Eksamen IN1010, våren 2023

Oppgavetekst med sensorveiledning

I denne oppgaven skal du skrive deler av et lite program som behandler tog med vogner og lokomotiv.

Les gjennom hele oppgaven før du begynner på besvarelsen. Tegning av klassehierarkiet skal leveres i oppgave 1, samtlige andre svar (det vil si all programkode) skrives i svarfeltet for oppgave 2-5. Angi gjerne hvilken oppgave du svarer på som kommentarer i programkoden.

Du kan bruke norske bokstaver i denne besvarelsen.

Sensorveiledning:

Skrivefeil ignoreres vanligvis, men feil som kan tyde på manglende forståelse (for eksempel kopiering av kode som ikke passer inn eller svarer på oppgaven) skal gi trekk.

• Ignoreres (eksempler):

- *Mangler (noen) ';' , '}' eller '{' om koden ellers gir mening (NB)*
- *Småfeil i (gjenkjennbare) navn*
- *Blander metodenavn for tilsvarende operasjon i ulike beholdere*

• Kan gi (noe) trekk

- *Bruker beholder på en måte som tyder på manglende forståelse*
- *Gjennomgående feil/ mangler*

Studentene har fått beskjed om at import-setninger, kommentarer, eller tilgangsmodifikatorer (public/ private/ protected) ikke er viktige på eksamen.

Oppgave 1. Klassehierarki for skinnegående materiell (5 poeng)

Alle vogner og annet materiell som kan kjøre på en skinnegang, representeres av klassen **Skinnegående** som har følgende metoder:

hentId returnerer en String som identifiserer dette skinnegående objektet.

hentSporvidde returnerer et heltall som angir hvor brede spor det kan kjøre på.

Sporvidden og identifikatoren skal være parametere til klassens konstruktør. Skinnegående objekter skal kunne hektes sammen i en dobbeltlenket liste i en senere oppgave, så du skal også ha med instansvariabler i klassen som gjør dette mulig.

I dette eksamenssettet er alt skinnegående materiell enten av klassen **Lokomotiv** eller klassen **Vogn**. Alle Vogn-objekter er enten av klassen **Godsvogn** eller klassen **Passasjervogn**.

Skinnegående materiell kan implementere et interface **Motordrevet** med følgende grensesnitt:

fossilt som returnerer true dersom fossilt brennstoff (bensin, diesel, kull, ved) benyttes, ellers false.
trekkraft som returnerer antall kW (kilowatt) som et heltall.

Alle vogner har en lengde angitt som heltall i centimeter. Godsvogner har en maks tillatt lastevekt, angitt som et desimaltall i kilogram, og passasjervogner har et heltall som representerer maks antall passasjerer det er plass til i vognen. Disse instansvariablene skal være parametere til konstruktørene til klassene.

- ⇨ Tegn et klassehierarki for situasjonen beskrevet over.

Sensorveiledning:

I denne oppgaven skal studenten vise at hen forstår og kan beskrive et klassehierarki. Klassehierarkiet må inneholde den abstrakte klassen Skinnegående på toppen og de andre klassene under denne med piler opp til superklassen, se løsningsforslaget. Interfacet Motordrevet må være tegnet (som et interface) og Lokomotiv må ha en pil opp til dette interfacet. Også klassen Vogn bør være abstrakt. Men andre fornuftige hierarkier er også OK. Om kandidaten har valgt å la flere klasser (enn Lokomotiv) implementere interface Motordrevet, er dette OK. Grunnen til dette er at det ikke står eksplisitt i oppgaven at vogner ikke skal ha motor.

Oppgave 2. Programmere klasser og interface (15 poeng)

I denne oppgaven skal du programmere klasser og interface som beskrevet i oppgave 1.

- ⇨ Programmer klassene **Skinnegående, Lokomotiv, Vogn**, **Godsvogn**, **Passasjervogn** og interface-et **Motordrevet**.

Sensorveiledning:

Hierarki: ***I*** *denne oppgaven skal studenten vise at hen kan programmere enkle klasser og et interface. Skinnegående og Vogn bør være abstrakt. Extends og implements må være med på riktige måter, se løsningsforslaget (7 poeng)*

Konstruktører*: Instansvariabler skal deklareres og konstruktørene skal kalle på super() på riktig måte. Det er et pluss om konstante instansvariabler er deklarert som final (8 poeng).*

Oppgave 3. Klassen Tog (40 poeng)

I oppgave 3 skal du skrive en klasse **Tog** med variabler og metoder. Et tog er en dobbeltlenket liste av Skinnegående-objekter og du skal bruke instansvariablene i disse objektene til å lage denne listen. I denne oppgaven kan du aksessere instansvariabler i Skinnegående-klassen og dens subklasser.

3a) 1 poeng

- ⇨ Skriv konstruktør og instansvariabler for klassen Tog med pekere til første og siste Skinnegående-objekt i listen.

Sensorveiledning: Her skal studenten vise at hen kan deklarere en svært enkel datastruktur i en beholder som skal inneholde en dobbelt lenket liste. Konstruktør kan være utelatt eller tom, eller sette første- og sistepeker. Første og siste-pekerne skal være av typen Skinnegående.

3b) 4 poeng

⇨ Skriv metoden **leggTil** i klassen Tog som legger til et Skinnegående objekt bakerst i listen. Parameter til metoden er en referanse til objektet som skal legges til.

Sensorveiledning: Her skal studenten vise at hen behersker enkel manipulering av referanser i en lenket liste. Det er viktig at programmet ikke leter gjennom listen, men bruker siste-pekeren til å sette inn sist.

3c) 7 poeng

⇨ Skriv metoden **taUt** i klassen Tog som tar ut et element i listen. Parameter til metoden er en referanse til objektet som skal tas ut. Du kan anta at objektet som tas ut, ligger i toget. Metoden skal returnere en referanse til det objektet som tas ut.

Sensorveiledning: Her skal studenten vise at hen behersker noe vanskeligere manipulering av referanser i en dobbeltlenket liste. Programmet må ta hensyn til alle spesialtilfeller (se løsningsforslaget). Det er viktig at dette gjøres riktig for denne listen av skinnegående objekter og bruker referansene som ligger i disse objektene (av klassen Skinnegående) til forrige og neste objekt i listen. Det skal ikke brukes Node-objekter. Løsninger som bærer preg av å være hentet fra andre programmer uten at studenten viser at hen kan og skjønner manipulering av referanser i en liste, gir ikke uttelling.

3d) 7 poeng

⇨ Skriv metoden **finnOgTaUt** i klassen Tog som finner et element i listen og tar det ut. Parameter til metoden er en String som er id-en til det Skinnegående-objektet som skal tas ut (du kan anta at id-en er unik). Metoden skal returnere en referanse til det objektet som tas ut.

Sensorveiledning: Her skal studenten også vise at hen behersker vanskelig manipulering av referanser i en dobbeltlenket liste. Det er også i denne oppgaven viktig at programmet viser at studenten skjønner hvordan dette skal gjøres for akkurat denne listen. Besvarelsen må inneholde en while-løkke som terminerer når programmet har funnet objektet som skal tas ut, og om et slikt ikke finnes, terminerer når listen er ferdig gjennomløpt. Det er mulig å bruke iteratoren som defineres i oppgave 3f), men dette gir bare et lite pluss om det er gjort riktig. Når riktig objekt er funnet, bør metoden kalle taUt-metoden fra oppgave 3c).

3e) 7 poeng

⇨ Skriv metoden **leggTilForan** i klassen Tog som har to parametere av typen Skinnegående. Den første parameteren er en referanse til et objekt du kan anta allerede ligger i toget, den andre er en referanse til et objekt som skal settes inn i listen foran objektet som refereres av den første parameteren.

Sensorveiledning: Dette er den siste oppgaven der studenten skal vise at hen behersker manipulering av referanser i en dobbelt lenket liste. Det er også i denne oppgaven viktig at programmet viser at studenten skjønner hvordan dette skal gjøres for akkurat denne listen. Besvarelsen må ta for seg spesialtilfellet at objektet skal legges til først i listen. Det er også her viktig at programmet ikke leter gjennom toget, men bare setter det nye objektet inn på riktig plass.

3f) 7 poeng

⇨ Skriv en iterator over alle Skinnegående-objektene i et Tog.

Sensorveiledning: Her skal studenten vise at hen kan skrive en standard iterator over en lenket liste. Siden det har vært undervist akkurat slike iteratorer i løpet av semesteret er det viktig at det gjøres riktig, at studenten viser at hen har skjønt hvordan det gjøres og har tilpasset iteratoren til akkurat denne listen som utgjør et tog. Selve Tog-klassen må implementere `Iterable<Skinnegående>`*, det må finnes en metode* `Iterator<Skinnegående> iterator ()` *som lager et nytt Iterator-objet og det må være deklarert en klasse som implementerer* `Iterator<Skinnegående>`*, se løsningsforslaget. I den sistnevnte klassen må metodene hasNext() og next() være deklarert og programmert riktig.*

3g) 7 poeng

⇨ Skriv metoden **Passasjervogn[] hentPassasjervogner ( )** i klassen Tog som returnerer en array med referanser til alle Passasjervogn-objekter i toget.
Hint: Tell antall passasjervogner før du oppretter arrayen.

Sensorveiledning: Her skal studenten vise at hen kan opprette og legge inn referanser i en array og returnere denne fra metoden. I tillegg skal studenten vise at hen kan undersøke om et objekt har visse egenskaper (ved bruk av instanceof) og kan typekonvertere (caste) (fra Skinnegaende til Passasjervogn), se løsningsforslaget.

Oppgave 4. Sporvidde (20 poeng)

Kun vogner og lokomotiv med samme sporvidde kan brukes sammen i et tog. Dersom sporvidden ikke er den samme for alle vognene og lokomotivene, så skal et unntak ("Exception") av typen **FeilSporvidde** kastes. I oppgave 4 skal du skrive kode for ulike måter å kontrollere dette på.

4a) 1 poeng

⇨ Skriv unntaket FeilSporvidde.

Sensorveiledning: Her skal studenten vise at hen kan skrive en klasse som er en subklasse av Exception eller RunTimeException.

4b) 7 poeng

Vi ønsker å kunne sjekke at alle objektene i et tog har samme sporvidde etter at de er lagt inn. Skriv en iterativ (ikke rekursiv før i 4d) metode **sjekkSporvidde** som går gjennom hele toget og sjekker at alle vognene har samme sporvidde. Hvis programmet oppdager et objekt med en annen sporvidde, skal unntaket du laget i oppgave 4a kastes.

⇨ Skriv metoden sjekkSporvidde i klassen Tog.

Sensorveiledning: Her skal studenten vise at hen kan skrive en metode som kaster et unntak om et unormalt objekt oppdages i listen. Metoden må spesialbehandle tilfellet at listen er tom. Om den ikke er tom, må ett av objektene anses som normalt (med sin sporvidde) (f.eks. det første objektet i listen) og listen løpes gjennom, f.eks. med iteratoren definert i oppgave 3f. Om et unormalt objekt (annen sporvidde) oppdages, kastes et FeilSporvidde-unntak. Om alle objekter har samme sporvidde, returnerer metoden normalt.

NB: Oppgaven skriver et sted vogn *i stedet for* objekt*. Om noen har lagt inn ekstra kode for dette, er det selvsagt OK.*

4c) 4 poeng

Bruk unntaket du laget i oppgave 4a) til å lage en **leggTilSikker**-metode som setter inn et nytt objekt bakerst i listen (slik som i oppgave 3b) og som kaster et unntak når en vogn eller et lokomotiv med en annen sporvidde enn de som allerede er i toget, blir forsøkt lagt til.

⇨ Skriv metoden leggTilSikker i klassen Tog

Sensorveiledning: I denne oppgaven skal studenten vise at hen kan skrive en metode som er en litt mer kompleks versjon av metoden i oppgave 3b. Det må sjekkes om listen er tom, og hvis ikke at det nye objektet har samme sporvidde som et av dem som allerede er i toget, f.eks. det første. Er dette i orden bør objektet settes inn med et kall på leggTil() fra oppgave 3b. Om det ikke er i orden, må programmet kaste et FeilSporvidde-unntak.

4d) 8 poeng

⇨ Skriv en rekursiv løsning for å sjekke at alle objektene i et tog har samme sporvidde, inkludert eventuelle endringer eller tillegg du vil gjøre i svar på tidligere oppgaver.

Sensorveiledning: I denne oppgaven skal studenten vise at hen kan skrive en rekursiv metode i en liste og kalle på denne metoden i det første (eller det andre) objektet i listen. Dette kan gjøres på svært mange måter, f.eks. slik det er gjort i løsningsforslaget ved å sjekke at to og to objekter som ligger etter hverandre i listen har samme sporvidde. Det kan f.eks. også gjøres ved å finne sporvidden til det første objektet i listen, og så bruke dette som parameter til den rekursive metoden som sjekker resten av listen. Om den rekursive metoden finner forskjellige sporvidder kastes et unntak. Om rekursjonen når slutten av listen uten å finne noe feil, returnerer metoden normalt. Alle rekursive løsninger gir uttelling, og om de er riktig og oversiktlig skrevet, gir de full uttelling.

Oppgave 5. Tråder (20 poeng)

I oppgave 5 skal du skrive én trådklasse som heter **Leter** og én som heter **Resultat**. Begge disse skal implementere Runnable. Du skal også skrive en klasse **Monitor** som skal være en monitor.

5 a) 5 poeng

Leter-trådene skal lete gjennom hvert sitt tog. De skal lete etter alle skinnegående objekter der Id-en begynner med en bestemt tekststreng. Hver gang en Leter-tråd har funnet et skinnegående objekt som passer, skal den legge en referanse til dette inn i monitoren ved å kalle metoden **leggTil** i monitoren. Når en Leter-tråd har lett gjennom hele sitt tog, skal den si fra til monitoren om dette ved å kalle metoden **ferdigLeting** i monitoren og så terminere.

Som parameter til Leter-klassens konstruktør skal du ha en referanse til toget tråden skal lete i, en referanse til monitoren og en referanse til String-objektet det skal letes etter.

Hint: Metoden startsWith(String s) i String-klassen returnerer true hvis String-objektet den kalles på, begynner med s.

⇨ Skriv trådklassen Leter

Sensorveiledning: I denne oppgaven skal studenten vise at hen kan skrive en trådklasse med tre instansvariabler som initialiseres av konstruktøren. Tråden må ha en run-metode som leter gjennom sitt tog ved å bruke togets iterator. Hver gang et objekt med riktig id finnes må dette legges inn i monitore

med et kall på leggTil(). Når run-metoden har iterert gjennom hele toget kaller den ferdigLeting() i monitoren før den terminerer.

5 b) 3 poeng

Det skal bare være én instans av tråden Resultat. Resultat-tråden skal hente Skinnegående-objekter fra monitoren ved å kalle metoden **hentNeste** i monitoren og skrive ut Id-en i terminalvinduet ved hjelp av System.out.println. Hvis hentNeste returnerer null er det ikke flere Skinnegående- objekter igjen å skrive ut, og tråden kan terminere. Resultat-tråden skal kunne startes opp samtidig med Leter-trådene og således kunne utføres i parallell med Leter-trådene.

Resultat-klassens konstruktør skal bare ha en referanse til monitoren som parameter

⇨ Skriv trådklassen Resultat

Sensorveiledning: I denne oppgaven skal studenten vise at hen kan skrive en trådklasse med en instansvariabel som initialiseres av konstruktøren. Klassens run-metode skal hente referanser til Skinnegående objekter fra monitoren helt til verdien null returneres. Når et objekt er hentet skal id-en skrives ut med System.out.println(). Når null returneres skal run-metoden terminere.

5 c) 12 poeng

Monitoren skal inneholde de tre metodene leggTil, ferdigLeting og hentNeste som beskrevet over. Som parameter til monitorens konstruktør skal du bare ha hvor mange Leter-tråder som finnes.

⇨ Skriv Monitor-klassen

Sensorveiledning:

***Låsing**: I denne oppgaven skal studenten vise at hen kan programmere en monitor med tre metoder og en betingelse. Monitoren må inneholde en beholder som kan ta vare på referanser til Skinnegående objekter, f.eks. en ArrayListe. Låsen og Condition-variabelen må være deklarert på riktig måte og alle metodene må låses og låses opp med låsen. Studentene har sett monitorer svært like denne i løpet av semesteret, så dette bør være rimelig riktig for å få god uttelling (5 poeng).*

***Ventekø**: Metoden hentNeste må inneholde et kall på await() i en while-løkke når beholderen er tom og det er mer å gjøre. Riktig eller nesten riktig while-test gir god uttelling. Kall på signal() må finnes i begge de to andre metodene. Metoden hentNeste må returnerer neste objekt i beholderen eller null når jobben er ferdig (7 poeng).*

(Du skal altså ikke skrive noe fullstendig program som starter opp noen tråder – du skal bare skrive disse tre klassene med metoder og variabler.)