DAT102 – Våren 2023

Øving 4
Utlevert 2. februar
Leveres inn sammen med Øving 5

Introduksjon

Denne oppgaven inngår som del av den andre obligatoriske øvingen. Vi henstiller at dere er 3-4 på gruppen, men aldri flere uten at dette er avklart med foreleserne. Hvis dere lager grupper på 4, kan 2 og 2 jobbe tett sammen i smågrupper med å programmere samtidig og diskutere underveis. De to gruppene blir enige om en felles innlevering slik at dere leverer inn som en gruppe på 4. Grupper som er mindre enn tre, vil få redusert tilbakemelding siden vi har begrenset rettekapasitet.

For å oppnå en best mulig faglig utvikling må du være aktiv på lab. **Det betyr at alle må kode.**

Oppgave 1

I denne oppgaven kan vi gjøre det enkelt ved å la innlesing og utskrift foregå i **`main().`** På GitHub finner du et prosjekt, kalt **`MengdeU`** der du skal fylle ut med den koden som mangler. Studer mappestrukturen. Studer interface for mengde. Det er også klienter, **`KlientBingo`** og noen versjoner av **`Ordliste`** som dere kjører når dere har fylt ut med nødvendig kode.

Kommentar: Alle klasser arver fra Object-klassen.

https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/Object.html

Object-klassen har en `equals`-metode. Den implementasjonen tester på om to objekter har like adresser i minnet til maskinen. Vi ønsker også å teste på om innholdet av to objekter er like. Da må vi overkjøre `equals` - metoden fra Object – klassen. Se tidligere notater. Når vi overkjører `equals`-metoden, er den en god regel å overjøre `hashCode`-metoden som også finnes i `Object`-klassen.

Hvis det eventuelt er andre metoder som ikke er laget ferdig og som ikke står her, må dere lage de.

a) Lag metodene equals og hashCode for både KjedetMengde og TabellMengde. Når det gjelder hashCode kan du bruke den som genereres fra Eclipse (velg «Source» fra menylinjen)

b) Lag metodene union, snitt, differens og undermengde for klassen KjedetMengde. Metoden union er laget, men den skal gjøres mer effektiv. Bruk iterator. Hverken this-mengden eller mengden gitt som parameter skal endres for snitt og differens (som for union).

c) Lag metodene union, snitt, differens og undermengde for klassen TabellMengde. Metoden union er laget, men den skal gjøres mer effektiv. Bruk iterator. Hverken this-mengden eller mengden gitt som parameter skal endres for union, snitt og differens (som for union).

d) Lag en "ufullstendig" testklasse for Mengde, der du kun lager testmetoder for union, snitt og differens. Det er frivillig å lage fullstendig testklasse (teste alle metodene på interfacet.

Det er flere måter å lage testmetoder for union, snitt og differens. **En mulighet**: I testmetoden for union lager dere en tom mengde **`begge`** som skal bli union av to mengder, **`m1(this)`** og **`m2.`** Lag mengder med ca. 5 elementer. Vi vet hva resultatet skal bli, så vi lager en "fasit"-mengde. Til slutt buker dere **`equals`**-metoden på mengden **`begge`** og "fasit"-mengden og

tilsvarende for å lage testmetoder for snitt og differens.

Lag to testmetoder for hver av de 3 mengdeoperasjonene der:

- **`m1`**og **`m2`** har felles elementer
- de ikke har felles elementer (mengdene er disjunkte).

Oppgave 2

Du kan gjerne utvide klassene **`KjedetMengde`** og **`Tabellmengde`** med **`toString()`-** metoden for bruk i denne oppgaven, men ikke ha den med i ADT'en. Metoden under er for **`KjedetMengde.`**

```
//Returnerer en streng som representerer mengden.
public String toString(){
// For klassen KjedetMengde String resultat = "";
    LinearNode<T> aktuell = start;
    while(aktuell != null){
        resultat += aktuell.getElement().toString() + "\t";
         aktuell = aktuell.getNeste();

     }
     return resultat;
}
```

Vi skal lage en del av et system for å administrere et datakontaktfirma. Systemet skal håndtere en **medlemstabell** som inneholder medlemmers interesser (mengde av hobbyer). Tabellen skal brukes for å finne medlemmer med felles interesser.

Følgende opplysninger om hvert medlem skal lagres:

- *navn* (som er forskjellig for hvert medlem).
- *hobbyer* som er referanse til et objekt av klassen **KjedetMengde** eller **TabellMengde**
- *statusIndeks* som angir indeks til partneren i medlemstabellen dersom medlemmet er "koblet", ellers er den lik -1.

Klassen *Hobby* kan defineres som:

```
public class Hobby {
     private String hobbyNavn;
     public Hobby(String hobby){
           hobbyNavn = hobby;
     }
     public String toString(){
           //… returnerer hobbynavnet med "<" foran og ">" bak som
           // String (Eksempel: <tegne, male>)
           // Merk: Kan også ha uten  «<» og «>», men forsøk med.
     }
     public boolean equals(Object hobby2){
           // eventuelt fylle ut først med "standard" kode
           // som vi ofte har med overkjøring av
           // equals-metoden generert av Eclipse
           Hobby hobbyDenAndre = (Hobby)hobby2;
           return(hobbyNavn.equals(hobbyDenAndre.getHobbyNavn()));
     }
}
```

Klassen *Medlem* kan defineres som:

```
public class Medlem {

      private String navn;
      private MengdeADT<Hobby> hobbyer;
      private int statusIndeks;

      //… Konstruktør

      //… Andre metoder

}
```

Lag et lite klientprogram for klassen Medlem med et eget `main`- program før du går videre. Lag en ekstra metode for utskrift til skjerm av alle medlemsdata for dette formålet.

OBS! Sjekk at du har en riktig konstruktør. Det er viktig.

Klassen Medlem skal i tillegg til konstruktør og nødvendige get- og set-metoder bl.a. ha følgende (du kan definere andre i tillegg):

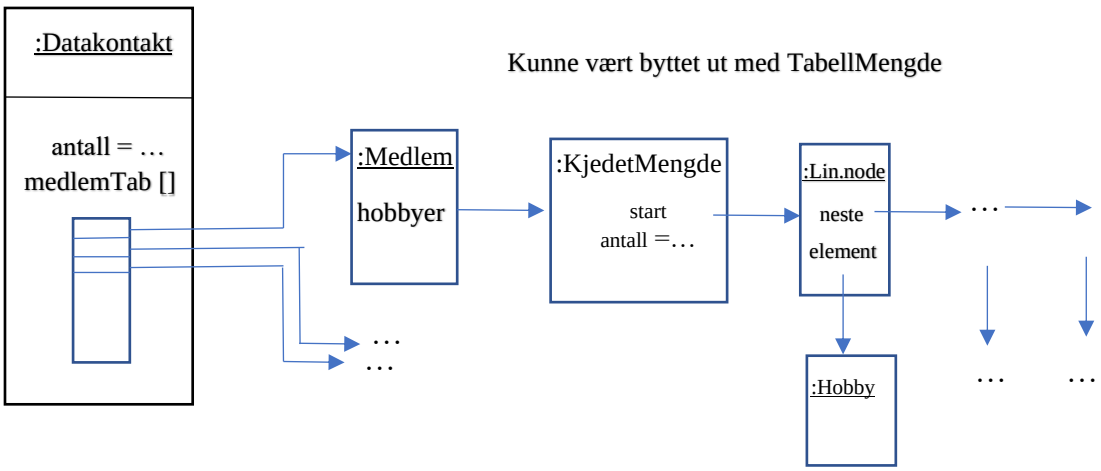
> **`passerTil(Medlem medlem2)`** som avgjør om to medlemmer passer til hverandre og altså kan danne et par. To medlemmer passer til hverandre dersom de har nøyaktig samme hobbyer (tips: like mengder).

Klassen **`Datakontakt`** **har en *medlemstabell*** som kan lagre opplysninger om medlemmer, og en variabel *antallMedlemmer* som angir antall registrerte medlemmer i tabellen til enhver tid.

Klassen **`Datakontakt`** skal bl.a. ha følgende metoder (du kan definere andre i tillegg):

- **`leggTilMedlem(Medlem person)`** legger til et nytt medlem i medlemstabellen.
- **`finnMedlemsIndeks(String medlemsnavn)`** som finner indeksen til medlemmet i medlemstabellen dersom medlemmet er registrert, ellers returneres indeks lik -1. (Noen vil kanskje si at vi denne metoden burde returnert en referanse slik at vi lettere kunne bytte ut til annen datastruktur, men det lar vi være) .
- **`finnPartnerFor(String medlemsnavn)`** som finner ut om et medlem (identifisert med medlemsnavn) passer med et annet medlem (dersom det finnes) i medlemstabellen. Dette medlemmet skal være det første som passer og ikke er "koblet". Metoden oppdaterer medlemstabellen dersom den finner en partner, og returnerer eventuell indeks til partneren i medlemstabellen (eller -1).
- **`void tilbakestillStausIndeks(String medlemsnavn)`** som oppdaterer medlemstabellen. Dersom medlemmet finnes og har en partner, så brytes koblingen dvs. begge får statusindeks –1.

Objektdiagram



Nedenfor finner du en skisse for en klasse Tekstgrensesnitt

```
public class Tekstgrensenitt{

       //Hvis du vil lage meny, kan du også legge det inn i Tekstgrensesnitt
       // leser opplysningene om et medlem fra tastatur

       public static Medlem lesMedlem(){

              …
              //f.eks. bruke Scanner.

       }


       // Skriver ut hobbylisten for et medlem
       public static void skrivHobbyListe(Medlem medlem) {
           System.out.println("Alle hobbyene ");
           System.out.println(medlem.getHobbyer().toString());
       }

      public static void skrivParListe (Datakontakt arkiv){

      /* skriver ut på skjermen en oversikt over medlemmer som er koblet
         til hverandre i medlemstabellen til enhver tid.

         Et slikt par skal kun vises én gang på utskriftlisten. Metoden
         skriver også ut antall par som er funnet.

         Eksempel på utskrift:

          PARNAVN                 HOBBYER
          Erna og Jonas           <ski, musikk, politikk>
          Eva og Adam             < epleplukking, paradishopping>
          …….
          Antall par funnet: 12

       */

   }
```

Du skal lag et enkelt klientprogram (et main - program som bruker klassen **`Datakontakt)`** slik at du får forsøkt alle metodene. Du kan gjerne lage andre metoder i klassene i tillegg.

Krav til innlevering for oppgave 2

En zippet fil med zippet prosjekt + pdf fil med utskrift av kjøring.

I tillegg: Studer det objektdiagrammet (lagt ut) som viser hvordan klassene Datakontakt, Medlemmer og hobbymengden er implementert.

Kommentar: Klassen **`Medlem`** har en indeks som attributt og har derfor gjort seg avhengig av en tabell-implementasjon (sml. tabell av medlemmer i klassen **`Datakontakt)`** som kanskje er mindre bra utforming av klassen. Egentlig kunne brukt referanse i stedet for indekser. Et medlem som er koblet refererer til et annet og tilsvarende for partner vha. objekt-referanser der null-referanse betyr ledig. MEN følg likevel opplegget i oppgave-teksten der vi bruker **tabellindekser**.

Frivillig

Bruk union, snitt og differens i oppgaven over for å finne ut hvem man passer best sammen med basert på mengden av hobbyer. For eksempel. En mulig passer-funksjon kan være:
antallHobbyerFelles – (antallEkstraHobbyerA * antallEkstraHobbyerB)/totaltAntallHobbyer

Eksempel: A: 1, 6, 7, 9, B:1, 2, 4, 7, 8 gir 2 – (2 * 3)/7 = 8/7

Frivillig

På github ligger prosjektet SorteringOgSokingU. Klassen har kun en metode som er en metode for lineært søk. I testklassen er det lagt inn testdata og kun en testmetode.
Lag en testplan og tilhørende testmetoder. Kjør testene.