



## HØGSKOLEN I SØR-TRØNDELAG

Avdeling for informatikk og e-læring -

AITeL

<b>Kandidatnr:</b>	
<b>Eksamensdato:</b>	6.mai 2009
<b>Varighet:</b>	0900-1300
<b>Emnekode:</b>	LO191D / LC191D
<b>Emnenavn:</b>	LO191D Videregående programmering (i Java), nettbasert LC191D Videregående programmering, campus
<b>Klasse(r):</b>	HING2008HA, nettstudenter, div. gjentak
<b>Studiepoeng:</b>	6
<b>Faglærer(e):</b>	Else Lervik
<b>Kontaktperson (adm.)</b>	Ingrid Island
<b>Hjelpemidler:</b>	Lærebøker, alle håndskrevne og trykte hjelpemidler.
<b>Oppgavesettet består av:</b>	4 oppgaver og 6 sider (inkludert forside og vedlegg)
<b>Vedlegg består av:</b>	2 sider
<b>Merknad: Oppgaveteksten kan beholdes av studenter som sitter eksamenstiden ut.</b>	
<b>Lykke til!</b>	

## **Les dette!**

All programmering skal skje i Java.

Det er tillatt å ha flere medlemmer i klassene enn det som er oppgitt i teksten.

Les gjennom hele oppgavesettet før du begynner å programmere. Pass på at du ikke gjør verken mer eller mindre enn det oppgavene spør etter. Men dersom du trenger flere metoder/konstruktører for å lage det oppgavene spør etter, skal du også programmere disse. En fornuftig oppdeling i metoder ut over det oppgaven spør etter, kan gi plusspoeng ved bedømmelsen.

Dersom du mener det mangler opplysninger, sett dine egne forutsetninger.

Oppgavene er laget slik at de alle sammen arbeider med den samme problemstillingen. Det vil derfor være slik at du i en oppgave må bruke klasser/metoder du skal ha laget i en annen oppgave. Dersom du av en eller annen grunn ikke har laget disse klassene/metodene, kan du, når du skal løse andre oppgaver, anta at de eksisterer.

## **Felles problemstilling for alle oppgavene**

Politikerne i ”*Nordens Miljøby*” har vedtatt nye regler for bomringen rundt byen. Hensikten er på den ene siden å redusere luftforurensningen og på den andre siden å bedre folks helse ved å premiere de som sykler eller går. Premieringen skjer ved at de får utbetalt en del av pengene som er kommet inn på bomstasjonene.

Innbyggerne betaler pr passering gjennom bomringen. De kan inngå én av følgende tre avtaler med bompengeselskapet:

### **1. Avtale Bilist**

Her betales en fast pris pr passering uavhengig av antall passeringer i måneden.

### **2. Avtale Kollektivist**

I denne avtalen inngår svært billig månedskort for byens kollektivtrafikk. (Månedskortene er ikke en del av denne oppgaven.)

Dersom en person med denne avtalen likevel må kjøre bil en dag, skal han betale etter følgende regler:

Han får et visst antall passeringer pr måned til en fast, relativt høy, pris. Deretter øker prisen med 10 prosentpoeng for hver passering.

Eksempel: De fem første passeringene koster 30 kr hver.

Passering nr 6 koster  $kr\ 30,00 * 1,10 = kr.\ 33,00$ .

Passering nr 7 koster  $kr\ 30,00 * 1,20 = kr.\ 36,00$ .

Passering nr 8 koster  $kr\ 30,00 * 1,30 = kr.\ 39,00$ .

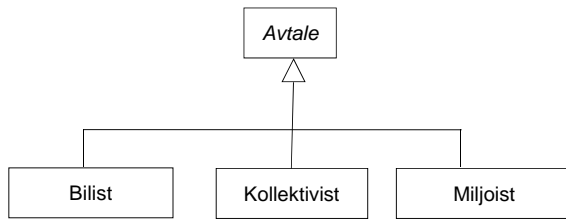
.. osv. ..

### **3. Avtale Miljoist**

Disse personene forplikter seg til å sykle og gå. De tjener opp såkalte miljøpoeng avhengig av antall ganger de sykler /går til byen, og hvor langt unna de bor. (Vi ser bort fra eventuelle problemer med kontroll her.) Hver gang systemet foretar avregning, får disse personene en viss prosentdel av bompengene som er betalt inn. Disse pengene fordeles etter antall miljøpoeng den enkelte har

opptjent. Hvis en person i denne kategorien kjører bil, må de betale dobbelt så mye som Kollektivistene.

## Oppgave 1 – vekt 20%



Figuren viser et klassetre som modellerer de ulike avtalene. Klassen *Avtale* er gitt i vedlegg 1. Denne klassen vil det være aktuelt å utvide for å løse oppgavene.

Hver avtale har et entydig nummer. Som det framgår av vedlegg 1 bruker vi her en klassevariabel som økes med 1 hver gang et nytt objekt blir laget. Dette gir entydighet, men ingen garanti for at alle nummer er i bruk.

I tillegg til nummeret skal vi lagre kun epostadressen til kunden. Epostadressen vil selvfølgelig være entydig. (For enkelhets skyld lagrer vi altså verken navn eller adresse eller andre personopplysninger.)

Programmer de tre subclassene, og utvid eventuelt klassen *Avtale* etter følgende spesifikasjoner:

- Alle klassene skal ha konstruktører som tar epostadressen som argument. I tillegg skal det være mulig å hente ut avtalenummeret og saldoen for påløpt bomavgift.
- Det må være mulig å registrere en passering gjennom bomringen. Da skal saldoen økes i henhold til reglene foran.

## Oppgave 2 – vekt 35%

Alle avtalene skal samles i en *ArrayList* i klassen *Avtaler*:

```
class Avtaler {
    private ArrayList<Avtale> avtaler = new ArrayList<Avtale>();
```

Lag følgende tre metoder i klassen *Avtaler*:

- a) En metode som registrerer en ny avtale. Metodehodet skal være som følger:

```
public boolean registrerNyAvtale(Avtale nyAvtale)
```

Metoden skal returnere *false* dersom avtale med denne epostadresse og/eller nr er registrert fra før.

- b) En metode som registrerer passeringer med privatbil gjennom bomringen. Metoden skal ha følgende hode:

```
public boolean[] registrerPasseringer(int[] nr)
```

Argumentet skal være en tabell med avtalenummer. Returnerdiene skal fortelle klienten hvilken av avtalene som ble registrert. (Verdi *false* vil bety at det aktuelle avtalenummeret ikke fins.)

c) En metode som gjør opp status.

Denne metoden kjøres en gang i måneden, og den skal gå gjennom alle avtalene og regne ut hva hver enkelt er skyldig (positivt beløp), eventuelt har til gode (negativt beløp, gjelder kun ”miljøister”).

Regler for avregning:

30% av de opptjente bompengene skal fordeles mellom ”miljøistene” avhengig av antall miljøpoeng opptjent. Eksempel: Hvis en person har 10% av totalt antall miljøpoeng denne måneden skal han også ha 10% av ”premien”.

Resultatet skal returneres i en *ArrayList* med objekter som inneholder avtalenummer, epost og beløp.

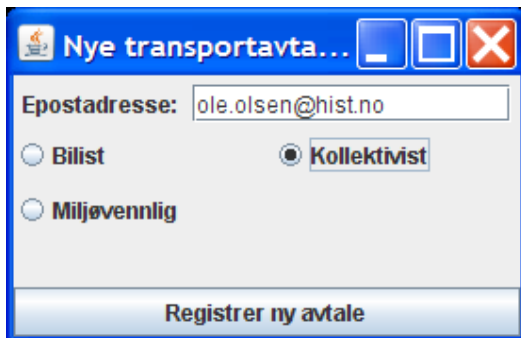
All informasjon knyttet til avregningen skal nullstilles for alle avtalene.

### Oppgave 3 – vekt 20%

Alle avtalene i et objekt av klassen *Avtaler* (oppgave 2) skal lagres på fil mellom hver kjøring av programmet. Skriv kodebiter (eventuelt metoder) som lagrer avtalene på fil, og som leser dem inn igjen. Kodebitene skal tas i bruk i oppgave 4.

### Oppgave 4 – vekt 25%

I vedlegg 2 finner du utdrag av et enkelt grafisk brukergrensesnitt for å registrere nye avtaler:



Som kjent så er det i Java mulig å vise alle knappene i en knappgruppe som ”ikke valgt” kun første gangen de vises i et vindu. Men hvis brukeren først velger en knapp, er det ikke mulig ”å slå av” alle knappene. Som det framgår av koden i vedlegg 2, så inneholder knappgruppen derfor fire knapper hvorav en av knappene (*knappAvtaleIkkeValgt*) er usynlig. Denne skal settes av programmet etter at en avtale er registrert. Dermed kan vi tvinge brukeren til hver gang bevisst å velge en av de tre avtalene.

Framvisning av vinduet skal sørge for at alle registrerte avtaler leses fra fil og inn i et objekt av klassen *Avtaler*. Brukeren kan så registrere flere avtaler. Hver registrering skal gi tilbakemelding om nummeret til den nye avtalen. Programmet skal lagre alle avtalene på fil i det brukeren avslutter programmet.

Programmer det som mangler i koden, og husk å vise hvor koden fra oppgave 3 skal plasseres. I koden er det skissert tre steder det kan være aktuelt å legge inn mer. Men du kan også legge inn kode andre steder. Bruk linjenumrene for referanse i besvarelsen.

## Vedlegg 1 – klassen Avtale – utvides ved behov

```
abstract class Avtale {
    /* Navngitte konstanter til bruk ved koding */
    public static final double PRIS_PASSERING_PROGRESSIV = 30.0;
    public static final int GRENSE_PROGRESSIV = 5;
    public static final double FAKTOR_PROGRESSIV = 0.1;
    public static final double PRIS_PASSERING_ORDINÆR = 20.0;

    private static int maksnr = 0; // for å lage entydig avtalenummer
    private int nr; // entydig pga maksNr som oppdateres i konstruktøren
    private String epost; // entydig
    private double saldo = 0.0;

    public Avtale(String startEpost) {
        maksnr++;
        nr = maksnr;
        epost = startEpost;
    }
    public static int finnMaksnr() {
        return maksnr;
    }
    public static void settMaksnr(int nyttMaksnr) {
        maksnr = nyttMaksnr;
    }
    public int finnNr() {
        return nr;
    }
    public String finnEpost() {
        return epost;
    }
    public double finnSaldo() {
        return saldo;
    }
    public void settSaldo(double nySaldo) {
        saldo = nySaldo;
    }
    public String toString() {
        return getClass().getName() + ", epost: " + epost + ", nr " + nr;
    }
}
```

## Vedlegg 2

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4. import static javax.swing.JOptionPane.*;
5. class GUI extends JFrame {
6.     private JRadioButton knappBilist = new JRadioButton("Bilist");
7.     private JRadioButton knappKollektivist = new JRadioButton("Kollektivist");
8.     private JRadioButton knappMiljø = new JRadioButton("Miljøvennlig");
9.     private JRadioButton knappAvtaleIkkeValgt = new JRadioButton();
10.    private JTextField epostfelt = new JTextField(15);
11.    private JButton registrerKnapp = new JButton("Registrer ny avtale");
12.    private Avtaler avtaler;
```

*... det kan være det mangler noen deklarasjoner ....*

```
13. public GUI() {
14.     setTitle("Nye transportavtaler");
15.     setLayout(new BorderLayout());
```

*... det kan være aktuelt å putte inn noe mer i konstruktøren ...*

```
16.     add(new ToppPanel(), BorderLayout.NORTH);
17.     add(new TypeAvtalePanel(), BorderLayout.CENTER);
18.     add(registrerKnapp, BorderLayout.SOUTH);
19.     pack();
20. }
21. private class ToppPanel extends JPanel {
22.     public ToppPanel() {
23.         setLayout(new FlowLayout());
24.         add(new JLabel("Epostadresse: "));
25.         add(epostfelt);
26.         epostfelt.requestFocus();
27.     }
28. }
29. private class TypeAvtalePanel extends JPanel {
30.     public TypeAvtalePanel() {
31.         ButtonGroup gruppe = new ButtonGroup();
32.         gruppe.add(knappBilist);
33.         gruppe.add(knappKollektivist);
34.         gruppe.add(knappMiljø);
35.         gruppe.add(knappAvtaleIkkeValgt);
36.         knappAvtaleIkkeValgt.setVisible(false);
37.         setLayout(new GridLayout(3, 1, 5, 5));
38.         add(knappBilist);
39.         add(knappKollektivist);
40.         add(knappMiljø);
41.         add(knappAvtaleIkkeValgt);
42.     }
43. }
```

*... og så mangler det vel noe her ....*

```
44. } // GUI

45. class TestAvtale {
46.     public static void main(String[] args) {
47.         GUI vindu = new GUI();
48.         vindu.setVisible(true);
49.     }
50. } // TestAvtale
```