

IN1010 Objektorientert programmering – eksamen våren 2023

(utsatt/ny eksamen)

Oppgavetekst fra UiOs eksamen våren 2023 (utsatt/ny eksamen), IN1010 (overskriften er eksamenssett.no sin; utsnittene under er klippet uendret ut av UiOs publiserte dokument)

Utsatt/ny eksamen IN1010 våren 2023

I dette oppgavesettet skal du ikke bruke andre klasser fra Java-biblioteket enn de som alltid følger med i `java.lang`, med unntak av klassen `NoSuchElementException` og det du trenger for å lage en iterator. På eksamen trenger du ikke skrive import-setninger for disse.

Pass på at du bare svarer på det hver enkelt oppgave ber om. Du skal ikke programmere et fullstendig program med en `main`-metode før på slutten av oppgavesettet (oppgave 9). Du kan gjerne bruke norske bokstaver. I noen oppgaver er det presisert hvilke læringsmål du skal vise du har nådd, men i de fleste oppgavene er dette opplagt.

Selvkjørende biler i kolonner

For å øke veikapasiteten og også sikkerheten til selvkjørende biler er det blitt foreslått at slike biler skal kjøre tett etter hverandre og på rekke i en kolonne (på engelsk: *platoon* eller *convoy*). I programmet ditt er alle biler selvkjørende, kan kjøre i kolonne og har egenskapene til klassen `Bil`. Enhver bil i programmet ditt er enten en personbil eller en lastebil. En kolonne består enten av bare personbiler eller av bare lastebiler. Kun spesielle biler kan kjøre først i en kolonne og lede denne, såkalte *lederbiler*. Denne egenskapen skal du beskrive ved hjelp av et interface du kaller `Lederbil`.

Oppgave 1 Tegning av klassehierarki (5 poeng)

- ⇒ Tegn et klassehierarki som kan representere ulike typer biler som beskrevet over. Begrunn valgene dine kort som kommentarer i programmet du skriver i oppgave 2.

Oppgave 2 Programmering av klassehierarki (15 poeng)

Bilene i en kolonne skal kunne hektes sammen i en dobbeltlenket liste ved hjelp av instansvariabler i bil-objektene. Hver bil skal også inneholde en referanse til den kolonnen den kjører i. Denne referansen skal være **null** når bilen ikke er i noen kolonne.

Enhver bil har en maksimal fart. Enhver lastebil har en maksimal lastevekt og personbiler har et maksimalt antall passasjerer. Disse instansvariablene skal initieres av klassenes konstruktører. Lederbiler har en egenskap kalt egnethet som er et heltall. Lederbilenes egnethet skal initieres av konstruktørene.

⇒ Programmer klassene og interfacet du tegnet i oppgave 1

Oppgave 3 Klassen Kolonne (20 poeng)

I denne oppgaven skal du skrive en klasse Kolonne. Objekter av denne klassen skal inneholde pekere til første og siste bil i en dobbeltkjedet liste av biler i kolonnen. Du skal bruke instansvariablene i bil-objektene til å lage denne listen. Metodene som setter inn og tar ut (dvs. fjerner) biler fra kolonnen, må sørge for at det alltid er en lederbil først i kolonnen (men det kan gjerne være én eller flere lederbiler inne i kolonnen også). I klassen Kolonne må programmet ditt sørge for at det

enten bare er lastebiler eller bare er personbiler i kolonnen, og dette bestemmes av den første bilen som settes inn.

Klassen Kolonne skal inneholde metoden settInn som skal sette inn bilen som refereres av parameteren på slutten av kolonnen, og metoden taUt for å ta ut (fjerne) en bil (referert av parameteren) av kolonnen. Metoden taUt må sjekke at bilen virkelig er i denne kolonnen. Metoden skal ikke lete etter bilen i kolonnen. Hvis taUt prøver å ta ut den første bilen i kolonnen, skal dette bare gå bra om også den neste bilen i kolonnen er en lederbil (eller kolonnen blir tom).

Deklarer en unntaksklasse som kastes når det settes inn eller tas ut en bil som gjør at det ikke blir en lederbil først i kolonnen, når en lastebil settes inn i en personbilkolonne (eller omvendt) eller bilen som skal ut, ikke er i kolonnen. Bruke en String-parameter til unntaksklassen for å skille de forskjellige unntakssituasjonene.

⇒ Skriv klassen Kolonne med de to metodene settInn og taUt. Skriv også unntaksklassen som beskrevet over.

Oppgave 4 Iterator (15 poeng)

- ⇒ Skriv en iterator i klassen Kolonne som itererer over alle bilene i kolonnen *som har leder-egenskaper*. Hvis next-metoden i iteratoren blir bedt om å returnere en bil og det ikke er flere lederbiler igjen å iterere over, skal det kastes et unntak av typen NoSuchElementException. I denne oppgaven skal du vise at du skjønner hvordan en iterator skal programmeres når ikke alle objektene i beholderen skal tas med.

Oppgave 5 Rekursiv metode (10 poeng)

En kolonne kan ikke kjøre raskere enn bilen med lavest maksimalfart.

- ⇒ Skriv metoden finnMaksfartR i klassen Kolonne som finner kolonnens maksimale fart ved å kalle en rekursiv metode i klassen Bil. Skriv også den rekursive metoden i klassen Bil.

Oppgave 6 Beste lederbiler (15 poeng)

I denne oppgaven skal du skrive en metode finnBesteEgnet i klassen Kolonne som returnerer en array med referanser til de 5 lederbilene i kolonnen som har størst (best) egnethet. Om det er færre enn 5 lederbiler i kolonnen, skal arrayen fylles opp med **null**-verdier. Husk at du ikke skal bruke klasser fra Java-biblioteket (annet enn de som er nevnt først i oppgaveteksten). Du kan bruke iteratoren beskrevet i oppgave 4 selv om du ikke har programmert denne. I denne oppgaven skal du vise at du behersker å bruke arrayer og kan programmere en noe mer kompleks algoritme.

- ⇒ Skriv metoden med signaturen Lederbil [] finnBesteEgnet () i klassen Kolonne.

Oppgave 7 Datastrukturtegning (5 poeng)

I denne oppgaven skal du tegne en datastruktur basert på klassen Kolonne i oppgave 3 og metoden finnBesteEgnet i oppgave 6. Anta at et objekt av klassen Kolonne inneholder fire lastebiler, og at både den første og den siste er lederbiler.

- ⇒ Tegn opp datastrukturen i dette Kolonne-objektet og tegn den arrayen det returneres en referanse til fra et kall på metoden finnBesteEgnet. Skriv på alle objektene hvilke klasser de er instanser av. Ikke tegn metoder. Skriv navn på variabler, men du trenger ikke skrive type og du trenger bare skrive/tegne innholdet i referansevariablene.

Oppgave 8 Generisk klasse (10 poeng)

Du skal nå skrive en ny klasse `KolonneG` som er en meget forenklet generisk versjon av klassen `Kolonne` slik at når det deklarerer et objekt av klassen `KolonneG`, bestemmes det om det skal være en kolonne av bare personbiler eller en kolonne av bare lastebiler. Klassen `KolonneG` skal ta vare på bilene på samme måte som klassen `Kolonne`, men for enkelhets skyld behøver det i klassen `KolonneG` ikke være en lederbil først i kolonnen og bil-objekter skal ikke peke på kolonnen sin. Du skal ikke gjøre forandringer i klassene i oppgave 2.

Klassen `KolonneG` skal ha to metoder:

- `settInn` skal legge inn en bil (referert av parameteren) bakerst i kolonnen.
- `taUtSiste` skal fjerne og returnere en referanse til den siste bilen i kolonnen. Hvis kolonnen er tom, returneres **null**.

Metodene skal ikke kaste noen unntak. I denne oppgaven er det viktig at du viser at du kan programmere en generisk klasse. Du kan legge mindre vekt på pekerlistehåndteringen (for det har du vist du kan i oppgave 3).

Tips: Ikke ta utgangspunkt i koden du skrev i oppgave 3 – programmer hele klassen på nytt.

⇒ Skriv klassen `KolonneG` med instansvariabler og de to metodene `settInn` og `taUtSiste`.

Oppgave 9 Bruk av generisk klasse (5 poeng)

I denne oppgaven skal du vise at du behersker å bruke en generisk klasse. Du skal skrive en klasse `Kolonnekjøring` som skal bruke klasser fra oppgave 2 og klassen `KolonneG` fra oppgave 8. Til sammen skal dette kunne virke som et fullstendig program. Klassen `Kolonnekjøring` skal inneholde en `main`-metode som

1. Oppretter en (tom) kolonne av lastebiler
2. Oppretter en lastbil som er en lederbil
3. Setter denne bilen inn i kolonnen
4. Tar den siste (og eneste) bilen ut av kolonnen og skriver ut denne lederbilen sin egnethet ved hjelp av `System.out.println()`.

⇒ Skriv klassen `Kolonnekjøring` med en `main`-metode som beskrevet over.