TOD063 Datastrukturer og algoritmer

Øving	: 3
Utlevert	: Uke 7
Innleveringsfrist	: 26. februar 2010
Klasse	: 1 Data og 1 Informasjonsteknologi

Gruppearbeid: 2-3 personer pr. gruppe.

Oppgave 1

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 to-og-to medlemmer med felles interesser (referert til som partnere når en har funnet og koblet to 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 (se vedlegg)
- *statusIndeks* som angir indeks til partneren i medlemstabellen dersom medlemmet er "koblet", ellers er den lik -1.

Klassen *Hobby* kan defineres som:

```
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 og male> )
    }
    public boolean equals(Object hobby2){
        … returnerer true hvis hobbynavnet i objektet er
           det samme som i parameteren
    }
}// end Hobby
```

Klassen *Medlem* kan defineres som:

```
class  Medlem {
    // Egenskaper
    private String navn;
    private KjedaMengde<Hobby> hobbyer;
    private int statusIndeks;
    …
}
```

Test 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).

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

- **`lesMedlem()`** som leser inn personnavn og alle hobbyer til et medlem

- **`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.

- **`skrivHobbyListe()`** som skriver ut alle hobbyene for et medlem (bruk toString-metoden i Mengde)

Klasse *Datakontakt* har en *medlemstabell* som kan lagre opplysninger om maksimalt 100 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):

- **`registrerMedlem()`** som leser inn data for et nytt medlem fra tastatur.
- **`finnMedlemsIndeks(medlemsnavn)`** som finner indeksen til medlemmet i medlemstabellen dersom medlemmet er registrert, ellers returneres indeks lik -1.
- **`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).
- **`skrivParListe()`** som 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 utskriftslisten. Metoden skriver også ut antall par som er funnet.

Eksempel på utskrift:

PARNAVN	HOBBYER
Erna og Siv	<ski> <musikk> <politikk>
Eva og Adam	<epleplukking> <paradishopping>
……	

Antall par funnet: 12

Et enkelt klientprogram (et main - program som bruker klassen *Datakontakt*) skal skrives på en slik måte at vi får testet alle metodene. Du kan få behov for å utvikle andre metoder i klassene. Programmet skal være menybasert.

Krav til innlevering for oppgave 1:

i) Klassekart/UML-diagram
ii) Figur/skisse som viser hvordan klassene *Datakontakt* og *Medlem* er implementert (objektdiagram)
iii) Programkode for hele programmet (font Courier New, size 10-12, minimaliser marger og unngå uheldige sidebrekk og linjebrekk – dvs. se til at det er pen layout og god lesbarhet i den koden dere leverer inn)
iv) Utskrift fra kjøring av programmet, spesielt fra utskrift av 'Parliste'

Oppgave 2 (Formål: Prøve ADTen *Stabel* fra kap. 4)

I denne oppgaven skal vi bruke stabel. Stabel er en sist-inn/først-ut datastruktur (LIFO).

Java tilbyr en rekke datstrukturer definert i ulike klasser. Bl.a. finner vi klassen Stack (bokas kap. 4.6) som dere kan bruke i denne oppgaven. Dere kan også bruke en av de to Stabel-implementeringene lagt ut under KodeEksempler/kap4.

Den innebygde Stack-klassen har bl.a. følgende metoder:

Metoder	Beskrivelse
`boolean empty()`	returnerer **true** hvis stabel er tom, ellers **false**
`Object peek()`	returnerer toppelement, men fjerner det ikke
`Object pop()`	stabler av toppelementet og returnerer det
`Object push(Object element)`	stabler *element* på toppen av stabel

(Unntaket **EmtyStackExecption** blir kastet når du kaller **pop()** hvis stabelen er tom. Den pop()-metoden som ligger på eksempel-siden i it's learning returnerer med null-peker i stedet).

Eksempel på bruk:.

```
import java.io.*;
import java.util.*;    //For å få tilgang til klassene java.util.Stack og
                       //java.util.EmptyStackException
class SnuStreng{

     public static void main(String args[]){
        String str = ".rutkurtsatad OFIL ne re relbatS";
        int len = str.length();
        Stack tegnStabel = new Stack();
        for (int i = 0; i < len;i++){
            tegnStabel.push(new Character(str.charAt(i)));
        }
        System.out.println(str);
        while(!tegnStabel.empty()){
            Character tegn_obj = (Character) tegnStabel.pop();
            System.out.print(tegn_obj);
        }
        System.out.println();
     }
}
```

Vi skal se på parentessetting i et Java-program. Hvis vi ser bort fra at det inne i kommentarer og strenger også kan være parenteser, *skal* parenteser alltid komme i par. Vi kan dele symbolene {, [, (, },] og) i *åpne*-symboler (venstreparentesene) og *lukke*-symboler (høyreparentesene). Når vi går gjennom et Java-program og finner et lukke-symbol, skal det passe med sist møtte åpne-symbol. Det vil si at

[…(…)…] er lovlig
[…(…]…) er ulovlig

For å sjekke om parentesene er rett satt kan vi bruke en stabel. Vi stabler på når vi finner et åpne-symbol og stabler av og sjekker at vi får et par når vi treffer på et lukke-symbol. Tre feilsituasjoner kan oppstå:

1. Symbolene danner ikke et par. Eksempel: Vi treffer på en "]" og øverst på stabelen er det en "{" eller en "(". Da må du skrive ut en passende feilmelding, men du skal likevel behandle resten av teksten også. (Dette kan da i noen tilfeller resultere i en rekke nye feilmeldinger selv om det bare var *en* feil i uttrykket.)

   Eks.: Lukkesymbol] på linje nr x, tegn nr y har feil åpnesymbol

2. Vi treffer på et lukkesymbol og stabelen er tom. Da vil det være rimelig å gi melding om at det er truffet på et lukke-symbol uten tilsvarende åpne-symbol.

   Eks.: Lukkesymbol] på linje x, tegn nr y mangler tilsvarende åpnesymbol

3. Stabelen er ikke tom når det er slutt på teksten. Her kan du gi melding om at det mangler (et eller flere) lukke-symbol.

   Eks.: Åpne-symbol (har ikke tilsvarende lukkesymbol.

   Åpne-symbol [har ikke tilsvarende lukkesymbol.

   Alternativ utskrift (dersom du også lagrer posisjon og linjenummer på stabelen i tillegg til parentesen):

   Åpnesymbol (på linje x, tegn nr y har ikke tilsvarende lukkesymbol

   Åpnesymbol [på linje z, tegn nr w har ikke tilsvarende lukkesymbol

   For å få til denne utskriften må du lage en klasse **ParentesData** der objektene inneholder både linjenummer og posisjon på linjen i tillegg til selve venstre-parentesen. Et slikt objekt lagres altså på stabelen når du finner en venstre-parentes i teksten.

Dere skal lage et Java-program som leser en fil med et Java-program og sjekker om parentesene er satt rett. Dere kan anta at det *ikke* er parenteser i String-uttrykk og i kommentarsetninger. Filnavnet leses inn fra tastatur. En Java-fil er tekstfil, en filtype vi også har brukt i tidligere øvinger. Se eksempel i vedlegget til slutt i oppgavesettet hvordan du åpner en slik fil og leser inn data linjevis til en variabel av type String. Ved feilsituasjonene ovenfor skal en prøve å gi presise feilmeldinger som vist i eksemplene.

Krav til innlevering for oppgave 2:

i) Klassekart/UML-diagram
ii) Skisse for implementert datastruktur
iii) Kildekode (som for oppgave 1)
iv) Utskrift av kjøring med to - tre ulike Java-filer som inndata. Minst en av Java-filene må ha flere parentesfeil.

Vedlegg til øving 3: Eksempel på lesing av tekstfil i Java

```
import java.io.*;
import java.util.*;
class TextFilTest {

   private static void skrivLinjeTegnForTegn(String linje) {
      for (int i = 0; i < linje.length(); i++){
         System.out.print(linje.charAt(i));
      }
      System.out.println();
   }

   public static void lesFraFil(String innFil) {
      FileReader fr = null;
      try {
         fr = new FileReader(innFil);
         //åpner tilgang til filen
      }
      catch(FileNotFoundException e) {
         System.out.println("Finnner ikke filen");
         System.exit(-1);
      }
      BufferedReader br = new BufferedReader(fr);

      while (true) {          //gjentas inntil slutt på fil (eof=true)
         String linje = null;
         try{
            linje = br.readLine();
         }
         catch (IOException ioe) {
            System.out.println("Feil ved innlesing!");
            System.exit(-1);
         }
         if (linje==null)       //undersøker om det var slutt på filen
            break;              //i så fall: terminerer while

         skrivLinjeTegnForTegn(linje);
      }
      try {
         fr.close();
      }
      catch (IOException ioe) {
         System.out.println("Feil ved lukking av fil!");
      }
   }


   public static void main (String args[]) {

      lesFraFil("MinFil.txt");

   }
}
```