# DAT102 – Våren 2023

## Øving 3

Utlevert 23. januar

# DAT102 – Obligatorisk nr 1

Består av både Øving 2 (utlevert 13.01) og Øving 3 (denne)

## Innleveringsfrist: 3. februar 2023

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. Pass på å veksle mellom hvem som skriver. De to smågruppene blir enige om en felles innlevering slik at dere leverer inn som en gruppe på 4.

OBS! Husk å ta med navn på alle gruppemedlemmene. Canvas tillater ikke å gjør det i ettertid.

Dere eksporterer javaprosjektene, zipper og leverer inn på Canvas. Prosjektene skal kunne kjøres. Teorioppgaver leverer dere inn på canvas som pdf-fil. I denne øvingen skal vi ikke bruke ferdiglagde samlinger (fra Collection i Java).

### Formål

- Interface
- Kjedet struktur, ny implementasjon
- Enhetstesting med JUnit

## Oppgave 1 og oppgave 2 er fra øving 2

På interface-filen tar du med programdokumentasjon, kort dokumentasjon av hva metodene skal gjøre. Lenke til tips for å skrive Javadoc:

http://www.oracle.com/technetwork/articles/java/index-137868.html.

I dette faget kan du utelate Javadoc dokumentasjonen i klassene når du har det med på interface-filen.

Annotasjoner er merker som beskriver andre data. Annotasjoner brukes under kompilering og kjøring og har bestemte merker. https://www.javatpoint.com/java-annotation. Vi kommer tilbake til annotasjoner ved testing.

Bruk annotasjonen **`@Override`** på alle metodene i implementasjonsfilen som implementerer metodespesifikasjonene i interface (kommer opp automatisk i Eclipse). En annen lenke til annotasjoner: https://crunchify.com/understanding-java-annotation-annotation-examples/

## Oppgave 3

I denne oppgaven skal du ta utgangspunkt i **`Filmarkiv`**-programmet fra øving 2 og implementere klassen **`Filmarkiv`** på en ny måte. Definer klassen **`Filmarkiv2`** med de samme operasjonene som i øving 2, 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 **`FilmarkivADT`** skal være det samme som i øving 2.

Klassen `Film` skal ikke endres fra øving 2. Klassen **`LinearNode`** kan hentes fra GitHub.

```
public class LinearNode <T>{
    private LinearNode<T> neste;
    private T element;
    …
}
```

Deler av av klassen `Filmarkiv2` ser slik ut:

```
public class Filmarkiv2 implements FilmarkivADT{
    private int antall;
    private LinearNode<Film> start;
    // Ingen referanse til siste, kun start
    …
    /*  Klassen skal ha de samme operasjoner som for Filmarkiv i
        øving 2, men pass på at implementeringen av alle metoder
        blir tilpasset den nye strukturen. */

}
```

De andre klassene skal i størst mulig grad være uendret. Metodene i klassen for Filmarkiv2 (implementert vha kjedet struktur) skal ha samme oppførsel som metodene i klassen for Filmarkiv (implementert vha av tabell).

La *k* være antall sjangre og *n* være antall filmer. Hva er tidskompleksiteten uttrykt i O-notasjon med *k* og *n* for de to metodene i) og ii) under? Begrunn svaret.

**i)** **`antallSjanger(Sjanger sjanger)`** og
**ii)** **`skrivUtStatistikk(FilmarkivADT film)`**

## Oppgave 4

a) Ingen innlevering på pkt a), men på pkt b)

**Ref:**
https://junit.org/junit5/docs/current/user-guide/

http://junit.sourceforge.net/javadoc/org/junit/Assert.html

## Innledning

Les forelesningen om testing. Hent ned prosjektene **`TestKalkulator`** og **`TestKarakter`** som er lagt ut på GitHub. Se godt på testene. Kjør testene. Observer. Du får «**grønt**» på alle testmetodene for

**`TestKalkulator`**, men **ikke** for **`TestKarakter`**. Rett opp feilen og se at du får «grønt» på alle testmetodene for **`TestKarakter`**. Testklassen skal ligge i en annen "kode (source)"-mappe enn kildekoden for klassen som skal testes, men skal ha `samme pakkestruktur, no.hvl.dat102.`

Hent ned prosjektet Stabel fra GitHub. Sjekk ut at alle metodene for de to implementasjonene (klassene) er kodet ferdig. Hvis ikke, så må du fylle ut det som mangler. Stå i prosjektmappen. Trykk høyre musetast. Velg Run As og velg JUnit Test. Se om du får grønt på alle metodene (når du trykker på de 2 testklassene fra JUnit-vinduet).

Se på de 2 test-metodene under. Hva uttrykker de?

i)

```
@Test
public void pushPopErTom() {
	try {
		stabel.push(e0);
		stabel.pop();
		assertTrue(stabel.erTom());

	} catch (EmptyCollectionException e) {
		fail("push eller pop feilet uventet " + e.getMessage());
	}
}
```

ii)

```
@Test
public void popFromEmptyIsUnderflowed() {
	Assertions.assertThrows(EmptyCollectionException.class, () -> {
		stabel.pop();
	});
}
```

Lenker
https://howtodoinjava.com/junit5/expected-exception-example
https://junit.org/junit5/docs/5.3.0/api/org/junit/jupiter/api/Assertions.html

b) **Innlevering av prosjektet KoeT med de nye testene du legger til.**

Hent ned prosjektet KoeT fra GitHub. Sjekk ut alle metodene for de 2 implementasjonene (klassene) er kodet ferdig. Hvis ikke, så må du fylle ut det som mangler.

Du skal lage en mappe Test. Se på prosjektet Stabel og følg det opplegget som gjelder organisering av katalogen og mappestruktur. Du kan ikke kopiere testmetodene for stabel (de vil oppføre seg helt annerledes enn for kø). Det er derfor viktig at du forstår hvordan køen oppfører seg. Bruk navn som er assosiert til kø i programkoden. Når du har laget alt inkludert testmetodene, så kjører du testene og ser at det fungerer. Har du husket å teste alle metodene for ulike tilfeller?

c) Frivillig
Vi har ikke har testet at utvidelsen fungerer for tabell-implementasjonen, altså når tabellen blir full (Det samme manglet i prosjektet Stabel). Lag en testmetode, `utviderSeg()` som tester om utvidelsen av tabellen fungerer.

Tips. Metoden kan ha en løkke der den legger et bestemt element inn i køen for eksempel. SIZE + 2 ganger, der SIZE er størrelsen på tabellen når den opprettes første gang. Deretter tar metoden ut av køen like mange ganger og til slutt sjekker den at tabellen er tom.

Metoden skal nå bare ligge i test-klassen for TabellSirkulaerKoe (som arver fra KoeADTTest). Du må endre synligheten fra private til protected for køen og testobjektene (altså for alle medlemsdata, men det gir Eclipse beskjed om). Kjør JUnit og se om testene fungerer.

==========

**Legge til JUnit i prosjektet (for å kunne kjøre JUnit – hvis den ikke ligger der allerede):**

- Stå i prosjektmappen. Trykk høyre musetast.
- Velg Build Path | Configure Build Path
- Velg Add Library.
- Da skal JUnit komme opp i listen. Legg den til.

**Litt om feilsøking**

Når du får kjørefeil og ikke vet hvor i koden feilen er, så bruk bruddpunkt (breakpoint). Da vil kjøringen stanse der du setter bruddpunktet (et sted i koden der du har mistanke om feil). Hvis du ikke har fått feil inntil bruddpunktet, så setter du nytt bruddpunkt videre i koden. På denne måten kan du lettere innsnevre det området der feil er.

http://www.vogella.com/tutorials/EclipseDebugging/article.html

I Eclipse, sett markøren helt til venstre på den linjen du vil ha bruddpunkt. Trykk venstre musetast. Velg Toggle breakpoint. Får blå kule. Velg Debug i Run-menyen. Svar ja på å skifte perspektiv. Du kan nå følge med på innholdet av variabler. Kjør single steg ved å velge Step over i Run-menyen. Prøv de andre valgene. Til slutt kan du fjerne bruddpunkt(ene) ved Remove All Breakpoints. Trykk Terminate for å avslutte. Lukk –debug –view. Gå til window-menyen og velg perspective | Java default.