IN1010 våren 2026

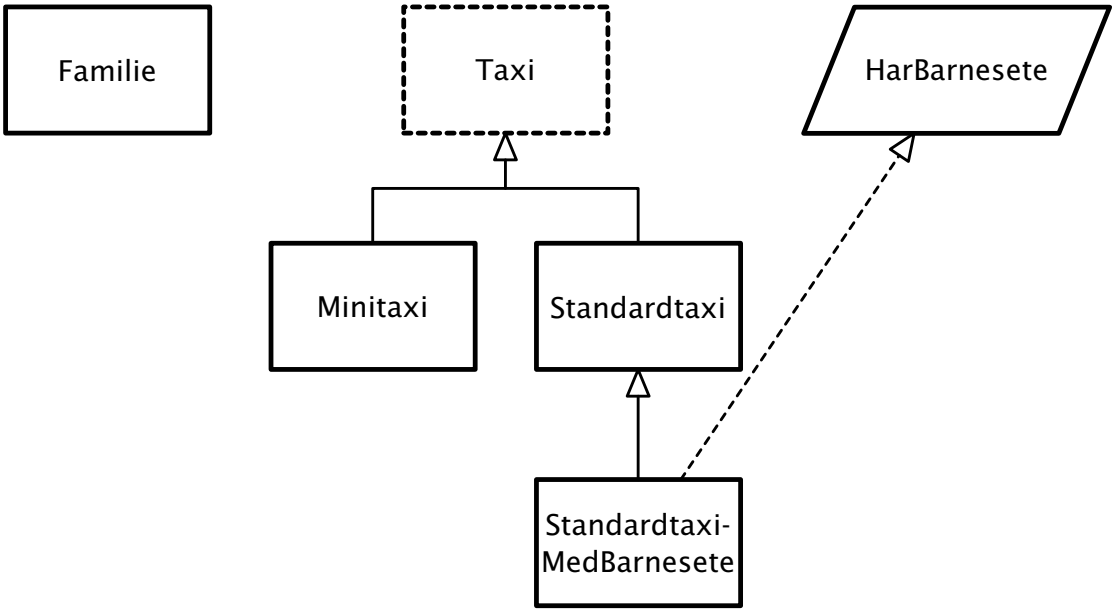
Dag Langmyhr
dag@ifi.uio.no

Løsningsforslag til eksamen 29. mai 2026

Sammendrag

Dette er et løsnings*forslag*. Det finnes mange andre fullgode løsninger.

Oppgave 1



Oppgave 2–8 og 10

```
// Disse import-spesifikasjonene er ikke nødvendig å ha med i besvarelsen.
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/* Oppgave 2 */
abstract class Taxi {
    final int maxAntallPersoner, maxAntallKofferter;
    Taxi forrige, neste;

    Taxi (int maxPers, int maxKoff) {
```

```
        maxAntallPersoner = maxPers;  maxAntallKofferter = maxKoff;
    }

    /* Oppgave 4 */
    Takstinformasjon takster;

    /* Oppgave 3 */
    boolean kanFrakte (Familie fam) {
        // Standardversjon for taxier uten barnesete:
        return fam.antallPersoner <= maxAntallPersoner &&
            fam.antallBarn == 0 &&
            fam.antallKofferter <= maxAntallKofferter;
    }

    /* Oppgave 5 */
    int beregnPris (Familie fam) {
        // Denne metoden fungerer for alle former for taxi,
        //   både med og uten barneseter, siden 'takster.hentTakst("per-barn")'
        //   returnerer 0 om det ikke er noen takst for barneseter.
        return takster.hentTakst("startpris") +
            fam.antallPersoner * takster.hentTakst("per-person") +
            fam.antallBarn * takster.hentTakst("per-barn") +
            fam.antallKofferter * takster.hentTakst("per-koffert") +
            fam.avstand * takster.hentTakst("per-km");
    }
}

class Minitaxi extends Taxi {
    Minitaxi () {
        super(3,1);

        /* Oppgave 4 */
        try {
            takster = new Takstinformasjon("minitaxi-takst.txt");
        } catch (FileNotFoundException e) {
            System.out.println("Kan ikke lese minitaxi-takst.txt.");
            System.exit(1);
        }
    }
}

class Standardtaxi extends Taxi {
    Standardtaxi () {
        super(4,3);

        /* Oppgave 4 */
        try {
            takster = new Takstinformasjon("standardtaxi-takst.txt");
        } catch (FileNotFoundException e) {
            System.out.println("Kan ikke lese standardtaxi-takst.txt.");
            System.exit(1);
        }
    }
}
```

```
class StandardtaxiMedBarnesete extends Standardtaxi implements HarBarnesete {
    int maxAntallBarn;

    StandardtaxiMedBarnesete (int antBarn) {
        maxAntallBarn = antBarn;
    }

    @Override
    public int antallBarneseter () {
        return maxAntallBarn;
    }

    /* Oppgave 3 */
    @Override
    boolean kanFrakte (Familie fam) {
        // Spesialversjon for taxier som kan sette inn barneseter:
        return fam.antallPersoner <= maxAntallPersoner &&
            fam.antallBarn <= maxAntallBarn &&
            fam.antallKofferter <= maxAntallKofferter;
    }
}

interface HarBarnesete {
    int antallBarneseter ();
}

class Familie {
    final int antallPersoner, antallBarn, antallKofferter, avstand;

    Familie (int nPers, int nBarn, int nKoff, int nKm) {
        antallPersoner = nPers;   antallBarn = nBarn;
        antallKofferter = nKoff;  avstand = nKm;
    }
}

/* Oppgave 4 */
class Takstinformasjon {
    private Map<String,Integer> takst = new HashMap<>();

    Takstinformasjon (String filnavn) throws FileNotFoundException {
        Scanner s = new Scanner(new File(filnavn));
        while (s.hasNextLine()) {
            String linje = s.nextLine();
            String[] data = linje.split(":");
            takst.put(data[0], Integer.parseInt(data[1]));
        }
        s.close();
    }

    int hentTakst (String hva) {
        Integer v = takst.get(hva);
        if (v == null) return 0;
        else return v;
```

```
    }
}

/* Oppgave 6 */
class Taxikø {
    Taxi førsteTaxi;

    /* Oppgave 10 */
    Lock lås = new ReentrantLock();
    Condition venterPåTaxi = lås.newCondition();

    void settInnSistIKøen (Taxi tx) {
        lås.lock();
        try {
            if (førsteTaxi == null) {
                førsteTaxi = tx;
            } else {
                Taxi tp = førsteTaxi;
                while (tp.neste != null)
                    tp = tp.neste;
                tp.neste = tx;
                tx.forrige = tp;
            }
            venterPåTaxi.signal();
        } finally {
            lås.unlock();
        }
    }

    Taxi hentBilligsteTaxi (Familie fam) {
        lås.lock();
        try {
            Taxi billigst;
            while (true) {
                ArrayList<Taxi> mulige = finnAlleAktuelle(fam);
                billigst = finnBilligsteTaxi(mulige,fam);
                if (billigst != null) break;
                venterPåTaxi.await();
            }
            fjern(billigst);
            return billigst;
        } catch (InterruptedException e) {
            return null;
        } finally {
            lås.unlock();
        }
    }

    /* Oppgave 6 */
    ArrayList<Taxi> finnAlleAktuelle (Familie fam) {
        ArrayList<Taxi> res = new ArrayList<>();
        Taxi tx = førsteTaxi;
        while (tx != null) {
            if (tx.kanFrakte(fam))
```

```
                res.add(tx);
            tx = tx.neste;
        }
        return res;
    }

    /* Oppgave 7 */
    Taxi finnBilligsteTaxi (ArrayList<Taxi> aktuelle, Familie fam) {
        // Denne metoden starter rekursjonen.
        return finnBilligsteRekursivt(0, aktuelle, fam);
    }

    Taxi finnBilligsteRekursivt (int fraOgMed, ArrayList<Taxi> aktuelle,
                                 Familie fam) {
        // Denne metoden leter rekursivt i 'aktuelle'
        // fra og med indeks 'fraOgMed'.
        if (fraOgMed >= aktuelle.size())
            return null;

        Taxi denne = aktuelle.get(fraOgMed);
        Taxi bestAvResten = finnBilligsteRekursivt(fraOgMed+1, aktuelle, fam);
        if (bestAvResten == null ||
            denne.beregnPris(fam) <= bestAvResten.beregnPris(fam))
            return denne;
        else
            return bestAvResten;
    }

    /* Oppgave 8 */
    void fjern (Taxi tx) {
        // Invariant: tx != null
        if (tx.forrige != null)
            tx.forrige.neste = tx.neste;
        if (tx.neste != null)
            tx.neste.forrige = tx.forrige;
        if (førsteTaxi == tx)
            førsteTaxi = førsteTaxi.neste;
        tx.forrige = tx.neste = null;
    }
}
```

Oppgave 9

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class LagGUIvindu {
    public static void main (String[] args) {
        GUIvindu demovindu = new GUIvindu();
    }
}

class GUIvindu {
```

```
    JFrame vindu;
    JPanel panel;
    JTextField antPersonerSvar = new JTextField(6),
        antSmåbarnSvar = new JTextField(6);

    GUIvindu () {
        try {
            UIManager.setLookAndFeel(
                UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { System.exit(1); }

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

        panel = new JPanel();
        panel.setLayout(new GridLayout(3,2));
        panel.add(new JLabel("Antall personer totalt:"));
        panel.add(antPersonerSvar);
        panel.add(new JLabel("Antall barn under 5 år:"));
        panel.add(antSmåbarnSvar);

        JButton bestill = new JButton("Bestill");
        class SkrivBestilling implements ActionListener {
            @Override
            public void actionPerformed (ActionEvent e) {
                System.out.println("Antall personer totalt: " +
                                   antPersonerSvar.getText());
                System.out.println("Antall småbarn under 5 år: " +
                                   antSmåbarnSvar.getText());
                System.exit(0);
            }
        }
        bestill.addActionListener(new SkrivBestilling());
        panel.add(bestill);
        vindu.add(panel);

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