Skriftelig prøve den 7. juni 2000 Kursusnummer 49104

Kursusnavn: Programmering.

Tilladte hjælpemidler: Alle *skriftlige* hjælpemidler — ingen computere!

Opgavesættet består af fire opgaver, der har følgende vægtning (cirka):

Opgave 1	25%
Opgave 2	15%
Opgave 3	25%
Opgave 4	35%

Sættet bedømmes efter 13-skalaen og tæller $1/3$ af den samlede karater for kursus 49104 og 49105.

! **Bemærk:** der er mange sider i sættet fordi der er brugt store bogstaver og indsat sideskift for at øge læsevenligheden.

Opgave 1

Spørgsmål 1.1

Vis, hvad Java-programmet på næste side udskriver på skærmen når det køres.
Vink: Lav en tabel for de mulige `i` og `j` værdier i løkken:

`i`	`j`	udskrift
`0`	`0`	+
	`1`	⋮
	`2`	
	`3`	
`1`	`0`	
	⋮	

```
public class Opgave1_2
{
   public static void main (String[] args)
      {
         for(int i=0; i<4; i=i+1)
         {
            for(int j=0; j<4; j=j+1)
            {
               if (i==0 || i==3)
               {
                  if (j==0 || j==3)
                     System.out.print("+");
                  else
                     System.out.print("-");
               }
               else // i is 1 or 2
               {
                  if (j==0 || j==3)
                     System.out.print("|");
                  else
                     switch((i+j)%3)
                     {
                         case 0:
                            System.out.print("o"); break;
                         case 1:
                            System.out.print("*"); break;
                         case 2:
                            System.out.print("x"); break;
                     }
               } // end if-else (i==0 || i==3)
            } // end for j
            System.out.println();
         }
      }
}
```

Husk, at `n%3` er resten ved division med 3. For eksempel er `5%3` lig med `2` og `6%3` giver 0.

Spørgsmål 1.2

I denne opgave betragter vi udskrift af figurer af samme slags som den følgende:

```
      *
     ***
    ** **
   **   **
  **     **
**       **
```

Første linje består af fem mellemrum og en stjerne; anden linje af fire mellemrum og tre stjerner. Tredje linje består af tre mellemrum, fulgt af et mellemrum og to stjerner; fjerde linje af to mellemrum, fulgt af tre mellemrum og to stjerner. Og så videre. Den sidste linje har ingen mellemrum inden stjernerne.

Skriv en metode `private static void myShape(int n)`, som udskriver en figur som den ovenfor. Parameteren `n` til `myShape` metoden skal angive antallet af linjer i figuren, d.v.s. `myShape(6)` skal udskrive ovenstående figur.

Markér mellemrum i strenge med ␣, som i `"a␣b"`.

Andre eksempler:

`myShape(1):`

```
*
```

`myShape(2):`

```
 *
***
```

`myShape(3):`

```
  *
 ***
** **
```

`myShape(4):`

```
   *
  ***
 ** **
**   **
```

Opgave 2

Når man arbejder med aktier bruger man ofte et *løbende gennemsnit* for aktiekurserne. Et løbende gennemsnit for en dag er gennemsnittet af dagens og de to foregående dages kurs, for eksempel:

Dag	1	2	3	4	5	6
Kurs	50	52	45	41	52	48
Løbende gns.	–	–	49	46	46	47

Vi bruger Javas heltalsafrunding, det vil sige decimalerne kastes væk, så `16.67` afrundes til `16`. Følgende er givet:

```
public class RunningAverage
{
   public static
   void main (String[] args)
      {
         int[] shareValues = {40,20,39,38,30,42,
                              48,53,60,51,44,36};

         System.out.println("The running average is: "
                            + running(shareValues));
      }

   private static
   String running(int[] a)
      {
          ...
      }

}
```

Når `running` metoden er færdiggjort (rigtig kode i stedet for `...`) skal programmet give følgende udskrift:

```
The running average is: - - 33 32 35 36 40 47 53 54 51 43
```

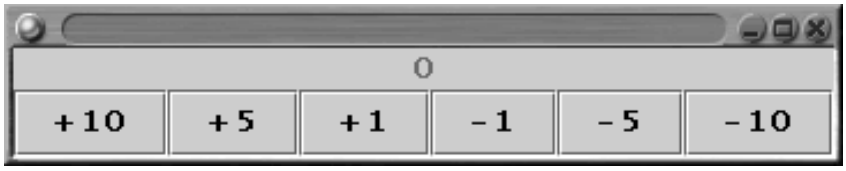
Man kan antage at argumentet `a` til `running` metoden har en længde på mindst 3.

Spørgsmål 2.1

Færdiggør `running` metoden.

Opgave 3

I denne opgave skal der laves et program, `Counter`, med grafisk brugergrænseflade. Et screenshot fra programmet kunne se således ud:



Der skal være et tal midt i øverste linje af vinduet — tallet repræsenterer værdien af en tæller, som kan justeres op og ned ved hjælp af knapperne i nederste linje.

Spørgsmål 3.1

Skitser hvordan man kan opnå et layout som vist ovenfor — angiv hvilke(n) layout manager(e) der skal bruges til vinduet og eventuelle undervinduer/paneler.

Spørgsmål 3.2

På næste side er et programskellet for `Counter`-programmet. Opgaven går ud på at færdiggøre dette program ved at løse spørgsmålene 3.2.A og 3.2.B nedenfor.

! Det er ikke nødvendigt at skrive hele programmet — nøjes med at skrive de metoder og klasser I laver.

Spørgsmål 3.2.A

Konstruktoren `Counter` skal initialisere instansdata og sørge for tilføjelse af de grafiske komponenter til dens *content pane* (husk at der er tale om en `JFrame`, så man kan ikke tilføje komponenter direkte).
Vink: Benyt metoden `String intToString(int n)`, der returner en strengrepræsentation af tallet `n` med et plus eller minus foran. For eksempel: `intToString(3)` giver strengen `+3`.

Spørgsmål 3.2.B

Listenerklassen `MyListener` skal lytte til knapperne, som justerer værdien af tælleren (den øjeblikelige værdi er gemt i `counterValue`).
Vink: Lav et listenerobjekt til hver enkelt knap og lad listenerobjektet holde styr på, hvor meget tællerens værdi skal ændres ved tryk på knappen som listenerobjektet lytter på.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Counter extends JFrame
{
   private JButton[] buttons;
   private JLabel label;
   private int counterValue;

   public Counter()
      {
          ...
      }

   private static
   String intToString(int n)
      {
         String res;
         if (n>0)
            res = "+";
         else
            res = "";
         return res + Integer.toString(n);
      }

   private class MyListener implements ActionListener
   {
       ...
      public
      void actionPerformed (ActionEvent e)
         {
             ...
         }
   } // MyListener

   public static
   void main (String[] args)
      {
         Counter f = new Counter();
         f.pack();
         f.show();
      }
}
```

Opgave 4

I denne opgave ser vi på fly og flyruter.

Spørgsmål 4.1

Denne delopgave omhandler repræsentation af fly i Java ved hjælp af følgende interface:

```
public interface Plane
{
   String id();
   int    capacity();
}
```

Metoden `id()` returnerer flyet identifikationskode f.eks. `OY-4388` og `capacity()` returnerer hvor mange passagerer der plads til på flyet.
Lav en klasse `DC_9` som implementerer `Plane` interfacet. Et DC-9 fly har plads til 100 passagerer. Konstruktoren skal have følgende signatur: `DC_9(String id)` — argumentet `id` angiver flyets identifikationskode.

Spørgsmål 4.2

For en flyrute mellem to lufthavne repræsenteres en specifik flyvning af klasser, som implementerer `Flight` interfacet:

```
public interface Flight
{
   Plane  plane();
   String from();
   String to();
   int    passengers();
   int    available();
   void   bookSeats(int seats) throws OverBooked;
}
```

Metoderne skal have følgende virkemåde:

`Plane plane()`	det fly, som bruges til denne flyvning.
`String from()`	afgangslufthavnen.
`String to()`	ankomstlufhavnen.
`int passengers()`	antal passagerer der har booket en plads på turen.
`int available()`	antal ledige pladser på flyet.
`void bookSeats(int seats) throws OverBooked`	booker `seats` pladser på turen, hvis der er plads, ellers kastes en `OverBooked` exception.

Definitionen af `OverBooked` exceptionen brugt i `Flight` interfacet ser således ud:

```
public class OverBooked extends Exception
{
   public
   OverBooked (String message)
      {
         super(message);
      }
}
```

Følgende del af klassen `MyFlight` som implementerer `Flight` interfacet er givet:

```
public class MyFlight implements Flight
{
   private Plane plane;
   private String from;
   private String to;
   private int passengers;
    ...
} // MyFlight
```

Skriv `MyFlight` klassens metoder `available` og `bookSeats`, hvor `bookSeats` metoden skal kaste en `OverBooked` exception, hvis antallet af nye bookinger plus de gamle overstiger flyets kapacitet. Ved oprettelse af et `OverBooked` objekt skal man kalde konstruktoren med en passende fejlbesked.

! Resten af metoderne skal *ikke* skrives.

Spørgsmål 4.3

Det er almen viden, at der ikke er direkte forbindelser mellem alle lufthavne; ofte bliver man nødt til at rejse via en anden lufthavn.

For eksempel: hvis der fra København til München er plads til 10 passagere og fra München til Saarbrücken er plads til 3 passagere, så derfor er det muligt at rejse fra København til Saarbrücken med mellemlanding i München.
NB: der skal være ledige pladser på begge flyene for at rejsen er mulig.

Denne opgave går ud på at færdiggøre `twoLegs` metoden nedenfor, som skal returnere antallet af mulige flyvninger mellem to byer med én mellemlanding.

For eksempel kan man rejse fra København til Saarbrücken med mellemlanding i Dortmund eller München — altså to muligheder.

Arrayet `fs` af typen `Flight[]`, som er argument til `twoLegs` metoden, repræsenterer alle de mulige flyvninger. Dette array skal gennemsøges for at finde alle mulige flyvninger med mellemlanding mellem `from` og `to` lufthavnene (andet og tredje argument til `twoLegs`).

```
int twoLegs(Flight[] fs, String from, String to)
{
    ...
}
```