# Eksamensinfo

# Utsatt og ny eksamen

# IN1010/INF1010 våren 2019

## PRAKTISK INFORMASJON

Hvis du synes noen deler av oppgaven er uklare, kan du legge dine egne forutsetninger til grunn og gjøre dine egne rimelige antagelser. Gjør i så fall rede for disse forutsetningene og antagelsene.

Les alle deloppgavene før du begynner å besvare dem. Det er informasjon i senere deloppgaver som kan brukes i besvarelsen av tidligere oppgaver.

Unngå å bruke en stor del av tiden din på oppgaver som gir deg få poeng.

En faglærer vil besøke eksamenslokalet for å oppklare eventuelle uklarheter og feil i oppgaveteksten omlag en time etter at eksamen har startet.

## Tillatte hjelpemidler

Alle trykte og skrevne hjelpemidler er tillatt. En enkel kalkulator finnes tilgjengelig i Inspera.

## Tegning på papir

I oppgave 1 og 9 skal svaret tegnes på papir (skisseark). Instruksjon om utfylling av skisseark finner du nederst på skjermen (ved siden av lenke til kalkulatoren). Husk å notere oppgavekoder, kandidatnummer og annen informasjon med én gang; Det blir IKKE gitt ekstratid for å fylle ut dette.

Om du har behov for å tegne andre datastrukturer (tilhørende andre oppgaver enn nr 1 og 9) for å vise sensor hvordan du tenker når du skriver algoritmer, så tegner du dette etter svaret på oppgave 1, skriver tydelig på tegningen hvilken/hvilke oppgave(r) tegningen tilhører, og leverer alt som svar på oppgave 1.

## Blandede tekst/kodesvar

I noen oppgaver kan du både skrive tekst og program. Du kan da skrive teksten inn i «programboksen» uten å passe på riktig Java-syntaks, dvs du behøver ikke kommentere ut teksten du skriver.

Norge er en stolt skipsfartsnasjon, og du søker om jobb i rederiet *Norgeslinjen*.

Som en del av intervjuprosessen skal du lage noen programdeler for innredning av båtene i rederiet. Det du lager i dette oppgavesettet, skal ikke være et fullstendig program. Ikke programmer mer enn det du blir uttrykkelig bedt om i oppgaveteksten. Les gjennom hele oppgaveteksten før du begynner å løse noen deler.

For alle metoder og variabler du deklarerer er det viktig at det klart fremgår av besvarelsen din hvilken klasse de hører hjemme i. Når du blir bedt om å skrive programbiter på nytt, er det også viktig at du markerer hva som er ny kode og hva som er gammel kode.

Det du lager skal kunne brukes til å innrede mange skip slik at det du programmerer må kunne brukes i mange båter.

Alle gjenstander om bord i kahytter og salonger beskrives av klassen Inventar. Ethvert inventar er kjennetegnet av en tekst (en String) og et heltall, henholdsvis navnet på inventaret og verdien. I tillegg har alt inventar en metode kalt finnForsikringsverdi. Fordi vi senere skal sette inventar inn i en liste, inneholder alle Inventar-objekter en referanse som kan peke på annet inventar og den har navnet neste. Alle Inventar-objekter skal kunne sammenlignes på verdi, og klassen Inventar implementerer derfor interfacet Comparable.

Noe inventar er leid ut fra et utleieselskap. Dette inventaret er i tillegg til annet inventar kjennetegnet av navnet på utleieselskapet (en String) og beskrives av klassen UtleieInventar. Annet inventar eies av rederiet selv og kalles *eget inventar*. Slikt inventar er i tillegg til annet inventar kjennetegnet av et innkjøpsår og beskrives av klassen EgetInventar. Metoden finnForsikringsverdi i UtleieInventar returnerer 100 % av inventarets verdi, mens metoden finnForsikringsverdi i EgetInventar gir 80 % av verdien.

Noe som henger på veggen kaller vi et *bilde*, og noe som står på gulvet kaller vi et *møbel*. Et bilde har en metode kalt bredde som forteller hvor mye veggplass det trenger. (Vi regner med at det bare går ett bilde i høyden; følgelig er det bare bildets bredde som er av betydning.) Et møbel har en metode kalt areal som forteller hvor mye gulvplass det trenger. Interfacene i programmet skal hete Bilde og Mobel.

I tillegg til de klassene som allerede er nevnt, har det fullstendige programmet klassene UtleieBilde, UtleieMobel, UtleieBildeMobel, EgetBilde, EgetMobel og EgetBildeMobel. Disse seks klassene beskriver eget inventar og utleieinventar som henger på vegg eller står på gulvet. Klassene UtleieBildeMobel og EgetBildeMobel beskriver inventar som må stå på gulvet inntil en vegg og som derfor har egenskapene til både et bilde og et møbel. (Merk at når et møbel står inntil en vegg, er det ikke plass til noe bilde der.) Alt inventar er enten bilder eller møbler (eller en kombinasjon av de to). Alle klasser skal ha konstruktører som initialiserer alle variabler og konstanter i klassene.

Alt inventar i et skip skal inngå i en enkel liste som pekes på av en variabel med navn altInventar. Alle salonger og kahytter i skipet er nummerert 0 til antallRom-1 og skal holdes orden på av en array indeksert fra 0 til antallRom-1 med navn rom. Denne tabellen skal peke på objekter av klassen Rom. Hvor mange rom det er i skipet, skal være en parameter når skips-objektet opprettes. Hvert rom-objekt skal ha et antall kvadratmeter gulvplass til møbler, antall meter vegg til bilder, antall kvadratmeter og antall veggmeter allerede i bruk, samt en beholder som innholder alt inventaret i dette rommet. Finn gjerne en passende klasse i Java-biblioteket for å lage denne beholderen.

# 1 Tegne klassehierarkiet

Tegn opp hele klasse- og interfacehiearkiet slik det er beskrevet.  Ta bare med navnene på interfacene og klassene.

**NB!** Det er snakk om klassene og interfacene, *ikke* objektene.

*I denne oppgaven skal du svare med digital håndtegning. Bruk eget skisseark (utdelt). Se instruksjon for utfylling av skisseark i lenken under oppgavelinjen.*

---

Maks poeng: 7

# 2 Programmere interface og klasser

Programmer de to interfacene **Bilde** og **Mobel** og klassene **Inventar**, **EgetInventar**, **EgetBilde** og **EgetBildeMobel**.

**NB!** Du skal *ikke* programmere de andre klassene.

---

Maks poeng: 13

# 3 Programmere Skip og Rom

Skriv klassen **Rom** som beskrevet og klassen **Skip** som beskriver hele skipet og som inneholder variablene **altInventar** og **rom**. Når et nytt **Rom**-objekt opprettes, er rommets gulvplass og veggplass parametre. For enkelhets skyld skal du anta at alle rommene er like store med 200 kvadratmeter gulvplass og 60 meter veggplass.

Metoder for å sette inventar inn i skipet og inn i rommene skal du ikke skrive her men i neste oppgave.

---

Maks poeng: 7

# 4 Programmere innsetting

Skriv metoden **settInn** i klassen **Skip.** Metoden tar en referanse til et inventar og et romnummer som parametere. Inventaret skal settes inn blant alt inventaret i skipet, dvs (hvor du vil) i den enkle pekerkjeden som variabelen med navn **altInventar** peker ut. Bruk variabelen med navn **neste** inne i objektet til denne pekerkjeden.

Inventaret skal også settes inn (hvor du vil) blant alt inventaret i dette rommet. Inventaret er et bilde og/eller et møbel, og antall gjenværende veggmeter og gulvareal i dette rommet skal oppdateres. Anta her i denne oppgaven at det er plass til inventaret i dette rommet.

---

Maks poeng: 5

# 5 Programmere unntak

Programmer metoden **settInn2** i klassen **Skip**. Du kan nå ikke anta at det er plass i rommet til denne varen. Metoden skal kaste ett av to egendefinerte unntak avhengig av om det er for lite gulvplass eller for lite veggplass. (Hvis det for lite av begge ressurser, er det det samme hvilket unntak som kastes.) Disse kontrollene skal du lage urealistisk enkle; du skal for eksempel ikke ta hensyn til hjørner i rommet eller formen på møblet på gulvet. Du skal kun sjekke gjenværende vegg- og/eller gulvplass. Er det ikke plass til inventaret i dette rommet, skal inventaret ikke settes inn.

---

Maks poeng: 7

# 6 Beregne forsikringsverdi

Skriv den rekursive metoden **finnSummenAvForsikringsverdier** i klassen **Inventar**. Metoden skal finne summen av forsikringsverdier i dette objektet og resten av objektene i den enkle listen (der neste objekt i listen pekes ut av variabelen med navn **neste**).

Skriv også metoden **totalForsikringsverdi** i klassen **Skip** som finner summen av forsikringsverdiene til alt inventaret i hele skipet ved å kalle **finnSummenAvForsikringsverdier**.

---

Maks poeng: 14

# 7 Programmere iterasjon

Skriv klassen **Skip** på nytt slik at den implementerer grensesnittet **Iterable** så det blir mulig å iterere gjennom alt inventaret i hele skipet ved hjelp av pekerlisten.

Husk at i følge Javas API så må en klasse som implementerer **Iterable** inneholde metoden **Iterator<E>  iterator()**.  En klasse som implementerer **Iterator** må igjen inneholde metodene **next()** og **hasNext()**. (Det er også vanlig at den inneholder metoden **remove()**, men den skal ikke du implementere.

Utsnitt av API-et for **Iterable**:

```
interface Iterable<E> {
   Iterator<E> iterator()
      Returns an iterator over elements of type E.
}
```

Utsnitt fra API-et for **Iterator**:

```
interface Iterator<E> {
   boolean hasNext()
      Returns true if the iteration has more elements.

   E next()
      Returns the next element in the Iteration.
```

---

Maks poeng: 10

# 8 Beregne forsikringsverdi med iterasjon

Skriv metoden **totalForsikringsverdi2** i klassen **Skip** som bruker iteratoren du har laget i forrige oppgave til å finne summen av alle forsikringsverdiene til alt inventaret i skipet.

---

Maks poeng: 4

Du skal i denne oppgaven lage en beholder som kan ta vare på objekter. Klassen skal hete PrioTabell. Beholderen skal være generisk, men bare objekter som implementerer grensesnittet Comparable (med seg selv) skal kunne legges inn i beholderen.

Datastrukturen i beholderen skal være en array med lengde lengde som oppgis som parameter til konstruktøren til beholderen. Beholderen skal ha de tre opplagte metodene settInn, taUt og antall.

Metoden taUt skal være en veldig enkel og rask operasjon som henter ut objektet som er størst uten å flytte på referansene til de andre objektene i beholderen. Vi skal få til dette ved å se på arrayen som en ring, der element 0 følger etter element lengde-1.

Metoden settInn skal sette inn et nytt element på sin rette plass i beholderen, og for å få det til kan det være nødvendig å flytte referansene til alle objekter som er mindre enn det objektet som settes inn. Det er derfor naturlig å starte med å sammenligne det nye objektet med det minste objektet i beholderen og flytte på referanser helt til letingen finner et objekt som er større (eller lik). Siden vi ser på arrayen som en ring, så kan letingen etter plassen der det nye objektet skal settes inn komme til den ene enden av arrayen og må fortsette i den andre.

Hvis det ikke er plass til et nytt objekt, skal settInn-metoden fjerne det minste objektet i beholderen (eller la være å sette inn det nye elementet, hvis det er aller minst).

**Eksempel**

Anta at du har en nyopprettet beholder med plass til 6 objekter og at det er foretatt 6 kall på settInn. Anta videre at arrayen din heter alleO. Da kan referansen til det største objektet ligge i alleO[0], det nest største i alleO[1] osv. Hvis du nå kaller taUt, vil referanseverdien i alleO[0] bli returnert. Et nytt kall på taUt vil returnere referanseverdien i alleO[1]. Nå er de to ledige elementene i arrayen alleO[0] og alleO[1].

Anta nå at du nå kaller settInn. Da må metoden starte å lete og sammenligne fra alleO[5] (fordi det er der det minste elementet ligger i dette eksemplet). Hvis settInn nå tilfeldigvis setter inn et objekt som er mindre enn alle som ligger i arrayen fra før, vil dette bli lagt inn i alleO[0] fordi vi ser på arrayen som en ring, og derfor er dette elementet mindre enn det som refereres av alleO[5]. Hvis derimot objektet som refereres av alleO[5] er mindre enn det nye objektet, må verdien i alleO[5] flyttes til alleO[0], før letingen fortsette med å sammenligne med alleO[4] osv.

# 9 Datastruktur

Tegn på et ark fire **Inventar**-objekter med fire forskjellige verdier i instansvariabelen **verdi**. Tegn også et objekt av klassen **PrioTabell** som kan ta vare på referanser til opp til 5 slike **Inventar**-objekter. Tegn arrayen **alleO** slik at den inneholder referanser til disse fire objektene der referansen til det største objektet ligger i **alleO[2]**.

*I denne oppgaven skal du svare med digital håndtegning. Bruk eget skisseark (utdelt). Se instruksjon for utfylling av skisseark i lenken under oppgavelinjen.*

---

Maks poeng: 7

# 10 Programmere prioritetstabell

Programmer den generiske beholderklassen **PrioTabell**.

---

Maks poeng: 26