Husk å fylle ut emneevaluering fra Fagutvalget!

Eksamen IN1010 vår 2026

Skoleeksamen i Silurveien

29. mai; 15:00-19:00
(sjekk hvilken sal i Studentweb)

Resten av semesteret

- Siste undervisning:
  - Eksamensoppgave fra 2022 (økt 1) gjennomgås på fellesøvelse i overmorgen
    Bruk Discourse for spørsmål frem til eksamen

- Eksamen 29. mai kl 15.00-19.00
  - Sensur innen 1. juli
  - Løsningsforslag og sensorveiledning legges ut samtidig (eller før) på semestersiden
  - Individuell begrunnelse sendes ut til alle (UiO-epost, trenger ikke be om det)

- Utsatt / ny eksamen i august (antakelig uke 33): Samme form som ordinær

Læringsmål fra emnebeskrivelsen

Hva lærer du? Emnebeskrivelse IN1010
https://www.uio.no/studier/emner/matnat/ifi/IN1010/#learning_outcomes

Etter å ha tatt IN1010:

- har du god oversikt over programmeringsspråket Java og du kan bruke det til å løse reelle problemer av middels størrelse
- behersker du avanserte objektorienterte mekanismer som subklasser, polymorfi og interface
- har du oversikt over noen grunnleggende datastrukturer (spesielt lenkete lister) og du kan programmere de viktigste operasjonene på dem
- kan du utvikle robuste og pålitelige programmer med godt objektorientert design, og du kan finne alternative løsninger for et gitt problem og vurdere fordeler og ulemper ved de ulike løsningene
- har du kunnskap om parallelle programmer med tråder og du kan benytte dette i enkle programmer
- kjenner du til hendelseshåndtering og kan skrive enkle programmer som håndterer hendelser

Pensum

Dette ble også presentert på forelesning før Midtveiseksamen

- Følg lenke fra semestersiden til Pensum/ litteratur i Leganto
  - Oversikt over alt relevant materiale

- Kjernepensum er alt som er forelest (unntak: Simula) og gitt i innleveringer

- NB: Læringskravet i IN1010 er ikke først og fremst å kunne et pensum, men å kunne skrive gode programmer (rimelig raskt) ved å sette seg inn i en oppgave og bruke programmeringsmetoder som er forelest og innøvd i de forskjellige studieaktivitetene.

- Har du deltatt på undervisningen og gjort innleveringer, dvs forstår og har trent på å bruke pensum, er du godt rustet!

Praktisk info – bla gjennom disse sidene!

- Skoleeksamen i Inspera
  innlogging, brukergrensesnitt, funksjoner, håndtegning

- Prøveeksamen er nyttig for å bli kjent med (friske opp) Inspera.
  - Mac-brukere: Norsk Windows-tastatur i Silurveien. Obs: {} [] etc.
  - Prøveeksamen stenges to dager før eksamen

- Silurveien: Oppmøtetid, transport, innganger, saler mm
- Studentweb (din plass)

Bruk semestersiden

- Ressurser
  - Lysark og programmer fra forelesningene i egne mapper
  - Opptak lenket fra Timeplanen/ aktuell forelesning
  - Notater i egen mappe
- Tidligere eksamensoppgaver
  - Ikke alt kommer hvert år, til sammen dekker de *det meste* av pensum
  - Oppgaver, løsningsforslag og sensorveiledning (hva skal sensor se etter, hva er viktig)
- På eksamenssiden
  - Praktisk/ formell informasjon om eksamen

Hva kan du få på eksamen?

- Programmeringsoppgaver som skal besvares i Inspera (som prøveeksamen)
- Tegning på papir som scannes inn (for eksempel datastruktur, klassehierarki)
  Hvis tegning: Sett av tid til å merke ark riktig før levering!
- Automatisk rettede oppgaver som på midtveis-eksamen

- Ett svarfelt per deloppgave. Henvis til eller kopier relevant kode ved behov.
- Hjelpemidler: Alle trykte og skrevne (dvs papir)

- Faglærer kommer innom alle saler en gang - fra ca en time etter eksamensstart

Hvordan vurderer sensor besvarelsene?

Du må vise forståelse for, og bruk av, sentrale deler av pensum

- Skrivefeil ignoreres vanligvis, men les gjennom det du skriver:
  - Feil som kan tyde på manglende forståelse vil gi trekk
  - Slurv kan gi opphav til misforståelser
- Alle navn på variabler/klasser/metoder vil være på norsk, også i engelsk oversettelse. Bruk disse!
- Ignoreres (eksempler):
  - Mangler (noen) ';' , '}' eller '{' om koden ellers gir mening (NB)
  - Småfeil i (gjenkjennbare) navn
- Kan gi trekk (eksempler)
  - Gjennomgående feil/ mangler
  - Bruker beholder på en måte som tyder på manglende forståelse
  - Kode som åpenbart er kopiert fra en annen løsning uten tilpasning
- Det legges ikke vekt på bruk av tilgangsmodifikatorer (public/ private)

Tidsbruk

Pass tiden – alt inkludert **id**
på tegninger må være klart
innen 4 timer

Bruk tid på å

- få oversikt over hele oppgavesettet
- lese teksten nøye for hver oppgave
- forstå datastrukturen før du programmerer (tegn, notér, pseudokode, …)
- vurdere tidsbruk underveis, mot poeng

Lesbar kode kan bidra til at sensor forstår hva du mener selv om det f eks mangler en '{'.

Før eksamen: Hva trenger du å ta med *som du forstår og rekker å finne frem i*

Ikke bruk tid på å

- gjenta oppgaveteksten
- skrive kode du ikke blir bedt om (les oppgaven!)
- unødige detaljer i evt. tegneoppgaver (les oppgaven!)
- kommentere (om du ikke trenger å tolke noe du synes er uklart)
- Med mindre oppgaven ber om det:
  - Test på lovlig input
  - Testprogrammer
  - Import-setninger

Prøveeksamen (=eksamen 2025)

Generelt om oppgaven

• I dette oppgavesettet skal du ikke bruke arrayer, ArrayList-er, HashMap-er eller andre av Java sine innebygde beholdere. I stedet skal du definere og bruke en egen datastruktur kalt Samling (som er beskrevet i første oppgave).

(Beholder)

• Du kan innføre flere parametre, variabler og metoder enn de som er omtalt i oppgaveteksten. Noen av oppgavene krever utvidelse av klasser skrevet i tidligere oppgaver. Beskriv hvor utvidelsen hører til i en kommentar.

• Du kan gjenbruke kode fra andre deloppgaver selv om du ikke har løst oppgaven. Anta i så fall at omtalte klasser og metoder finnes.

• Hvis du synes noe i oppgaveteksten er uklart, kan du gjøre dine egne antagelser. Skriv en kommentar om dette i svaret ditt.

Oppgave 1 Klassen Samling (vekt 10 poeng)

Klassen Samling skal deklareres slik:

```
class Samling<E> {
    private Node start;
    public Samling (int n) { ... }
    private class Node {
        ...
    }
    public void settInn (E ny) { ... }
    public void fjern (E ref) {
    /* skal ikke skrives, du kan anta at den er implementert */
    }
}
```

Konstruktøren skal opprette en lenkeliste med n noder. Etter at konstruktøren er ferdig vil det aldri bli laget nye noder eller fjernet noen.

Alle nodenes datareferanse skal initieres til null av konstruktøren. Generelt gjelder at om en datareferanse er null, skal det angi at det ikke er noe element tilknyttet noden akkurat nå.

Metoden settInn skal sette inn et nytt element i en ledig node, dvs en node som har datareferanse null. Det er det samme hvilken node som velges. Du kan anta at det alltid finnes en ledig node.

Metoden fjern fjerner det gitte elementet fra listen ved å sette dets datareferanse til null. Denne metoden skal du ikke skrive.

(I den virkelige verden ville klassen hatt mange flere metoder, men i denne eksamensoppgaven er dette tilstrekkelig.)

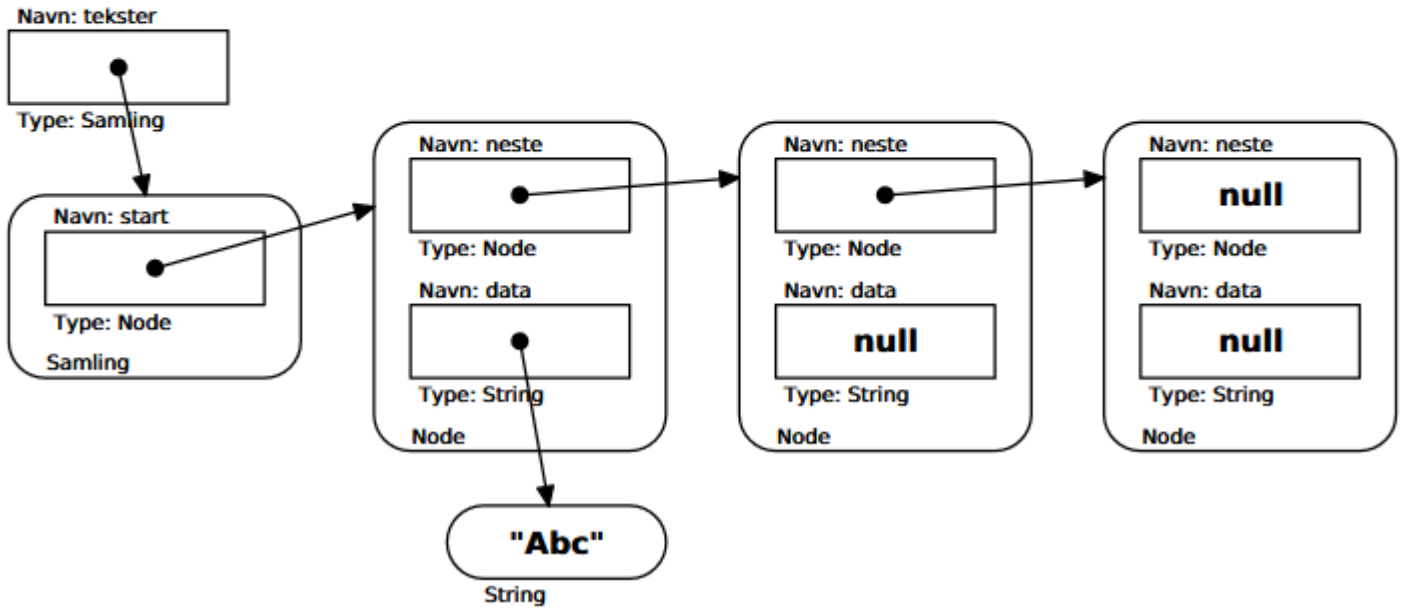
Oppgave 1 Klassen Samling (datastruktur)

Oppgave Skriv ferdig klassen `Samling` med de tre forekomstene av «`...`» fylt ut.

Eksempel Etter å ha utført denne koden:

```
Samling<String> tekster = new Samling<>(3);
tekster.settInn("Abc");
```

vil vi kunne få en datastruktur som denne:



Oppgave 1 Klassen Samling

Klassen Samling skal deklareres slik:

```
class Samling<E> {
   private Node start;
   public Samling (int n) { ... }
   private class Node {
      ...
   }
   public void settInn (E ny) { ... }
   public void fjern (E ref) {
   /* skal ikke skrives, du kan anta at den er implementert */
   }
}
```

```
Samling (int n) {
    for (int i = 1; i <= n; ++i) {
        Node ny = new Node();
        ny.neste = start;
        start = ny;
    }
}
```

```
    private class Node {
        Node neste = null;
        E data = null;
    }
```

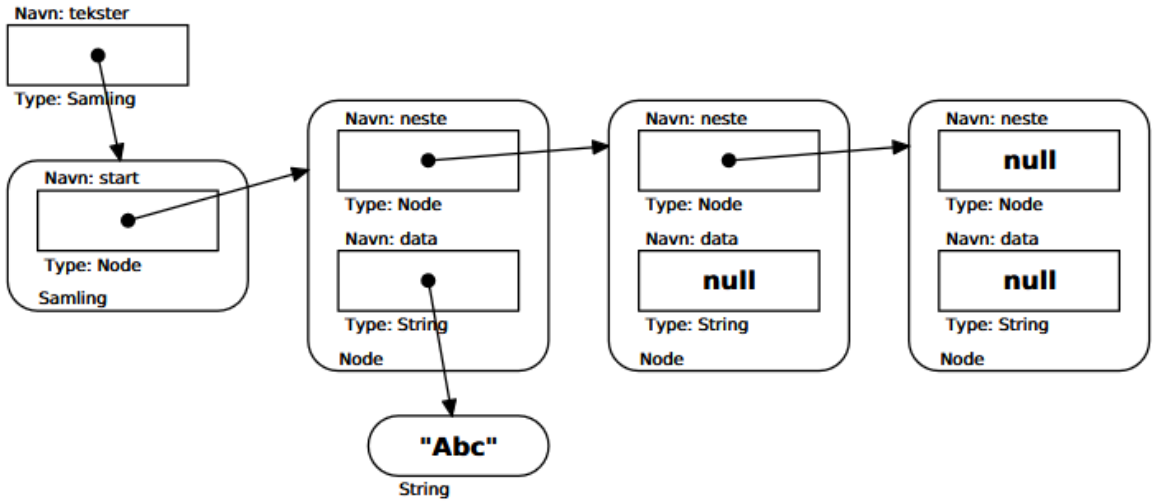
```
    public void settInn (E ny) {
        Node peker = start;
        while (peker.data != null) { // Alltid ledig
            peker = peker.neste;
        }
        peker.data = ny;
    }
```

Oppgave 2 Iterator

Eksempel Etter å ha utført denne koden:

```
Samling<String> tekster = new Samling<>(3);
tekster.settInn("Abc");
```

vil vi kunne få en datastruktur som denne:



Oppgave 2 Iterator (vekt 10 poeng)

Utvid klassen `Samling` fra oppgave 1 med en iterator som kan iterere over alle elementene lagret der.

NB! Bare elementer som er noe annet enn **`null`** skal tas med.

Husk:

- Beholderklassen (her `Samling`) må implementere interface `Iterable` - dvs
  - angi `implements` … i klassedeklarasjonen
  - ha en metode `iterator` som returnerer et iterator-objekt for beholderen

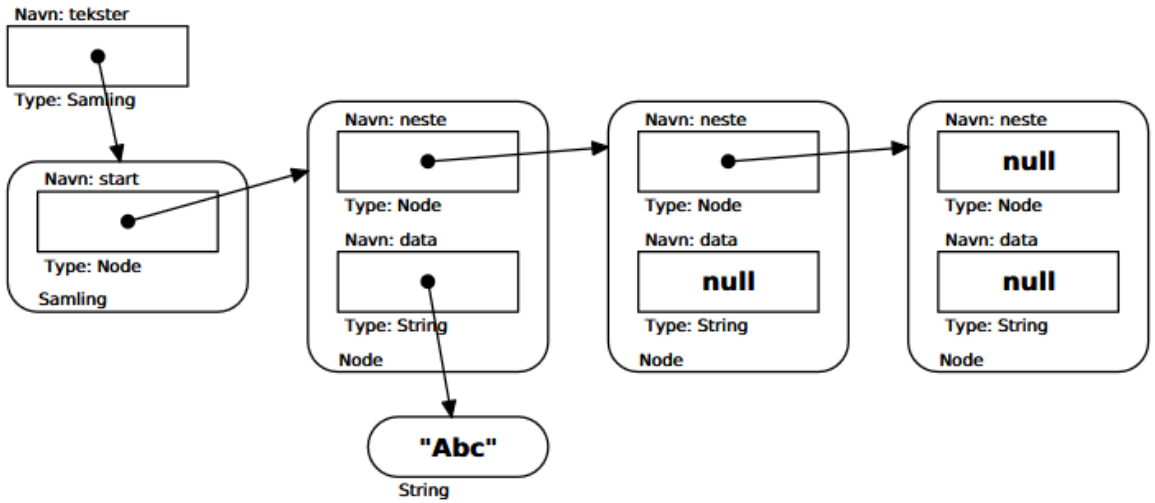
- Vi må dermed lage en iterator-klasse(1) i `Samling` som er tilpasset beholderen (2)
  1. Implementerer `Iterator` dvs metodene `hasNext` og `next`
  2. Hopper over tomme noder

Oppgave 2 Iterator

Eksempel Etter å ha utført denne koden:

```
Samling<String> tekster = new Samling<>(3);
tekster.settInn("Abc");
```

vil vi kunne få en datastruktur som denne:



Oppgave 2 Iterator (vekt 10 poeng)

Utvid klassen `Samling` fra oppgave 1 med en iterator som kan iterere over alle elementene lagret der.

NB! Bare elementer som er noe annet enn **`null`** skal tas med.

Husk:

Beholderklassen (her `Samling`) må implementere interface `Iterable` - dvs

- angi `implements` … i klassedeklarasjonen
- ha en metode `iterator` som returnerer et iterator-objekt for beholderen

```
class Samling<E> implements Iterable<E>{

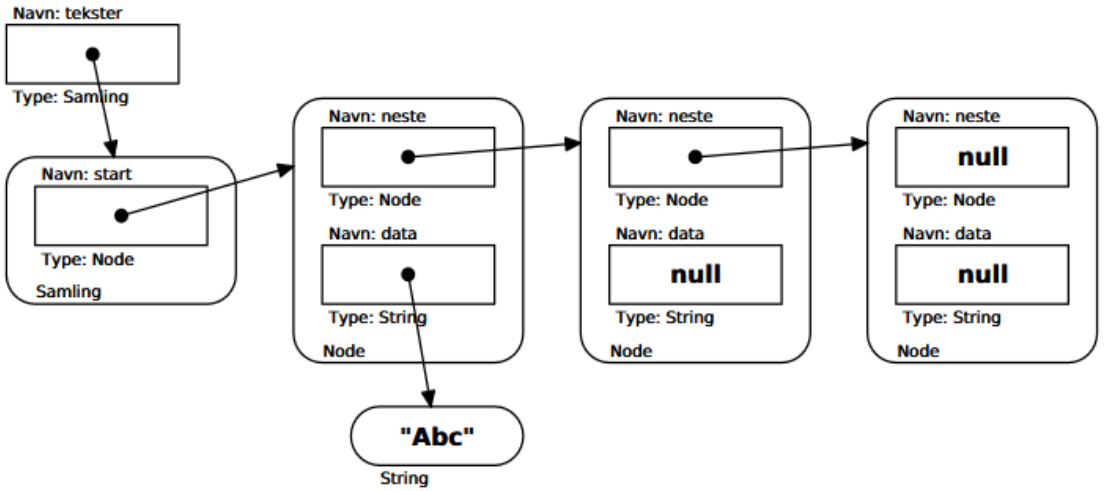
    @Override
    public Iterator<E> iterator() {
        return new SIterator();
    }
```

Oppgave 2 Iterator forts

Eksempel Etter å ha utført denne koden:

```
Samling<String> tekster = new Samling<>(3);
tekster.settInn("Abc");
```

vil vi kunne få en datastruktur som denne:



Oppgave 2 Iterator (vekt 10 poeng)

Utvid klassen `Samling` fra oppgave 1 med en iterator som kan iterere over alle elementene lagret der.

NB! Bare elementer som er noe annet enn **`null`** skal tas med.

Vi må dermed lage en iterator-klasse(1) i `Samling` som er tilpasset beholderen (2)

1. Implementerer `Iterator` dvs metodene `hasNext` og `next`
2. Hopper over tomme noder

```
    private class SIterator implements Iterator<E> {
        private Node denne;
        SIterator() {
            denne = start;
            while (denne != null && denne.data == null) {
                denne = denne.neste;
            }
        }
        @Override
        public E next() {
            Node tmp = denne;
            denne = denne.neste;
            while (denne != null && denne.data == null) {
                denne = denne.neste;
            }
            return tmp.data;
        }
        @Override
        public boolean hasNext() {
            return (denne != null);
        }
    }
```

Situasjonsbeskrivelse og oppgave 3

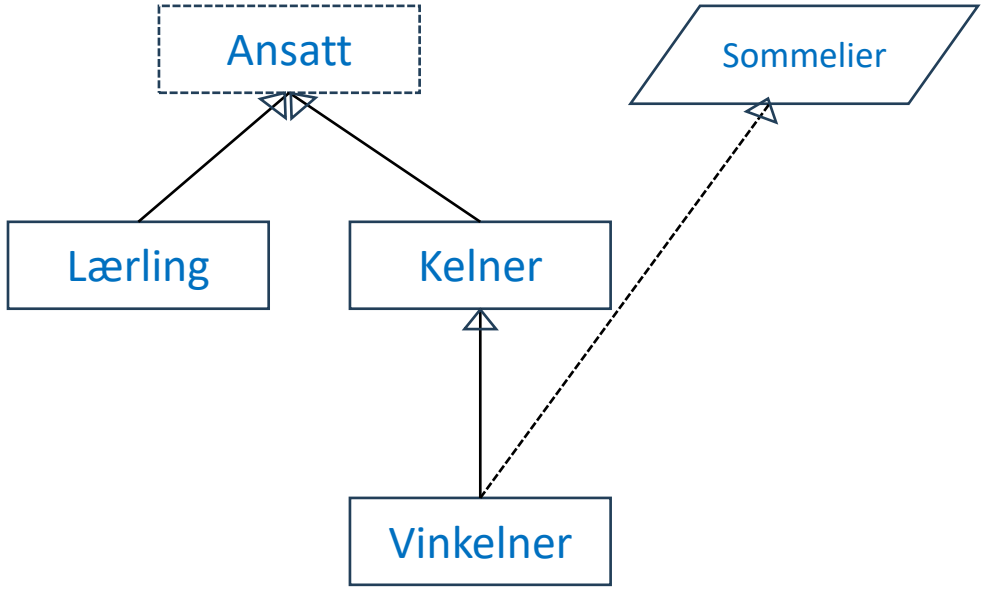
Situasjonsbeskrivelse for resten av oppgaven

Du skal lage deler av et Java-program for å administrere en restaurant.
I dette restaurantsystemet er alle ansatte enten kelnere eller lærlinger. Alle ansatte har et navn (en String). Noen kelnere har tatt et såkalt sommelier-kurs om vin og blitt vinkelner. Egenskapen sommelier er definert som et interface:

```
interface Sommelier {
    String spesialitet();
    // F eks "spansk vin" eller "fransk cognac"
}
```

Oppgave 3 Klassediagram (vekt 8 poeng)

Tegn et klassediagram for klassene Ansatt, Kelner, Vinkelner og Lærling og interfacet Sommelier.



Oppgave 4
Klassedeklarasjoner

På restauranten er det 15 bord. Hvert bord har et gitt antall plasser (gitt som parameter til konstruktøren). Når bordet er i bruk, skal det ha en referanse til gruppen av gjester som sitter der og til kelneren som serverer.

Restauranten har en Samling med inntil 25 ansatte, og hver kelner har en Samling av inntil 10 bord som hen betjener.

Til restauranten kommer det grupper med gjester; det kan være inntil 20 gjester i hver gruppe. Den enkelte gjest opplyser om vedkommende ønsker vin til maten (angitt som parameter til konstruktøren til gjesten).

Skriv klassene Restaurant, Ansatt, Kelner, Vinkelner, Lærling, Bord, Gjest og Gjestegruppe med instansvariabler og konstruktører, men ikke andre metoder.

```
class Restaurant {
    Samling<Bord> bordene = new Samling<>(15);
    Samling<Ansatt> ansatte = new Samling<>(25);
}

class Bord {
    final int antPlasser;
    Gjestegruppe gjester = null;
    Kelner servitør = null;

    Bord(int ant) {
        antPlasser = ant;
    }
}

class Gjest {
    private boolean vilHaVin;
    Gjest(boolean vin) {
        vilHaVin = vin;
    }
}

class Gjestegruppe {
    Samling<Gjest> personer = new Samling<>(20);
}
```

Oppgave 4 forts. Klassedeklarasjoner

```
interface Sommelier {
    String spesialitet ();
    // F eks "spansk vin" eller "fransk cognac"
}

abstract class Ansatt {
    final String navn;
    Ansatt(String id) {
        navn = id;
    }
}


class Lærling extends Ansatt {
    Lærling(String id) {
        super(id);
    }
}
```

```
class Kelner extends Ansatt {
    Samling<Bord> serverer = new Samling<>(10);
    Kelner(String id) {
        super(id);
    }
}

class Vinkelner extends Kelner implements Sommelier {
    private String minSpesialitet;
    Vinkelner(String id, String spes) {
        super(id);
        minSpesialitet = spes;
    }

    @Override
    public String spesialitet() {
        return minSpesialitet;
    }
}
```

Oppgave 5 Søking

Utvid klassen Gjest med metoden boolean ønskerVin().

Utvid deretter klassen Gjestegruppe med metoden boolean noenØnskerVin() som finner ut om minst én i gruppen ønsker vin.

Metoden skal benytte iteratoren fra oppgave 2 til å sjekke gjestene.

```
class Gjest {
    private boolean vin;

    public Gjest(boolean vin) {
        this.vin = vin;
    }

    public boolean ønskerVin() {
        return vin;
    }
}
```

```
class Gjestegruppe {
    Samling<Gjest> gjester = new Samling<>(20);

    public boolean noenØnskerVin() {
        for (Gjest denne:gjester) {
            if (denne.ønskerVin()) {
                return true;
            }
        }
        return false;
    }
}
```

Oppgave 6 Finn beste

Skriv metoden
`Kelner finnMinstÅGjøre(boolean erVinkelner)`
i klassen Restaurant som finner den kelneren som betjener færrest antall gjester.

(NB! færrest antall gjester, ikke færrest antall bord.)

Hvis erVinkelner er sann, er bare Vinkelner-e interessante. Vi kan anta at det alltid er minst én som fyller kriteriene.

For hver kelner:
- Oppfyller Vinkelner-krav?
- Håndtere referanse-typer ansatte
- Finne antall gjester per kelner
  - per bord (per gjestegruppe)

Oppgave 6 forts.

For hver kelner:
- Oppfyller Vinkelner-krav?
  - Sammenlign med minste hittil (om noen)
    - Hvor mange gjester?
  - Trenger Kelner-referanse

Trenger å finne:
- Antall gjester per kelner
  - antall per bord (=gjestegruppe)
    - antall i Samling

Her er det en fordel om man har sett over oppgavesettet først!

class Restaurant

```
public Kelner finnMinstÅGjøre(boolean erSommelier) {
    Kelner minstHittil = null;
    for (Ansatt ax : ansatte) {
        if (erSommelier && ax instanceof Vinkelner ||
            !erSommelier && ax instanceof Kelner) {
            Kelner kx = (Kelner) ax;
            if (minstHittil == null ||
                kx.antallServerer() < minstHittil.antallServerer()) {
                minstHittil = kx;
            }
        }
    }
    return minstHittil;
}
```

class Kelner

```
public int antallServerer() {
    int sum = 0;
    for (Bord bx : serverer) {
        sum += bx.gjester.antall();
    }
    return sum;
}
```

class Gjestegruppe

```
public int antall () {
    return personer.antall();
}
```

Gjennomgang av prøveeksamen: Rekursjon, GUI og tråder

Magnus Skjegstad

IN1010, våren 2026

Oppgave 7 Rekursjon (vekt 10 poeng)

Utvid klassen `Samling` fra oppgave 1 med en metode `antall` som finner antall elementer. Dette skal gjøres ved å gå gjennom samlingen *rekursivt*.

- Vi må telle antallet noder **med data** i den lenkede listen i en Samling
- Flere måter dette kan gjøres på, viktig at løsningen er rekursiv – altså en metode som kaller seg selv

Rekursjon

- Vi kan lage en metode i **Node** som rekursivt teller antallet fremover i listen (med seg selv som start).

```
int antallFraOgMedDenne () {
    int res = 0;
    if (data != null) ++res;
    if (neste != null) res += neste.antallFraOgMedDenne();
    return res;
}
```

- Nå kan vi i **Samling**-klassen legge til en antall-metode som teller fra start-noden:

```
public int antall () {
    if (start == null) return 0;
    return start.antallFraOgMedDenne();
}
```

Oppgave 8 Ta imot gjester (vekt 12 poeng)

Skriv metoden `taImotGjester(Gjestegruppe gruppe)` i klassen `Restaurant`. Den skal gjøre følgende:

1. Finn et ledig bord med nok plasser. (Du kan her anta at det alltid finnes et slikt bord.)
2. Finn den kelneren som for tiden har minst å gjøre. Ta hensyn til om noen av gjestene i gruppen ønsker vin.
3. Oppdater referansene mellom gjestegruppen, kelneren og bordet.

Gjør bruk av metoder definert tidligere, og tilføy eventuelle hjelpemetoder i andre klasser ved behov.

(Kort gjennomgang av denne fordi vi trenger den for tråder litt senere)

Hjelpemetoder - ikke nødvendig, men gjør det mer oversiktlig

Hjelpemetode i **Restaurant**:

```
Bord finnEgnetBord (int ant) {
    for (Bord bx: bordene) {
        if (bx.erLedig() && bx.antPlasser >= ant)
            return bx;
    }
    return null;
}
```

Hjelpemetode i **Bord**:

```
public boolean erLedig () {
    return gjester == null;
}
```

```
public Bord taImotGjester (Gjestegruppe gruppe) {
    // 1. Finn et bord
    int antGjester = gruppe.antall();
    Bord detteBordet = finnEgnetBord(antGjester);

    // 2. Finn kelneren med minst å gjøre
    Kelner denneKelneren = finnMinstÅGjøre(gruppe.noenØnskerVin());

    // 3. Oppdater referansene
    denneKelneren.serverer.settInn(detteBordet);
    detteBordet.gjester = gruppe;
    detteBordet.servitør = denneKelneren;

    // Returner hvilket bord gjestene fikk (nyttig i oppgave 10)
    return detteBordet;
}
```

(Mer detaljer i løsningsforlaget)

Oppgave 9 GUI (vekt 10 poeng)

Endre metoden `taImotGjester` i klassen `Restaurant` slik at hvis det ikke finnes noe ledig bord med nok plass til gruppen av gjester, skal programmet lage et GUI-vindu med teksten “Beklager, restauranten er full!", samt en OK-knapp som lukker vinduet. Vinduet skal vises på skjermen og kan se slik ut:



Plan

- Vi må lage
  - et vindu
  - et panel
  - en tekst
  - en knapp med en handling
  - + det som alltid ligger rundt GUI-programmer
- Vi skal utvide metoden taImotGjester() i Restaurant-klassen
- Vi må sjekke om det er ledig bord, så vise vindu ved feil



Hvis fullt, vis vindu

```
public Bord taImotGjester (Gjestegruppe gruppe) {
    // 1. Finn et bord
    int antGjester = gruppe.antall();
    Bord detteBordet = finnEgnetBord(antGjester);

    // Oppgave 9
    // ---------

    if (detteBordet == null) {
        visSkilt();  return null;
    }
                :
```

Vi bruker en egen metode her (visSkilt()) for å gjøre det mer oversiktlig

```
private void visSkilt () {
    try {
        UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName());
    } catch (Exception e) { System.exit(9); }

    JFrame vindu = new JFrame("Melding");
    vindu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JPanel panel = new JPanel();
    vindu.add(panel);

    JLabel tekst = new JLabel("Beklager, restauranten er full!");
    panel.add(tekst);

    class Avslutter implements ActionListener {
        @Override
        public void actionPerformed (ActionEvent e) {
            vindu.setVisible(false);
        }
    }

    JButton ok = new JButton("OK");
    ok.addActionListener(new Avslutter());
    panel.add(ok);

    vindu.pack();
    vindu.setLocationRelativeTo(null);
    vindu.setVisible(true);
}
```

Setter felles utseende

Lager vinduet og angir at programmet avsluttes når det lukkes

Lager panelet og legger det til i vinduet

Lager merkelappen med tekst og legger til i panelet

Handlingen som utføres når man trykker på knappen (ActionListener)

Knappen lages og legges til

Pakk sammen, sett posisjon og vis vindu



Det som var viktigst i denne oppgaven

- Riktig struktur på GUI-oppsettet (inkludert vindu, panel, tekst, knapp, visning av vindu)
- Brukerinteraksjon med OK-knappen (ActionListener-knappen og actionPerformed-metoden)
- Småfeil i klassenavn, metodenavn osv er ikke viktig

Oppgave 10 Tråder (vekt 14 poeng)

I denne oppgaven skal du implementere venting på ledig bord ved hjelp av tråder. Du skal ta utgangspunkt i koden du hadde *før* du gjorde endringen i `taImotGjester` i forrige oppgave.

Endre klassen `Restaurant` så den blir en monitor. Metoden `taImotGjester` skal nå fungere slik: Hvis det ikke er noen ledige bord som har nok plasser, må kallet vente på en `Condition` til et slikt bord blir ledig. Utvid også klassen `Restaurant` med en ny metode `forlatBordet(Bord bord)` som registrerer at bordet er ledig og signalerer ventende gjestegrupper.

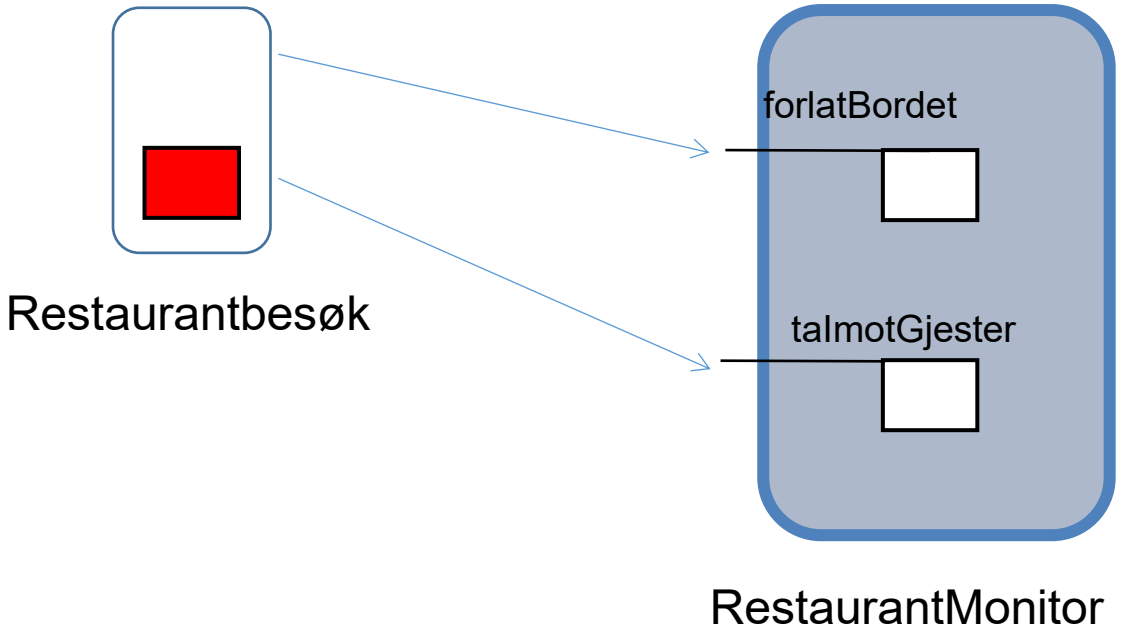
Skriv også trådklassen `Restaurantbesøk` der konstruktøren skal ha to parametre: `Restaurant restaurant` og `Gjestegruppe gruppe`. `run`-metoden skal gjøre dette:

1. Kalle metoden `restaurant.taImotGjester`. (Siden `Restaurant` nå er en monitor, vil dette kallet ikke returnere før det finnes et ledig bord.)

2. Spise (men dette skal du ikke programmere)

3. Kalle metoden `restaurant.forlatBordet`.

- Vi trenger:
  - Restaurantbesøk-klasse som implementerer Runnable og har konstruktør med to parametere for Restaurant og Gjestegruppe
  - Kritiske regioner i Restaurant så den er trådsikker (vi kaller denne RestaurantMonitor i løsningsforslaget)
  - Condition-variabel for å kunne vente



Restaurantbesøk-tråder:

1. venter på bord (**talmotGjester**)
2. spiser (ikke vist)
3. forlater bordet (**forlatBordet**)

Passiv venting og låser i RestaurantMonitor (fra tråder 1)

Samling<Bord> bordene;

Lock lås = new **ReentrantLock**();
Condition ventPåBord = lås.newCondition();

void **forlatBordet** (Bord bordet)

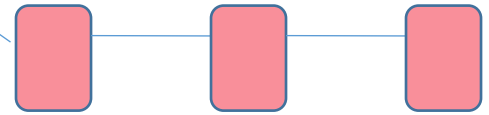
```
lås.lock();
try {
    // noen har gått, så det er helt sikkert ikke tomt:
  ventPåBord.signal();
} finally {
  lås.unlock();
}
```

Bord **taImotGjester** (Gjestegruppe gjester) throws InterrupedException

```
lås.lock();
try {
   while (detteBordet == null) {
     ventPåBord.await();
     detteBordet = finnEgnetBord(..);
   }
} finally {
  lås.unlock();
}
```



Tråder som venter på låsen for å få komme inn første gang



Restaurantbesøk som venter på at det skal bli ledige bord

RestaurantMonitor må vente i **taImotGjester** hvis restauranten er full

```
class Restaurantbesøk implements Runnable {
    RestaurantMonitor restMonitor;


    Gjestegruppe gruppe;

    Restaurantbesøk (RestaurantMonitor rest, Gjestegruppe gr) {
        restMonitor = rest;  gruppe = gr;
    }

    @Override
    public void run () {
        Bord vårtBord = restMonitor.taImotGjester(gruppe);
        // Spis
        restMonitor.forlatBordet(vårtBord);
    }
}
```

Restaurantbesøk kan implementeres ganske rett fram fra oppgaven

RestaurantMonitor

- Vi trenger lås og Condition-variabel:

```
class RestaurantMonitor {
    Samling<Ansatt> ansatte = new Samling<>(25);
    Samling<Bord> bordene = new Samling<>(15);

    // Nytt i oppgave 10:
    Lock lås = new ReentrantLock();
    Condition ventPåBord = lås.newCondition();
```

```
public Bord taImotGjester (Gjestegruppe gruppe) {
    // Må låses fordi den oppdaterer monitorens data.
    lås.lock();
    try {
        // 1. Finn et bord
        int antGjester = gruppe.antall();
        Bord detteBordet = finnEgnetBord(antGjester);

        // Vent til et egnet bord er ledig.
        while (detteBordet == null) {
            ventPåBord.await();
            detteBordet = finnEgnetBord(antGjester);
        }

        // 2. Finn kelneren med minst å gjøre
        Kelner denneKelneren = finnMinstÅGjøre(gruppe.noenØnskerVin());

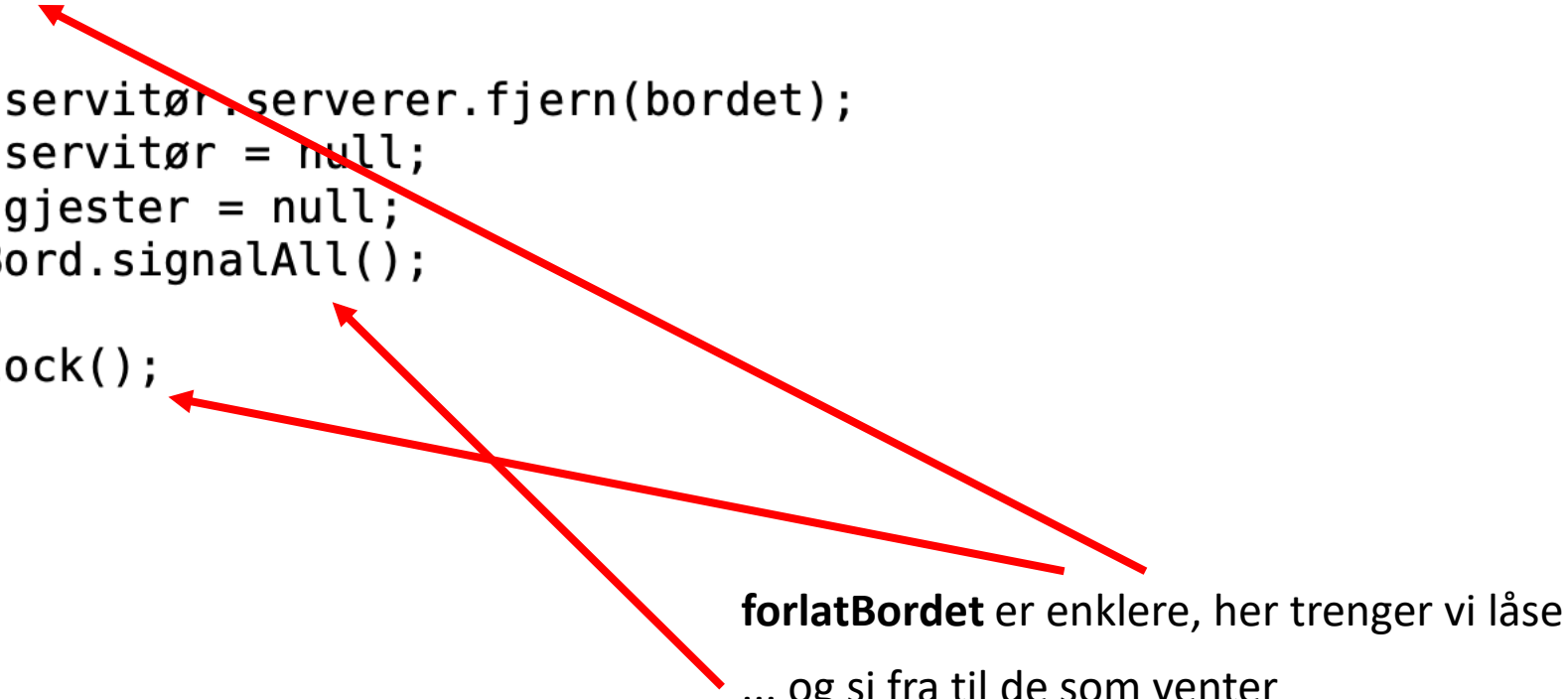
        // 3. Oppdater referansene
        denneKelneren.serverer.settInn(detteBordet);
        detteBordet.gjester = gruppe;
        detteBordet.servitør = denneKelneren;
        return detteBordet;
    } catch (InterruptedException e) {
        System.exit(1);
        return null;  // Kreves av kompilatoren!
    } finally {
        lås.unlock();
    }
}
```

taImotGjester må låse

... og vente på bord når det er fullt

Når vi venter må vi også håndtere InterruptedException

```
public void forlatBordet (Bord bordet) {
    // Må låses fordi den endrer monitorens data.
    lås.lock();
    try {
        bordet.servitør.serverer.fjern(bordet);
        bordet.servitør = null;
        bordet.gjester = null;
        ventPåBord.signalAll();
    } finally {
        lås.unlock();
    }
}
```



forlatBordet er enklere, her trenger vi låse

... og si fra til de som venter

```
public Kelner finnMinstÅGjøre (boolean erSommelier) {
    // Må låses fordi den aksesserer monitorens data.
    lås.lock();
    try {
        Kelner minstHittil = null;
        for (Ansatt ax: ansatte) {
            if (erSommelier && ax instanceof Vinkelner ||
                ! erSommelier && ax instanceof Kelner) {
                Kelner kx = (Kelner)ax;
                if (minstHittil == null ||
                    kx.antallServerer() < minstHittil.antallServerer()) {
                    minstHittil = kx;
                }
            }
        }
        return minstHittil;
    } finally{
        lås.unlock();
    }
}
```

finnMinstÅGjøre er public og må derfor også låses

Vi trenger ikke låse finnEgnetBord (den er private i løsningsforslaget og brukes bare internt i RestaurantMonitor)

Det som var viktigst i denne oppgaven

- Å lage en monitor av Restaurant-klassen med riktig låsing i taImotGjester og forlatBordet
- Riktig bruk av synkronisering med Condition-variabelen
- Klassen Restaurantbesøk med run-metoden
- Riktig håndtering av unntak