UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i:	INF1000 — Grunnkurs i objektorientert programmering
Eksamensdag:	Torsdag 4. desember 2014
Tid for eksamen:	14.30 (4 timer)
Oppgavesettet er på:	7 sider
Vedlegg:	Ingen
Tillatte hjelpemidler:	Alle trykte og skrevne

Bokmål

- Kontroller at oppgavesettet er komplett, og les nøye gjennom oppgavene før du løser dem.
- Du kan legge dine egne forutsetninger til grunn og gjøre rimelige antagelser, så lenge de ikke bryter med oppgavens «ånd». Gjør i så fall rede for disse forutsetningene og antagelsene.
- Poengangivelsen øverst i hver oppgave angir maksimalt antall poeng. Sammenlagt gir alle oppgavene maksimalt 100 poeng. Unngå å bruke en stor del av tiden din på oppgaver som gir deg få poeng.
- Svarene skal skrives på gjennomslagspapir. Skriv hardt nok til at besvarelsen blir mulig å lese på alle gjennomslagsarkene, og ikke legg andre deler av eksamensoppgaven under når du skriver.
- Du beholder selv underste ark etter levering av de to øverste til eksamens-inspektøren.

Oppgave 1 (4 poeng)

a) Hva er verdien til **tall** etter at følgende kode er utført?

```
int tall=(5+3)*2;
tall = tall+2;
```

Svar: **18**

b) Anta at følgende programsetninger utføres. Hva skrives ut på skjermen?

```
String a = "ab";
String b = "b" + a;
a = a + b;
System.out.println(a);
```

Svar: **abbab**

c) Hva er verdien til **x** etter at følgende kode er utført?

```
int x = 6;
int y = 15;
while (x < 30) {
  if (x < y) {
    x = x * 2;
  } else {
    y += 5;
    x -= 5;
  }
}
```

Svar: **38**

Oppgave 2 (6 poeng)

a) Hva skrives ut her?

```
class MinRegneKlasse {
  public static void main(String args[]) {
    System.out.println(regnUt(4, 10));
  }

  public static int regnUt(int a, int b) {
    int svar;
    if (a == b) {
      svar = a * 2;
    } else {
      svar = a;
    }
    return svar;
  }
}
```

Svar: **4**

b) Anta at programmet nedenfor utføres. Hva blir utskriften på skjermen?

```
class BrukMin {
  public static void main(String[] args) {
    MinKlasse mittObjekt = new MinKlasse(5);
    mittObjekt.leggTilVerdi(3);
    int verdi = mittObjekt.hentVerdi();
    mittObjekt.leggTilVerdi(verdi);
    System.out.println(mittObjekt.hentVerdi());
  }
}

class MinKlasse {
  private int minVerdi;

  MinKlasse(int startVerdi) {
    minVerdi = startVerdi;
  }

  void leggTilVerdi(int tillegg){
    minVerdi += tillegg;
  }

  int hentVerdi(){
    return minVerdi;
  }
}
```

Svar: **16**

c) (Vanskelig)
Det er en logisk feil i metoden **stoerst** nedenfor, som gjør at den ikke alltid returnerer det største av de tre tallene den får som parameter. Gi et eksempel på et kall med tre tall som fører til at metoden returnerer et annet tall enn det største.

```
static int stoerst(int a, int b, int c) {
  int svar;
  if (a>b && a>c) {
    svar = a;
  }
  else if (b>a && b>c) {
    svar = b;
  } else {
    svar = c;
  }
  return (svar);
}
```

Svar: Metoden gir galt svar om de to første parametrene er like og den tredje er mindre enn de to, for eksempel **stoerst(2, 2, 1)**.

Oppgave 3 (4 poeng)

a) Skriv binærtallet 1100 som et desimaltall. Svar: **12**
b) Skriv desimaltallet 18 som et binærtall. Svar: **10010** evt **0001 0010**
c) Skriv summen av de to binærtallene 110 og 11 som et binærtall. Svar: **1001**
d) Skriv det heksadesimale tallet 2A som et desimaltall. Svar: **42**

Oppgave 4 (5 poeng)

Skriv en metode

```
int pris (boolean gratis, int alder)
```

Dersom parameteren **gratis** har verdien **true**, skal metoden *alltid* returnere 0. Dersom parameteren **gratis** har verdien **false** og verdien av **alder** er mindre enn 18, skal metoden returnere 100, ellers 200. Altså skal f.eks. kallet **pris (true, 10)** returnere 0, kallet **pris(false,10)** returnere 100 og kallet **pris(false, 50)** returnere 200.

Løsningsforslag:

```
int pris(boolean gratis, int alder) {
    int svar;
    if (gratis) {
        svar = 0;
    } else if (alder < 18) {
        svar = 100;
    } else {
        svar = 200;
    }
    return svar;
}
```

Oppgave 5 (5 poeng)

Skriv en metode som har en **int**-array som parameter og som returnerer en verdi av type **boolean**. Metoden skal sjekke om alle verdiene i arrayen er i stigende rekkefølge (sortert). Dersom alle verdiene er i sortert rekkefølge skal metoden returnere **true**, ellers skal metoden returnere **false**. Du kan anta at alle verdiene i arrayen er ulike. Metoden trenger altså ikke ta hensyn til eventuelle like verdier.

Løsningsforslag:

```
boolean oppg5(int[] tall) {
    for (int i = 0;    i < tall.length-1;    i++) {
        if (tall[i+1] < tall[i]) {
            return false;
        }
    }
    return true;
}
```

Oppgave 6 (6 poeng)

a) Skriv en metode med en **int**-array som parameter og som returnerer en verdi av type **int**. Dersom alle verdiene i arrayen er like, skal metoden returnere denne verdien. Dersom ikke alle verdiene er like, skal den returnere tallet -1. Du kan anta at arrayen inneholder minst en verdi.

Løsningsforslag:

```
int oppg6(int[] tall) {
    for (int i = 0;    i < tall.length-1;    i++) {
        if (tall[i+1] != tall[i]) {
            return -1;
        }
    }
    return tall[0];
}
```

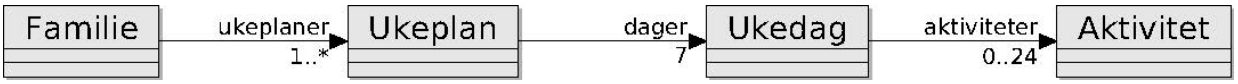
b) Dersom du kaller metoden fra a) med en ikke-tom array og får -1 tilbake, kan du da være sikker på at ikke alle tallene i arrayen du sendte inn var like? Begrunn svaret.

Svar: Nei; du kan ha sendt inn en array hvor alle tall er -1.

Oppgave 7 Familiens ukeplaner (50 poeng)

Du har påtatt deg å lage ukeplaner for hele familiens faste aktiviteter. For å gjøre det enkelt antar vi at alle aktiviteter starter på en hel time (kl 00.00, 01.00, 02.00 etc) og at de alle varer nøyaktig 1 time.

Du skal bruke klassene vist i dette UML klassediagrammet:



a) Klassen **Aktivitet** skal ha en representasjon med disse to objektvariablene: en **String** med navn **aktNavn** som brukes til å beskrive hva slags aktivitet det gjelder (for eksempel "svømming" eller "trener fotball 1A"), og en **int** med navn **start** som inneholder starttidspunktet for aktiviteten. Klassen skal også inneholde denne konstruktøren:

```
Aktivitet (String hva, int kl) { … }
```

Skriv klassen **`Aktivitet`** inkludert konstruktør (som må programmeres ferdig).

b) Skriv klassen **Ukedag** med dette foreløpige grensesnittet (dvs disse metodene):

```
Ukedag(String dag)                // Konstruktør

void settInn(String hva, int kl) // Oppretter og setter inn ny
                                  // aktivitet på angitt
                                  // klokkeslett, og gir feilmelding på
                                  // terminal om tidspunktet er opptatt

int tidligste()                   // Returnerer når på dagen den første
                                  // aktiviteten starter (-1 hvis det
                                  // ikke finnes noen aktivitet denne
                                  // dagen)

int seneste()                     // Returnerer når seneste aktivitet
                                  // den dagen starter (-1 hvis ingen
                                  // aktiviteter)

int antall()                      // Beregner og returnerer antall
                                  // aktiviteter den dagen
```

Du skal programmere alle disse metodene. Husk også å ta med klassens datarepresentasjon (dens objektvariabler).

c) Klassen **Ukedag** skal nå utvides med metoden **settInnLedig**:

```
void settInnLedig(String hva)
```

som setter inn en aktivitet for en ledig time den dagen. Hvis kalenderen er full, skal metoden skrive ut en feilmelding. Hvis det ikke er noen aktiviteter fra før den dagen, skal aktiviteten starte kl 12.00. Ellers

skal metoden helst sette inn den nye aktiviteten i en ledig time mellom tidligste og seneste aktivitet; hvis det ikke er mulig, skal den settes rett etter den hittil seneste aktiviteten; om heller ikke det er mulig, settes aktiviteten inn rett før den hittil tidligste.

Skriv metoden **settInnLedig**.

d) Skriv klassen **Ukeplan** med følgende grensesnitt (dvs metoder):

```
Ukeplan(String hvem)        // Konstruktør
Ukedag travleste()          // Finner dagen med flest aktiviteter
```

Husk å ta med datarepresentasjonen (dvs objektvariablene) til klassen.

e) Klassen **Ukeplan** skal nå utvides med metoden **skrivUt** som har denne signaturen:

```
void skrivUt()
```

Metoden skal sørge for at ukeplanen skrives ut, for eksempel slik:

```
Ukeplan for Aud:
Mandag
  Kl 9: Forelesning
  Kl 10: Forelesning
  Kl 12: Lunsj
Onsdag
  Kl 17: Trening
```

Skriv metoden **skrivUt**. Du vil sannsynligvis trenge nye metoder i flere av de andre klassene også; skriv disse metodene og angi i hvilken klasse den enkelte metoden hører hjemme. Selve formatteringen av utskriften er ikke viktig, så ikke bruk mye tid på det.

f) (Vanskelig)
Skriv klassen **Familie** med nødvendig datarepresentasjon og metoden (som du skal skrive):

```
void skrivAktiviteter()
```

Denne metoden skal skrive ut en liste over alle de ulike aktivitetene (fra **aktNavn**-variabelen) familien er med på. Aktiviteter med samme navn (dvs har lik verdi i **aktNavn**-variabelen) skal kun skrives ut én gang selv om de forekommer flere ganger hos samme eller forskjellige familiemedlemmer.

Hint: Bruk en **HashMap** til å holde orden på hvilke aktivitetsnavn som har vært skrevet ut.

Løsningsforslag:

```
import java.util.ArrayList;
import java.util.HashMap;


class Eksamen {
   public static void main(String[] arg) {
     // Testprogram; ikke eksamensoppgave.
```

```
      Familie fam = new Familie();

      fam.skrivUkeplaner();

      System.out.println("Alle aktivitetene:");
      fam.skrivAktiviteter();
   }
}


class Familie {
   private ArrayList<Ukeplan> ukeplaner = new ArrayList<>();

   void skrivAktiviteter() {
      HashMap<String,String> skrevetUt = new HashMap<>();

      for (int i = 0;  i < ukeplaner.size();  i++) {
         ukeplaner.get(i).skrivAktiviteter(skrevetUt);
      }
   }

   void skrivUkeplaner() {
      // Til testing; ikke eksamensoppgave.

      for (int i = 0;  i < ukeplaner.size();  i++) {
         ukeplaner.get(i).skrivUt();
      }
   }
}


class Ukeplan {
   private Ukedag[] dager = new Ukedag[7];
   private String eier;

   Ukeplan(String hvem) {
      eier = hvem;
      dager[0] = new Ukedag("Mandag");
      dager[1] = new Ukedag("Tirsdag");
      dager[2] = new Ukedag("Onsdag");
      dager[3] = new Ukedag("Torsdag");
      dager[4] = new Ukedag("Fredag");
      dager[5] = new Ukedag("Loerdag");
      dager[6] = new Ukedag("Soendag");
   }

   Ukedag travleste() {
      Ukedag travlestHittil = dager[0];
      int hvorTravel = travlestHittil.antall();

      for (int i = 1;  i < 7;  i++) {
         if (dager[i].antall() > hvorTravel) {
            travlestHittil = dager[i];
            hvorTravel = travlestHittil.antall();
         }
```

```
      }
      return travlestHittil;
    }

    void skrivUt() {
      System.out.println("Ukeplan for " + eier + ":");
      for (int i = 0;  i < 7;  i++) {
         dager[i].skrivUt();
      }
    }

    void skrivAktiviteter(HashMap<String,String> skrevetUt) {
      for (int i = 0;  i < 7;  i++) {
         dager[i].skrivAktiviteter(skrevetUt);
      }
    }
}


class Ukedag {
    private Aktivitet[] aktiviteter = new Aktivitet[24];
    private String ukedag;

    Ukedag(String dag) {
      ukedag = dag;
    }

    void settInn(String hva, int kl) {
      if (aktiviteter[kl] != null) {
         System.out.println(ukedag + " kl " + kl + " er opptatt!");
      } else {
         aktiviteter[kl] = new Aktivitet(hva,kl);
      }
    }

    int tidligste() {
      for (int i = 0;  i <= 23;  i++) {
         if (aktiviteter[i] != null) return i;
      }
      return -1;
    }

    int seneste() {
      for (int i = 23;  i >= 0;  i--) {
         if (aktiviteter[i] != null) return i;
      }
      return -1;
    }

    int antall() {
      int n = 0;

      for (int i = 0;  i <= 23;  i++) {
         if (aktiviteter[i] != null) {
           n++;
         }
```

```
      }
      return n;
    }

    void settInnLedig(String hva) {
      // Er hele dagen ledig?
      if (antall() == 0) {
          aktiviteter[12] = new Aktivitet(hva,12);
          return;
      }

      // Sjekk om det er noen pauser i loepet av dagen:
      int tidlig = tidligste();
      int sen = seneste();
      for (int i = tidlig+1;  i < sen;  i++) {
          if (aktiviteter[i] == null) {
            aktiviteter[i] = new Aktivitet(hva,i);
            return;
          }
      }

      // Proev senere paa kvelden:
      if (sen < 23) {
          aktiviteter[sen+1] = new Aktivitet(hva,sen+1);
          return;
      }

      // Proev tidligere paa morgenen:
      if (tidlig > 0) {
          aktiviteter[tidlig-1] = new Aktivitet(hva,tidlig-1);
          return;
      }

      // Intet ledig!
      System.out.println("Intet ledig paa " + ukedag + "!");
    }

    void skrivUt() {
      if (antall() > 0) {
          System.out.println(ukedag);
          for (int i = 0;  i <= 23;  i++) {
            if (aktiviteter[i] != null) {
               aktiviteter[i].skrivUt();
            }
          }
      }
    }

    void skrivAktiviteter(HashMap<String,String> skrevetUt) {
      for (int i = 0;  i <= 23;  i++) {
          if (aktiviteter[i] != null) {
            aktiviteter[i].skrivAktiviteter(skrevetUt);
          }
      }
    }
}
```

```
class Aktivitet {
   private String aktNavn;
   private int start;

   Aktivitet(String hva, int kl) {
     aktNavn = hva;
     start = kl;
   }

   void skrivUt() {
     System.out.println("  Kl " + start + ": " + aktNavn);
   }

   void skrivAktiviteter(HashMap<String,String> skrevetUt) {
     if (! skrevetUt.containsKey(aktNavn)) {
        System.out.println(aktNavn);
        skrevetUt.put(aktNavn,"ja");  // Kan sette inn hva vi vil.
     }
   }
}
```

Oppgave 8 (10 poeng)

Det er utviklet en nettjeneste som kan benyttes av personer som ønsker å bli blodgivere. Tjenesten stiller en rekke spørsmål for å avdekke eventuelle hindringer for å gi blod, og dermed minske risikoen for et forgjeves besøk hos Blodbanken for mulige nye blodgivere. Spørsmålene er hentet fra Blodbankens standard skjema for nye blodgivere, og handler blant annet om reiser til bestemte områder, sykdommer og seksuelle forhold. Svarene lagres fortløpende av tjenesten sammen med navn, fødselsnummer og annen informasjon som kreves for ferdigstilling av et komplett skjema.

Dersom analysen avdekker hindringer for å gi blod får brukeren beskjed om dette. I noen tilfeller vil sykdom og seksuell kontakt kun være midlertidige hindringer, og dataene blir da lagret hos tjenesten mens bruker får beskjed om å forsøke igjen om én eller seks måneder. Om alt ser greit ut, får man tilsendt et ferdig utfylt skjema i PDF-format å ta med til Blodbanken, slik at man slipper å bruke tid på manuell utfylling.

a) Personopplysningsloven § 2.1 forteller hva som er en personopplysning, og § 3 beskriver Saklig virkeområde. Gjelder loven for den løsningen som er beskrevet ovenfor?
   Svar: Ja

b) Inngår det sensitive personopplysninger i dataene som behandles? Begrunn med henvisning til Personopplysningsloven.
   Svar: Ja, ref §2-8, spesielt c, d og a.

c) Kreves det konsesjon fra Datatilsynet for systemet som implementerer denne tjenesten? Henvis til Personopplysningsloven.
   Svar: Ja, ref §33

Som et alternativ til det nettbaserte systemet beskrevet ovenfor har du utviklet en smarttelefon-applikasjon som blodgivere kan bruke for selv å holde rede på når man tidligere har gitt

blod, når man tidligst kan gi blod igjen, og forhold som kan gjøre at man må vente en viss tid før man kan gi blod neste gang, inklusive reiser til bestemte områder, sykdommer og skader eller seksuelle forhold. Applikasjonen har en dialog der blodgiveren svarer på relevante spørsmål. Svarene lagres kun lokalt på blodgiverens smarttelefon, og brukes av applikasjonen til å beregne når blodgiveren tidligst kan gi blod neste gang. Resultatet av beregningen blir vist for blodgiveren på smarttelefonens skjerm.

d) Gjelder personopplysningsloven for smarttelefon-applikasjonen beskrevet ovenfor? Begrunn.
Svar: Nei, så lenge personopplysningene ikke gjøres tilgjengelig for andre, er dette behandling av personopplysninger som den enkelte foretar for rent personlige formål.

Oppgave 9 (10 poeng)

Dersom du kaster 3 terninger, er det 6*6*6=216 mulige utfall av antall øyne på de tre terningene (1-1-1, 1-1-2, …, 6-6-6). Bare i 6 av disse 216 utfallene er det samme antall øyne på alle de tre terningene (1-1-1, 2-2-2 osv).

Skriv de nødvendige programlinjene for å printe ut alle kombinasjoner av antall øyne på de tre terningene på terminalen . Du trenger ikke skrive et komplett program med klasse og metodesignatur, bare anta at det er på plass. Print til slutt ut hvor mange kombinasjoner som hadde minst 2 like terninger. Merk at 1-1-2 og 1-2-1 i denne sammenhengen er to ulike kombinasjoner, slik at begge skal printes ut og telle med i antall kombinasjoner med minst 2 like terninger.

Løsningsforslag:

```
int antToLike = 0;
for (int t1 = 1;    t1 <= 6;    t1++) {
   for (int t2 = 1;    t2 <= 6;    t2++) {
      for (int t3 = 1;    t3 <= 6;    t3++) {
         System.out.println("Utfall: " + t1 + "- " + t2 + "- " + t3);
         if ((t1==t2) || (t1==t3) || (t2==t3)) {
            antToLike++;
         }
      }
   }
}
System.out.println("Antall minst to like: " + antToLike);
```