INF1010 2014 — Obligatorisk oppgave 1

Versjon 1.1 Rettet en feil i metoden skrivUtAltOmMeg(). Versjonsnummeret økes når flere eksempler og spørsmål føyes til nederst. Selve oppgaven endres ikke med mindre det finnes direkte feil.

Du skal skrive et javaprogram med fire personobjekter. Disse fire objektene representerer inf1010-studentene Ask, Dana, Tom og deg selv (bruk gjerne ditt eget navn). Objektene skal alle være objekter/instanser av den samme klassen, klassen Person:

```
class Person {

    private String navn;
    private Person [] kjenner;
    private Person [] likerikke; // = uvenner
    // venner blir da alle personer som pekes på av kjenner
    // unntatt personen(e) som pekes på av likerikke
    private Person forelsketi;
    private Person sammenmed;

    Person(String n, int lengde) {
      ...
    }

    public String hentNavn() { return navn; }

    // Andre metoder som er tilgjengelige utenfra, se nedenfor
}
```

Et objekt av denne klassen har mange egenskaper: variabler og metoder. De fleste variablene er *pekere*. En peker er en variabel som kan peke på et objekt. (Vi tegner denne som en firkant som inneholder en pil som peker på et objekt). sammenmed er eksempel på en slik peker. Videre har vi to arrayer av pekervariable. En av dem heter kjenner. Dette er en pekervariabel som peker på en *array av pekere*. Hver indeks av en pekerarray peker til null eller til et objekt av klassen Person.

Tegn først et objekt av klassen Person. navn skal peke på et String-objekt med verdien ditt eget navn. Ta med alle egenskaper som er med programskissen ovenfor. La arrayene ha lengde tre, og la alle pekere til Person peke til null.

Noen av variablene er pekere som kan peke på andre personobjekter slik at vi kan lage et forenklet nettverk av personer.

Når et personobjekt opprettes, sender vi med to parametre til konstruktøren:

- en tekststreng som er navnet til personen objektet representerer
- et heltall (int lengde) som bestemmer lengden til arrayene.

Du skal programmere ferdig klassen ved å fullføre konstruktøren og metodene som skal være tilgjengelig utenfra. Noen få metoder er gitt ferdig programmert. Du må gjerne lage flere hvis du ønsker. Alle metoder som skal være tilgjengelig utenfra skal deklareres public. Trenger du andre variable

i klassen, må du gjerne legge dem til. Disse skal også være deklarert som private.

Nedenfor tenker vi oss virkningen av metodene definert for en konkret person («inne i» personobjektet), nedenfor **i objektet med navnet «Dana»**. Parameteren p peker også på et personobjekt. Klassen Person skal ha følgende metoder her forklart med signaturer og virkning:

```
public boolean erKjentMed(Person p)
// Sann hvis Dana kjenner personen p peker på.

public void blirKjentMed(Person p)
// Dana blir kjent med p, bortsett fra hvis p peker
// på Dana (Dana kan ikke være kjent med seg selv).

public void blirForelsketI(Person p)
// Dana blir forelsket i p, bortsett fra hvis p peker på Dana

public void blirUvennMed(Person p)
// Dana blir uvenn med p, bortsett fra hvis p peker på Dana

public boolean erVennMed(Person p)
// returnerer sann hvis Dana kjenner p og ikke er uvenner med p

public void blirVennMed(Person p)
// samme virkning som blirKjentMed(p), men hvis Dana ikke
// liker p dvs. (likerikke[i] == p) for en gitt i
// blir likerikke[i] satt til null.

public void skrivUtVenner()
// skriver ut navnet på dem Dana kjenner, unntatt dem hun ikke liker.

public ... hentBestevenn() // returtypen skal du bestemme
// returnerer en peker til Danas bestevenn.
// En persons bestevenn er for enkelhets skyld definert til å være
// det objektet som pekes på av kjennerarrayens indeks 0.

public ... hentVenner()
// returnerer en array som peker på alle Danas kjente
// Arrayen skal være akkurat så lang at lengden er lik antallet venner,
// og rekkefølgen skal være den samme som i kjenner-arrayen.

public ... antVenner()
// returnerer hvor mange venner Dana har

// Disse metodene trenger du ikke lage selv, men gjør det gjerne for
// øvelsens skyld:

public void skrivUtKjenninger() {
    for (Person p: kjenner)
        if ( p!=null) System.out.print(p.hentNavn() + "␣");
    System.out.println("");
}
```

```
public void skrivUtLikerIkke() {
    for (Person p: likerikke)
        if ( p!=null) System.out.print(p.hentNavn() + " ");
    System.out.println("");
}

public void skrivUtAltOmMeg() {
    System.out.print(navn + " kjenner: ");
    skrivUtKjenninger();
    if (forelsketi != null)
        System.out.println(navn +
            " er forelsket i " + forelsketi.hentNavn());
    System.out.print(navn + " liker ikke ");
    skrivUtLikerIkke();
}
```

Du skal også skrive en klasse som oppretter datastrukturen. I denne klassen lages de fire objektene, pekt på av variablene jeg, ask, dana og tom. Sørg deretter for at følgende *tilstandspåstander* holder i datastrukturen, ved å kalle på metodene du laget i personklassen:

Tilstandspåstander

- Jeg kjenner de tre andre og er ikke forelsket i noen og liker alle.
- Ask kjenner Dana, meg og Tom. Han er forelsket i meg. Han liker ikke Dana og Tom.
- Dana kjenner Ask, Tom og meg. Hun er forelsket i Tom og liker ikke deg.
- Tom kjenner meg, Ask, Dana. Han liker verken Ask eller meg og er forelsket i Dana.

Kall på skrivUtAltOmMeg i alle fire objekter i rekkefølgen jeg, ask, dana og tom, skal da gi utskriften:

```
Jeg kjenner: Ask Dana Tom
Ask kjenner: meg Tom Dana
Ask er forelsket i meg
Ask liker ikke Dana Tom
Dana kjenner: Ask Tom meg
Dana er forelsket i Tom
Dana liker ikke meg
Tom kjenner: meg Ask Dana
Tom er forelsket i Dana
Tom liker ikke Ask meg
```

Du trenger ikke bruke alle metodene i Person for å få til denne tilstanden. Du kan erstatte Jeg og meg med ditt eget fornavn for å gjøre det litt enklere.

Vanligere enn å beskrive én spesiell tilstand i datastrukturen, er det å oppgi generelle regler som gjelder for det datastrukturen beskriver. Slike regler

kan formuleres som *invariante* (uforanderlige) tilstandspåstander om datastrukturen. Eksempler på slike påstander er f.eks. at en person ikke kan kjenne en annen person mer enn en gang, eller at man ikke kan være forelsket i eller mislike seg selv osv. Ofte er det regler som gjør at datastrukturen stemmer overens med virkeligheten. Tilstandspåstander er til stor hjelp når vi skal programmere og vi vil komme tilbake til dem i senere oppgaver. **Du skal ikke ta hensyn til slike regler i denne oppgaven.**

Oppsummering av det som skal leveres

- en tegning av datastrukturen for et personobjekt med ditt navn.
- klassen Person med konstruktør og alle egenskapene (variable og metoder).
- en testklasse som oppretter datastrukturen med de 4 personobjektene slik at datastrukturen tilfredsstiller tilstandspåstandene. Testklassen skal til slutt gi utskriften ovenfor.
- en klasse med main-metoden som starter det hele ved å opprette et objekt av testklassen.
- Tegningen skannes/fotograferes og leveres som en A4-side i pdf- eller jpg-format. De tre klassene samles i en java-fil og leveres som et kjørbart program. Filen skal hete Oblig1.java. Innleveringen vil dermed bestå av 2 filer. Flere detaljer, se generell info om levering av obliger på semestersida.

Flere tester (frivillig)

Skriv gjerne flere tester. Her er et eksempel:
Hvilket navn skal skrives ut med følgende programkode (i testklasen):

```
Person [] danasVenner = dana.hentVenner();
System.out.println(danaKjenner[1].hentBestevenn().hentNavn());
```

Spørsmål til oppgaven som kan være interessante for flere

I fjor sendte en student følgende spørsmål: «Skal relasjonen mellom instansene av Person-klassen anses til å være symetrisk? Kan/skal vi gjøre det slik at Ego kjenner Dana impliserer at Dana også kjenner Ego? Og Ego er forelsket i Dana impliserer at Dana også er forelsket i Ego. Fordi, hvis ikke så vil det at Dana er uvenn med Ego ikke implisere at Ego også er uvenn med Dana, og da vil Ego.erVennMed(Dana) være lik true, og det vil føre til at Ego.bliVennMed(Dana) gå an, til tross for deres vennskap som egentlig ikke eksisterer. Kan vi løse oppgaven slik at relasjonene blir symetriske der de bør være, eller er det 'ikke lov'.?»

Svar: Dette henger sammen med det som kalles invariante tilstandspåstander i oppgaven. Hvilke regler (fra virkeligheten) vil vi skal reflekteres i programmet (datamodellen av relasjoner mellom 4 personer). I obligen skal du ikke ta hensyn til dette. Tilstanden i datastrukturen skal være akkurat slik som utskriften sier. At dette er unaturlig eller kunstig, gjør ikke noe.

Hvis du vil ha «symmetriske» relasjoner må du gjerne gjøre det, gjør det da i en kopi av programmet som du ikke leverer. Å formulere slike regler

er ikke lett. F.eks. vil utsagn som «Dana og Ask er venner» vanligvis innebære kjennskap begge veier, mens utsagn som «Ask er forelsket i deg» ikke nødvendigvis betyr at du er forelsket i ham eller kjenner ham. Noen av oss opplever dessverre også at noen vi trodde var en venn, viste seg å ikke være det. Å lage regler som treffer virkeligheten 100% er (nesten?) umulig.

Vi vil i senere obliger lage personnettverk som modellerer virkeligheten bedre.