DAT102 ALGORITMER OG DATASTRUKTURER SIST OPPDATERT 14.2

Vår-20 Øving 3 (del av obligatorisk 2) Utlevert 14.02. Alle øvinger er pensum.
Bli ferdig med øving 3 til fredag 28.02. Bruk ca. 1 uke pr. oppgave.
Prosjektene skal senere leveres inn sammen med øving 4. Se planen.
Grupper på 2-3 medlemmer, ikke flere enn 3. Senere blir det opprettet en obligatorisk 2 på Canvas for innlevering av øving 3 og øving 4.

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`** dvs. du skal fylle ut med kode for noen metoder.
Studer mappestrukturen. Studer interface for mengde.
Det er også to klienter, **`KlientBingo`** og **`Ordliste`** som dere kjører når dere har fylt ut med nødvendig kode.

- Kommentar: Alle klasser arver fra Object-klassen.
  Object-klassen har en equals-metode. Den implementasjonen tester på om to objekter har like adresser. Vi ønsker å teste på om innholdet av to objekter er like. Da må vi overkjøre equals()-metoden fra Object –klassen.

a)

Lag **`equals og fjern -`** metodene for de to klassene **`TabellMengde og KjedetMengde`**.
For **`equals-`**metoden bruker du iterator på **parameter-mengden.**

Metoden **`equals`** i klassen **`TabellMengde`** og **`KjedetMengde`** tester om t**`his`**-mengden er lik parameter. To mengder er like om det er samme antall elementer i mengdene og det elementene er nøyaktig de samme. Det er selvsagt tilstrekkelig med siste del av definisjonen, men for en effektiv implementasjon er det også fornuftig å bruke første del.

Klassen ***String*** har implementert metoden **`equals`** slik at vi kan teste om to String-objekter er like. Klassen **`Bingo`** har implementert **`equals`** som tester på at to bingokuler er like.
Kjør så **`KlientBingo`** både for **`TabellMengde`** og **`KjedetMengde`**.

b)

Det et er laget en private-metode, **`settInn for klassen KjedetMengde,`** som setter inn et nytt element i mengden. Denne metoden kan brukes for å gjøre **union** mer effektiv og kan brukes for å lage **snitt** og **differens**. I klassen **`TabellMengde`** fins den allerede.
Vi betegner **`this-`**mengden som **m1** og parameter-mengden som **m2**.

Polymorfisme: En interface-referanse kan referere til objekter av ulike klasser dersom klassene implementerer interface'et. Husk, fra en interface-referanse kan vi bare kalle på de metodene som står i interface'et.
OBS! Hvis vi kaller på **`settInn-`** metoden som altså ikke står i interface'et, må vi typekonvertere til objektreferanse. Vi deklarerer **`begge`** som en interface-referanse.
Se lærebok øverst side-501. Eks: **`(KjedetMengdeADT<T>)begge).settInn(element);`**
Bruk **iterator** for å lage union, snitt og differens.

i) Lag en effektiv union for begge klassene, **`TabellMengde`** og **`KjedetMengde`**.
Tips kladdeoppgaveNr3. Kjør klienten **`OrdListe3 eller OrdListe`** og se at det virker.

ii) Snittet av to mengder m1 og m2 er mengden av de elementer som fins både i m1 og m2.
Lag metoden **`snitt`** for de to klassene **`TabellMengde`** og **`KjedetMengde`** som lager en ny mengde som er snittet av m1(this) og m2 (parameter).

iii) Differansen av to mengder, m1 og m2, (m1- m2) er mengden av de elementer i m1 som ikke fins i m2. Lag metoden **`differens`** som lager en ny mengde som er differensen av denne mengden m1 og parameteren m2.

Kjør så **`OrdListe`** både for **`TabellMengde`** og **`KjedetMengde`** for å se snitt og differens virker

iv) **Frivillig**. For de flittigste**:** Lag en metode **`undermengde`** for de to klassene **`TabellMengde`** og **`KjedetMengde`** som avgjør om parameteren, `m2` er en undermengde av **`this`**-mengden. Bruk iterator på parameter-mengden.

c)

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 interface'et)

En testklasse bør være i en egen "source"-mappe. Se tidligere prosjekter med testing.

Hvordan **kan** vi lage testmetoder for union, snitt og differens? **En mulig måte**: 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 mengdeoperasjon: 1) Der **`m1`**og **`m2`** har felles elementer og 2) der de ikke har felles elementer (disjunkte). Kjør testene og dokumenter utskrift til innlevering senere.

Krav til innlevering for oppgave 1: pdf fil med utskrift av kjøring av klienter og kjøring av testmetodene + zippet prosjekt.

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;
  }
```

Husk, hvis vi bruker objektreferanse når vi alle public-metodene i klassen. Hvis vi bruker interface-referanse når vi kun de public-metodene som er spesifisert i interface'et. Ved å bruke interface-referanse slipper vi gjøre mange endringer i koden når vi bytter implementasjonen.

Oppgave
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:

- *medlemmets 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 en String (Eksempel: <tegne, male>)
    }
    public boolean equals(Object hobby2){ //
        Hobby hobbyDenAndre = (Hobby)hobby2;
     return(hobbyNavn.equals(hobbyDenAndre.getHobbyNavn()));
    }

 }// end Hobby
```

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 formål**).**
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(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 returnerte en referansen slik at vi lettere kunne bytte ut til annen datastruktur, men det lar vi være) .

- **`finnPartnerFor(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 det finner en partner, og returnerer eventuell indeks til partneren i medlemstabellen (eller -1).

- **`tilbakestillStausIndeks(medlemsnavn)`** som oppdaterer medlemstabellen, slik at dette medlemmet (identifisert ved medlemsnavn) og dets partner, dersom det fins, ikke lenger er "koblet" (dvs. begge får statusindeks –1).

Nedenfor finner du en skisse for en klasse Tekstgrensesnitt

```
public class Tekstgrensenitt{// Klasse for inn/ut terminal
//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.

Frivillig: For de flittigste.
Bruk union, snitt og differens i oppgaven over, altså for mengden av hobbyer.
For eksempel. Utvide programmet med å finne snittet av hobbyer mellom to medlemmer osv.

Krav til innlevering for oppgave 2: pdf fil med utskrift av kjøring + zippet prosjekt:
I tillegg: Studer det objektdiagrammet som (lagt ut) 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)`** **s**om kanskje er mindre bra utforming av klassen. Egentlig kunne man tenkt å bruke referanse i stedet for indekser. Et medlem som er koblet refererer til et annet og tilsvarende for partner vha. objektreferanser der null-referanse betyr ledig. MEN følg likevel opplegget i oppgave-teksten der vi bruker **tabellindekser**.