Øving 1 - DAT102 Datastrukturer og algoritmer v-20

Øving : 1 sist oppdatert 22.01
Utlevert : 17. januar
Innleveres senere sammen med **Øving 2** (se forelesningsplan)
Klasse : 1 Data og 1 Informasjonsteknologi

Gruppearbeid: 2-3 personer pr. gruppe. Vi henstiller at dere er 3, men ikke flere enn 3. Du kan levere alene, men kan da ikke regne med særlig tilbakemelding. Dette er pga. rettekapasitet!

For å oppnå en best mulig faglig utvikling må du være aktiv på lab. **Det betyr at alle må kode.**
Eks på github https://github.com/dat100hib/dat102. Disse prosjektene vil bli oppdatert etter hvert.

OBS! Denne oppgaven inngår som del av den første obligatoriske øvingen og det er derfor viktig at du kommer raskt i gang og jobber ferdig med denne øvingen. Generelle råd:

- Les gjennom øvingen med en gang den blir utlevert.
- Start med å jobbe individuelt.
- Diskuter med de andre i gruppen.
- Legg en plan for arbeidet. Se notatet Metodikk.
- Husk å opprette et eget workspace for dette faget.

Oppgave 1 (Denne oppgaven vil inngå som del av den første obligatoriske)

En abstrakt datatype (ADT) er en spesifikasjon (beskrivelse) av en samling av data av samme type og en mengde lovlige operasjoner på disse. En ADT uttrykker hva som skal gjøres, men ikke hvordan og er derfor uavhengig av implementasjonen (derfor ordet abstrakt).

Formål: Bruk av interface. **Bruk av flere klasser**
Siden arkivet vårt kun skal være et arkiv av film'er er det mindre aktuelt å bruke generisk type her.

Vi definerer en klasse **Film** som følger:
Data:

- Et entydig Film-nummer (heltall, ingen krav om bestemte nummerserier e.l.)
- Navn på produsent(filmskaper)
- Tittel på Film
- År for lansering (heltall)
- Sjanger () av type enum. Se på slutten av oppgaven om enum.
- Filmselskap

Konstruktører:

- opprette en "tom" Film
- opprette en ny Film med de data som er gitt ovenfor

Videre spesifiserer vi ADT-en **Filmarkiv**: **Se side 3!**

Data:

- En samling filmer-er

Operasjoner:

- Legge inn en ny film. Det er frivillig om du vil teste på om den ligger der fra før.
- Slette en film
- Søker og henter filmer-er med en gitt delstreng i tittelen
- Søke og hente en produsent med en gitt delstreng i tittelen

- Henter antall filmer

Filmarkiv-klassen skal implementere interface'et FilmarkivADT. Du kan også lage andre metoder i klassen ved behov. Klassen skal ha en konstruktør som oppretter et tomt-arkiv med plass til et gitt antall filmer.

Klassen Tekstgrensesnitt skal bl.a. ha disse metodene:

- Skrive ut alle filmer med en spesiell delstreng i tittelen
- Skrive ut alle filmer av en produsent / en gruppe
- Skrive ut en enkel statistikk som inneholder antall filmer totalt og hvor mange det er i hver sjanger.

Klassen Fil skal ha metoder for lesing og skriving til tekstfil med poster av filminformasjon.
Du skal altså lage klasser, interface og et menystyrt program for å administrere et Filmarkiv. I tillegg skal det lages egne klasser som håndterer data fra/til terminal og for fil. Se forslag lenger nede.
Når vi starter programmet, skal vi få spørsmål om vi ønsker å jobbe med et eksisterende arkiv eller om vi vil opprette et nytt. Mellom hver gang vi bruker programmet, lagrer vi arkivet på fil, dvs. også når vi avslutter programmet. I denne oppgaven behøver vi ikke sette inn sorert.

Med det vi har lært til nå, vil det være naturlig å bruke en tabell for å lagre filmene i arkivet, men main-metoden skal være slik at den mest mulig er uavhengig av hvordan filmarkivet er lagret. NB! Det skal ikke brukes ferdige datastrukturer som f.eks. ArrayList i Java. OBS! Etter at vi har gjennomgått teorien om bla. kjedet datastrukturer, skal du i neste øving lage en ny implementasjon av filmarkiv-klassen. **Interfacet skal være det samme. Det betyr at det som står på interface-filen skal gjenbrukes og ikke endres.**

Nyttige tips:

Du skal lagre opplysningene om en film på en *tekstfil*. I prosjektet TekstFil på github kan dere bruke koden i filen Fil.java, MEN dere Må tilpasse koden til variabelnavn dere bruker!

I denne øvingen kan vi bruke et kort skille**streng** (f.eks."#") etter hvert felt i posten unntatt siste. Bakgrunnen for skillestreng er at vi brukers split-metoden i `String`-klassen (Vi kan bruke et skille**tegn** '#', men da må vi bruke `substring` og `indexOf`-metodene som blir mer teknisk). Eks:

…
```
Knut#Olsen#180.0#false
```
…

For å kunne sette av passelig med plass til film-tabellen, kan det være fornuftig å lagre antall filmer først på filen. Om det ikke er plass for å sette inn en ny film, skal det oppretter plass til en 10% større tabell og videre kopiere referansene til de gamle filmene inn i den nye utvidete tabellen.

Ha gruppemøter der dere i hver gruppe:

- Blir enige om utforming av klassene og relasjonene mellom de.
- Blir enige om hvorledes datastrukturen være.

Etter implementering og testing:

- Gå kritisk gjennom koden til hverandre og bli enige om en endelig kode.

Videre, husk:

- Programmet skal være lett å endre / lese / forstå.
- Unngå feil / oppdage feil så tidlig som råd er.

Utviding av en tabell.

```
public void leggTil(Film
element){
   if (antall == tab.length){
       utvidKapasitet();
   }
   tab[antall] = element;
   antall++;
}
```

```
private void utvidKapasitet(){//eks. utvide 10%
   Film[] hjelpetabell = new Film[(int)Math.ceil(1.1 *
tab.length)];
   for (int i = 0; i < tab.length; i++){
      hjelpetabell[i] = tab[i];
    }
    tab = hjelpetabell;
 }
```

Krav til innlevering til neste øving:
Krav til innlevering, se eget notat. I tillegg til programmet leverer dere en kort rapport (ca. 1 side) om de erfaringene dere har gjort og hvordan gruppearbeidet har fungert, et klassekart og et objektdiagram som viser hvordan datastrukturen er implementert.

Klassene og interface for adt skal være på egne filer i egne pakker.

- På implementasjonsfilene: `package no.hvl.data102;`
  `package no.hvl.data102.klient;`
- På adt-filen: `package no.hvl.data102.adt;`

Bemerk: *I deklarasjonene skriver vi **FilmarkivADT filmarkiv** i stedet for **Filmarkiv filmarkiv** som også hadde fungert, men vi skal senere bytte ut tabellimplementasjon av filmarkivet med såkalt kjedet implementasjon. Den skal da implementere det samme interface'et som Filmarkiv, dvs. grensesnittet for klassene skal være den samme. Klienten skal ikke behøve å endres vesentlig.*

Her følger et mulig opplegg som du gjerne kan endre noe på, men programmet skal være oppdelt i klasser og det skal brukes interface for ADT slik at vi lett kan bytte ut til ny implementasjon:

```
public interface FILMarkivADT {// Bruk gjerne javadoc her
i
                              // stedet for vanlige
                              // kommentarlinjer som her
// Returnere en tabell av Filmer
  Film[] hentFilmTabell();

// Legger til en ny Film
  void leggTilFilm(Film nyFilm);

 // Sletter en Film hvis den fins
  boolean slettFilm(int filmNr);

  // Søker og henter Filmer med en gitt delstreng
  Film[] sokTittel(String delstreng);

  // Søker og henter produsenter med en gitt delstreng
  Film[] sokProdusent(String delstreng);


  // Henter antall Filmer for en gitt sjanger
   int antallSjanger(Sjanger sjanger);
 }
  // Returnerer antall Filmer
  int antall();

 }//interface
```

```
public class Film {
  …
  private Sjanger sjanger;
  //Konstruktør
  …
  public getSjanger() {
  return sjanger;
  …
  }
}
```

```
public class Filmarkiv implements FilmarkivADT {
    //Instansvariable
    private Film[] filmTabell;
    private int antall;
    //Konstruktører og andre metoder
    //…fyll ut
}//class
```

```
public class Tekstgrensesnitt {

 // lese opplysningene om en FILM fra tastatur
   public Film lesFilm(){…}

// vise en film med alle opplysninger på skjerm (husk tekst for sjanger)
   public void visFilm(Film film){…}

// Skrive ut alle Filmer med en spesiell delstreng i tittelen
   public void skrivUtFilmDelstrengITittel(FilmarkivADT filma, String delstreng){…}

// Skriver ut alle Filmer av en produsent / en gruppe
   public void skrivUtFilmProdusent(FilmarkivADT filma, String delstreng) {…}

// Skrive ut en enkel statistikk som inneholder antall Filmer totalt
// og hvor mange det er i hver sjanger
   public void skrivUtStatistikk(FilmarkivADT filma){…}
//… Ev. andre metoder

}//class
```

I prinsippet kunne vi nå enkelt bytte ut den tekstbaserte klassen med et grafisk basert (GUI) som vi ikke gjør her.
Det er også vanlig å ha et eget interface for Fil, men det trenger vi heller ikke benytte i denne oppgaven.
Metodene i klassen Fil kan gjerne være **`static.`**
Du kan opprette et Scanner-objekt i lesFilm og ha det åpent hele tiden og eventuelt lukke det på slutten i Meny når all lesing er ferdig.

```
import …; // se fileksemplene i samme mappen som øvingen
public class Fil {
   final String SKILLE = "#";            // Eventuelt ha som parameter i
                                         // metodene.
   // Lese et Filmarkiv fra tekstfil
     public static void lesFraFil(FilmMarkivADT filmarkiv, String filnavn){…}
Alt: public static FilmarkivADT lesFraFil(String filnavn) {
         FILMarkivADT filma = null;
         …;
         filma = new Filmarkiv(n);
         …;
         return filma
}
// Lagre et Filmarkiv til tekstfil
  public static void skrivTilFil(FilmarkivADT filmarkiv, String filnav){…}

}//class
```

```
public class KlientFilmarkiv{
  public static main((String[] args){
  //… meny
  FilmarkivADT filma = new Filmarkiv();
  Meny meny = new Meny(filma);
          …
 }
}//class
```

```
Alt:
public class KlientFilmarkiv {
  public static void main(String[] args){
    …
    Meny meny = new Meny(filma);
    meny.start();
  }
}//class
```

```
public class Meny{
   private Tekstgrensesnitt tekstgr;
    private FilmarkivADT filma;
    public Meny(FilmarkivADT filma){
       tekstgr = new Tekstgrensenitt();
       this.filma = filma;
  }
…
  public void start(){…}

}//class
```

```
@Override
// Søker og returnerer antall filmer med en gitt
sjanger
public int antallSjanger(Sjanger sjanger) {
   int antallSjanger = 0;
   for (int i = 0; i < antall; i++) {
       if (filmTabell[i].getSjanger() == sjanger) {
          antallSjanger++;
       }
    }
    return antallSjanger;
}
```

Dere kan lage klient- og menyklassen som dere ønsker bare den er fornuftig utformet.

Husk, tekstfilen kan se slik ut som under, altså på den første linjen står antall film-informasjoner:
Eks:

```
12
Nr#Produsent#Tittel#år#Sjanger#Plateselskap
...
```

Se forelesning om enum og koden på github:

```
// Gir antall enum-objekter dvs. antall sjangre
int lengde = Sjanger.values().length;

// Gir en tabell av referanser til enum-objekter som er opprettet.
Sjanger[] sjangTab = Sjanger.values();
...
```

Javadokumentasjon:

https://docs.oracle.com/javase/10/docs/api/java/lang/Enum.html

https://docs.oracle.com/javase/10/docs/api/java/lang/String.html#split-java.lang.String-

---

Bemerkning:

På en interface-fil har vi "tomme" metoder altså uten implementasjon. I implementasjonsfilen ligger metodene implementert.

OBS! Hvis vi utvider med nye public-metoder på implementasjonsfilen som **ikke** står i interface-filen, kan vi ikke nå de ved en interface-referanse. Da må vi bruke objekt-referanse. Men vi skal bruke interface-referanse som parameter for å nå metoder i Filmarkivet, så dere skal **ikke** utvide klassen senere med andre public-metoder enn de som står i oppgaven for ADT. Grunnen til det er at når vi bruker interface-referanse trenger vi ikke å modifisere parametertype når vi endrer implementasjonen. Vi skal i neste øving endre mplementasjonen fra tabell til kjedet struktur. Det er veldig viktig at dere gjør ferdig øving1 før øving2 blir utdelt!

Fra oppgaven skriver vi:

```
public class KlientFilmarkiv{
   public static main((String[]args){
  //… meny
  FilmarkivADT filma = new Filmarkiv();  // Her konverteres automatisk fra
                                         //  obj.-ref til interface-ref.
  Meny meny = new Meny(filma);           …
  }
}//class
```

Trimming av tabell og sletting.

Det kan være lurt å ha en hjelpemetode i Filmarkiv-klassen som trimmer en tabell dvs. at vi alltid har en full tabell av referanser til objekter. Noen metoder i Filmarkiv-klassen kan bruke den.

Når vi skal slette et element i en tabell, kan vi gjøre det ved å la referansen til det slettede elementet referere til det siste elementet og minke antall med 1. Tegn opp så ser du det. Da vil alle referanser som refererer til objekter komme sammenhengende etter hverandre (hvis tabellen da ikke er blitt tom). Da kan vi enkelt trimme tabellen som vist i metoden under. Vi får da at eventuelle null-referanser alltid vil være på slutten av tabellen.

```
private Film[] trimTab(Film[] tab, int n) { // n er antall elementer
    Film[] filmtab2 = new Film[n];
    int i = 0;
    while (i < n) {
         filmtab2[i] = tab[i];
         i++;
   }//while
   return filmtab2;
}//
```

Robusthet: Det er ikke noe krav om å teste datatyper ved registrering. Vi må bruke unntak i forbindelse med filer.

Oppgave 2 (Effektivitetsanalyse fra kapittel 2)
Denne oppgaven vil inngå som del av den første obligatoriske oppgaven.

Se notatene om analyse, effektivitetsmål for algoritmer.

La oss anta det foreligger en algoritme for et spesielt problem der n (positivt heltall) er et mål for størrelsen på problemet. Tidsforbruket t(n) til algoritmen er ofte avhengig av problemstørrelsen.

Vi sier at t(n) er av orden f(n) hvis det fins positive konstanter c og N slik at $t(n) \leq cf(n)$ for alle $n \geq N$ der t(n) ikke er negativ. I stor O-notasjon er skrivemåten t(n) er O(f(n)), der t(n) er vekstfunksjonen.

Det betyr at når $n \geq N$ vil grafen til t(n) ligge under grafen til cf(n). Følgende kan vises:

O(k(f(n)) er O(f(n)); her er k er en positiv konstant
O(f(n)) + O(g(n)) er O(f(n) + g(n))
O(f(n))O(g(n)) er O(f(n)g(n))

Vi har også at O(k) der k er en konstant er O(1), for eksempel O(10000) er O(1). Det betyr at tiden er uavhengig av størrelsen på problemet.

Eks: Gitt en algoritme som bruker $3n^2 + 9n$ operasjoner, dvs. $t(n) = 3n^2 + 9n$.
La oss vise at $t(n) = O(n^2)$ ut fra definisjonen over.
Siden $9n \leq n^2$ for alle $n \geq 9$ har vi at $3n^2 + 9n \leq 4n^2$ for alle $n \geq 9$.
Vi lar $\mathbf{f(n) = n^2}$, c = 4 og N = 9. Se definisjon over som er uthevet.
Altså har vi at $t(n) = 3n^2 + 9n$ er $O(\mathbf{n^2})$.

Kommentar: Vi ser at det er det dominerende leddet (det leddet som vokser raskest når n øker) her $\mathbf{n^2}$, som vi angir med O-notasjonen. Det betyr at i oppgaver der vi bare skal angi tidsforbruket til en algoritme uttrykt i O-notasjon behøver vi ikke finne c og N. Men noen ganger der vi skal måle tidsforbruket mer nøyaktig må vi ta hensyn til konstanten i det dominerende leddet + eventuelt de lavere ordens leddene.

En aritmetisk rekke er sum av tall der avstanden mellom tallene er den samme. Formelen for en aritmetisk rekke er ((førsteLedd + sisteledd)/2)*antallLedd . Disse rekkene vil ofte dukke opp i faget.

$1 + 2 + 3 + \ldots + n\text{-}1 + n \quad = n(n + 1)/2$ og dermed har vi også
$1 + 2 + 3 + \ldots + n\text{-}2 + n\text{-}1 = (n\text{-}1)n/2$

Oppgave 2 a)

Hva er størrelsesorden uttrykt i O-notasjon (dvs. vi behøver ikke finne c og N) for algoritmen når vekstfunksjonene er gitt som :

i) $4n^2 + 50n - 10$
ii) $10n + 4 \log_2 n + 30$
iii) $13n^3 - 22n^2 + 50\,n + 20$
iv) $35 + 13 \log_2 n$

Oppgave 2 b)

Gitt følgende algoritme:

```
for (int i = 1; i <= n; i++){
  for (int j = 1; j <= n; j++) {
        sum = sum + 1;
   }
}
```

Finn antall tilordninger (=) for algoritmen og effektiviteten uttrykt i O-notasjon.
Begrunn svaret. Vi ser kun på løkkekroppen når vi analyserer løkker!

Oppgave 2 c)

Gitt tre algoritmer A, B og C som alle beregner summen $1 + 2 + \ldots + n = n(n+1)/2$, n heltall > 0.

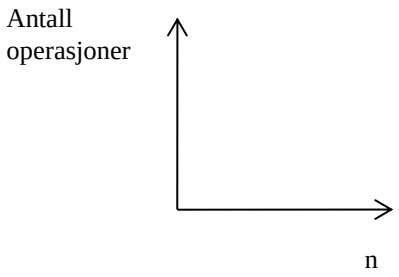
Algoritme A	Algoritme B	Algoritme C

`sum = 0`
`for (int i = 1; i <= n; i++){`
`  sum = sum + i;`
`}`
	`sum = 0`
	`for (int i = 1; i <= n; i++){`
	`  for (int j = 1; j <= i; j++){`
	`  sum = sum + 1`
	`  }`
	`}`
		`sum = n*(n + 1)/2`

Finn antall operasjoner (tilordninger, addisjoner, multiplikasjoner, divisjoner) for hver av algoritmene og fyll ut tabellen under. **Vi tar ikke med tilordning av løkkeindeks**

	Algoritme A	Algoritme B	Algoritme C
Tilordninger =			
Addisjoner +			
Multiplikasjoner *			
Divisjoner /			
Totalt antall operasjoner			

OBS! Forenkling når vi sidestiller operasjonene.
Plott inn resultatene som tre grafer for de tilsvarende algoritmene. (Velg enhet lik 1 på n-aksen og 4 på den vertikale aksen).



Oppgave 2 d)

Vi antar at det foreligger 5 algoritmer av ulik orden. I tabellen er det gitt de tilhørende vekst-funksjonene. Vi er interessert i å finne ut tiden det tar å prosessere 10^6 elementer for de ulike algoritmene på en prosessor som kan utføre 10^6 operasjoner pr. sekund. log n betyr her $\log_2 n$; $\log_2 n = \ln n / \ln 2$. Fyll ut siste kolonne i tabellen under og angi med de enheter som er angitt.

t(n)	**$t(10^6)/ 10^6$**
$\log_2 n$	… [sekunder]
n	… [sekunder]
$n \log_2 n$	… [sekunder]
n^2	… [dager]
n^3	… [år]

Oppgave 2 e)

Følgende metode avgjør om en tabell inneholder minst en duplikat:
Finn antall sammenligninger i verste tilfelle for algoritmen og effektiviteten uttrykt i O-notasjon.
Begrunn svaret.

```
boolean harDuplikat (int tabell[], int n){// n er antall elementer
   for (int indeks = 0; indeks <= n-2; indeks++){
      for (int igjen = indeks + 1; igjen <= n-1; igjen++){
          if(tabell[indeks] == tabell[igjen])
               return true;
      }
   }
   return false;
}//metod
```

Oppgave 2 f)

Vi ser på tidskompleksiteten for vekstfunksjoner til 4 ulike algoritmer (for en viktig operasjon) der n er antall elementer.

i) $t_1 = 8n + 4n^3$ iii) $t_3 = 20n + 2n\log_2 n + 11$
ii) $t_2 = 10 \log_2 n + 20$ iv) $t_4 = 4\log_2 n + 2n$

Hva er O-notasjonen for de ulike vekstfunksjonene?
Hvilken av algoritmene over er mest effektiv og hvilken er minst effektiv (i begge tilfeller for store n)?
Grunngi svaret.

Oppgave 2 g)

Gitt følgende metode:

```
public static void tid(long n) {
//…fyll ut
   long k = 0;
   for (long i = 1; i <= n; i++) {
       k = k + 5;
   }
 //…fyll ut
}
```

Vi ønsker å måle tiden for n = 10 mill, 100 mill og 1000 mill. Bruk metoden javametoden,

`public static long currentTimeMillis()`.

Hvorfor er vekstfunksjonen her T(n) = cn, der c er en konstant.

Bruk disse kallene: Kjør programmet noen ganger og se om du da får bedre resultater.

```
public static void main(String[] args) {
   tid(100000000L);
   tid(1000000000L);
   tid(10000000000L);
 }
```

Hvordan stemmer resultatene?
Det kan ikke bli helt nøyaktige tider fordi **`currentTimeMillis`** er basert på systemklokken og ikke på prosessortiden. Det er flere kilder som kan forstyrre måling av tiden basert på systemklokken.