DAT102 – Våren 2023

Øving 2
Utlevert 13. januar

Leveres inn sammen med Øving 3

Introduksjon

Denne oppgaven inngår som del av den første obligatoriske øvingen. Vi henstiller at dere er 3-4 på gruppen, men aldri flere. 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.**

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» som du finner i modulen Informasjon på Canvas.
- Husk å opprette et eget workspace for dette faget.

Oppgave 1 (inngår i den første obligatoriske)

Formål: Bruk av grensesnitt (interface). Bruk av flere klasser.

Definisjon: 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).

Vil skal se på hvordan et filmarkiv kan implementeres både ved hjelp av tabell (denne oppgaven) og senere ved hjelp av en kjedet struktur.

Opprett et nytt Javaprosjekt på vanlig måte. Vi har vi stort sett hatt alt i **samme** pakke. Når programmene blir større, er det fornuftig å dele de i pakker. Lag følgende pakker i prosjektet: Unngå æ, ø og å i koden.

- `package no.hvl.data102;`
  Klasser for Film og Filmarkiv
- `package no.hvl.data102.klient;`
  Klasser for å kunne bruke arkivet
- `package no.hvl.data102.adt;`
  For grensesnitt fil

Vi definerer en klasse Film som følger:

- Data:
  - Et entydig filmnr (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 et "tomt" film-objekt
    - opprette et nytt film-objekt med de data som er gitt ovenfor
- Metoder
    - Get/set-metoder for alle objektvariablene
    - `equals`-metode som overkjører tilsvarende metode i Object-klassen. To filmer er like om de har samme nummer. Se equals-metoden i Person-klassen i prosjektet Bag.
    - `hashCode`-metode som overkjører tilsvarende metode i Object-klassen. Når man overkjører `equals`-metoden, skal man også overkjøre `hashCode`-metoden slik at objekter som er like også har samme `hashCode`. Dette har vi ikke sett på enda, men det kommer senere. Om nr er navnet på medlemsvariabelen, kan vi skrive `hashCode` slik:

```
@Override
public int hashCode() {
    Integer temp = nr;
    return temp.hashCode();
}
```

Lag klassen `Film`.

Videre spesifiserer vi ADT-en Filmarkiv. Data er en samling filmer. Lovlige operasjoner er gitt som et grensesnitt (interface) i Java som er vist under. Det er vanlig å bruke Javadoc-kommentarer framfor metodene for å angi en kort forklaring av hva metoden skal gjøre. Deretter kan man bruke ulike tags for å forklare eventuelle parametre og returverdi.

For å starte en Javadoc-kommentar startar man med «/**» og trykker ny linje (enter). Da får man opp en mal med tags for eventuelle parametre (@param) og returverdi (@return) som kan fylles ut. Det finnes flere tags. Når man senere implementer grensesnittet, får man opp første linje av metodene med forklaring. Javadoc brukes også for å generere dokumentasjon.

```
public interface FilmarkivADT {
	/**
	 * Hente en film med gitt nr fra arkivet
	 *
	 * @param nr nummer på film som skal hentes.
	 * @return film med gitt nr. Om nr ikke finnes, returneres null.
	 */
	Film finnFilm(int nr);

	/**
	 * Legger til en ny film.
	 * @param nyFilm
	 */
	void leggTilFilm(Film nyFilm);

	/**
	 * Sletter en fil med gitt nr
	 * @param filmnr nr på film som skal slettes
	 * @return true dersom filmen ble slettet, false ellers
	 */
	boolean slettFilm(int filmnr);
```

```
    /**
     * Søker og henter Filmer med en gitt delstreng i tittelen.
     * @param delstreng som må være i tittel
     * @return tabell med filmer som har delstreng i tittel
     */
    Film[] soekTittel(String delstreng);

    /**
     * Finner antall filmer med gitt sjanger
     * @param sjanger
     * @return antall filmer av gitt sjanger.
     */
    int antall(Sjanger sjanger);

    /**
     *
     * @return antall filmer i arkivet
     */
    int antall();

}
```

Dersom du kopierer koden, slett et par av metodene med tilhørende Javadoc og skriv de inn på nytt slik som angitt ovenfor. Da får du se hvordan du selv kan gjøre dette fram grunnen av.

Når du har laget grensesnittet `FilmarkivADT`, skal du lage Filmarkiv-klassen. Denne skal implementere grensesnittet `FilmarkivADT` ved hjelp av en tabell for å lagre filmene. Filmene skal lagres sammenhengende i starten av tabellen. Senere skal vi se på hvordan det kan gjøres ved en kjedet struktur. Klassen skal ha en konstruktør med som oppretter et tomt arkiv med plass til et gitt antall filmer (gitt som parameter til konstruktøren).

Dersom tabellen er full når det skal legges til en ny film, oppretter dere en tabell som er dobbelt så stor og kopierer filmene over i denne tabellen. Deretter lar dere objektvariabelen referere/peke til den nye tabellen. Dette kan gjerne gjøres i en privat hjelpemetode, `utvid`.

Tabellene som returneres fra søkemetodene bør være fulle. Se trimming av tabeller i slutten av oppgaven.

Lag klassen `Filmarkiv`.

Dersom vi skulle ha administrert et filmarkiv, måtte vi lagret filmene på en fil eller i en database mellom hver gang vi kjørte programmet. I stedet for å bruke tid på filbehandling (som ikke er en sentral del av kurset), kan vi lage en metode som erstatter det å lese filmer fra fil med en metode som setter inn en del forhåndsdefinerte filmer i arkivet. Det betyr at vi ikke får lagret eventuelle nye filmer.

Det finnes litt ulike måter å organisere bruken av filmarkivet på. En måte kan være å lage klassene:

- `KlientFilmarkiv`
- `Tekstgrensesnitt`
- `Meny`

Da kan deler av implementasjonen se slik ut (resten må fylles ut):

```
public class KlientFilmarkiv {

	public static void main(String[] args) {
		FilmarkivADT filma = new Filmarkiv(100);
		Meny meny = new Meny(filma);
		meny.start();
	}

}

public class Meny {
	private Tekstgrensesnitt tekstgr;
	private FilmarkivADT filmarkiv;

	public Meny(FilmarkivADT filmarkiv){
		tekstgr = new Tekstgrensesnitt();
		this.filmarkiv = filmarkiv;
	}

	public void start(){
		// legg inn en del forhåndsdefinerte filmer
		// TODO
	}

}

public class Tekstgrensesnitt {
	// lese opplysningene om en FILM fra tastatur
	public Film lesFilm(){
		// TODO
		return null;
	}

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

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

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

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

}
```

Enum (oppramstype)

Du finner forslag til en Enum for sjanger på GitHub. To nyttige metoder

```
// 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();
```

Det finnes flere måter å konvertere fra tekst til enum og andre veien. Her er tips til hvordan det kan gjøres:

- Konvertering fra tekst til enum
  ```
  String s = tekst;
  Sjanger sjanger = Sjanger.valueOf(s);
  ```

  Det finnes en metode i `String`-klassen som gjør en streng om til bare store bokstaver. Du kan eventuelt bruke denne før konvertering.

- Konvertering fra enum til tekst
  Det letteste er å bruke `toString()`-metoden. Standardimplementasjon for denne er ut bruke navnet på konstantene, for eksempel KRIM.

Trimming av tabeller

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

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

Oppgave 2 (inngår i første obligatoriske)

Introduksjon

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. Et eksempel: Dersom problemet er sortering, vil størrelsen på problemet være antall elementer som skal sorteres. Tidsforbruket *t(n)* til algoritmen er ofte avhengig av problemstørrelsen.

> **Definisjon**: 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:

$O(kf(n))$ er $O(f(n))$ der 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.

Eksempel: Gitt en algoritme som bruker $3n^2 + 9n$ operasjoner, dvs. $t(n) = 3n^2 + 9n$. La oss vise at $t(n)$ er On^2) ut fra definisjonen over.

Siden $9n \leq n^2$ for alle $n \geq 9$ har vi at $3n^2 + 9n \leq 3n^2 + n^2 = 4n^2$ for alle $n \geq 9$. Vi lar $f(n) = n^2, c = 4, N = 9$ i definisjonen over. Altså har vi at $t(n) = 3n^2 + 9n$ er $O(n^2)$.

Kommentar: Vi ser at det er det dominerende leddet (det leddet som vokser raskest når n øker) her 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. Noen ganger der vi skal måle tidsforbruket mer nøyaktig må vi ta hensyn til konstanten i det dominerende leddet + eventuelt de lavere ordens leddene (leddene som vokser senere enn det dominerende leddet).

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

$$1 + 2 + \cdots + n = \frac{(1+n)}{2}n = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$$

Siste steget har vi tatt med så det blir lettere å se hvor raskt rekken vokser. Noen ganger stopper vi rekken på *n-1*. Hva blir summen da?

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. $13n^3 + 22n^2 + 50n + 20$
iv. $35 + 13\log_2 n$

Oppgave 2 b

Gitt følgende algoritme:

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

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 følgende algoritme:

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

Finn antall tilordninger (=) for algoritmen og effektiviteten uttrykt i O-notasjon. Begrunn svaret.

Oppgave 2 d

Vi ser på en sirkel med radius r. Da vil areal og omkrets være gitt med formlene:

$2\pi r^2$ og $2\pi r$

Angi i stor O-notasjon hvordan areal og omkrets vokser. Dette har ikke direkte med en algoritme å gjøre, men er med for å sjekke om dere har forstått begrepene vekstfunksjon og stor O-notasjon. Oppgaven er svært lett om dere har skjønt begrepene.

Oppgave 2 e

Følgende metode avgjør om en tabell med n elementer inneholder minst en duplikat:

```
boolean harDuplikat(int tabell[], int n) {

      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;
}
```

Finn antall sammenligninger i verste tilfelle for algoritmen og effektiviteten uttrykt i O-notasjon. Begrunn svaret.

Oppgave 2 f)

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

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

Hva er O-notasjonen for de ulike vekstfunksjonene?

Ranger vekstfunksjonene etter hvor effektive de er (fra best til verst). Anta at *n* er stor.

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
}
```

Det finnes flere heltalstyper i Java. I DAT100 har vi stort sett brukt `int`, men husk at alle heltalstyper i Java har en øvre grense (i motsetning til i matematikk). Om vi trenger større heltall enn ca $2*10^9$, kan vi bruke heltalstypen `long`. For å angi et `long`-heltall skriver vi en stor «L» bak tallet, for eksempel: 10000000000L.

```
public static long currentTimeMillis().
```

Denne metoden kan kalles før og etter `tid()`-metoden og så beregner man tiden det har gått mellom de to kallene.

Hvorfor er vekstfunksjonen tid()-metoden *T(n) = cn*, der c er en konstant?

Vi ønsker å måle tiden for n =10^7, 10^8 og 10^9 når vi kaller `tid()`-metoden. Tidsmålingene blir ikke helt nøyaktige siden `currentTimeMillis` er basert på systemklokken og ikke på prosessortiden. Det er flere kilder som kan forstyrre måling av tiden basert på systemklokken. Du får mer nøyaktige tider ved å kjøre metoden flere ganger og så finne gjennomsnittet.

Hvordan stemmer resultatene vekstfunksjonen? Diskuter.