

Tentamen på

Kurs:

DA339A, Objektorienterad programmering

Provkod: 2002 Tentamen 2, 2,5 hp
220816 kl 08.15 – 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 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.

Lärare besöker tentan ungefär vid följande tider: 9.00 och 10.00

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 sekvensdiagram är ett dynamiskt diagram.
B	Vid generalisering fyller multiplicitet inget syfte i ett klassdiagram.
C	Objekt kan dela identitet och klass med varandra.
D	Om klasserna A och B är associerade med en komposition, där A är helheten, så gäller att om ett objekt av A raderas så ska objekt av klassen B som har en länk till objektet av A också raderas.
E	En klass B kan inte vara superklass till en annan klass C och subklass till en tredje klass A samtidigt.
F	Om två klasser har flera associationer mellan sig så får dessa inte vara av samma typ.
G	Ett interface kan implementera (implements) ett annat interface
H	När en array eller ett annat objekt skickas som argument till en metod, tillämpas en mekanism som heter pass-by-value , vilket betyder att objektets data skickas till metoden.
I	Överskuggning (overriding) innebär att en metod i en subklass ska användas i stället för den metod som subklassen har arvt från en överordnad klass. En överskuggande metod har samma namn och samma argument som metoden som den överskuggar.
J	När typen av objekt är bestämt vid kompilering, sker en statisk bindning.
K	Vid hantering av undantag, kan det finnas flera try och finally block men bara ett (1) catch block.
L	Alla metoder i ett interface är implicit public och abstract.
M	Överlagring (overloading) är ett annat namn för överskuggning (overriding), dvs.båda används vid arv.
N	Instansvariabler är inte tillåtna i ett interface men det går bra att använda konstanter. Konstanterna är public , static , final .
O	Antag att metoden <code>public void setNumbers(int[] numbers) { // kod }</code> finns i en klass. Metoden kan anropas från en annan metod så här: <code>// antag att int[] numbers är deklarerad och initierad setNumbers(numbers);</code>

P	Om vi inte definierar metoden toString finns ändå metoden tillgänglig, eftersom den ärvt av klassen Object som är superklass till alla klasser.
Q	Nyckelordet throws används i en metods signatur för att tala om att ett undantag kan inträffa i metoden.
R	Ett interface innehåller vanligtvis implementation av metoder.
S	Nyckelordet extends används för att implementera ett interface och nyckelordet implements används för att ärva en superklass.
T	En superklass konstruktor anropas och exekveras alltid först och sen exekveras subklassens egen konstruktor.

Svar

Korrekta/sanna: A, B, D, I, J, L, N, O, P, Q, T

Felaktiga/falska: C, E, F, G, H, K, M, R, S

Uppgift 2 2p

Vi kan dela in klasser i de olika typerna/stereotyperna boundary, control och entity. Förklara kortfattat vad som skiljer de olika typerna åt och vad som utmärker respektive typ av klass.

Svar: Se slides 6-8 i F16

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.

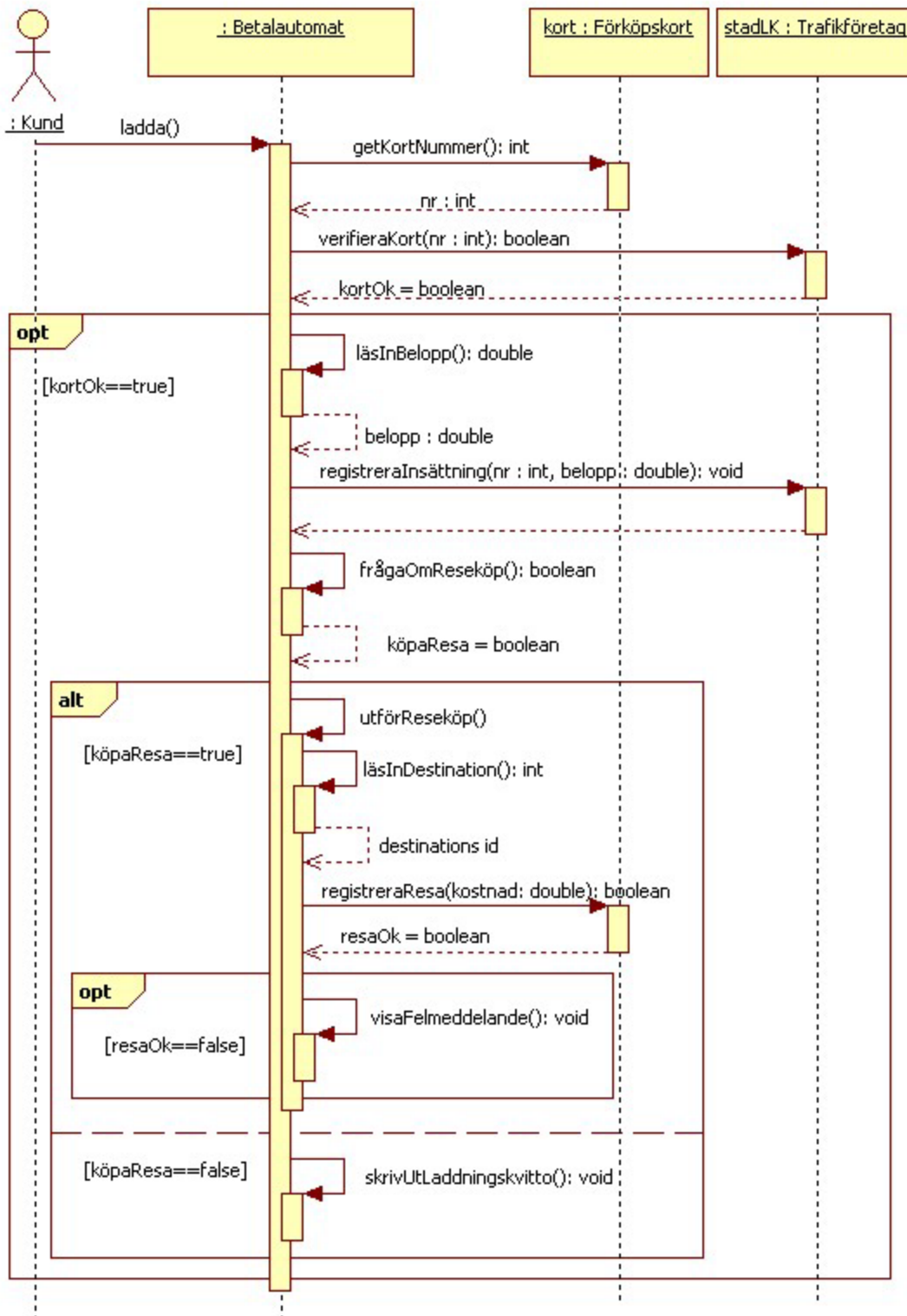
Kommentar runt svar:

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

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 Trafikföretag {  
    public boolean verifieraKort(int nr) {...}  
    public void registreraInsättning(int nr, boolean belopp) {...}  
}
```

```
class Förköpskort {  
    public int getKortNummer() {...}  
    public boolean registreraResa(double kostnad) {...}  
}
```

```
class Betalautomat {  
    public void ladda() {  
        nr = kort.getKortNummer();  
        kortOk = stadLK.verifieraKort(nr);  
        if(kortOk){  
            belopp = läsInBelopp();  
            stadLK.registreraInsättning(nr, belopp);  
            köpaResa = frågaOmReseköp();  
            if(köpaResa){  
                utförReseköp();  
            }  
            else{  
                skrivUtLaddningskvitto();  
            }  
        }  
    }  
}
```

```
... void utförReseköp() {  
    ... = läsInDestination();  
    resaOk = kort.registreraResa(...);  
    if(!resaOk){  
        visaFelmeddelande();  
    }  
}  
... double läsInBelopp() {...}  
... boolean frågaOmReseköp() {...}  
... int läsInDestination() {...}  
... void visaFelmeddelande() {...}  
... void skrivUtLaddningskvitto() {...}  
}
```

Uppgift 5 4p

5a 1p

Klassen **Literature** innehåller de instansvariabler som visas i koden nedan. Du skall komplettera med en konstruktor som initierar instansvariablerna med hjälp av argumenten i konstruktorn. Obs! "hårdkodning" av värden i instansvariablerna är ej tillåtet.

```
public class Literature {  
    int numPages;  
    String title;  
    String author;  
  
    // Komplettera med en konstruktor här  
}
```

5b 1p

Skriv en överlagrad (override) **toString**-metod som returnerar en sträng med information om objektets data.

5c 2p

Skriv en metod som kontrollerar att värdet på *numPages* är större än noll (0) och korrigera värdet till ett (1) om så inte är fallet. Lägg till/ändra i konstruktorn du skrev i 5a för användning av metoden.

Lösningsförslag:

```
public class Literature {  
    int numPages;  
    String title;  
    String author;  
  
    Literature(String title, String author, int numPages)  
    {  
        this.title=title;  
        this.author=author;  
        this.numPages=checkNumPages(numPages);  
    }  
  
    private int checkNumPages(int num)  
    {  
        if (num > 0)  
            return num;  
        else  
            return 1;  
    }  
  
    @Override  
    public String toString() {  
        return title + ": antal sidor = " + numPages + ", författare = " + author;  
    }  
}
```

Uppgift 6 4p

6a 1p

Skapa en klass Book (Bok). Denna klass skall ha en instansvariabel isbn (String) och skall ärvä från klassen Literature i uppgift 5.

6b 2p

Skapa en konstruktor för klassen Book med nödvändiga parametrar för att initiera alla instansvariabler.

6c 1p

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

```
public class Main {  
    public static void main(String[] args) {  
        Literature b = new Book("Röda rummet", "August Strindberg", 308, "9789188680501");  
        System.out.println(b);  
    }  
}
```

output:

Röda rummet: antal sidor = 308, författare = August Strindberg, isbn = 9789188680501

Lösningsförslag:

```
public class Book extends Literature{  
    String isbn;  
    Book(String title, String author, int numPages, String isbn) {  
        super(title, author, numPages);  
        this.isbn = isbn;  
    }  
  
    @Override  
    public String toString() {  
        return super.toString() + ", isbn = " + isbn;  
    }  
}
```

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.

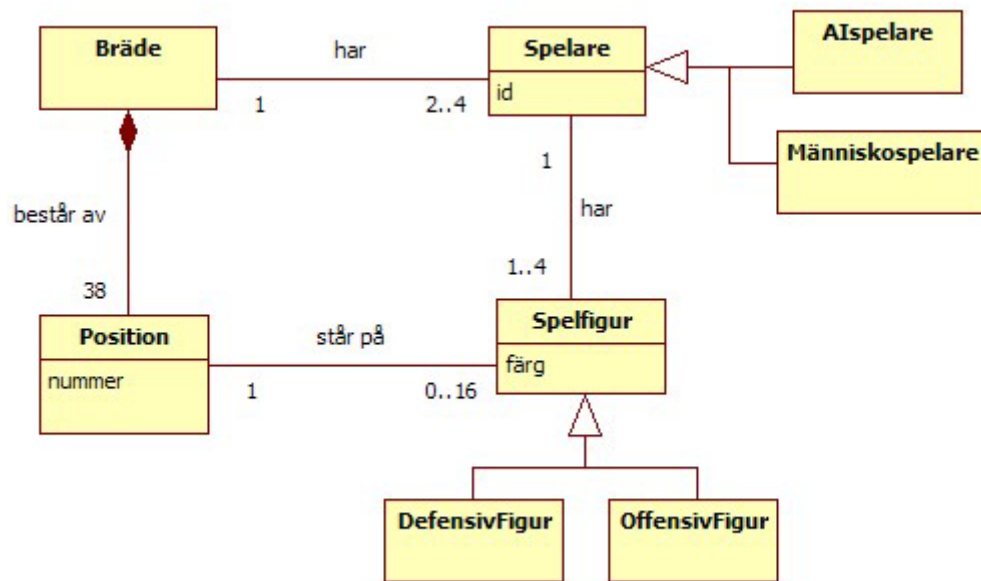
Det är inte nödvändigt att visa attribut eller operationer i klassdiagrammet. Det räcker att skriva ut klassnamn och namn på associationer och multiplicitet. Korrekt UML-notation ska användas (se Bilaga 1).

Systembeskrivning

Ett spel spelas på ett spelbräde med 38 möjliga positioner för spelfigurerna. Varje position identifieras av ett nummer. Man kan spela spelet mot en AI-spelare eller mot en annan spelare som styrs av en människa. Minst två spelare krävs varav minst en är en människa. Som mest kan fyra spelare spela spelet. En spelare kan bara spela på en spelplan åt gången. Spelare skiljs åt genom ett unikt id.

En spelfigur som är i spel står alltid på någon position. Spelfigurer kan ha samma position. Det finns två olika sorters spelfigurer - offensiva spelfigurer och defensiva spelfigurer. Dessa har mycket gemensamt, bland annat att de har en färg och styrs av en spelare men reglerna för hur de får röra sig på spelplanen är helt olika för de två typerna.

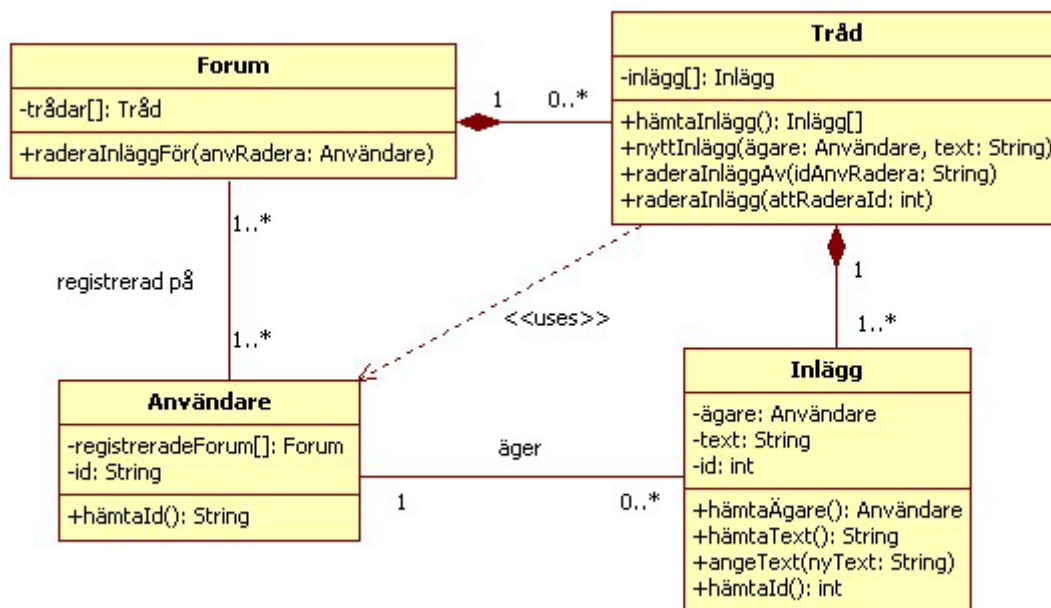
Varje spelare har 1 till 4 egna spelfigurer att flytta. Spelaren startar med 4 spelfigurer och efterhand som spelet pågår så tas spelfigurer ur spel. Om en spelares sista spelfigur slås ut av en annan spelare så är den spelare som förlorade sin sista spelfigur ute ut spelet. Vinnare är den spelare som är ensam kvar på spelbrädet.

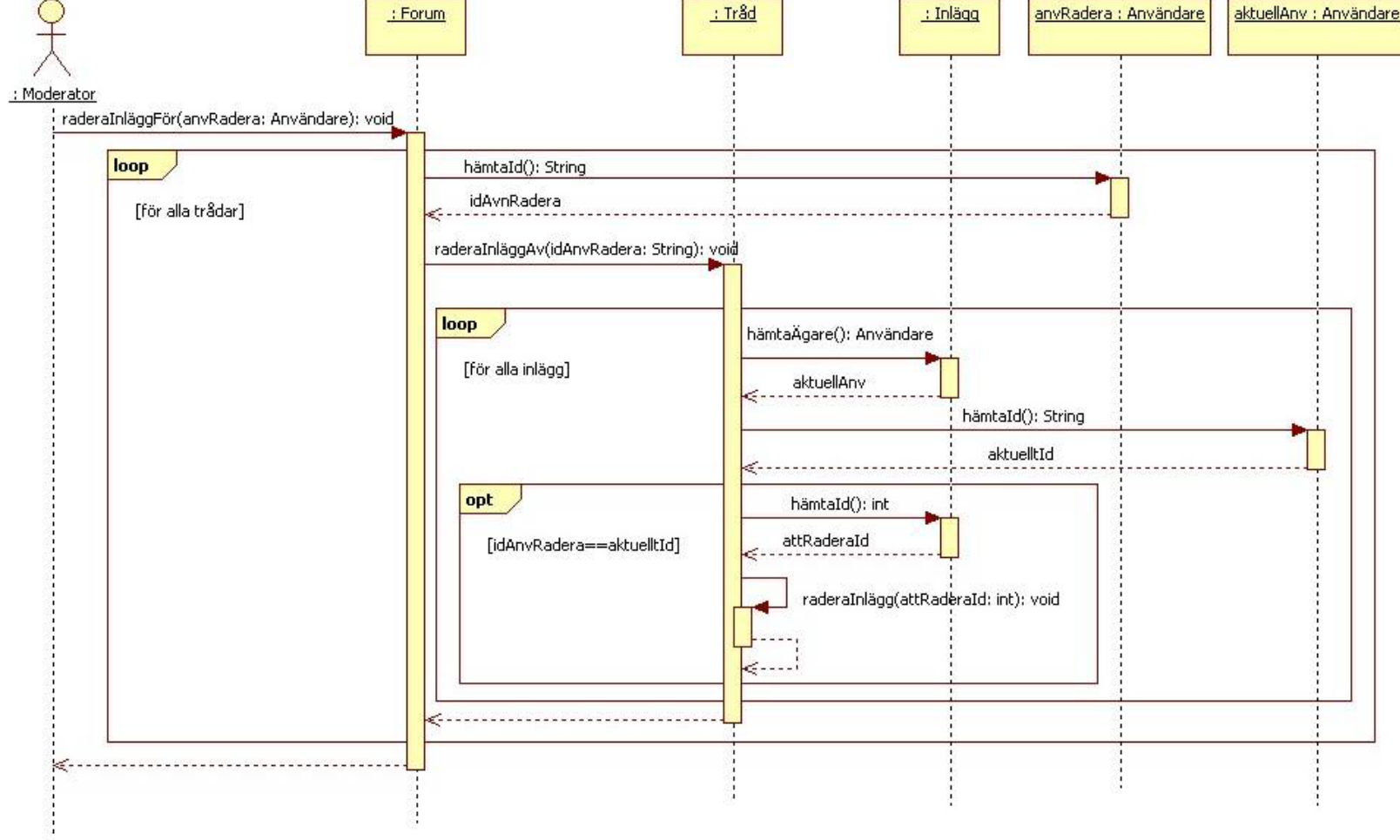


Uppgift 8 7p

Klassdiagrammet nedan beskriver ett system som hanterar ett diskussionsforum. Givet detta rita ett sekvensdiagram som visar vad som sker när en moderator får i uppdrag att radera alla inlägg gjorda av en viss given användare på ett forum.

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 den mänskliga moderatoren och som interagerar med klassen Forum genom att initiera metoden `raderaInläggFör(anvRadera: Användare)` via något gränssnitt.





Uppgifter för betyg VG

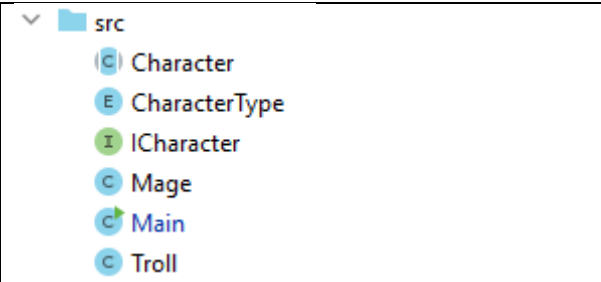
Uppgift 9

Du skall komplettera ett litet program för slumpvis generering av spelkaraktärer.

Enumerationen **CharacterType** är given. Du skall skriva en klass **Character**, som använder enumerationen och också implementerar interfacet **ICharacter**. 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.

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, strukturerar metoderna på rätt sätt och använder rätt syntax. Till höger i tabellen nedan visas de källfiler som skall kompletteras i uppgiften.

<pre>public enum CharacterType { TROLL, ELF, HUMAN, MAGE }</pre>	
--	---

Uppgift 9a: interface

Komplettera interfacet **ICharacter** enligt nedan, punkt 1 till 3.

```
public interface ICharacter {  
    // En karaktär (Character) måste ha en typ enligt enum-typen CharacterType  
    // 1. Komplettera med en getter- och en settermetod,  
    // som klasserna (Character) måste implementera.  
  
    // En karaktär behöver ett namn.  
    // 2. Komplettera med en getter- och en settermetod, som  
    // klasserna (Character) måste implementera.  
    // Settermetoden skall ta förnamn och efternamn som argument.  
    // För att i de implementerande klasserna lagras ned i en (1) String.  
  
    // 3. Skriv en metod getInfo som skall returnera en String  
}
```

Uppgift 9b: abstrakt klass och abstrakt metod

Skriv en klass **Character**, som implementerar interfacet i 9a. Klassen skall innehålla all kod som implementationen av interfacet kräver, med undantag av metoden **getInfo**, som skall definieras som en abstrakt metod i klassen. Du får gärna deklarera variabler och skriva flera metoder ifall du behöver dem. En egenskap är dock nödvändig: Karaktärens hela namn (för- och efternamn) som skall lagras ned i en (1) String variabel. Deklarera en variabel för detta.

Definiera **getInfo** i klassen **Character**, som skall returnera en String. I denna del, skriv bara de ändringar som du behöver göra i klassen **Character**.

Uppgift 9c: abstrakt klass och abstrakt metod

Skriv två klasser **Troll** och **Mage**, som subklasser till **Character**. Deklarera en instansvariabel i varje klass enligt följande:

- Klassen **Troll**: strength(int), som innehåller ett värde på trollets styrka.
- Klassen **Mage**: mana(int), som innehåller ett värde på magikerns "magi".

Båda klasser skall överskugga (override) den abstrakta metoden **getInfo**. Metoden kan returnera en test sträng som: "Jag är en magiker" respektive "Jag är ett troll", alternativt skriva ut all information om karaktären. Se testutskrift i slutet på uppgiften (efter 9d).

Klasserna behöver inte vara stora utan de skall vara så pass kompletta så de tillsammans med **Character** och **ICharacter** kan kompileras. Små syntax-fel är tillåtna.

Uppgift 9d: dynamisk bindning

Skriv färdigt metoden **createCharacter**, se nedan, så main-metodens anrop fungerar i att randomisera 10 karaktärer enligt hur koden är skriven. Se också testutskrift.

```
import java.security.SecureRandom;
public class Main {
    // https://www.fantasynamengenerators.com/
    static String[] firstNames =
        {"Falco", "Badegisel", "Dodo", "Thietmar", "Adalolf"};
    static String[] lastNames =
        {"Bottomhill", "Chubb", "Longriver", "Took-Took", "Finnagund"};

    public static void main(String[] args) {
        SecureRandom rnd = new SecureRandom();
        for (int i=0; i<10; i++){
            int firstIndex = rnd.nextInt(firstNames.length);
            String firstName = firstNames[firstIndex];
            int lastIndex = rnd.nextInt(lastNames.length);
            String lastName = lastNames[lastIndex];
            int characterIndex = rnd.nextInt(CharacterType.values().length);
            CharacterType cType = CharacterType.values()[characterIndex];

            createCharacter(firstName, lastName, cType);
        }
    }

    private static void createCharacter(String firstName, String lastName, CharacterType cType) {
        // Komplettera
        // Krav:
        //     dynamisk bindning
        //     metoderna getInfo och setName måste anropas här.
    }
}
```

Testutskrift från main-metoden, som kan se olika ut beroende på randomiseringen, exakt utskrift enligt nedan är ej ett krav:

```
** ELF is not implemented **
A mage with name Adalolf Bottomhill has a mana of 200
** HUMAN is not implemented **
** HUMAN is not implemented **
A troll with name Thietmar Took-Took has a strength of 50
A troll with name Adalolf Bottomhill has a strength of 50
** HUMAN is not implemented **
** ELF is not implemented **
** ELF is not implemented **
** ELF is not implemented **
```

Lösningsförslag:

```
public enum CharacterType {
    TROLL,
    ELF,
    HUMAN,
    MAGE
}

import java.security.SecureRandom;
public class Main {
    // https://www.fantasynamengenerators.com/
    static String[] firstNames =
        {"Falco", "Badegisel", "Dodo", "Thietmar", "Adalolf"};
    static String[] lastNames =
        {"Bottomhill", "Chubb", "Longriver", "Took-Took", "Finnagund"};

    public static void main(String[] args) {
        SecureRandom rnd = new SecureRandom();
        for (int i=0; i<10; i++){
            int firstIndex = rnd.nextInt(firstNames.length);
            String firstName = firstNames[firstIndex];
```

```
        int lastIndex = rnd.nextInt(lastNames.length);
        String lastName = lastNames[lastIndex];
        int characterIndex = rnd.nextInt(CharacterType.values().length);
        CharacterType cType = CharacterType.values()[characterIndex];

        createCharacter(firstName, lastName, cType);
    }
}

private static void createCharacter(String firstName, String lastName, CharacterType
cType) {
    Character character = null;
    switch (cType)
    {
        case TROLL:
            character = new Troll();
            ((Troll)character).setStrength(50);
            break;
        case MAGE:
            character = new Mage();
            ((Mage)character).setMana(200);
            break;
        default:
            System.out.println(" ** " + cType + " is not implemented **");
    }

    if (character != null)
    {
        character.setName(firstName, lastName);
        character.setCharacterType(cType);
        System.out.println(character.getInfo());
    }
}

}

public interface ICharacter {
    void setName(String first, String last);
    String getName();
    CharacterType getCharacterType();
    void setCharacterType(CharacterType cType);
    String getInfo();
}

public abstract class Character implements ICharacter {
    private String name;
    private int hp;
    private CharacterType cType;

    @Override
    public void setName(String first, String last) {
        this.name = first + " " + last;
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public CharacterType getCharacterType() {
        return cType;
    }

    @Override
    public void setCharacterType(CharacterType cType) {
        this.cType = cType;
    }
}
```

```
@Override
    public abstract String getInfo() ;
}

import java.util.Locale;
public class Troll extends Character {

    private int strength=0;

    public double getStrength()
    {
        return strength;
    }

    public void setStrength(int strength)
    {
        this.strength = strength;
    }

    @Override
    public String getInfo() {
        return "A " + getCharacterType().toString().toLowerCase(Locale.ROOT) + " with name " +
            getName() + " has a strength of " + strength;
    }
}

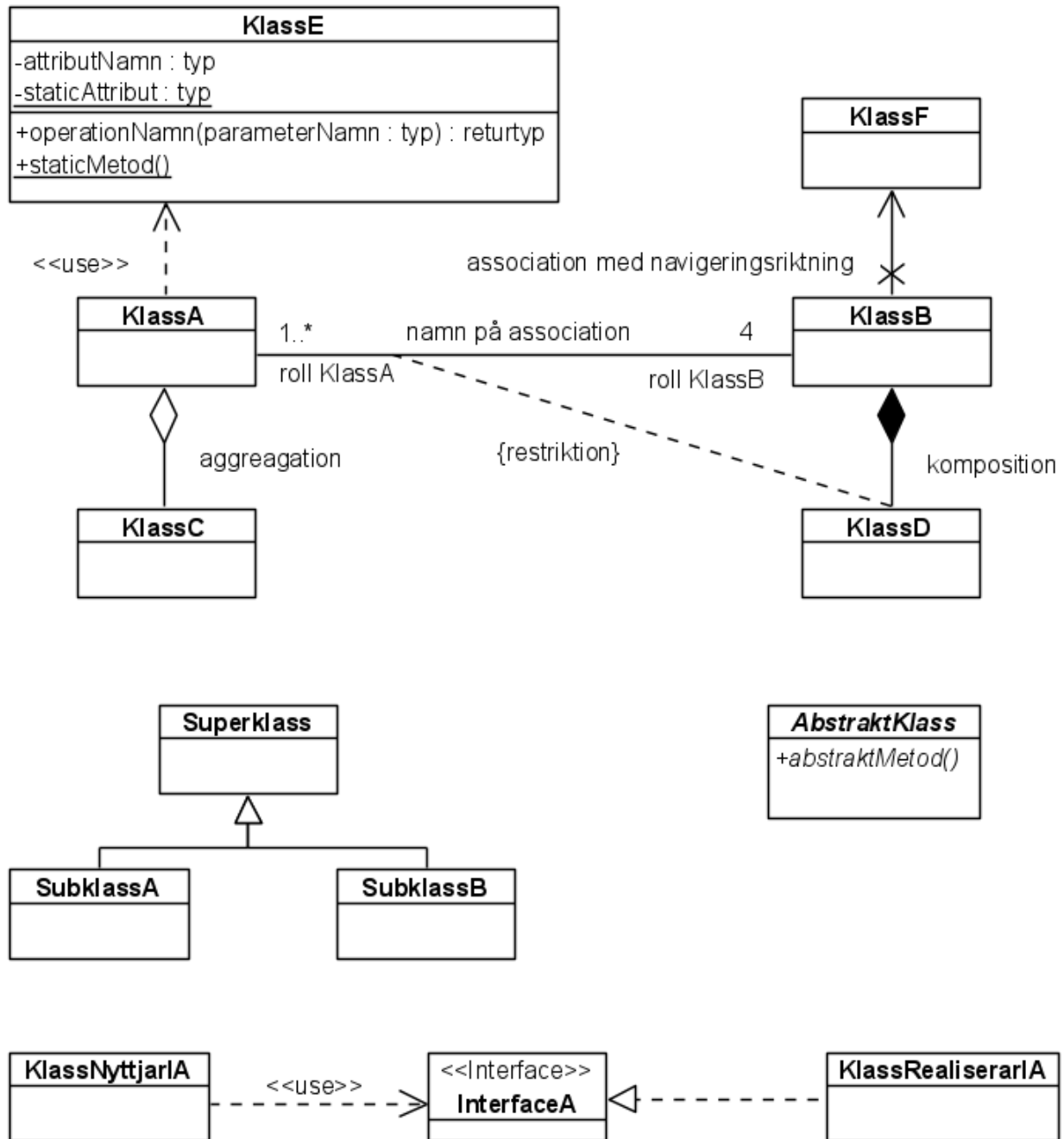
import java.util.Locale;
public class Mage extends Character{
    private int mana = 0;

    public int getMana(){
        return mana;
    }

    public void setMana(int mana){
        this.mana = mana;
    }

    @Override
    public String getInfo() {
        return "A " + getCharacterType().toString().toLowerCase(Locale.ROOT) + " with name " +
            getName() + " has a mana of " + mana;
    }
}
```


Bilaga 1 UML-notation klassdiagram



Bilaga 2 UML-notation sekvensdiagram

