CHALMERS TEKNISKA HÖGSKOLA
Datavetenskap TDA547

Tentamen i Objektorienterad programmering

Fredagen 9 mars 2012, 8.30 – 12.30.
Jourhavande lärare: Björn von Sydow, tel 0722/391401.

Inga hjälpmedel.

Lösningar till uppgifterna behöver ej kommenteras. Ej heller behöver nödvändiga importer anges. Triviala syntaxfel tolereras utan poängavdrag vid rättningen. Skriv läsligt!

Skrivningen kan ge maximalt 40 p; 20 p ger med säkerhet godkänt. 27 p ger betyget 4 och 33 p betyget 5.

1. (a) En klass vars instanser fungerar som tärningar kan ha följande struktur:

```
public class Dice {
  int value;

  public Dice() {...}

  public void roll() {...}
  public int getValue() {...}
}
```

Ett anrop av `getValue` föregånget av ett anrop av `roll` ska ge ett slumpmässigt heltal mellan 1 och 6. Mellan anropen av `roll` ska nya anrop av `getValue` ge samma resultat. Komplettera klassen så att den blir fullständig. (2 p)

(b) Utskriften från ett testprogram som testar klassen `Dice` kan se ut så här:

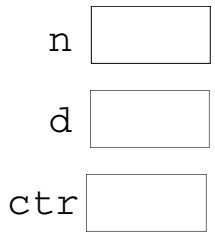
```
> java Uppgift1 10000
0  0
1  1734
2  1614
3  1666
4  1597
5  1694
6  1695
7  0
8  0
9  0
```

Programmet simulerar 10000 tärningskast och skriver ut antalet gånger värdena 0, 1, 2 . . . 9 förekommer. Förstås bör värdena 0, 7, 8 och 9 inte förekomma, men vi testar ju klassen! Eftersom slumpen är inblandad kan utskrifterna variera från körningstillfälle till körningstillfälle.

Testprogrammet, som ges nedan, använder sig av ett objekt av klassen `Counters`, som håller rätt på antalet gånger värdena 0, 1, 2, ... 9 förekommer. Din uppgift är att definiera klassen `Counters`. Vilka metoder som ska finnas och vilken parameter konstrueraren tar framgår av testprogrammet. (4 p)

```
public class Uppgift1 {
   public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);
      Dice d = new Dice();
      Counters ctr = new Counters(9); \\ count values up to 9
      for (int i=0; i<n; i++) {
         d.roll();
         ctr.count(d.getValue());
      }
      for (int i=0; i <= ctr.getMaxValue(); i++))
         System.out.println(i + "  " + ctr.getCounter(i));
   }
}
```

(c) Rita en principskiss av hur minnet ser ut då testprogrammet körs, just före `for`-loopen med utskrifterna. Din skiss bör utgå från



och fylla i innehållet i minnescellerna. Referenser anges med pilar till andra minnesceller, vars innehåll också måste anges. Man vet förstås inte tärningens värde vid tillfället; anta något rimligt värde. (3 p)

2. Programmet `Uppgift2` läser ett antal tal på kommandoraden och skriver ut deras medelvärde:

```
 > java Uppgift2 13.2 9.4 11 5.4 8
Medelvärdet är 9.4
 >
```

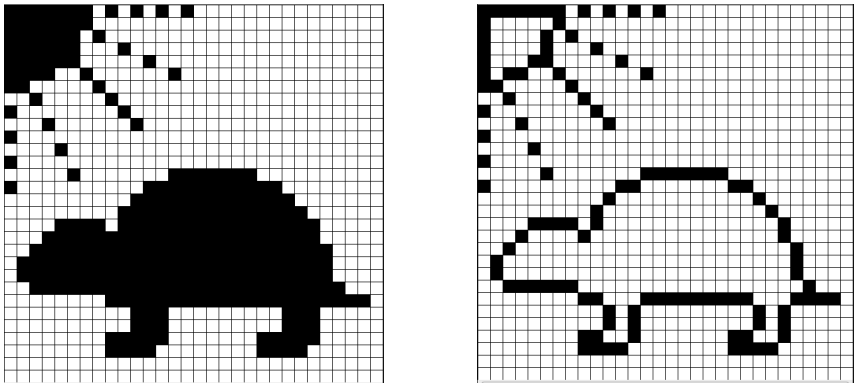
(a) Definiera en funktion
`public static double average(double [] a)`
som returnerar medelvärdet av talen i fältet `a`. Du får anta att fältet har längd minst ett, så att division med noll undviks. (3 p)

(b) Komplettera metoden från (a) till en fullständig klass `Uppgift2` som testar metoden `average` som i ovanstående exempel. (3 p)

(c) Om vi till exempel har en fil med mätvärden som vi vill beräkna medelvärdet av kan det vara bättre att använda en funktion
`public static double average(Scanner in)`
för medelvärdesberäkningen. Vi behöver då inte spara alla tal i ett fält, vilket är olämpligt om vi har väldigt många värden. Definiera denna funktion. (3 p)

3. En svartvit bild kan representeras av en matris av booleska värden, dvs av en variabel av typen `boolean[][]`, där vi bestämmer att ett värde `true` i matrisen betyder att motsvarande pixel är svart.

   Givet en bild `image` bestämmer vi dess *kontur*, som också är en bild, på följande sätt: Pixel (x,y) i konturen är svart om dels pixel (x,y) i `image` är svart, dels denna pixel har minst en vit granne i `image`. Som grannar räknar vi här bara de fyra pixlarna till vänster om, höger om, under och ovanför (x,y). Nedan visas ett exempel; bilden till höger är konturen av bilden till vänster.



   Som man kan se längst upp till vänster i bilderna anser vi att alla kantpixlar har en vit granne utanför bilden.

   Definiera en funktion

```
public static boolean[][] contour(boolean[][] image)
```

   som beräknar konturen av den bild som fås som parameter. Funktionen ingår som (än så länge) enda funktion i biblioteksklassen `ImageLib`. (6 p)

4. I denna uppgift ska vi betrakta ett program för att interaktivt skapa sådana bilder som i föregående uppgift. Användargränssnittet visar endast en bild som ovan (plus de vanliga fönsterdekorationer som finns högst upp i alla fönster).

   I en första version interagerar användaren med programmet endast genom att klicka med musen i någon av de små kvadraterna; denna byter då färg från svart till vit eller tvärtom. Detta program kan byggas upp av tre klasser:

   - En modellklass `BitmapModel`, vars objekt som innehåller representationen av en bitmap som en matris av booleska värden. I konstrueraren anges bitmappens storlek, dvs antalet rader och kolumner. Klassen har i övrigt metoderna
     `int getWidth()` som ger antalet kolumner i matrisen.
     `int getHeight()` som ger antalet rader.
     `boolean getPixel(int x, int y)` som ger värdet av pixeln i rad x, kolumn y.
     `void flip(int x, int y)` som byter färg på pixeln i rad x, kolumn y.
   - En vyklass `BitmapView` för att rita upp bilden; denna klass är en subklass till `JPanel` och dess objekt fungerar som rityta; i `paintComponents` ritas rutmönstret och de svarta kvadraterna genom att konsultera modellobjektet. Här finns också muslyssnaren. Vi bryr oss inte om detaljerna i denna klass.

- En huvudklass som innehåller `main`-rutinen:

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

public class BitmapMain {

  public static void main(String [] args) {
    JFrame f = new JFrame("Bitmaps");
    BitmapModel m = new BitmapModel(30,30);
    BitmapView v = new BitmapView(m);
    f.add(v,BorderLayout.CENTER);
    f.pack();
    f.setVisible(true);
  }
}
```

(a) Implementera modellklassen `BitmapModel`. (4 p)

(b) Vi vill nu förbättra programmet så att man enkelt kan få konturen till den aktuella bilden. För detta lägger vi till en knapp under rutnätet; när användaren klickar på knappen ersätts den aktuella bilden med sin kontur. Gör de ändringar i modellklass och huvudklass som behövs.

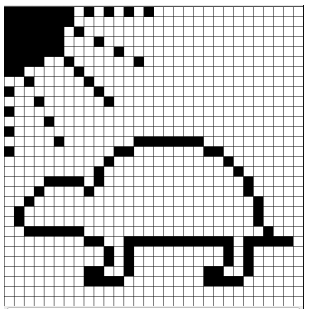
Observera att ni här får använda funktionen `contour` från uppgift 3 även om ni inte gjort den uppgiften. (6 p)

5. Vi fortsätter med de svartvita bilderna. Din uppgift är nu att definiera subrutinen

```
public static void fill(int x, int y, boolean[][] image)
```

som nästan är en invers till `contour`: subrutinanropet `fill(x,y,image)` skall, om pixeln (x,y) i `image` är vit, fylla det vita området i bilden runt (x,y) fram till närmaste del av konturen med svarta pixlar. Om (x,y) är svart har `fill` ingen effekt.

Om vi som exempel antar att `image` representerar den högra bilden av sköldpaddan i uppgift 3, så ska anropet `fill(3,3,image)` fylla solen igen och ge resultatet till höger (origo är högst upp till vänster, så pixeln $(3,3)$ är inne i solen). Däremot fylls inte sköldpaddan. Anropar vi `fill` en gång till med denna bild och (x,y) innanför sköldpaddans kontur, så återfår vi ursprungsbilden. Subrutinen `fill` fyller alltså bara ett delområde, begränsat av en svart kontur.



Vi tänker oss att denna rutin läggs till i biblioteksklassen `ImageLib`, dvs ni behöver inte lägga till denna möjlighet till det grafiska programmet i föregående uppgift. (6 p)