Eksamen IN1010 med løsningsforslag

Dag Langmyhr
dag@ifi.uio.no

4. juni 2025

Sammendrag

Dette er eksamensoppgaven gitt 4. juni 2025 sammen med et løsningsforslag. Det er viktig å merke seg at dette er et løsnings*forslag* og at det finnes flere alternative og like gode løsninger på de fleste oppgavene.

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).
- 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.

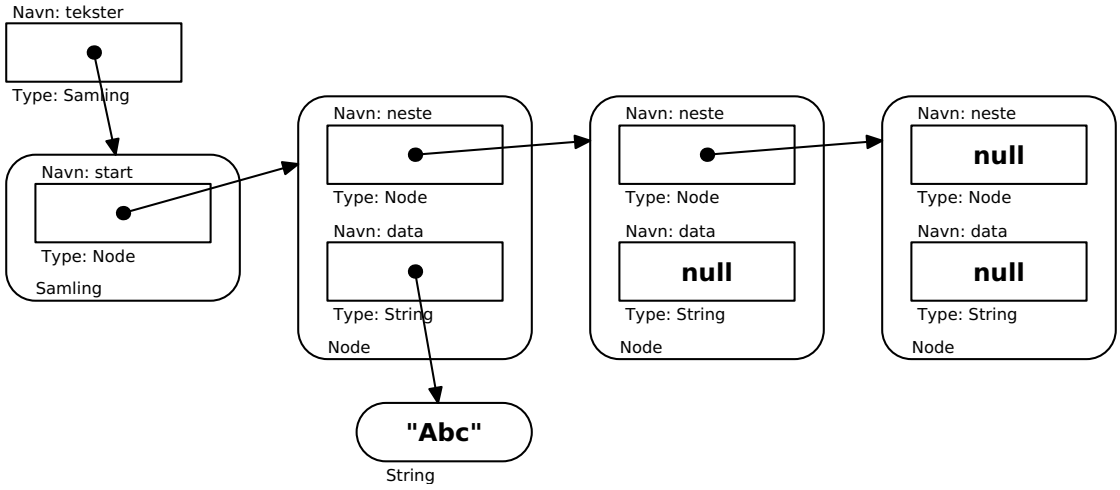
(I den virkelige verden ville klassen hatt mange flere metoder, men i denne eksamensoppgaven er dette tilstrekkelig.)

Oppgave 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:



Løsningsforslag

```
class Samling<E>
{
    private Node start;

    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 np = start;
        while (np.data != null) np = np.neste;
        np.data = ny;
    }
}
```

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.

Løsningsforslag

Først må vi angi at `Samling` har en iterator:

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

Så må vi definere metoden `iterator` og tilhørende `Iterator`-klasse. Det viste seg nyttig med en liten hjelpemetode `finnNesteIkkeNull` for å unngå å gjenta kode.

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

    private class SamlingIterator implements Iterator<E> {
        private Node denne;

        SamlingIterator () {
            denne = start;  finnNesteIkkeNull();
        }

        private void finnNesteIkkeNull () {
            while (denne != null && denne.data == null)
                denne = denne.neste;
        }

        @Override
        public boolean hasNext () {
            return denne != null;
        }

        @Override
        public E next () {
            E res = denne.data;
            denne = denne.neste;  finnNesteIkkeNull();
            return res;
        }
    }
```

Situasjonsbeskrivelse for resten av oppgaven

Du skal lage deler av et Java-program for å administrere en restaurant.

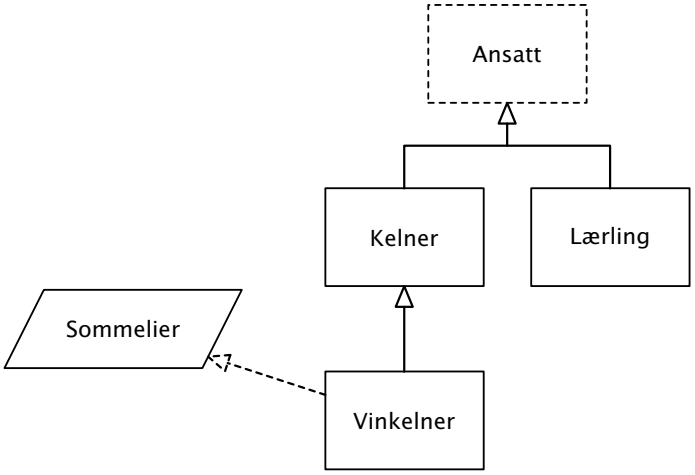
I dette restaurantsystemet er alle ansatte enten kelnere eller lærlinger. Alle ansatte har et navn (en `String`). Noen kelnere har tatt et såkalt *sommelier*-kurs om vin og blitt vinkelner. Egenskapen *sommelier* er definert som et **`interface`**:

```
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`.

Løsningsforslag



Oppgave 4 Klassedeklarasjoner (vekt 10 poeng)

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.

Løsningsforslag

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

abstract class Ansatt {
```

```
    final String navn;

    Ansatt (String id) {
        navn = 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;
    }
}

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

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 5 Søking (vekt 8 poeng)

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.

Løsningsforslag

I klassen `Gjest`:

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

I klassen `Gjestegruppe`:

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

Oppgave 6 Finn beste (vekt 8 poeng)

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.

Løsningsforslag

```
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;
}
```

Her bruker jeg en nyttig hjelperutine i `Kelner` som beregner hvor mange gjester en kelner serverer:

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

og en i `Gjestegruppe` som beregner antallet gjester i gruppen:

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

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*.

Løsningsforslag

Selve metoden `antall` ligger altså i `Samling`-klassen:

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

Den kaller hjelpemetoden `antallFraOgMedDenne` i `Node`-klassen som går rekursivt gjennom alle `Node`-ne:

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

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.

Løsningsforslag

```
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;
    }
```

I denne metoden bruker jeg to hjelperutiner for å gjøre koden mer lettlest (etter min mening): `finnEgnetBord` i klassen `Restaurant` som leter gjennom alle bordene til den finner et ledig bord med nok plasser:

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

og `erLedig` i klassen `Bord` som rett og slett angir om et bord er ledig:

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

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:



Løsningsforslag

`taImotGjester` starter nå slik:

```
    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;
        }
                 ⋮
```

Selve opprettelsen av GUI-vinduet skjer i en egen metode, også den i `Restaurant`-klassen:

```
    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);
    }
```

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`.

Løsningsforslag

Trådklassen `Restaurantbesøk` kan skrives rett frem etter beskrivelsen:

```
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);
    }
}
```

Jeg endret navnet på den modifiserte `Restaurant`-klassen til `RestaurantMonitor` for å unngå forvirring.

```
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 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();
        }
    }

    private Bord finnEgnetBord (int ant) {
        // Trenger ikke låses fordi den er privat.
        // (Da kan den bare kalles fra metoder som allerede er beskyttet.)
        for (Bord bx: bordene) {
            if (bx.erLedig() && bx.antPlasser >= ant)
                return bx;
        }
        return null;
    }

    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();
        }
    }

    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();
        }
    }
}
```