

# Mappeoppgave

## IDATx1003 Programmering 1 - Del 2

Høst 2023

## Innhold

Dette dokumentet inneholder 4 deler:

1. Noen avklaringer/opklaringer fra Del 1
2. Sjekkpunkter fra Del 1
3. Hva som skal gjøres i del 2 av 3
4. Viktige sjekkpunkter

## Noen avklaringer fra Del 1

Her følger noen oppklaringer/avklaringer fra Del 1 etter innkomne kommentarer/spørsmål fra dere studenter:

## Unikt tognummer

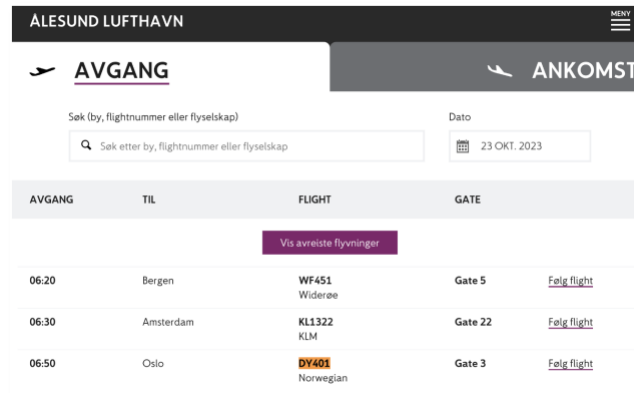
Tognummeret skal ikke forveksles med f.eks. registreringsnummer på bil. M.a.o. det er ikke et nummer som er unikt for akkurat det togsettet (lokomotiv + vogner), men for den spesifikke **togavgangen** som går på det spesifikke tidspunktet på den spesifikke strekningen. Vy opererer f.eks. med tognummer 61, 63, 601, 65, 67 og 605 på strekningen Oslo - Bergen. Motsatt vei (Bergen - Oslo) er tognumrene snarlige, men partall (62, 602 osv).

<b>F4 Oslo-Bergen</b>	
Tog nr	61 63 601 65 65 67 605
07.08.2023-09.12.2023	Δ Δ X Δ Δ X Δ Δ X Δ Δ X Δ Δ X Δ Δ X
Periode	1.5-30.9. 1.10-8.12. K =
Mandag-Fredag	M-F M-F M-F F M-F M-F
Lørdag	L L L L L L
Søndag	S S S S S S
Oslo S	0625 0825 1203 1425 1425 1625 2303
Sandvika	0639p 0839p 1218p 1439p 1439p 1639p 2318p
Asker	0646p 0846p 1224p 1446p 1446p 1646p 2325p
Drammen	0700p 0900p 1238p 1500p 1500p 1700p 2338p

<b>F4 Bergen-Oslo</b>	
Tog nr	62 602 64 66 66 66 606
07.08.2023-09.12.2023	Δ Δ X Δ Δ X Δ Δ X Δ Δ X Δ Δ X Δ Δ X
Periode	1.10-8.12. 1.5-30.9. K =
Mandag-Fredag	M-F M-F M-F F M-F M-F
Lørdag	L L L L L L
Søndag	S S S S S S
Bergen	0814 1143 1541 1645 1645 1645 2300
Arna	0823p 1152p 1550p 1654p 1654p 1654p 2310p
Dale	1230           2344
Voss	0922 1301 1657 1807 1807 1807 0028
Myrdal	1002 1350 1738 1848 1848 1848 0112

Dette tilsvarer egentlig samme system som for flyavganger:



AVGANG	TIL	FLIGHT	GATE
06:20	Bergen	WF451 Widerøe	Gate 5
06:30	Amsterdam	KL1322 KLM	Gate 22
06:50	Oslo	DY401 Norwegian	Gate 3

Her vist ved eksempel på avgang fra Ålesund lufthavn: WF451, KL1322, DY401 osv.

Det må være mulig for brukeren av applikasjonen din å oppgi dette unike tognummeret, så ikke la applikasjonen din automatisk generere det unike tognummeret

## Linje

I oppgaven for Del 1 står det at linje er "*en tekst på formen "L1", "F4" osv*". Det betyr ikke at **linje** skal bestå av en bokstav etterfulgt av et tall, men at linje **kan** være kombinasjoner av bokstaver og tall. Noen av dere har begynt å se på bruken av *regulære uttrykk (regular expressions)* for å verifisere at verdien til **linje** følger akkurat mønsteret med "bokstav etterfulgt av tall". Ikke gjør det!

## Sjekkpunkter fra Del 1

Fikk du gjort følgende i Del 1?:

1. Satt opp prosjektet riktig, på et egnet sted på din harddisk slik at du har full kontroll på hvor prosjektet befinner seg på din datamaskin?
2. Lagt prosjektet ditt til **versjonskontroll** på GitHub/GitLab ?
3. Implementert **entitetsklassen** for togavgang med alle felt, gode konstruktør(er), aksessor- og eventuelle mutator-metoder? Og med god **dokumentasjon** av både klasse- og **samtlig** public metoder iht. JavaDoc standarden? M.a.o. gir CheckStyle feilmeldinger?
4. Opprettet **enhetstest-klasse** for entitetsklassen, der du har laget gode **positive- og negative tester**? Husk at testdekning (test coverage) på 100% IKKE er et kvalitetsmål for testene dine overhodet. Det er kun en eventuell verifisering for å hjelpe deg til å avdekke hvor stor del av databasen din som blir testet av testene dine (hvilket også er viktig), men den sier **ingenting** om hvor gode testene er (kvalitet).

Hvis ikke: IKKE gå videre til Del 2 før dette er på plass!

## Mappe del 2 (av 3)

Del 2 av mappen bygger videre på Del 1.

Basert på tilbakemeldingen du har fått på del 1: gjennomfør nødvendig **refaktorisering** (eng: **refactoring**) av koden din fra del 1.

I denne delen av mappen skal du implementere **registeret** som skal holde på alle **togavgangene**, samt en tilhørende **enhetstest-klasse** som tester registerklassen både med negative og positive tester.

## Register over togavganger

Implementer en klasse som er ansvarlig for å holde på en samling av **togavganger** med tilhørende funksjonalitet.

Du skal selv:

- Velge navn på klassen som skal representere registeret.
- Vurdere hvilken klasse fra Java SDK du tenker er passende å bruke for å lagre alle togavgangene i (ArrayList, HashSet, HashMap osv). Husk å begrunn i **rapporten** hvorfor du valgte nettopp denne klassen fra SDK'en.

Fra listen over **funksjonelle krav** fra kravspesifikasjonen i Del 1, bør register-klassen som minimum ha støtte for følgende funksjonalitet:

- En metode for å legge til en togavgang i registeret. Dersom det allerede finnes en togavgang med samme tog-nummer som den avgangen som forsøkes lagt til, skal togavgangen ikke legges til registeret og metoden bør gi en eller annen form for tilbakemelding (NB! IKKE til brukeren) til koden som kaller metoden.
- En metode for å søke opp en togavgang basert på det unike tog-nummeret.
- En metode for å søke opp togavgang(er) med en gitt **destinasjon**.
- En metode som fjerner togavgang(er) som har avreisetidspunkt tidligere enn et gitt klokkeslett. Husk å også ta med forsinkelse i vurderingen om avgangen skal fjernes.
- En metode som returnerer alle togavganger som en **sortert liste** av togavganger sortert stigende på avreisetidspunkt. Her skal du ikke ta hensyn til eventuell forsinkelse. Merk at vi ikke ber om at registeret i seg selv nødvendigvis må være sortert til enhver tid, men at denne metoden skal returnere en sortert samling (eller en iterator til en sortert samling) basert på togavgangene i registeret ditt.

**NB! Beskriv i rapporten løsningen du valgte å implementere her. Beskriv hvordan du kom frem til løsningen (søkte på nett, ChatGPT, GitHub CoPilot, spurte en venn osv), og forklar med egne ord koden du har implementert (dette for å vise at du forstår koden du har implementert).**

**Dersom kode er inspirert av en løsning fra andre (f.eks, nettside) er det viktig at kilde skal henvises og begrunnes.**

## Enhetstest av registerklassen

Opprett enhets-tester for å teste registerklassen. Husk at det er smart å planlegge på forhånd **hva** du skal teste i registerklassen og **hvordan**, og spesielt hvordan du skal teste for å få utført **negativ** testing. Dette er det lurt å skrive i JavaDoc'en til testklassen **FØR** du begynner å kode testene.

Husk både **positiv(e)** og **negativ(e)** tester. Kontroller gjerne dekningsgrad (test-coverage) for å sjekke at mest mulig av koden i register-klassen dekkes av testene.

## Oppdatere applikasjonsklassen

Oppdater **init()** og **start()** metodene i klassen som representerer applikasjonen (og på sikt vil representere det tekstbaserte brukergrensesnittet):

- **init()**: Her legger du inn all kode som er nødvendig for å **initialisere** applikasjonen ved oppstart, som f.eks. å opprette instansen av register-klassen.
- **start()**: Oppdater denne metoden til å ta i bruk den nye register klassen. Opprett 3-4 togavganger som du legger til i registeret. Test deretter noe av funksjonaliteten til registeret. Du kan f.eks. allerede nå tenke på å implementere en metode som skriver ut **informasjonstavla** til konsollet basert på registrerte togavganger. Det er lurt å lage en egen metode for å skrive ut denne oversikten. Vent med å implementere full meny med input fra brukeren. Dette kommer i del 3 😊

# Viktige sjekkpunkter

Når du løser oppgaven, bør du dobbeltsjekke følgende:

- Kompilering/bygging og prosjektstruktur:
  - Er prosjektet et Maven-prosjekt med en ryddig og riktig katalogstruktur?
  - Kan prosjektet **kompile** uten feil (*mvn compile*)?
  - Kan samtlige enhets-tester i prosjektet kjøres uten at de feiler?
- Versjonskontroll med git:
  - Er prosjektet underlagt versjonskontroll med sentralt repository i GitHub eller GitLab?
  - Finnes det flere innsjekkinger (commits) ?
  - Beskriver commit-meldingene endringene på en kort og konsis måte?
- Enhetstester:
  - Har enhetstestene beskrivende navn som dokumenterer hva testene gjør?
  - Følger de mønstret Arrange-Act-Assert?
  - Tas det hensyn til både positive og negative tilfeller?
  - Er testdekningen god nok?
  - Kjører samtlige tester uten feil?
- Er klassene for togavgang, register og applikasjon implementert iht oppgavebeskrivelsen?
- Kodekvalitet:
  - Er koden godt dokumentert iht JavaDoc-standard, og skrevet i henhold til Google sin kodelstil? (Tips: bruk CheckStyle)
  - Er koden robust (validering mm)?
  - Har variabler, metoder og klasser beskrivende navn?
- Er klassene gruppert i en logisk pakkestruktur?