

# DAT102 – Våren 2026

Øvingsoppgaver uke4 (19.-25. januar)

## Introduksjon

**Oppgavene i dette settet inngår som del av den første obligatoriske øvingen!**

Vi anbefaler at dere er 3-4 på gruppen, men aldri flere. Hvis dere lager grupper på fire, kan to og to 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 fire. Grupper som er mindre enn tre, kan 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.

Mer informasjon om selve innleveringen kommer senere.

## Oppgave 1

**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 ved hjelp av en lenket struktur (neste oppgave).

Opprett et nytt Javaprojekt 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.

- no.hvl.data102.filmarkiv.adt; Interfacet **FilmarkivADT**
- no.hvl.data102.filmarkiv.impl; Klassene **Film**, **Filmarkiv** og (enum) **Sjanger**
- no.hvl.data102.filmarkiv.klient; Klassene **FilmarkivMain**, **Meny** og **Tekstgrensesnitt**
- no.hvl.data102.filmarkiv.test; Klassen (junit-test) **FilmarkivTest**

### 1.1

Lag klassen **Film**. Denne skal inneholde

- Data (private)
  - 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.
  - Navn på filmselskap
- Konstruktører
  - opprette et "tomt" film-objekt
  - opprette et nytt film-objekt med de data som er gitt ovenfor
- Metoder
  - get- og set-metoder for alle objektvariablene
  - equals- og hashCode-metoder som overkjører tilsvarende metoder i Object-klassen. To filmer er like om de har samme nummer. Når man overkjører equals-metoden, skal man også overkjøre hashCode-metoden slik at objekter som er like også har samme hashCode. (Du kan bruke Eclipse generere disse for deg)

### 1.2

Videre spesifiserer vi ADT-en for et filmarkiv. Data er en samling filmer. Lovlige operasjoner er gitt som et grensesnitt (interface) i Java som vist under. Det er vanlig å bruke Javadoc-kommentarer til metodene for å angi en 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 starter 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 implementerer grensesnittet, får man opp første linje av metodene med forklaring. Javadoc brukes også for å generere dokumentasjon.

## Lag interfacet FilmarkivADT

```
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 delstrek i tittelen.  
     * @param delstrek som må være i tittel  
     * @return tabell med filmer som har delstrek i tittel  
     */  
    Film[] soekTittel(String delstrek);  
  
    /**  
     * Søker og henter filmer med en gitt delstrek i filmprodusent  
     * @param delstrek  
     * @return  
     */  
    Film[] soekProdusent(String delstrek);  
  
    /**  
     * 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 fra grunnen av.

### 1.3

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. Klassen skal ha en konstruktør med som oppretter et tomt arkiv med plass til et gitt antall filmer (gitt som parameter til konstruktøren).

Husk, Filmarkiv skal implementere alle metodene på filen FilmarkivADT.java (interface), men kan også ha andre metoder (hvis dere har bruk for).

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 hjelpekode, utvid.

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

### 1.4

Det er lurt å teste om ting virker underveis når man skriver kode. **Bruk JUnit til å enhetsteste metodene i Filmarkiv etter hvert som dere lager dem.** Det er ikke nødvendig at testene er «komplette». Det viktige er at dere blir vant til å bruke JUnit til testing.

### 1.5

Det finnes litt ulike måter å organisere bruken av filmarkivet på.

En måte kan være å lage klassene:

- **FilmarkivMain**, som inneholder main()-metoden for å starte programmet
- **Meny**, som starter med å legge inn en del filmer i arkivet\*, og deretter går i en (evig) løkke, spør brukeren hva hun ønsker å gjøre, og får utført dette. Ting som skal kunne gjøres i programmet må omfatte bruk av **alle** metodene i **Filmarkiv**.
- **Tekstgrensesnitt** er en hjelpeklasse som Meny bruker for å få utført tingene man ønsker å gjøre.

\**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.*

**NYTT 2026: Dere trenger ikke å lage et komplett main-program (det er for tidkrevende). Det er tilstrekkelig at dere lager fungerende menypunkter for et par sentrale funksjoner.**

Skjeletter for klassene **FilmarkivMain**, **Tekstgrensesnitt** og **Meny** står nedenfor. Utvid (og fullfør) disse slik at dere får et fungerende program for å utprøve **alle Filmarkiv** sine metoder.

```

public class FilmarkivMain {

    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 for å teste metodene
        // ..
        // TODO
    }
}

public class Tekstgrensesnitt {

    // Leser inn opplysninger om en film fra tastatur og returnere et Film-objekt
    public Film lesFilm(){
        // TODO
    }

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

    // Skriver ut alle filmer med en spesiell delstrek i tittelen
    public void skrivUtFilmDelstrekITittel(FilmarkivADT arkiv, String delstrek) {
        // TODO
    }

    // Skriver ut alle Filmer av en produsent (produsent er delstrek)
    public void skrivUtFilmProdusent(FilmarkivADT arkiv, String delstrek) {
        // TODO
    }

    // Skriver ut en enkel statistikk som inneholder antall filmer totalt
    // og hvor mange det er i hver sjanger.
    public void skrivUtStatistikk(FilmarkivADT arkiv) {
        // TODO
    }

    // osv ... andre metoder
}

```

## Enum (oppramstype)

En enum for sjanger kan f.eks. se slik ut:

```
public enum Sjanger {  
    ACTION, DRAMA, HISTORY, SCIFI;  
  
    public static Sjanger finnSjanger(String navn) {  
        for (Sjanger s : Sjanger.values()) {  
            if (s.toString().equals(navn.toUpperCase())) {  
                return s;  
            }  
        }  
        return null;  
    }  
}
```

Legg til og fjern sjangre slik du synes listen bør være.

Et par nyttige tips:

```
// En tabell av referanser til enum-objekter som er opprettet.  
Sjanger[] sjangTab = Sjanger.values();  
  
// Antall enum-objekter dvs. antall sjangre  
int lengde = Sjanger.values().length;
```

Det finnes flere måter å konvertere fra tekst til enum og andre veien.

- 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 DRAMA.

## Trimming av tabeller

Det kan være lurt å ha en hjelpekode 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

I denne oppgaven skal du ta utgangspunkt i **Filmkiv**-programmet fra forrige oppgave og implementere klassen **Filmkiv** på en ny måte. Definer klassen **Filmkiv2** med de samme operasjonene som i Oppgave 1, men de enkelte filmene skal nå lagres i en lineær kjedet struktur (i stedet for i en tabell).

Definer klassene nedenfor. Flere av metodene må nå få ny implementasjon. Husk, metodene skal ha navn, parametertype og returtype som i interfacet. Interface **FilmkivADT** skal være det samme som i Oppgave 1.

Klassen Film skal ikke endres fra Oppgave 1. Klassen LinearNode kan se slik ut:

```
public class LinearNode<T> {  
  
    public T data;  
    public LinearNode<T> neste;  
  
    ...  
}
```

eller en variant av dette.

Deler av av klassen Filmkiv2 ser slik ut:

```
public class Filmkiv2 implements FilmkivADT {  
  
    private int antall;  
    private LinearNode<Film> start;  
  
    ...  
}
```

De andre klassene skal i størst mulig grad være uendret. Metodene i klassen for Filmkiv2 (implementert vha kjedet struktur) skal ha samme oppførsel som metodene i klassen for Filmkiv (implementert vha av tabell).

**Bruk JUnit til å enhetsteste metodene i Filmkiv2 etter hvert som dere lager dem.** Dere skal i teorien kunne gjenbruke de samme testene som i Oppgave1.