



HØGSKOLEN I SØR-TRØNDELAG

Avdeling for informatikk og e-læring - AITeL

Kandidatnr:	
Eksamensdato:	1. desember 2010
Varighet:	0900-1300
Emnekode:	LO191D / LC191D
Emnenavn:	LO191D Videregående programmering (i Java), nettbasert LC191D Videregående programmering, campus
Klasse(r):	nettstudenter, div. gjentak
Studiepoeng:	6
Faglærer(e):	Else Lervik
Kontaktperson (adm.)	Ingrid Island
Hjelpemidler:	Lærebøker, alle håndskrevne og trykte hjelpemidler.
Oppgavesettet består av:	4 oppgaver og 8 sider (inkludert forside og vedlegg)
Vedlegg består av:	2 sider
Merknad: Oppgaveteksten kan beholdes av studenter som sitter eksamenstiden ut.	
Lykke til!	

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 du introduseres til en problemstilling nedenfor. Denne problemstillingen gjelder for oppgave 1, 2 og 3. I oppgave 4 justeres problemstillingen noe. Det vil kunne være slik at du i en oppgave med fordel kan bruke klasser/metoder du skal ha laget i en tidligere 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.

Eventuell feilsjekking skal utføres, men dersom annet ikke er oppgitt, skal feilene behandles på enkleste måte, dvs. at du kan la metodene returnere `-1`, `false` eller `0`, avhengig av returtypen. Du trenger ikke å utføre feilkontroll i konstruktører eller i `set`-metoder.

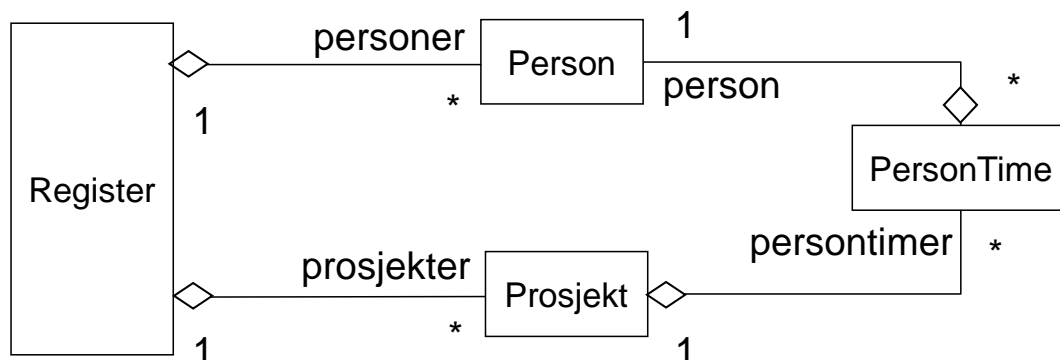
Du trenger ikke sette opp `import`-setninger.

Felles problemstilling for oppgave 1, 2 og 3

I vedlegg 1 finner du klassene `Person` og `PersonTime`. Disse klassene skal brukes i kode som håndterer deler av et enkelt prosjektverktøy.

Et objekt av klassen `Person` inneholder informasjon knyttet til en person. Personen identifiseres med nr (`persNr`).

Et objekt av klassen `PersonTime` forteller hvor mange timer en person totalt har jobbet. I oppgaven skal du lage klassen `Prosjekt` (mer om det nedenfor), til hvert prosjektobjekt skal det kunne knyttes mange `PersonTime`-objekter. Der vil man altså kunne finne ut hvor mange timer hver enkelt prosjektdeltaker har arbeidet på prosjektet.



Både personer og prosjekter skal samles i et register. Figuren over viser sammenhengen mellom klassene.

Oppgave 1 – vekt 25%

Programmer klassen *Prosjekt* etter følgende spesifikasjoner:

Klassen skal ha følgende objektvariabler:

```
private final int prosjNr; // prosjektidentifikasjon
private final String prosjNavn;
private ArrayList<PersonTime> persontimer = new ArrayList<PersonTime>();
```

- Lag en konstruktør som tar prosjektnr., navn og en *ArrayList* med *PersonTime*-objekter som argument. Konstruktøren skal sørge for at *PersonTime*-objektene er beskyttet fra eventuelle endringer utenfor *Prosjekt*-klassen.
- Lag *toString()*-metode som gir utskrift omtrent på følgende format (formatering av desimaltall er ikke viktig):

```
Prosjekt: 1234 Testprosjekt med deltakere:
3: Eva Jensen, kr. 250,00, antall timer: 10,5
5: Berit Iversen, kr. 295,00, antall timer: 15,0
6: Ingrid Bø, kr. 305,00, antall timer: 20,5
```

Her er prosjektnummeret *1234*, prosjektnavnet er *Testprosjekt*, og dette prosjektet har knyttet til seg tre personer som har jobbet henholdsvis 10,5, 15,0 og 20,5 timer på prosjektet.

toString()-metoden som du lager, skal være en omdefinering av metoden *toString()* arvet fra klassen *Object*.

- Lag *equals()*-metode som returnerer *true* dersom prosjektnr og/eller navn er like. Metoden som du lager, skal være en omdefinering av metoden *equals()* arvet fra klassen *Object*.
- Lag en metode som regner ut prosjektkostnadene ved å løpe gjennom alle *PersonTime*-objektene og summerer opp antall timer * timelønn * faktor, der *faktor* sørger for påslag som svarer til feriepenger, arbeidsgiveravgift og fortjeneste. Eksempel: Med prosjektdeltakerne fra utskriften i oppgave b) og faktor = 1,5 ser regnestykket slik ut:

$$\text{sum} = ((250 * 10,5) + (295 * 15,0) + (305 * 20,5)) * 1,5$$

Metoden skal ha følgende hode:

```
public double finnTimekostnader(double faktor)
```

Oppgave 2 – vekt 20%

Du skal lage metoder i klassen *Register*. Klassen har disse objektvariablene:

```
private ArrayList<Person> personer = new ArrayList<Person>();  
private ArrayList<Prosjekt> prosjekter = new ArrayList<Prosjekt>();
```

De tre metodene du skal lage har følgende spesifikasjoner:

Metode a)

Metodehode:

```
public Person[] hentPersoner()
```

Metoden skal returnere en tabellreferanse til objektene som ligger i ArrayListen *personer*.

Svar også på følgende spørsmål: Er det noen grunn til å bruke dyp kopiering her? Begrunn svaret.

Metode b)

Metodehode:

```
public boolean registrerNyttProsjekt(Prosjekt nyttProsjekt)
```

Metoden skal legge prosjektet inn i ArrayListen *prosjekter* forutsatt at prosjekt med dette nummer og/eller navn ikke er registrert fra før. Metoden skal returnere *true* eller *false* avhengig av om prosjektet ble registrert i listen.

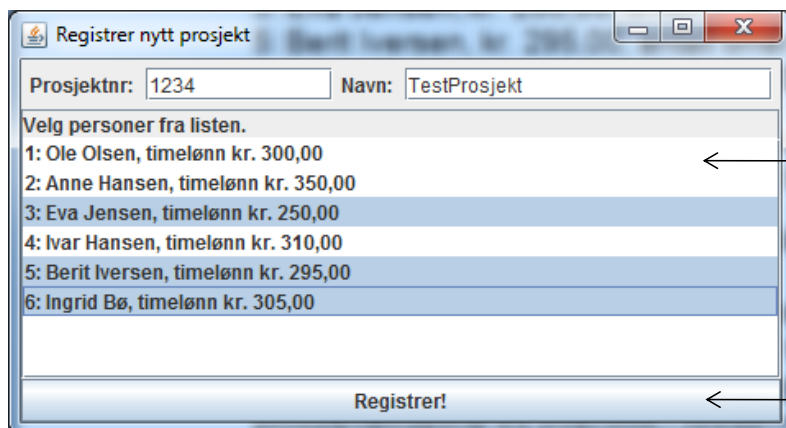
Metode c)

Metodehode:

```
public double finnTotaleTimekostnader(double faktor)
```

Metoden skal regne ut timekostnadene for alle registrerte prosjekter. Parameteren *faktor* er lik for alle prosjektene og er den samme som i oppgave 1d)

Oppgave 3 – vekt 20%



Listen skal vise alle objektene i Array-Listen "personer" i klassen Register

Denne knappen skal knyttes til en ActionListener

I denne oppgaven skal du jobbe med dette brukergrensesnitt for å registrere et nytt prosjekt. Brukeren skal skrive inn prosjektnummer og –navn i de to øverste feltene. Deretter skal prosjektdeltakerne velges fra listen. (Det skal være mulig å ikke velge noen prosjektdeltakere.)

Til slutt trykker brukeren på knappen *Registrer!*.

Brukeren skal få tilbakemelding (bruk enkel showMessageDialog) hvis

- Prosjektet ble registrert
- Prosjektet ikke ble registrert, og med begrunnelse
 - Prosjektnummeret ikke er et heltall.
 - Navnefeltet ikke er fylt ut.
 - Prosjekt med dette nummer og/eller navn er registrert fra før.

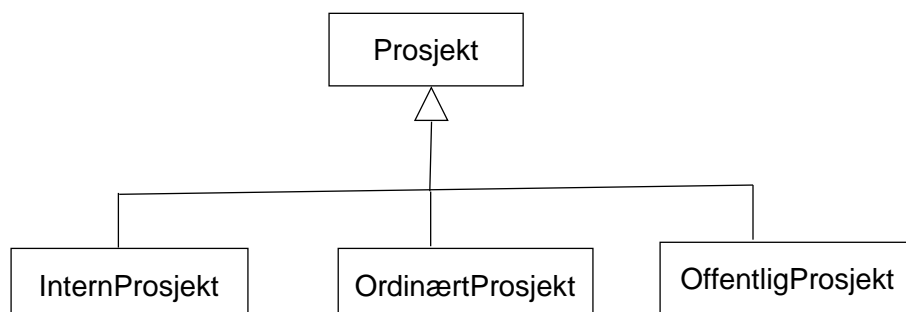
Du finner utsnitt av koden i vedlegg 2.

Du skal programmere det som mangler, det vil si:

- Noen linjer i GUI-konstruktøren
- Innholdet i metoden *actionPerformed()*.

Oppgave 4 – vekt 35%

Her reviderer vi problemstillingen noe.



Vi skal nå klassifisere prosjektene i tre kategorier som vist i klassesdiagrammet over.

Avhengig av prosjekttype beregnes nå prosjektkostnadene noe forskjellig. Framgangsmåten er imidlertid den samme som i oppgave 1d), ved at en summerer over alle personer.

Her følger forskjellene:

Internt prosjekt: Prosjektkostnadene beregnes som i oppgave 1d), men med en fast faktor på 1.4.

Ordinært prosjekt: Her forhandles det fram en faktor for hvert enkelt prosjekt. Denne lagres i et objekt av klassen *OrdinærtProsjekt*. For øvrig er beregningene som i oppgave 1d).

Offentlig prosjekt: Dette er prosjekt med offentlig oppdragsgiver. Kostnadene er ikke direkte avhengig av personens lønn, men knyttet til personens formelle kompetanse. Anta at det i *Person*-klassen eksisterer en metode *finnKompetanse()* som gir en tallverdi som symboliserer kompetansen (du skal ikke programmere metoden *finnKompetanse()*). Timekostnadene er gitt av kompetanseverdien:

- Kompetanse 1 eller 2: Timekostnader 600 kr.
- Kompetanse 3 eller 4: Timekostnader 700 kr.
- Kompetanse 5 eller høyere: Timekostnader 900 kr.

Oppgave

Klassen *Prosjekt* har nå metoden *finnTimekostnader()* uten parameter.

Programmer de tre subclassene slik at forskjellene beskrevet over framkommer, og ta også med eventuelle endringer som må gjøres i klassene *Prosjekt* og *Register* for at dette skal fungere som et system der det er mulig å legge inn prosjekter av forskjellig type.

Hvordan kan du på enkleste måte få inn dette valget i klassen GUI fra oppgave 3? Tegn en skisse av revidert brukergrensesnitt, og vis hvilke endringer som må gjøres i koden.

VEDLEGG 1

```
class Person {
    private final int persNr;
    private final String navn;
    private final double timelønn;

    public Person(int persNr, String navn, double timelønn) {
        this.persNr = persNr;
        this.navn = navn;
        this.timelønn = timelønn;
    }

    public int getPersNr() {
        return persNr;
    }

    public String getNavn() {
        return navn;
    }

    public double getTimelønn() {
        return timelønn;
    }

    public String toString() {
        java.util.Formatter f = new java.util.Formatter();
        f.format("%.2f", timelønn);
        return persNr + ": " + navn + ", timelønn kr. " + f.toString();
    }
}

class PersonTime {
    private final Person pers;
    private double antTimer = 0.0;
    public PersonTime(Person pers) {
        this.pers = pers;
    }

    public Person getPerson() {
        return pers;
    }

    public double getAntTimer() {
        return antTimer;
    }

    public void setAntTimer(double nyAntTimer) {
        antTimer = nyAntTimer;
    }
}
```

```
}  
}
```

VEDLEGG 2

```
class GUI extends JFrame {  
    private Register reg;  
    private JTextField idFelt = new JTextField(10);  
    private JTextField navnefelt = new JTextField(20);  
  
    private JList liste;  
    private JButton knapp = new JButton("Registrer!");  
    public GUI(Register reg) {  
        this.reg = reg;  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setTitle("Registrer nytt prosjekt");  
        setLayout(new BorderLayout());
```

***... det som mangler i konstruktøren kan settes inn her,
... merk det (*) i besvarelsen***

```
JScrollPane personliste = new JScrollPane(liste);  
JViewport jvp = new JViewport();  
jvp.setView(new JLabel("Velg personer fra listen.)); // tabelloverskrift  
personliste.setColumnHeader(jvp);
```

... og/eller her, merk det (**) i besvarelsen

```
    pack();  
}
```

```
public class FeltInput extends JPanel {  
    public FeltInput() {  
        setLayout(new FlowLayout());  
        add(new JLabel("Prosjektnr: "));  
        add(idFelt);  
        add(new JLabel("Navn: "));  
        add(navnefelt);  
    }  
}
```

```
public class Knappelytter implements ActionListener {  
    public void actionPerformed(ActionEvent hendelse) {
```

... innholdet i denne metoden skal programmeres ...

```
    }  
}
```