

# LC191D/L0191D Videregående programmering – mai 2010

## Løsningsforslag

### Oppgave 1

Transporttype er en tekst som er felles for klassene AnnenEgenTransport og Kollektivtransport. Vi legger den derfor i klassen Transport.

Vi lar prisen beregningen være abstrakt i klassen Reiseelement, derfor blir også klassen Transport abstrakt.

Klassene i klassesettet ser nå slik ut:

```
abstract class Reiseelement {  
    public abstract double finnPris(); // metoden får implementasjon i subclassene  
}
```

```
class AndreUtgifter extends Reiseelement {  
    private final String tekst;  
    private final double pris;  
  
    public AndreUtgifter(String tekst, double pris) {  
        this.tekst = tekst;  
        this.pris = pris;  
    }  
  
    public String getTekst() {  
        return tekst;  
    }  
  
    public double finnPris() {  
        return pris;  
    }  
}
```

```
abstract class Transport extends Reiseelement { // REVIDERT UTGAVE  
    private final Etappe etappe;  
    private final String transportmiddel;  
  
    public Transport(Etape etappe, String transportmiddel) {  
        this.etappe = etappe;  
    }  
}
```

```

    this.transportmiddel = transportmiddel;
}

public String getTransportmiddel() {
    return transportmiddel;
}
}

class Kollektivtransport extends Transport {
    private final double pris;

    public Kollektivtransport(Etappe etappe, String transportmiddel, double pris) {
        super(etappe, transportmiddel);
        this.pris = pris;
    }

    public double finnPris() {
        return pris;
    }
}

class Transporttype { // HJELPEKLASSE, se kommentar nedenfor
    private final String type;
    private final double pris;

    public Transporttype(String type, double pris) {
        this.type = type;
        this.pris = pris;
    }

    public String getType() {
        return type;
    }

    public double getPris() {
        return pris;
    }
}

class AnnenEgenTransport extends Transport {
    private final double antKm;

    public static Transporttype[] TRANSPORT_TYPER =
        {new Transporttype("motorsykkkel", 2.8), new Transporttype("moped", 1.55),
        new Transporttype("snøscooter", 6.6),

```

```

    new Transporttype("båt med motor minst 50 hk", 6.5),
    new Transporttype("båt med motor under 50 hk", 3.5),
    new Transporttype("EL-bil", 4.0), new Transporttype("Annet", 1.5));

public AnnenEgenTransport(Etappe etappe, String transportmiddel, double antKm) {
    super(etappe, transportmiddel);
    this.antKm = antKm;
}

public double finnPris() {
    return antKm * finnPrisPrKm(getTransportmiddel());
}

private static double finnPrisPrKm(String type) { // hjelpemetode
    for (int i = 0; i < TRANSPORT_TYPER.length; i++) {
        if (TRANSPORT_TYPER[i].getType().equals(type)) {
            return TRANSPORT_TYPER[i].getPris();
        }
    }
    return 0.0;
}
}

```

**Kommentar til klassene `Transporttype` og `AnnenEgenTransport`:**

Studenten bør lage en datastruktur for de ulike transporttypene. En rekke med if-else-setninger er ikke akseptabelt. Alle noenlunde fornuftige strukturer er vurdert omtrent likeverdige. Det mest elegante er nok å bruke ENUM – det ble imidlertid før eksamen opplyst at det ikke var pensum, men det tas med her for eksemplets skyld:

```

enum Transporttype {
    MOTORSYKKEL("motorsyssel", 2.8), MOPED("moped", 1.55), SNØSCOOTER("snøscooter", 6.6),
    BÅT_MED_STOR_MOTOR("båt med motor minst 50 hk", 6.5),
    BÅT_MED_LITEN_MOTOR("båt med motor under 50 hk", 3.5),
    EL_BIL("EL-bil", 4.0), ANNET("Annet", 1.5);

    Private final String navn;
    private final double godtgjPrKm;
    Transporttype(String navn, double godtgjPrKm) {
        this.navn = navn;
        this.godtgjPrKm = godtgjPrKm;
    }

    public String toString() { // objektnavnet som tekst
        return navn;
    }
}

```

```

public double getGodtgjPrKm() {
    return godtgjPrKm;
}
}

```

```

class AnnenEgenTransport extends Transport {
    private final double antKm;
    private final double godtgjPrKm;

```

```

public AnnenEgenTransport(Etappe etappe, Transporttype type, double antKm) {
    super(etappe, type.toString());
    this.godtgjPrKm = type.getGodtgjPrKm();
    this.antKm = antKm;
}

```

```

public double finnPris() {
    return antKm * godtgjPrKm;
}
}

```

Vi lager et objekt av klassen **AnnenEgenTransport** slik:

```

Reiseelement r = new AnnenEgenTransport(
    new Etappe("kl 0810 10.01.2010", "kl 0900 10.01.2010", "Oslo S", "Fugleøya"),
    Transporttype.BÅT_MED_STOR_MOTOR, 10);

```

## Oppgave 2

Klassen **Person**:

```

public double finnPrisAlleReiser() {
    double sum = 0.0;
    for (Reise r : reiser) {
        sum += r.finnReisensTotalpris();
    }
    return sum;
}

```

```

public int finnTotaltAntOvernattinger() {
    int ant = 0;
    for (Reise r : reiser) {
        ant += r.finnAntOvernattinger();
    }
    return ant;
}

```

Klassen **Reise**:

```
public static final double DAGPRIS1 = 270.0;
public static final double DAGPRIS2 = 580.0;

public int finnAntOvernattinger() {
    return Math.abs(startDato.dagerForskjell(sluttDato));
}

public double finnKostgodtgjØrse() {
    int antDager = finnAntOvernattinger();
    return (antDager == 0) ? DAGPRIS1 : DAGPRIS2 * (antDager + 1);
}

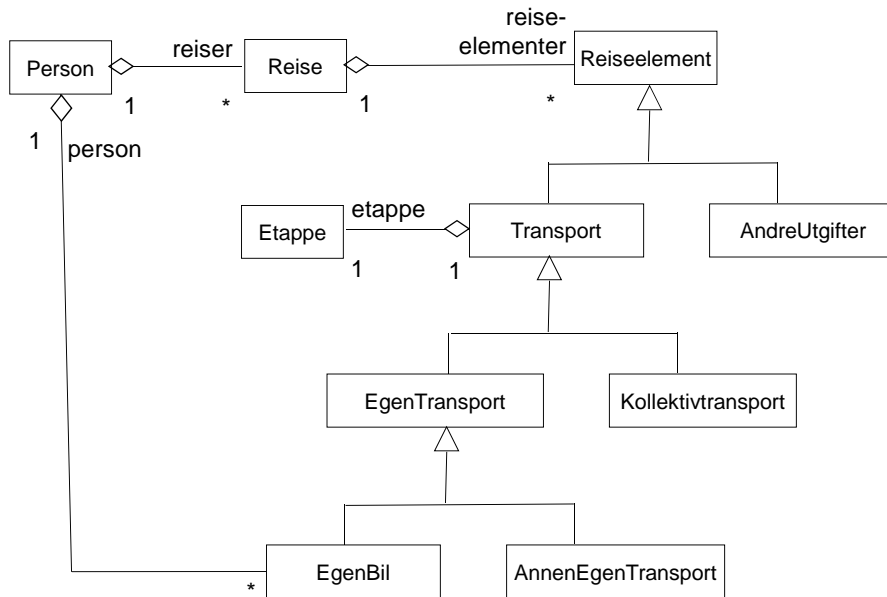
public double finnNattTillegg() {
    return prisPrNatt * finnAntOvernattinger();
}

public double finnReisensTotalpris() {
    double sum = 0.0;
    for (Reiseelement re : reiseelementer) {
        sum += re.finnPris();
    }
    return sum + finnKostgodtgjØrse() + finnNattTillegg();
}
```

### Oppgave 3

Reglene for bruk av privatbil var dessverre så dårlig formulert, at det måtte tas hensyn til dette ved sensureringen. Reglene er slik at grensen på 9000 km gjelder pr. person, og da slik å forstå at **en må summere alle reisene for denne personen før grensen kan sjekkes**. (Pussig at ingen spurte om dette på eksamen, å kjøre 9000 km på en og samme tur innenlands er da ganske uvanlig, med det ble vel bare et tall for dere uten at dere tenkte mer over det. Av 62 studenter var det kun 4 som hadde skjØnt hva oppgaven innebar ... og for de av disse som hadde programmert dette, ble det regnet at de hadde mindre tid enn de Øvrige til å gjøre resten av oppgavesettet.)

Dere er ikke bedt om å tegne revidert klassesdiagram, men det kan se slik ut:



Klassen EgenBil er ny, og vi får antall kilometer som felles attributt for klassene EgenBil og AnnenEgenTransport. Denne velger vi å generalisere ut og legge i en egen superklasse, klassen EgenTransport, felles for disse to klassene. Videre må vi kople klassen EgenBil til Person slik at vi kan holde orden på antall kilometer den enkelte har fått godtgjørelse for.

Klassene EgenTransport og EgenBil ser dermed slik ut:

```

abstract class EgenTransport extends Transport {
    private final double antKm;

    public EgenTransport(Etappe etappe, String transportmiddel, double antKm) {
        super(etappe, transportmiddel);
        this.antKm = antKm;
    }

    public double getAntKm() {
        return antKm;
    }
}

```

```

class EgenBil extends EgenTransport {
    public static double GRENSE_EGENBIL = 9000.0;
    public static double PRIS1 = 3.65;
    public static double PRIS2 = 3.0;
    private Person person;
    private double beregnetPris;
}

```

```

public EgenBil(Etappe etappe, double antKm, Person person) {
    super(etappe, "Egen bil", antKm);
    this.person = person;
    beregnPris(); // må gjøre det her ettersom algoritmen er avhengig av antall km kjørt
}

public double finnPris() {
    return beregnetPris;
}

private void beregnPris() {
    double antKmHittil = person.getAntKmHittil();
    double nyAntKmHittil = antKmHittil + getAntKm();
    double antKmPris1 = 0.0;
    double antKmPris2 = 0.0;
    if (nyAntKmHittil <= GRENSE_EGENBIL) {
        antKmPris1 = getAntKm();
    } else if (antKmHittil >= GRENSE_EGENBIL) {
        antKmPris2 = getAntKm();
    } else { // litt av hvert
        antKmPris1 = GRENSE_EGENBIL - antKmHittil;
        antKmPris2 = nyAntKmHittil - GRENSE_EGENBIL;
    }
    beregnetPris = antKmPris1 * PRIS1 + antKmPris2 * PRIS2;
    person.setAntKmHittil(nyAntKmHittil);
}
}

```

Klassen **AnnenEgenTransport** blir som før, men nå altså subklasse til **EgenTransport**.

I klassen **Person** har vi følgende utvidelser:

```

private double antKmHittil = 0.0;
public double getAntKmHittil() {
    return antKmHittil;
}
public void setAntKmHittil(double nyAntKmHittil) {
    antKmHittil = nyAntKmHittil;
}

```

## Oppgave 4

```
class GUI extends JFrame {
    private Person person;
    private JList oversiktsliste = new JList();
    private JScrollPane rulleOversiktsliste = new JScrollPane(oversiktsliste);
    private JList detaljliste = new JList(); // initieres som tom liste
    private JScrollPane rulleDetaljliste = new JScrollPane(detaljliste);
    public GUI(Person person) {
        this.person = person;
        setTitle("Ekamen mai 2010");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(2, 1, 5, 5));
        initierOversiktsliste();
        initierDetaljliste();
        add(rulleOversiktsliste);
        oversiktsliste.addListSelectionListener(new Listelytter());
        add(rulleDetaljliste);
        pack();
    }

    private class Listelytter implements ListSelectionListener {
        public void valueChanged(ListSelectionEvent hendelse) {
            int indeks = oversiktsliste.getSelectedIndex();
            if (indeks >= 0) {
                String[] detaljer = person.hentDetaljer(indeks); // se nedenfor
                detaljliste.setListData(detaljer);
            }
        }
    }

    private void initierOversiktsliste() {
        String[] s = person.lagOversikt();
        oversiktsliste.setListData(person.lagOversikt()); // se nedenfor

        JViewport jvp = new JViewport();
        jvp.setView(new JLabel("Reiser foretatt av " + person.getNavn())); // tabelloverskrift
        rulleOversiktsliste.setColumnHeader(jvp); // ok om denne ikke var med i besvarelsen
        oversiktsliste.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    }

    private void initierDetaljliste() {
        JViewport jvp = new JViewport();
        jvp.setView(new JLabel("Detaljer, valgt reise")); // tabelloverskrift
        rulleDetaljliste.setColumnHeader(jvp); // ok om denne ikke var med i besvarelsen
    }
}
```

```

    detaljliste.setEnabled(false); // låser detaljlisten for brukeren
}
}

```

Ang. **JList**: Det er helt ok om studenten brukte listemodell for datainnholdet. Men akkurat i dette tilfellet er det unødvendig ettersom hele listen skal byttes ut. Til det bruket har vi metoden `setListData()`. Det som imidlertid ikke fungerer er å opprette et nytt `JList`-objekt. For husk at det er det første `JList`-objektet som er lagt inn i listen med GUI-komponenter. Skisse:

```

JList detaljliste = new JList();
JScrollPane rulleDetaljliste = new JScrollPane(detaljliste);
add(rulleDetaljliste); // her legges JList-objektet inn i GUI-beholderen
detaljliste = new JList(...); // dette endrer ikke innholdet i GUI-beholderen

```

Prinsippet er det samme som for eksempelvis `ArrayList`. Kun referansene kopieres inn, ikke det de peker til.

### Å få fram teksten i øverste liste (formattering ikke del av oppgaven)

Klassen `Person`:

```

public String[] lagOversikt() {
    String[] liste = new String[reiser.size()];
    for (int i = 0; i < reiser.size(); i++) {
        liste[i] = reiser.get(i).toString();
    }
    return liste;
}

```

Klassen `Reise`:

```

public String toString() {
    java.util.Formatter f = new java.util.Formatter();
    f.format("Kost: NOK %.2f, natt: NOK %.2f, totalpris: NOK %.2f", finnKostgodtgjorelse(),
        finnNattTillegg(), finnReisensTotalpris()); // fra oppgave 2
    return hensikt + ", periode: " + startDato.format() + " - " + sluttDato.format() + ". "
        + f.toString() + ".";
}

```

### Å få fram teksten i nederste liste (formattering ikke del av oppgaven)

Klassen `Person`:

```

public String[] hentDetaljer(int indeks) {
    if (indeks >= 0 && indeks < reiser.size()) {
        return reiser.get(indeks).hentReisedetaljer();
    }
}

```

```
}  
return new String[0];  
}
```

#### Klassen Reise:

```
public String[] hentReisedetaljer() {  
    String detaljer[] = new String[reiseelementer.size()];  
    for (int i = 0; i < reiseelementer.size(); i++) {  
        detaljer[i] = reiseelementer.get(i).finnDetaljer();  
    }  
    return detaljer;  
}
```

#### Klassen Reiseelement:

```
public abstract String finnDetaljer();
```

#### Klassen AndreUtgifter:

```
public String finnDetaljer() {  
    java.util.Formatter f = new java.util.Formatter();  
    f.format("%.2f", finnPris());  
    return "Andre utgifter: " + tekst + ": NOK " + f.toString();  
}
```

#### Klassen Transport:

```
public String finnDetaljer() {  
    java.util.Formatter f = new java.util.Formatter();  
    f.format("%.2f", finnPris());  
    return "Transport: " + transportmiddel + ", tid fra - til: " + etappe.getStartTid() + " - "  
        + etappe.getSluttTid() + ", sted fra - til: " + etappe.getFraSted() + " - " + etappe.getTilSted()  
        + ", NOK: " + f.toString() + ".";  
}
```

#### Kommentar ang. toString() :

Noen har deklært toString() abstrakt i en av klassene, det er ikke tillatt – ettersom den allerede har en standardimplementasjon (som arves, dersom du ikke lager din egen).