Universitetet i Bergen
Det matematisk-naturvitenskapelige fakultet
Institutt for informatikk

Bokmål

Eksamen i emnet INF100
Grunnkurs i programmering
Torsdag 27. november 2014
Tid: 09:00 – 14:00

Tillatte hjelpemidler: Alle trykte og skrevne
Oppgavesettet består av 4 oppgaver og er på 20 sider.

Viktige generelle råd og kommentarer:

- Kontroller at oppgavesettet er fullstendig.
- Les nøye gjennom hele oppgavesettet før du begynner å besvare de enkelte delspørsmålene.
- Mange av deloppgavene gjør bruk av metoder fra andre deloppgaver. Derfor er det viktig å merke seg hva de forskjellige metodene i deloppgavene gjør og hvilke verdier de returnerer. Det er ikke nødvendig å innføre ytterligere feltvariabler eller metoder i klassene for å besvare oppgavene i oppgavesettet.
- Det er viktig å holde seg til momenter som er nevnt i oppgaven, og ikke trekke inn andre aspekter.
- Dersom det er noen deler som du ikke får besvart, kan du likevel fortsette med de øvrige deloppgavene og anta at disse er tilgjengelige.
- Det er en fordel å kladde algoritmebeskrivelse og/eller diagram for bedre forståelse av program du utvikler.
- Koden din bør være formatert slik at den er leselig og enkel å forstå.
- Kontroller parameterverdier der det er hensiktsmessig.
- I kildekoden kan du anta at alle nødvendige import-setninger er inkludert.
- Synes du at oppgaveteksten er uklar eller ufullstendig noe sted, må du lage dine egne presiseringer og angi disse i besvarelsen.
- Prosentsatsene ved hver deloppgave angir omtrentlig vektlegging ved sensur.

Lykke til!

Fredrik Manne Atle Geitung

Oppgave 1a (7%)
Metoden finnMinste er en del av et større programsystem og skal finne og skrive ut verdien og plassen til det minste tallet i en tabell med flyttall. Du kan anta at tabell[] er en tabell med double-verdier som er korrekt deklarert og initialisert. Linjenumrene er bare gitt for at det skal være lettere å referere til en linje. De påvirker ikke koden.

```
1:  static void finnMinste(double[] tabell) {
2:    int N = tabell.length;
3:    double minVerdi = 0;
4:    int minPlass = 0;
5:    for(int i = 1;i < N; i++) {
6:      if (tabell[i] < minVerdi)
7:        minVerdi = tabell[i];
8:        minPlass = i;
9:    }

10:   System.out.print("Den minste verdien er tabell[");
11:   System.out.println(minPlass + "] = " + minVerdi);
12: }
```

Beskriv tre feil i koden som enten vil føre til et ukorrekt svar eller til at programmet vil krasje. Skriv maksimalt 10 ord for hver feil.

Løsningsforslag:
1. Linje 3: Dersom tabell inneholder negative tall vil disse ikke bli returnert.
   Kunne byttet linje 3 med: `double minVerdi = tabell[0];`
   Da forsvinner neste feil:
2. Linje 5: Løkken begynner på 1 og ikke på 0.
3 (- 4). Linje 7-8: Mangler {...} rundt linje 7 og 8.

Oppgave1b (8%)
Skriv en metode som tar en rektangulær tabell med flyttall og skriver ut summen av verdiene nedenfor hoved-diagonalen. En hver rektangulær tabell har eksakt en hoved-diagonal.
I de første tre eksemplene er elementene på hoved-diagonalen merket med D, mens elementene nedenfor hoved-diagonalen er merket med x. Det er disse du skal summere. I det siste eksemplet er tallene vist og svaret for denne tabellen vil være 19.0.
Du kan gå ut ifra at tabellen er korrekt definert og initialisert.

```
D o o o
x D o o
x x D o
x x x D

D o o o
x D o o
x x D o
x x x D
x x x x
x x x x

D o o o o o
x D o o o o
x x D o o o
x x x D o o

1.0  2.0  3.0
4.0  5.0  6.0
7.0  8.0  9.0
```

```
static void sum(double[][] tabell) {
// Skriv kode her
}
```

Løsningsforslag:

```
static void sum(double[][] tabell {
  int n = tabell.length;
  int m = tabell[0].length;
  int minste;

  if (n < m)
    minste = n;
  else
    minste = m;

  double sum = 0.0;
  for(int i=0;i<minste;i++)
    for(int j=i+1;j<n;j++)
      sum += tabell[i][j];

  System.out.println(sum);
}
```

Oppgave 1c (8%)

Hva blir skrevet ut når dette programmet kjøres? Du trenger ikke å grunngi svaret ditt.

```
public class ShellGame {

  public static void main(String[] args) {
    Kopp kopp1 = new Kopp(true);
    Kopp kopp2 = new Kopp(false);
    Kopp kopp3 = new Kopp(false);

    byttOm(kopp1,kopp2,kopp3);

    System.out.println(kopp1.hentErt());
    System.out.println(kopp2.hentErt());
    System.out.println(kopp3.hentErt());
  }

  static void byttOm(Kopp k1,Kopp k2,Kopp k3) {
```

```
    boolean b1 = k1.hentErt();
    boolean b2 = k2.hentErt();
    boolean b3 = k3.hentErt();

    k1.settErt(b2);
    k2.settErt(b3);
    k3.settErt(b1);

    Kopp temp = k1;
    k1 = k3;
    k3 = k2;
    k2 = temp;
  }
}

/** klasse for en kopp som kan inneholde en ert */
public class Kopp {

  /** Sann dersom koppen inneholder en ert, ellers falsk */
  boolean harErt;

  Kopp(boolean harErt) { this.harErt = harErt; }

  boolean hentErt() { return this.harErt; }

  void settErt( boolean verdi) { this.harErt = verdi; } }
```

Løsningsforslag:

false
false
true

Opplysninger til oppgavene 2, 3 og 4.

I de resterende oppgavene skal det skrives metoder for å administrere et reisebyrå som selger billetter (klassen `Billett`) til togruter (klassen `TogRute`). For hver togrute har vi et billettsystem (klassen `BillettSalg`) som organiserer selve salget og holder orden på billettene.
En togrute går mellom to stasjoner (klassen `Stasjon`) og kan stoppe på andre stasjoner underveis. Toget kan ikke snu og går derfor bare i en retning. På hver stasjon gjør toget et opphold for å slippe av og ta på passasjerer.

Vi bruker følgende klasser med tilhørende feltvariabler:

`TogRute`: Informasjon om en bestemt togrute.

`stoppeSteder`	En tabell med samtlige stasjoner langs ruten.

`antallStasjoner`	Antall stasjoner langs ruten (inkludert utgangspunkt og endepunkt).
`avgangsTid`	Klokkeslettet når toget forlater første stasjon.
`stoppeTid`	Hvor lenge toget stopper på hver stasjon (unntatt første og siste stasjon).
`antallSeter`	Antall seter på toget.

`Stasjon`: Informasjon om en bestem stasjon.

`stasjonsNavn`	Navnet til stasjonen.
`reiseTid`	Reisetid til neste stasjon (0 hvis det er siste stasjon).

`Billett`: Informasjon om en billett.

`fraStasjon`	Navnet på stasjonen som billetten gjelder fra.
`tilStasjon`	Navnet på stasjonen som billetten gjelder til.
`avgangsTid`	Klokkeslettet når toget går fra <`fraStasjon`>.
`ankomstTid`	Klokkeslettet når toget ankommer <`tilStasjon`>.
`pris`	Hva billetten koster.
`tog`	Hvilken togrute billetten gjelder for.

`BillettSalg`: Informasjon om billettsalg for en bestemt togrute.

`tog`	Togruten som det selges billetter for.
`reservertePlasser`	En tabell som viser hvor mange plasser som er opptatt mellom hvert par av påfølgende stasjoner. Dvs at `reservertePlasser[i]` inneholder hvor mange som reiser mellom stasjon i og stasjon i+1.
`solgteBilletter`:	En tabell med alle billetter som er solgt.

`antallSolgteBilletter`: Antall billetter som er solgt.

Alle tidspunkt representeres som minutter (heltall). Det gjør at vi enkelt kan legge sammen tidspunkt. Det er kun ved utskrift vi konverterer tidspunkt tilbake til timer og minutter. For å gjøre dette bruker vi en hjelpeklasse `Tid` med to statiske metoder, en for å konvertere timer og minutter til minutter og en som konverterer minutter til en tekststreng på formatet `tt:mm`. Disse metodene er gitt i oppgaven. Klassen `ReiseByraa` er gitt helt til slutt i oppgavesettet. Denne klassen inneholder en `main`-metode som setter opp en togrute fra Bergen til Voss med tilhørende billettsalg. Deretter selges det tre billetter før togruten og billettsalget blir skrevet ut. Det gir følgende utskrift:

```
Tog fra Bergen til Voss med avgang 20:29 og ankomst 21:25.
Antall plasser: 12
Rute:
Bergen - Arna
Avgang 20:29, Ankomst 20:39
Arna - Dale
Avgang 20:42, Ankomst 21:02
Dale - Voss
Avgang 21:05, Ankomst 21:25
```

```
Billettsalg for tog fra Bergen til Voss med avgang klokken 20:29.
Antall solgte billetter: 3
Omsetning: 685.0 kroner.
Seteutnyttelse: 0.17.
```

I oppgave 2, 3 og 4 skal du nå skrive kode for flere av metodene som brukes. Hva metodene gjør er beskrevet i tilhørende kommentarer.

Oppgave 2
Skriv kode for følgende metoder i klassen `TogRute`:
a) (5%) `TogRute`
b) (6%) `finnStasjonsIndeks`
c) (7%) `finnAnkomstTid`
d) (6%) `finnAvgangsTid`
e) (7%) `skrivTogRute`

Oppgave 3
Skriv kode for følgende metoder i klassen `Billett`:
a) (5%) `Billett`
b) (6%) `finnPris`

Oppgave 4
Skriv kode for følgende metoder i klassen `BillettSalg`:
a) (5%) `BillettSalg`
b) (6%) `ledigSete`
c) (6%) `seteReservasjon`
d) (6%) `samletOmsetning`
e) (6%) `seteUtnyttelse`
f) (6%) `skrivBillettSalg`

Her følger koden for de ulike klassene.

```
/** Klasse for å håndtere tid. Innholder ingen oppgaver.*/

public class Tid {

  /**
    * Statisk metode for å regne om et tidspunkt gitt som timer og minutter til
    * bare minutter.
    *
    * @param timer         antall timer.
    * @param minutter     antall minutter.
    *
    *  @return tidspunkt gitt som minutter. */

  public static int tilMinutter(int timer,int minutter) {

    return timer*60 + minutter;

  }


  /**
    * Statisk metode for å formatere et tidspunkt gitt i minutter til en tekstreng
    * på formen tt:mm.
    *
    * @param timer         antall timer.
    * @param minutter     antall minutter.
    *
    * @return tidspunkt angitt som tekststreng på formen tt:mm. */

  public static String formater(int minutter) {

    // String.format(streng, liste) oppfùrer seg akkurat som System.out.printf(streng,liste)
    // bare at strengen returneres istedenfor å skrives ut til skjermen.

    return String.format("%02d:%02d",minutter/60, minutter%60);
  }
}
```

```
/** Klasse for å representere en togstasjon. Inneholder ingen oppgaver.*/

public class Stasjon {

  /** Navn på stasjonen. */
  private String stasjonsNavn;

  /** Reisetid i minutter til neste stoppested, 0 dersom dette er siste stopp. */
  private int reiseTid;

  /**
   * Konstruktùr for å opprette en stasjon.
   *
   * @param stasjonsNavn gir navn på stasjonen
   * @param reiseTid gir reisetid i minutter til neste stasjon */

  public Stasjon(String stasjonsNavn, int reiseTid) {

    this.stasjonsNavn = stasjonsNavn;
    this.reiseTid = reiseTid;

  }

  /** @return navn til stasjonen. */
  public String hentNavn() { return stasjonsNavn; }

  /** @return reisetid til neste stasjon. */
  public int hentReiseTid() { return reiseTid; }

}
```

```
/** Klasse for å representere en togrute. Inneholder oppgave 2. */

public class TogRute{

  /** Tabell med samtlige stasjoner langs ruten (inkludert fùrste og siste stasjon).
      Stasjonene ligger i samme rekkefùlge som toget vil besùke dem. */
  private Stasjon[] stoppeSteder;

  /** Antall stasjoner langs ruten (inkludert fùrste og siste stasjon). */
  private int antallStasjoner;

  /** Tidspunktet (i minutter) når toget kjùrer fra fùrste stasjon. */
  private int avgangsTid;

  /** Antall minutter som toget står på hver stasjon fùr det kjùrer videre. */
  private int stoppeTid;

  /** Antall seter på toget (merk vi har ingen ståplasser). */
  private int antallSeter;

  /**
   *  Konstruktùr for å opprette en togrute.
   *
   *  @param avgangstid    tidspunktet (i minutter) når toget starter fra fùrste stasjon.
   *  @param stoppeTid     antall minutter toget står på hver stasjon.
   *  @param antallSeter   antall seter på toget.
   *  @param stoppeSteder tabell med alle stasjoner langs ruten. */

 public TogRute(int avgangsTid,int stoppeTid,int antallSeter,Stasjon[] stoppeSteder) {
  /**************
    Oppgave 2a
  **************/

    this.avgangsTid = avgangsTid;
    this.stoppeTid = stoppeTid;
    this.antallSeter = antallSeter;
    this.stoppeSteder = stoppeSteder;
    this.antallStasjoner = stoppeSteder.length;
  }

  /** @return antall stasjoner. */
  int hentAntallStasjoner() { return antallStasjoner; }

  /** @return antall seter. */
```

```
int hentAntallSeter() { return antallSeter; }

/** @return avgangstid (i minutter) fra fùrste stasjon. */
int hentAvgangsTid() { return avgangsTid; }

/** @return den i©te stasjonen på togruten. Fùrste stasjon er nummer 0. */
Stasjon hentStopp(int i) { return stoppeSteder[i]; }


/**
 *  Gitt navnet på et stoppested, så returneres posisjonen til stoppestedet i
 *  stasjonstabellen. Dersom stoppestedet ikke finnes returneres -1.
 *
 *  @param navn   gir navnet til stasjonen.
 *
 *  @return posisjonen til stasjonen.  */

public int finnStasjonsIndeks(String navn){

/**************
  Oppgave 2b
 **************/

  for(int i=0;i<antallStasjoner;i++)
    if (stoppeSteder[i].hentNavn().equals(navn))
      return i ;

  return -1 ;
}



/**
 *  Metode for å bestemme når et tog ankommer en stasjon. Returnerer -1 hvis
 *  stasjonen ikke ligger på ruten eller hvis "navn" er fùrste stasjon.
 *
 *  @param navn   gir navnet på stasjonen.
 *
 *  @return tidspunkt (i minutter) for når toget ankommer stasjonen. */

public int finnAnkomstTid(String navn){

/**************
   Oppgave 2c
 **************/
```

```
  int indeks = finnStasjonsIndeks(navn);
  if ((indeks == -1) || (indeks == 0))
    return  -1;

  int ankomst = this.avgangsTid;

  for (int i=0;i<indeks;i++) {
    ankomst += stoppeSteder[i].hentReiseTid();
  }

  ankomst += (indeks-1) * this.stoppeTid;
  return ankomst;
}


/**
 *  Metode for å bestemme når et tog forlater et stoppested. Returnerer -1 hvis
 *  stasjonen ikke finnes på ruten, eller hvis "navn" er siste stasjon på ruten.
 *
 *  @param navn    gir navnet på stasjonen.
 *
 *  @return tidspunkt (i minutter) for når toget forlater stasjonen. */

public int finnAvgangsTid(String navn){

/***************
   Oppgave 2d
 ***************/
  int indeks = finnStasjonsIndeks(navn);

  if ((indeks == -1) || (indeks == antallStasjoner-1))
    return -1;

  if (indeks == 0)
    return this.avgangsTid;

  return finnAnkomstTid(navn) + this.stoppeTid;
}
```

```
  /**
   *  Utskrifts-metode for TogRute.
   *
   *  (Se eksemplet som er gitt i oppgavebeskrivelsen)
   *  Utskriften skal se ut som fùlger:
   *
   *  Tog fra <fùrste stasjon> til <siste stasjon> med avgang tt:mm og ankomst tt:mm.
   *  Antall plasser: <antall plasser>.
   *  Rute:
   *  <navn til fùrste stasjon> - <navn til andre stasjon>
   *  Avgang tt:mm, Ankomst tt:mm
   *  <navn til andre stasjon> - <navn til tredje stasjon>
   *  Avgang tt:mm, Ankomst tt:mm
   *  ...
   *  <navn til nest siste stasjon> - <navn til siste stasjon>
   *  Avgang tt:mm, Ankomst tt:mm  */

  public void skrivTogRute() {

  /**************
     Oppgave 2e
   **************/

    String start = stoppeSteder[0].hentNavn();
    String stopp = stoppeSteder[antallStasjoner-1].hentNavn();

    String str = "Tog fra " + start + " til " + stopp;
    str += " med avgang " + Tid.formater(avgangsTid) + " og ankomst ";
    str += Tid.formater(finnAnkomstTid(stopp)) + ".\n";

    str += "Antall plasser: " + antallSeter + "\n";

    str += "Rute: \n";
    for(int i=0;i<antallStasjoner-1;i++) {
      String fra = stoppeSteder[i].hentNavn();
      String til = stoppeSteder[i+1].hentNavn();
      str += fra + " - " + til + "\n";
      str += "Avgang " + Tid.formater(finnAvgangsTid(fra)) + ", ";
      str += "Ankomst " + Tid.formater(finnAnkomstTid(til)) + "\n";
    }
    System.out.println(str);
  }
}
```

```
/** Klasse for å representere en togbillett. Inneholder oppgave 3. */

public class Billett {

  /** Fra hvilken stasjon billetten gjelder */
  private String fraStasjon;

  /** Til hvilken stasjon billetten gjelder */
  private String tilStasjon;

  /** Avreisetidspunkt */
  private int avgangsTid;

  /** Ankomsttidspunkt */
  private int ankomstTid;

  /** Prisen til billetten */
  private double pris;

  /** Togruten som billetten gjelder for */
  private TogRute tog;


  /**
   *  Konstruktùr for å opprette en billett.
   *
   *  @param fraStasjon   stasjonen som billetten gjelder fra.
   *  @param tilStasjon    stasjonen som billetten gjelder til.
   *  @param tog           togruten som billetten gjelder for. */

  public Billett(String fraStasjon, String tilStasjon, TogRute tog){

  /**************
     Oppgave 3a
   **************/

    this.fraStasjon = fraStasjon;
    this.tilStasjon = tilStasjon;
    this.tog = tog;
    this.avgangsTid = tog.finnAvgangsTid(fraStasjon);
    this.ankomstTid = tog.finnAnkomstTid(tilStasjon);
    this.pris = finnPris();
  }
```

```
  /** @return pris til billetten. */
  public double hentPris() { return pris; }


  /**
   *  Metode for å bestemme pris på en billett. Du kan anta at feltvariablene "fraStasjon"
   *  og "tilStasjon" angir lovlige stasjoner slik at "fraStasjon" kommer fùr "tilStasjon".
   *  Startprisen for en billett er 30 kroner. Dessuten betaler man 5 kroner for hvert
   *  minutt reisen varer.
   *
   *  @return prisen på billetten. */

  private double finnPris() {

  /**************
    Oppgave 3b
   **************/

    return 30.0 + (this.ankomstTid ±this.avgangsTid)*5.0;
  }
}
```

```
/** Klasse for å organisere billettsalget til en togrute. Inneholder oppgave 4.*/

public class BillettSalg{

  /** Det maksimale antallet billetter som kan selges for en togrute. */
  final private static int MAX_BILLETTER = 500;

  /** Togruten som det selges billetter til. */
  private TogRute tog;

  /** Antall opptatte plasser mellom to påfùlgende stasjoner.
      reservertePlasser[i] gir antall opptatte plasser mellom stasjon i og stasjon i+1. */
  private int[] reservertePlasser;

  /** Tabell med alle solgte billetter. */
  private Billett[] solgteBilletter;

  /** Samlet antall solgte billetter. */
  private int antallSolgteBilletter;



  /**
   *  Konstruktùr for å opprette et billettsalg.
   *
   * @param tog  togruten som det selges billetter for. */

  public BillettSalg(TogRute tog) {

  /**************
    Oppgave 4a
  **************/

   this.tog = tog;
   int antallStasjoner= tog.hentAntallStasjoner();
   this.reservertePlasser = new int[antallStasjoner-1];
   this.antallSolgteBilletter = 0;
   this.solgteBilletter = new Billett[MAX_BILLETTER];
  }
```

```
/**
 *  Metode for å bestemme om det finnes en sitteplass mellom to stasjoner gitt ved sine
 *  indekser langs togruten. Returnerer sann dersom det finnes plass, ellers usann.
 *  Du kan anta at "fraIndeks" og "tilIndeks" er gyldige verdier.
 *
 *  @param fraIndeks    indeks til stasjonen det reises fra.
 *  @param tilIndeks    indeks tilstasjonen det reises til.
 *
 *  @return sant om det finnes ledig plass, ellers falsk. */

public boolean ledigSete(int fraIndeks, int tilIndeks) {

/**************
  Oppgave 4b
 **************/

  for(int i=fraIndeks;i<tilIndeks;i++)
    if (reservertePlasser[i] >= tog.hentAntallSeter())
      return false;

  return true;
}




/**
 *  Metode for å reservere et sete og få returnert en billett for ùnsket reise dersom den
 *  er mulig å gjennomfùre. Billetten lagres også i billett-tabellen. Dersom reisen ikke
 *  lar seg gjennomfùres skrives det ut en passende feilmelding og det returneres en
 *  null-referanse.
 *
 *  @param fra    stasjonen som det ùnskes en billett fra.
 *  @param til    stasjonen som det ùnskes en billett til.
 *
 *  @return billett for reisen, eventuelt null dersom reisen ikke lar seg gjennomfùre. */

public Billett seteReservasjon(String fra,String til){

/**************
  Oppgave 4c
 **************/

  int f = tog.finnStasjonsIndeks(fra);
```

```
  int t = tog.finnStasjonsIndeks(til);

  if ((t<=f) || (f == -1) || (t == -1)) {
    System.out.println("Reisen lar seg ikke gjennomfùre. \n");
    return null;
  }

  if (antallSolgteBilletter >= MAX_BILLETTER ) {
    System.out.println("Kan ikke selge flere billetter! Billetsystemet er fullt.");
    return null;
  }

  if (ledigSete(f,t)) {
    for(int i=f;i<t;i++)
      reservertePlasser[i]++;
  }
  else {
    System.out.println("Det er ikke plass på toget. \n");
    return null;
  }

  Billett b = new Billett(fra,til,tog);
  solgteBilletter[antallSolgteBilletter++] = b;
  return b;
}


/**
 *  Metode for å beregne samlet omsetning i kroner basert på billetter solgt så langt.
 *
 *  @return samlet omsetning. */

public double samletOmsetning() {

/**************
   Oppgave 4d
 **************/

  double omsetning = 0.0;
  for(int i=0;i<this.antallSolgteBilletter;i++) {
    omsetning += solgteBilletter[i].hentPris();
  }
  return omsetning;
}
```

```
/**
 *  Metode for å beregne seteutnyttelse. Seteutnyttelsen mellom to påfùlgende stasjoner
 *  er antall reserverte seter for denne strekningen delt på totalt antall seter. Dvs at
 *  dersom det finnes 100 seter på toget og det er reservert 25 seter mellom stasjon i og
 *  stasjon i+1 så er seteutnyttelsen for denne strekningen 25/100 = 0.25.
 *  Metoden skal beregne og returnere gjennomsnittlig seteutnyttelse for alle par av
 *  stasjoner som kommer direkte etter hverandre.
 *
 *  @return gjennomsnittlig seteutnyttelse. */

public double seteUtnyttelse() {

/**************
  Oppgave 4e
 **************/

  int antallStasjoner = tog.hentAntallStasjoner();
  double sum = 0.0;
  for(int i=0;i<antallStasjoner-1;i++) {
    sum += (double) reservertePlasser[i]/tog.hentAntallSeter();
  }
  return sum/(antallStasjoner-1);
}
```

```
/**
 *  Utskrifts-metode for BillettSalg.
 *
 *  (Se eksemplet som er gitt i oppgavebeskrivelsen)
 *  Utskriften skal se ut som fùlger:
 *
 *  Billettsalg for strekning <fùrste stasjon> til <siste stasjon>.
 *  Antall solgte billetter: <antall solgte billetter>.
 *  Samlet omsetning: <omsetning> kroner.
 *  Seteutnyttelse: <seteutnyttelse>.  */

public void skrivBillettSalg() {

/**************
  Oppgave 4f
 **************/
```

```
    int antallStasjoner = tog.hentAntallStasjoner();
    String str = "Billettsalg for tog fra " + tog.hentStopp(0).hentNavn();
    str += " til " + tog.hentStopp(antallStasjoner-1).hentNavn();
    str += " med avgang klokken " + Tid.formater(tog.hentAvgangsTid()) + ".\n";
    str += "Antall solgte billetter: " + antallSolgteBilletter + ".\n";
    str += "Omsetning: " + samletOmsetning() + " kroner.\n";
    str += "Seteutnyttelse: " + String.format("%.2f",seteUtnyttelse()) + ".\n";
    System.out.println(str);
  }
}

/**  Klasse for å håndtere setereservasjon på et tog. Inneholder ingen oppgaver. */

public class ReiseByraa {

  public static void main(String[] args) {

    Stasjon[] BergenVoss = new Stasjon[4];

    BergenVoss[0] = new Stasjon("Bergen",10);
    BergenVoss[1] = new Stasjon("Arna",20);
    BergenVoss[2] = new Stasjon("Dale",20);
    BergenVoss[3] = new Stasjon("Voss",0);

    TogRute BV = new TogRute(Tid.tilMinutter(20,29),3,12,BergenVoss);
    BillettSalg billetterBV = new BillettSalg(BV);

    Billett b1 = billetterBV.seteReservasjon("Bergen","Voss");
    Billett b2 = billetterBV.seteReservasjon("Arna","Voss");
    Billett b3 = billetterBV.seteReservasjon("Arna","Dale");

    BV.skrivTogRute();
    System.out.println();
    billetterBV.skrivBillettSalg();
  }
}
```