UiO : Universitetet i Oslo

Løsningsforslag ukeoppg. 7: 11. - 15. okt (INF1000 - Høst 2010)

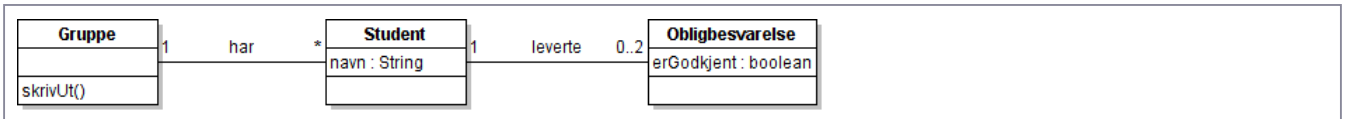
Mer om klasser og objekter *(kap. 8.17 - 8.18),* ***UML*** *(kap. 12.1 - 12.9)*

NB! *Disse er bare forslag til løsninger, dine svar kan være like gode eller bedre selv om de ser annerledes ut.*

Mål
Få mer øvelse i bruk av klasser og objekter, en innføring i UML-klassediagrammer, og tips til Oblig 3.

Oppgaver til teoritimen

1. **UML og behandling av obliger:**
   (a) Lag et **UML-klassediagram** for et lite program som kan brukes til å holde orden på obliger og studenter i én INF1000-gruppe. Bruk tre klasser: Student, Obligbesvarelse, og Gruppe. Navngi forholdene og skriv antall på begge sider av disse. Plassér følgende variabler i klassene der de passer best: `navn` på studentene, og en boolean `erGodkjent` som angir status på leverte obliger. Hva slags dataelementer (variabler, arrayer, eller lignende) ville du lagt til for å holde orden på studentene i gruppen og obligbesvarelsene de leverer? Anta at det bare er én oblig i kurset, men at studentene kan levere to utgaver av obligen hvis den første ikke blir godkjent.



Klassediagrammet ble laget med Violet UML Editor

   Tips: Se eksemplet på side 236 i læreboka.

   (b) Skriv et **program** for systemet beskrevet i del (a), og lag en testmetode som legger inn 3 studenter i gruppen, og obligbesvarelser for to av dem. Registrér én av disse besvarelsene som godkjent. (Anta som i (a) at kurset bare har én oblig, og at hver student kan levere opptil 2 besvarelser på den). Til slutt i testmetoden kaller du en annen metode i klassen Gruppe, kalt `skrivUt()`, som går gjennom alle studentene som er registrert i gruppen, og for hver av de skriver ut antall obliger de leverte (0, 1, eller 2), og om de fikk godkjent.

```
class Student {
    String navn;
    Obligbesvarelse[] oblig = new Obligbesvarelse[2];
}

class Obligbesvarelse {
    boolean erGodkjent; // Java initialiserer denne med verdien false
}

class Gruppe {
    Student[] studenter = new Student[40];
    int antStudenter;

    public static void main(String[] args) {
        Gruppe g = new Gruppe();
        g.testMetode();
    }

    void testMetode() {
        Student s1 = new Student();
        s1.navn = "Magnus Carlsen";
        s1.oblig[0] = new Obligbesvarelse();
        s1.oblig[0].erGodkjent = true;
        studenter[antStudenter++] = s1; // Lagrer s1 og setter antStudenter = 1

        Student s2 = new Student();
        s2.navn = "Moland French";
        s2.oblig[0] = new Obligbesvarelse();
        // erGodkjent blir automatisk satt til false for oblig[0]
        studenter[antStudenter++] = s2;

        studenter[antStudenter] = new Student();
        studenter[antStudenter++].navn = "Jens Obama";
        // Ingen levert oblig, oblig[0] og oblig[1] har default-verdi null

        skrivUt(); // Kaller metoden skrivUt()
    }


    void skrivUt() {
        // Løkke som går gjennom alle registrerte studenter, og skriver ut
        // for hver av dem: antall leverte obliger, og om de fikk godkjent.
        for (int i = 0; i < antStudenter; i++) {

            Student stud = studenter[i]; // Midlertidig peker

            System.out.println("Student: " + stud.navn);

            int antLevert = 0;
            if (stud.oblig[1] != null) {
                antLevert = 2;
            } else if (stud.oblig[0] != null) {
                antLevert = 1;
            }
            System.out.println("Leverte: " + antLevert + " obliger");

            String godkjent = "nei";
            if ((stud.oblig[0] != null && stud.oblig[0].erGodkjent)
                || (stud.oblig[1] != null && stud.oblig[1].erGodkjent)) {
                godkjent = "ja";
            }
            System.out.println("Fikk godkjent: " + godkjent + "\n");
        }
    }
}

/* KJØREEKSEMPEL:
> java Gruppe
Student: Magnus Carlsen
Leverte: 1 obliger
Fikk godkjent: ja

Student: Moland French
Leverte: 1 obliger
Fikk godkjent: nei

Student: Jens Obama
Leverte: 0 obliger
Fikk godkjent: nei
*/
```

(c) **Utvidelser:** Diskutér hvordan man kan utvide programmet til å kunne håndtere flere oblignr. i kurset, og enda flere besvarelser fra samme student på en gitt oblig. Evt. også flere grupper.

Det fins mange forskjellige måter å håndtere flere oblignr. på, noen av alternativene er:

Innføre en ny variabel `oblignr` i klassen Obligbesvarelse som angir hvilken oblig-oppgave besvarelsen gjelder.

En annen, bedre måte er å endre arrayen `oblig[]` i klassen Student til å være en 2D-array hvor første indeks angir oblignr. og andre indeks besvarelsesnr., f.eks. `oblig[0][1]` vil da angi studentens 2. besvarelse på første oblig. Alternativt kunne man ha flere 1D-arrayer, f.eks. `oblig1[]`, `oblig2[]`, osv.

En annen måte er å innføre en ny klasse Obligoppgave, som f.eks. kan ha koblinger til alle besvarelser på en obligoppgave (f.eks. i form av en array med pekere til hver registrerte besvarelse på obligen, og med alle besvarelse-utgaver fra alle studentene slått sammen i samme array). Da bør man også innføre en ny kobling som går *fra* Obligbesvarelse *til* Student, slik at man kan lettere finne studenten som leverte en besvarelsen når man går gjennom besvarelsene vha. ovennevnte array i klassen Obligoppgave.

Håndtering av flere enn 2 besvarelser på samme oblig kan ordnes ved å la den aktuelle arrayen hvor besvarelsene lagres ha flere enn 2 elementer.

Hvis man vil ha flere Grupper så er en god løsning å innføre en overordnet klasse, f.eks. kalt Kurs, som inneholder pekere til de forskjellige gruppene.

(d) **Konstruktør og this:** Lag en *konstruktør* i klassen Student, med minst én parameter, navn, og bruk nøkkelordet this for å skille mellom parameteren "navn" og objektvariabelen "navn" i konstruktøren. Skriv kode som benytter denne konstruktøren. *En konstruktør er en metode med samme navn som klassen den er i, og som blir utført automatisk når man oppretter objekter av klassen vha. nøkkelordet `new`.* Husk å endre alle setninger i programmet som inneholdt uttrykket "`new Klassenavn()`", slik at de benytter den nye konstruktøren (dvs. at du legger til en parameter i parentesene etter `new Klassenavn...`).

Her er et eksempel på klassen Student med en enkel konstruktør, som har én parameter: navn.

```
class Student {
    String navn;
    Obligbesvarelse[] oblig = new Obligbesvarelse[2];

    Student(String navn) {
        this.navn = navn;
    }
}
```

Legg merke til det reserverte ordet **this**, det er en peker til nåværende objekt og brukes ofte i konstruktører for å skille mellom parameter og objektvariabel hvis de har samme navn. Når man har innført denne konstruktøren kan man bytte følgende to linjer:

```
studenter[antStudenter] = new Student();
studenter[antStudenter++].navn = "Jens Obama";
```

...med bare én:

```
studenter[antStudenter++] = new Student("Jens Obama");
```

2. **Lesing/skriving av datafil:**
Utvid programmet fra oppgave nr. 1. (b) ovenfor med lesing og skriving av dataene til en datafil. Finn på en passende måte å formatere dataene på slik at du kan putte alt i en og samme fil.

Det nye her er metodene `skrivFil()` og `lesFil()`, og filformatet, beskrevet i metoden `skrivFil()`. I tillegg viser følgende kode hvordan du kan lage en konstruktør med 4 parametre (se klassen Student), og hvordan dette forenkler koden for å opprette Student-objekter (vist i den nye konstruktøren i klassen Gruppe).

```
import easyIO.*;

class Student {
    String navn;
    Obligbesvarelse[] oblig = new Obligbesvarelse[2];

    // Eksempel på konstruktør med 4 parametre:
    Student(String navn, int antLevert, boolean godkjent1, boolean godkjent2) {
        this.navn = navn;
        for (int i = 0; i < antLevert; i++) {
            oblig[i] = new Obligbesvarelse();
            // Hvis i == 0 brukes verdien fra godkjent1, ellers godkjent2:
            oblig[i].erGodkjent = ((i == 0) ? godkjent1 : godkjent2);
        }
    }
}

class Obligbesvarelse {
    boolean erGodkjent; // Java initialiserer denne med verdien false
}

class Gruppe {
    final int MAX_STUDENTER = 40;

    Student[] studenter = new Student[MAX_STUDENTER];
    int antStudenter;

    public static void main(String[] args) {
        new Gruppe();
    }

    Gruppe() {
        studenter[antStudenter++] =
            new Student("Magnus Carlsen", 1, true, false);
        studenter[antStudenter++] =
            new Student("Moland French", 1, false, false);
        studenter[antStudenter++] =
            new Student("Jens Obama", 0, false, false);

        skrivFil(); // Skriver dataene til fil

        studenter = null; // Sletter alle dataene
        lesFil(); // Leser data fra fil

        skrivUt(); // Skriver dataene på skjerm
    }


    void skrivUt() {
        // Løkke som går gjennom alle registrerte studenter, og skriver ut
        // for hver av dem: antall leverte obliger, og om de fikk godkjent.
        for (int i = 0; i < antStudenter; i++) {

            Student stud = studenter[i]; // Midlertidig peker

            System.out.println("Student: " + stud.navn);

            int antLevert = 0;
            if (stud.oblig[1] != null) {
                antLevert = 2;
            } else if (stud.oblig[0] != null) {
                antLevert = 1;
            }
            System.out.println("Leverte: " + antLevert + " obliger");

            String godkjent = "nei";
            if ((stud.oblig[0] != null && stud.oblig[0].erGodkjent)
                || (stud.oblig[1] != null && stud.oblig[1].erGodkjent)) {
                godkjent = "ja";
            }
            System.out.println("Fikk godkjent: " + godkjent + "\n");
        }
    }
```

```
    void skrivFil() {
        /* Filformat:
         *   <antall_studenter>
         *   <navn>; <ant_leverte>; <nr1_godkjent?>; <nr2_godkjent?>
         *
         * Eksempel på datafil:
         *   3
         *   Magnus Carlsen; 1; ja; nei
         *   Moland French; 1; nei; nei
         *   Jens Obama; 0; nei; nei
         */
        Out utfil = new Out("obligdata.txt");

        utfil.outln(antStudenter);
        for (int i = 0; i < antStudenter; i++) {
            Student stud = studenter[i]; // Midlertidig peker

            int antLevert = 0;
            if (stud.oblig[1] != null) {
                antLevert = 2;
            } else if (stud.oblig[0] != null) {
                antLevert = 1;
            }
            utfil.out(studenter[i].navn + "; " + antLevert + "; ");

            // Antar at man aldri leverer 2 besvarelser hvis nr. 1 ble godkjent
            String godkjent1 = "nei", godkjent2 = "nei";
            if (antLevert >= 1 && studenter[i].oblig[0].erGodkjent) {
                godkjent1 = "ja";
            } else if (antLevert >= 2 && studenter[i].oblig[1].erGodkjent) {
                godkjent2 = "ja";
            }
            utfil.outln(godkjent1 + "; " + godkjent2);
        }
        utfil.close();
    }

    void lesFil() {
        In innfil = new In("obligdata.txt");

        studenter = new Student[MAX_STUDENTER];

        antStudenter = innfil.inInt();
        for (int i = 0; i < antStudenter; i++) {
            String navn = innfil.inWord(";");

            int antLevert = innfil.inInt("; ");
            boolean godkjent1 = innfil.inWord("; ").equals("ja");
            boolean godkjent2 = innfil.inWord("; ").equals("ja");

            studenter[i] = new Student(navn, antLevert, godkjent1, godkjent2);

        }
        innfil.close();
    }
}

/* KJØREEKSEMPEL:
> java Gruppe
Student: Magnus Carlsen
Leverte: 1 obliger
Fikk godkjent: ja

Student: Moland French
Leverte: 1 obliger
Fikk godkjent: nei

Student: Jens Obama
Leverte: 0 obliger
Fikk godkjent: nei

> more obligdata.txt
3
Magnus Carlsen; 1; ja; nei
Moland French; 1; nei; nei
Jens Obama; 0; nei; nei
*/
```

3. **Vanlige feilmeldinger i Oblig 3:**

Finn feilene i følgende program og foreslå hvordan de kan rettes. Programmet er en forenklet utgave av det fra oppgave 1. (b), hvor det ikke er noen obliger, men bare håndtering av studentene i én INF1000-gruppe. Programmet består av to klasser: Student, som bare har én objektvariabel: navn; og Gruppe, som har en array med opptil 40 studenter og en *konstruktør* med programkode som illustrerer bruk av klasser og objekter.

> *// NB!* Disse er løsningsforslagene til oppgave 3 uke 7, klikk her for å se oppgaven uten fasit.

```
class Student {
   String navn;
}

class Gruppe {
   Student[] studenter = new Student[40]; // Array med Student-objekter
   int antStudenter; // Antall plasser i arrayen over som er i bruk

   public static void main(String[] args) {
      Gruppe g = new Gruppe();

      skrivAntall();
      /* Spørsmål:
       * a) Hvorfor gir linjen over følgende feilmelding?  Hvordan unngå det?
       *       Gruppe.java:12: non-static method skrivAntall() cannot
       *                       be referenced from a static context
       *           skrivAntall();
       *           ^
       *  Tips: Du kan lese mer om dette på side 199 (og 155) i læreboka.
       *    Problemet skyldes at main-metoden er en "klassemetode", dvs. deklarert
       *    med "static", og derfor kan man ikke referere til "objektvariabler"
       *    eller "objektmetoder" (dvs. variabler og metoder som ikke er deklarert
       *    me nøkkelordet static) direkte fra en static metode uten å gå
       *    via en peker, f.eks. pekeren g ovenfor.  Dette illustrerer
       *    hvorfor vi vanligvis bare har noen få programsetninger i
       *    main-metoden som bare setter programmet i gang, ved å kalle
       *    på andre metoder i programmet.
       */
```

Svar:
a) Globale variabler og metoder deklarert uten nøkkelordet **static** kalles for "objektvariabler" og "objektmetoder". Slike variabler og metoder kan ikke aksesseres direkte fra main (uten å gå via en objektpeker) fordi main-metoden er static - den er en såkalt "klassemetode" og ikke "objektmetode".

I dette tilfellet skulle kallet på objektmetoden skrivAntall() ha gått via en objektpeker, f.eks. slik: g.skrivAntall(), fordi kallet på metoden er gjort i static-metoden main.

```
   }
   Gruppe() {
      // b) Den vanligste feilmeldingen i Java er "cannot find symbol".
      //    Følgende linje gir feilmeldingen vist under.  Hva er feil i dette
      //    tilfellet, og hvordan kan vi rette det?
      antallStudenter = 0;
      /*                Gruppe.java:32: cannot find symbol
       *                symbol  : variable antallStudenter
       *                location: class Gruppe
       *                        antallStudenter = 0;
       *                        ^
       *  Tips: Årsaken til feilmeldingen "cannot find symbol" er alltid
       *    at noe ikke er deklarert riktig (f.eks. at det ikke er deklarert
       *    på riktig sted eller er stavet feil).  Feilen er heldigvis
       *    lett å fikse: se i feilmeldingen hva det var som ikke var
       *    deklarert, og sjekk at du har deklarert det riktig (på riktig
       *    sted i programmet, og stavet riktig).  Du kan finne hva det er
       *    som ikke var riktig deklarert på 2. linje i feilmeldingen, der
       *    det står "symbol".  I eksemplet over er problem-symbolet
       *    "variable antallStudenter".  Legg også merke til linjenummeret
       *    og posisjonen på linja som Java-kompilatoren også gir deg.
       */
```

b) antStudenter er stavet feil.

```
        // c) Følgende linje gir "NullPointerException" her.  Hvilken setning
        //    har vi glemt å utføre før dette?  (Husk at studenter[] er en array)
        studenter[antStudenter].navn = "Trine";
        antStudenter++;
        /*  Tips: Feilmeldingen NullPointerException skyldes oftest at man har
         *    forsøkt å bruke prikk-notasjon på en peker som hadde verdien null.
         *    Det er ikke lov (det gir ingen mening å si f.eks. null.navn).
         *
         *  Slik fikser du NullPointerException-feil:
         *    1. Se i feilmeldingen hvilket linjenummer Java fant feilen i.
         *    2. Se hvilke pekere i den angitte linjen som kan ha vært null
         *       under kjøring av programmet, særlig blandt de som har prikk-
         *       notasjon etter seg, f.eks. hvis linjen inneholder uttrykket:
         *         hyblene[rad][kol].leietager.navn  ...så kan null-pekeren
         *       være hyblene[rad][kol] eller hyblene[rad][kol].leietager
         *    3. Endre programmet slik at pekeren som forårsaket feilen ikke
         *       kan være null når programmet kommer til den aktuelle linjen.
         *       Hvis problemet var en peker med prikknotasjon kan du legge
         *       til en if-test på at pekeren != null før den aktuelle linjen.
         *    Det som står foran prikken må altså være noe som peker på
         *    et allerede-opprettet objekt av riktig klasse (og ikke null), på
         *    det tidspunktet under programutførelsen når setningen blir utført.
         *
         *    4. Hvis linjen har flere pekere og du ikke finner hvilken som ga
         *       NullPointerException kan du legge inn en testutskrift rett før
         *       linja, og skrive ut én av dem, f.eks.
         *         System.out.println("test1:" + hyblene[rad][kol].leietager);
         *       Med denne fremgangsmåten vil du alltid kunne finne nullpekeren.
         */
```

```
c) Her mangler å opprette studentobjekt, f.eks. slik:
    studenter[antStudenter] = new Student();
```

```
        // d) Skriv det som mangler for at studenter[1].navn skal bli "Martin".
        Student s1 = new Student();
        s1 _________ = "Martin";
        studenter[antStudenter++] = _________ ;
```

```
d) s1.navn = "Martin";
studenter[antStudenter++] = s1 ;
```

```
        // e) Hva mangler her for at if-testen skal kunne gi true?
        //    Tips: Husk at tekster må sammenlignes med andre tekster.
        if (studenter[1].equals("Martin")) {
            System.out.println(true);
        }
```

```
e) Det mangler .navn:
if (studenter[1].navn.equals("Martin")) {
```

```
        // f) Hvorfor gir følgende løkke NullPointerException på linjen med
        //    System.out..? (anta at eneste kode som er blitt utført når
        //    programmet kommer hit er det som står i linjene over, med feilene
        //    rettet).  Også, hvordan kan betingelsen i første linje endres for å
        //    unngå NullPointerException?  Tips: Tenk arrayplasser og != null.
        for (int i = 0; i < studenter.length; i++) {
            Student s2 = studenter[i];
            System.out.println(s2.navn);
        }
```

```
f) Her blir det NullPointerException fordi det bare er noen få elementer
   i arrayen som inneholder Student-objekt, og resten har null-peker.
   Problemet er at "i" går for langt, og forbi plassene i arrayen der det
   ligger data.  En løsning er å endre betingelsen i for-løkka slik at "i"
   bare teller opp til antStudenter:
      for (int i = 0; i < antStudenter; i++) {

   En annen mulig løsning er å sjekke at elementet i arrayen
   studenter[] på plass "i" ikke er null:
      for (int i = 0; studenter[i] != null && i <
   studenter.length; i++) {
```

```
        // g) Finn en annen måte å unngå NullPointerException her uten å endre
        //    første linje under, men ved å legge til en if-setning inne i løkka:
        for (int i = 0; i < studenter.length; i++) {
            Student s3 = studenter[i];
```

```
g)  if (s3 != null) {
        System.out.println(s3.navn);
    }
```

```
    }

    // h) Hvorfor gir følgende linje i programkoden denne feilmeldingen:
    //    ArrayIndexOutOfBoundsException: 40
    studenter[40] = new Student();
```

h) Fordi indeksene i arrayen går fra 0 til 39.

```
    // i) Hva vet vi om uttrykket merkert med "___" på neste linje hvis
    //    linjen gir feilmeldingen "ArrayIndexOutOfBoundsException: -1":
    s1 = studenter[ ___ ];
```

i) Uttrykket har verdi -1, så det kan f.eks. ha vært:
`s1 = studenter[j];` ...der j har verdien -1 på dette tidspunktet.

```
    // j) Hva er feil her?  Feilmeldingen dette gir er:
    //                  Gruppe.java:103: incompatible types
    //                found   : java.lang.String
    //                required: Student
    studenter[2] = "Eva";
```

j) Dette er et forsøk på å lagre en String-verdi inne i en Student-peker.
Det går ikke. Tekster må lagres i tekst-variabler, f.eks.:
`studenter[2].navn = "Eva";`

```
    // k) Hva er feil her?  Feilmeldingen er: cannot find symbol
    //                  symbol  : constructor Student(java.lang.String)
    Student lars = new Student("Lars");
```

k) Som du ser av feilmeldingen, så finner ikke Java-kompilatoren noen konstruktør i klassen Student med en String som parameter. Løsningen er enten å lage en slik konstruktør i klassen Student (med én String som parameter), eller bruke den vanlige automatisk-definerte konstruktøren (som ikke har parametre), og lagre navnet i en egen programsetning:

```
Student lars = new Student();
lars.navn = "Lars";
```

```
    // l) Hva er feil her?  Feilmeldingen dette gir er:
    //                  Gruppe.java:114: incompatible types
    //                  found   : int
    //                  required: boolean
    if (antStudenter = 0) {
        system.out.println("Ingen student registrert!");
    }
```

l) Det skulle stått to lik-tegn: "==". Når det bare står ett lik-tegn er hele uttrykket en tilordning, som gir 0 som verdi, og det passer ikke som betingelse i en if-setning (betingelser må ha boolsk verdi).

```
    // m) Hvorfor klager Java også med: "package system does not exist"?
```

m) System er stavet feil.

```
    // n) Når vi har rettet alle feilene i a) til m) ovenfor, så gir fortsatt
    //    if-setningen under NullPointerException.  Hvordan unngår vi det?
    boolean funnet;
    for (int i = 0; i < studenter.length; i++) {
        Student stud = studenter[i];
        if (stud.navn.equals("Eva")) {
            funnet = true;
            System.out.println(stud.navn);
        }
    }
```

n) Ved f.eks. å legge til en ekstra-betingelse i if-setningen, f.eks.:

```
if (stud != null && stud.navn.equals("Eva")) {
```

```
    // o) Løkken ovenfor stopper ikke når navnet "Eva" blir funnet.
    //    Hvordan kan vi få løkken til å stoppe når navnet blir funnet?
```

o) Ved å legge til "! funnet" som en ekstra-betingelse i selve for-løkken:

```
for (int i = 0; i < studenter.length && !funnet; i++) {
```

```
    // p) Løkken ovenfor gir ingen melding til bruker når navnet ikke blir
    //    funnet.  Hvordan kan vi programmere utskrift av en slik melding?
    //    Og hvordan kan vi unngå at Java da skal klage om at "variable
    //    funnet might not have been initialized"?
```

```
p) Ved å sette startverdien "false" i funnet, og legge til rett etter
    løkken en if-setning som skriver ut en melding om "ikke funnet"
    hvis !funnet.
```

```
        // q) Anta at navnet til studenter[2] er "Eva".  Hvorfor endrer ikke
        //    følgende kode navnet til studenter[2]?  Hvordan kan det ordnes?
        //    (slik at studentobjektet i "ny" overføres til studenter[2]).
        Student ny = new Student();
        Student studPeker;
        ny.navn = "Heidi";
        studPeker = studenter[2];
        studPeker = ny;
    }
```

```
q) Vi har 2 student-objekter i minnet: Eva som er pekt på av studenter[2],
   og Heidi som er pekt på av ny.  Det som kodebiten gjør er å sette
   studPeker til å peke på det første, og deretter flyttes (bare) denne
   studPeker til å peke på ny i stedet.  Det endrer ikke på hva pekeren
   studenter[2] peker på.  For å ordne det må man bruke en tilordning
   hvor studenter[2] står på venstre side av "="-tegnet, f.eks. ved
   å legge til følgende som en ny linje rett etter kodebiten i oppgaven:
       studenter[2] = studPeker;
```

```
    void skrivAntall() {
        System.out.println("Antall studenter: " + antStudenter);
    }

    // r) Hvorfor får vi feilmeldingen: "<identifier> expected" på denne linjen:
    System.out.println("Slutt");
```

```
r) Det er bare deklarasjoner (av variabler og metoder) som
   kan plasseres her, i dette området utenfor metoder.  Andre typer
   setninger, som "System.out.." er bare lov å plassere inne i metoder.
```

```
}
```

Flere debuggings-tips:
Både feilmeldingene fra kompilatoren (`javac`) og kjøresystemet til Java (`java`) gir deg vanligvis linjenummeret der feilen oppsto, og navnet til feilen. Disse opplysningene er nyttige for å finne og rette feilen. Start alltid med å rette den første feilen som Java fant.

Tips til krøllparentes-feil:
En annen type feil som er lett å gjøre i Java er krøllparentes-feil, f.eks. å glemme en krøllparentes et sted, ha en for mye, en som "vender" feil vei, o.l. Disse problemene gir opphav til mange typer feilmeldinger, bl.a.:

«**class, interface, or enum expected**», «**<identifier> expected**»
«**reached end of file while parsing**», «**'}' expected**»,
«**illegal start of type**», «**illegal start of expression**», «**'else' without 'if'**», m.fl...

To gode tips for å unngå og rette slike feil er:

1. Alltid holde programmet riktig formatert med "innrykk" underveis mens du skriver det: Start et nytt "nivå" med innrykk rett etter hver "{" (åpnings-krøllparentes) du skriver, dvs. at alle linjene etter en slik krøllparentes får f.eks. 4 flere mellomrom foran enn linjene over ...frem til første "}", da du forminsker ett innrykk-nivå, osv. *Alle programmer i disse ukeoppgavene er formatert på denne måten.*
2. Når et av de ovennevnte feilene dukker opp, bruk funksjonen i Emacs (eller Eclipse, e.l.) som setter inn riktig innrykk i hele programmet. I Emacs kan det gjøres ved å markere hele programmet (Ctrl-x h) og velge "Java > Indent Line or Region" fra menyen øverst i Emacs-vinduet. Da blir det satt riktig innrykk i hele programmet og det blir lett å se hvor feilen er, ved å starte fra toppen og fortsette nedover til du finner noe som har havnet på feil innrykksnivå: Sjekke at alle klasser havnet i "første innrykks-nivå" (dvs. at det ikke står noen mellomrom før nøkkelordet `class`), og at første linje i alle metoder havnet i 2. nivå av innrykk (dvs. at det er 4 mellomrom rett før nøkkelordet `void`), og at selve innmaten i metodene er i 3. nivå (har 8 mellomrom foran), osv.

Flere tips til feilsøking og -retting finner du i Marit Nybakkens feilmeldinger.pdf.

Oppgaver til terminaltimen

1. **UML og behandling av obliger:**
   (Samme oppgave som i nr. 1. (b)-(d) og punkt 2. for teoritimen)

2. **Vanlige feilmeldinger i Oblig 3:** *(Oblig3-relevant!)*
   (Samme oppgave som i nr. 3. for teoritimen)

3. **Fortsett med Oblig 3:**
   Les gjennom oppgaveteksten til Oblig 3, og ut fra denne:

   (a) Tegn et *UML-klassediagram* (dette skal ikke leveres i Oblig 3, men kan være nyttig for å få oversikt over programmet, særlig hvis du vil legge til flere klasser enn de som er minstekravet, eller hvis du vil ha oversikt over variablene eller metodene du ønsker å ha i hver klasse.)

   (b) Bestem *forholdene* mellom klassene, dvs. hvilke klasser skal ha pekere til hvilke klasser, og hvor mange (f.eks. om pekerne skal være enkle variabler eller

arrayer). Navngi forholdene og sett på antall i hver ende.

(c) Begynn å skrive koden for Oblig 3, og finn ut hvilke *metoder* du vil ha i de ulike klassene. Til å begynne med skriver du tomme deklarasjoner (metoder uten noe kode inni) slik at du kan kompilere programmet og teste det underveis mens du skriver det. Revidér og gjenta punkt (a) til (c) inntil programmet fungerer som det skal.

4. **Ukens nøtt** er ekstraoppgavene i Oblig 3.

Tibakemelding om dette oppgavesettet kan du skrive i bloggen eller sende på mail til josek *[a]* ifi.uio.no