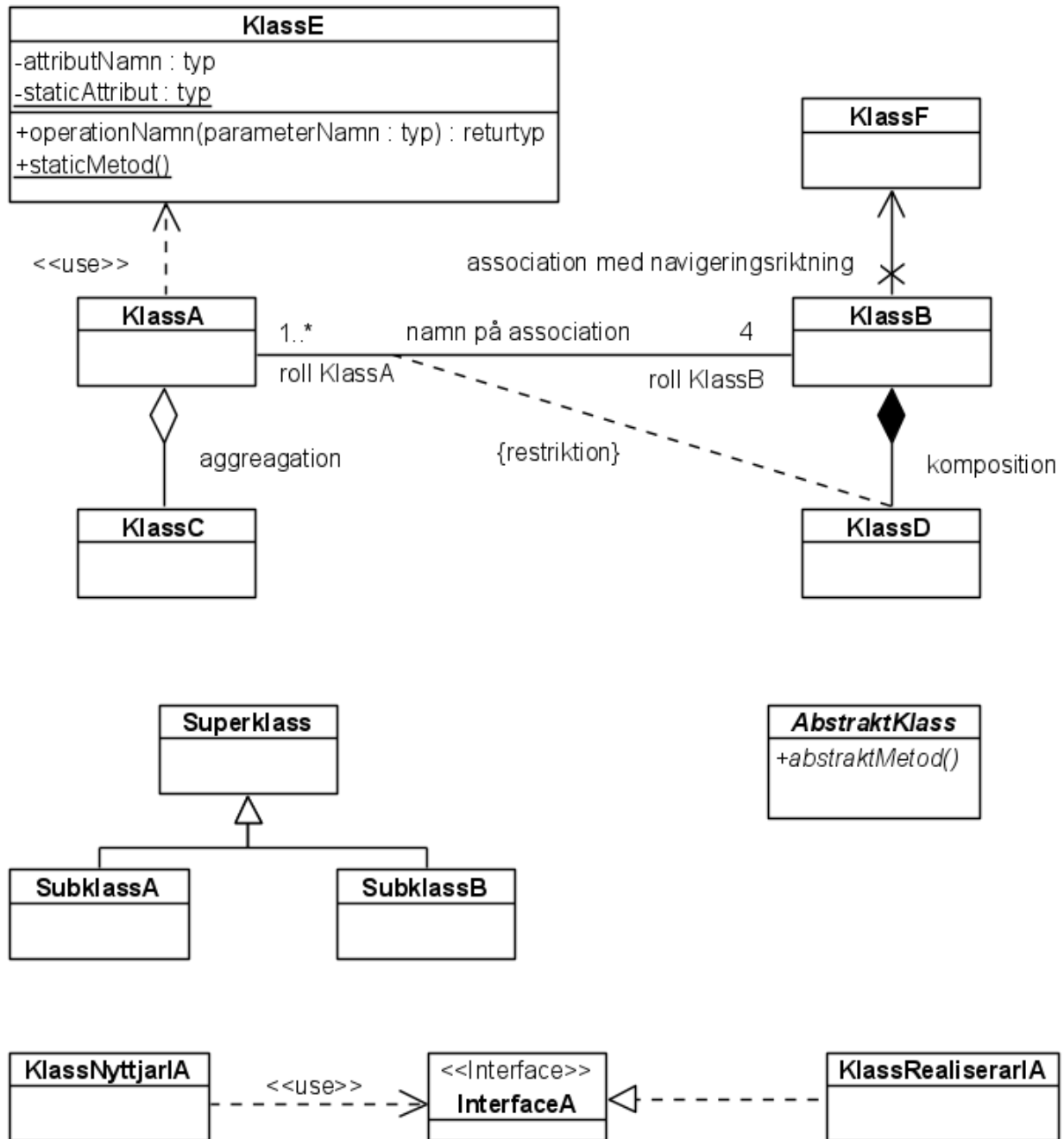
    public static void createVehicle(VehicleType vehicleType)
    {
        Vehicle vehicle = null;
        switch (vehicleType)
        {
            case Car:
                vehicle = new Car();
                ((Car)vehicle).setNumOfSeats(5);
                vehicle.setPower(200);
                vehicle.setWeight(2000);
                break;
            case Bike:
                vehicle = new Bike();
                ((Bike)vehicle).setNumOfGears(21);
                vehicle.setPower(5);
                vehicle.setWeight(5);
                break;
            default:
                System.out.println("No such vehicle implemented");
                break;
        }

        if (vehicle != null)
        {
```



```
        vehicle.setVehicleType(vehicleType);  
        System.out.println(vehicle.getVehicleInfo());  
        System.out.println("Vehicle has a power-to-weight ratio of " +  
            vehicle.calculatePowerToWeight() + " hp/kg");  
    }  
}
```

Bilaga 1 UML-notation klassdiagram



Bilaga 2 UML-notation sekvensdiagram

