CHALMERS TEKNISKA HÖGSKOLA
Datavetenskap TDA547

Tentamen i Objektorienterad programmering

Lördagen 12 mars 2011, 8.30 – 12.30.
Jourhavande lärare: Björn von Sydow, tel 0762/981014.
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. Betrakta följande Javaprogram:

```
public class Uppgift1 {

    public static void incr(int n) {n++;}
    public static void incrFst(int[] a) {a[0]++;}

    public static void main(String[] args) {
        int k = 2;
        int[] v = {1,2,3};
        incr(k);
        System.out.println(k);
        incr(v[1]);
        System.out.println(v[1]);
        incrFst(v);
        System.out.println(v[0]);
    }
}
```

   Vilken utskrift fås när programmet körs? Förklara kortfattat. (4 p)

2. Definiera en klass `Uppgift2` med följande metoder:

   (a) `public static double length(double[] v),`
   som beräknar längden av vektorn `v`. Längden av en vektor $(x_0, x_1, \ldots x_{n-1})$ ges av $\sqrt{x_0^2 + x_1^2 + \ldots + x_{n-1}^2}$. (3 p)

   (b) `public static double[] normalized(double[] v),`
   som returnerar en normaliserad variant av `v`. Man normaliserar en vektor genom att dividera varje komponent med vektorns längd (vilket gör att den normaliserade vektorn har längd 1). (3 p)

   (c) En `main`-rutin som deklarerar och skapar en vektor, samt skriver ut den normaliserad. Du kan välja vektorn själv i `main` om du vill; den behöver inte läsas in från användaren. (3 p)

3. Vi betraktar ett program som hanterar information om hästar och deras släktskap[1]. En central klass i programmet är klassen `Horse`, vars objekt representerar en enskild häst. Klassen erbjuder följande metoder:

   - `public String getName();` returnerar hästens namn.
   - `public int getNrOfChildren();` returnerar antalet barn hästen har.
   - `public Horse getChild(int i);` returnerar barn nummer `i`.
   - `public void addChild(Horse h);` lägger till `h` som ytterligare ett barn.

   En realistisk klass skulle innehålla många fler metoder (för att ge födelsedatum, kön, föräldrar, ras, mm), men vi bortser här från detta. Implementera klassen `Horse` med ovanstående metoder; du får själv välja lämpliga parametrar till konstrueraren. (5 p)

4. Vi vill utveckla ett program som listar alla ord som finns i en engelsk text, med ett ord per rad. Ett ord definieras som en följd (engelska) bokstäver; vi bortser från komplikationen att ett ord kan fortsätta på nästa rad (och då avslutas med bindestreck på den första raden).

   (a) Definiera först en statisk metod

   ```
   public static void printWords(Scanner in);
   ```

   som på terminalen listar alla ord som förekommer i `in` i den ordning de kommer. Om samma ord återkommer listas det igen.

   Observera att texten innehåller annat än bokstäver och blanktecken, såsom punkt, komma, semikolon, mm. Det räcker därför inte att använda `in.next()` för att hitta nästa ord. Vi påminner om metoden `String findInLine(String pattern)` i `Scanner`. Denna metod söker efter en sträng som matchar `pattern` på den aktuella raden.

   Denna rutin kommer alltså att återge texten nästan oförändrad, men med skiljetecken borttagna och bara ett ord per rad. (3 p)

   (b) I Java Collections Framework finns förutom listor också interfacet SortedSet<E>. En klass som implementerar detta interface kan ses som en mängd av element av typen `E`. Bland annat finns metoderna

   ```
   public boolean add(E e);
   public Iterator<E> iterator();
   ```

   Anropet `s.add(e)` lägger till `e` till `s` och returnerar `true` om `e` inte redan fanns där, annars returneras `false`. Som namnet `SortedSet` antyder kommer iteratorn att ge elementen i sorterad ordning, efter den naturliga ordningen.

   En implementation av detta interface finns också, klassen TreeSet<E>. Standardkonstrueraren i denna klass skapar en ny, tom mängd.

   Använd denna klass för att definiera

   ```
   public static void printWordsSorted(Scanner in);
   ```

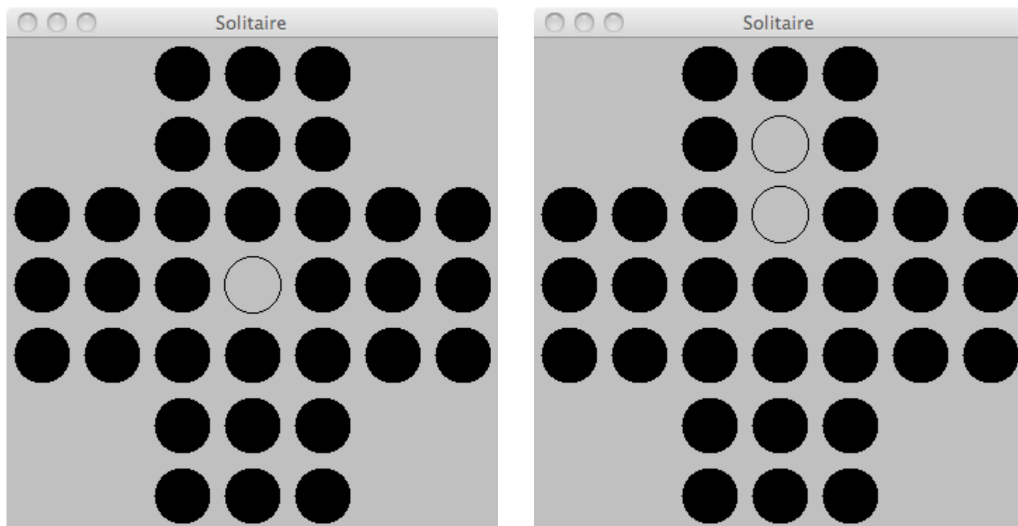
   som skriver ut alla ord som finns i `in` i sorterad ordning. Dessutom ska vi undvika att få med samma ord två gånger om det förekommer både som första ord i en mening (och därför börjar med stor bokstav) och inuti en mening. Detta görs genom att göra om alla ord till att bara bestå av små bokstäver. (4 p)

   (c) Lägg slutligen till en `main`-rutin som läser en text från standard input och skriver ut alla ord i texten i sorterad ordning. (1 p)

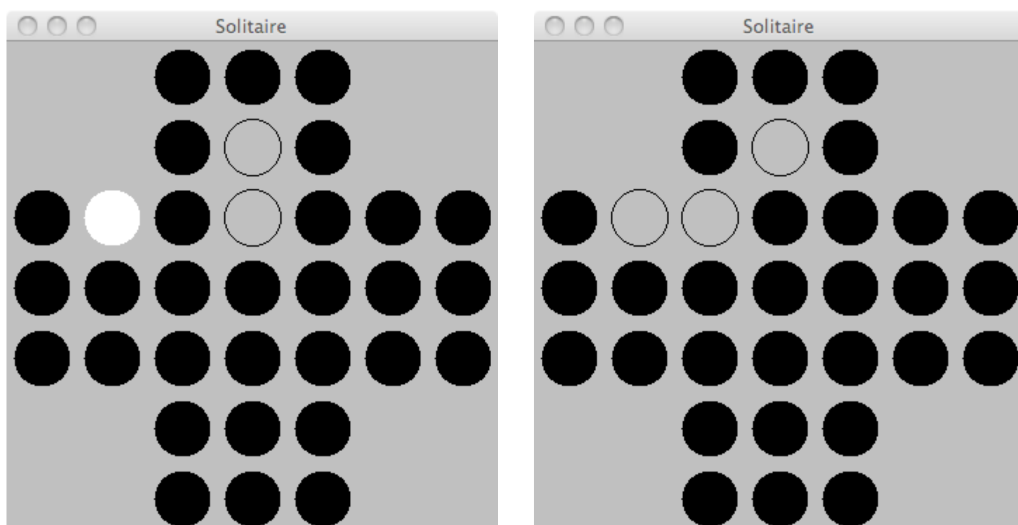
[1]Problemförfattaren vet mycket litet om hästavel. T ex säger man kanske inte barn om hästars avkomma?

5. *Solitär* är ett brädspel för en ensam spelare där 32 (svarta) kulor är placerade på en spelplan med 33 små fördjupningar som i bilden till vänster nedan; fördjupningen i mitten är tom. Ett drag i spelet består av att en kula hoppar över en grannkula (ovanför, nedanför, till vänster eller till höger) till platsen på andra sidan den överhoppade kulan, som måste vara tom. Den överhoppade kulan tas bort från spelplanen och spelet går ut på att få bort alla kulor utom en.

   I startläget är alltså fyra drag möjliga; man kan välja att flytta den kula som finns två steg från mitten åt något håll till mittpositionen. Den andra bilden visar läget efter det första draget om kulan två steg ovanför mitten valdes.

   Användaren gör ett drag i två steg; först klickar man på den kula man vill flytta; denna blir då *vald*, vilket syns genom att den visas vit, som till vänster nedan. Därefter klickar man på den tomma plats man vill hoppa till och draget utförs. Därefter är ingen kula vald, vilket visas till höger.

   Om draget är otillåtet (den nya platsen är inte tom, den ligger inte två steg ifrån eller den mellanliggande platsen är tom) sker inget mer än att ingen kula längre är vald. Om man ångrar ett påbörjat drag kan man alltså göra det ogjort genom att fullfölja med ett otillåtet drag.

Programmet utnyttjar en uppräkningstyp för situationen i varje plats på spelplanen:

```
public enum State {OCCUPIED, EMPTY, OUTSIDE}
```

En plats kan vara upptagen, tom eller *utanför*. Det sista alternativet gör det möjligt att betrakta spelplanen som en 7×7-matris av positioner; 16 platser i hörnen kommer alltid att vara utanför.

Dessutom finns tre klasser:

- En modellklass `SolitaireModel`, som representerar spelets tillstånd:

```
public class SolitaireModel {

    private static final int SIZE = 7;

    private State [][] board;

    public SolitaireModel() {
        board = new State[SIZE][SIZE];
        for (int r=0; r < SIZE; r++)
            for (int c=0; c < SIZE; c++)
                if ((r < 2 || r > SIZE-3) && (c < 2 || c > SIZE-3))
                    board[r][c] = State.OUTSIDE;
                else
                    board[r][c] = State.OCCUPIED;
        board[3][3] = State.EMPTY;
    }

    public void move(int r0, int c0, int r, int c) {
     // Överhoppas här
    }

    public State getState(int r, int c) {
        return board[r][c];
    }

    public int getSize() {
        return SIZE;
    }
}
```

  Den överhoppade metoden `move` tar som parametrar två positioner för ett drag, startpositionen `(r0,c0)` och slutpositionen `(r,c)`. Om det är ett giltigt drag att flytta en kula i startpositionen till slutpositionen så genomförs detta drag; annars sker ingenting.

  Notera att modellklassen inte vet någonting om valda platser; först när bägge positionerna för ett tänkt drag är kända anropas `move`.

- En vyklass `SolitaireView`, som visualiserar spelplanen:

```
public class SolitaireView extends JPanel {

    private static final int MARGIN = 5;
    private static final int CIRCLE_SIZE = 40;
```

```
    private static final int GRID_SIZE = CIRCLE_SIZE+2*MARGIN;
    private SolitaireModel model;
    private boolean hasSelection;
    private int selRow, selCol;

    public SolitaireView(SolitaireModel model) {
        this.model = model;
        setPreferredSize(new Dimension(model.getSize()*GRID_SIZE,
                                       model.getSize()*GRID_SIZE));
        setBackground(Color.LIGHT_GRAY);
        addMouseListener(new MyListener());
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        for(int r = 0; r < model.getSize(); r++)
            for(int c = 0; c < model.getSize(); c++) {
                g.setColor(Color.BLACK);
                State state = model.getState(r,c);
                if (state == State.OCCUPIED)
                    g.fillOval(MARGIN+c*GRID_SIZE,MARGIN+r*GRID_SIZE,
                               CIRCLE_SIZE,CIRCLE_SIZE);
                else if (state == State.EMPTY)
                    g.drawOval(MARGIN+c*GRID_SIZE,MARGIN+r*GRID_SIZE,
                               CIRCLE_SIZE,CIRCLE_SIZE);
            }
        if (hasSelection) {
            g.setColor(Color.WHITE);
            g.fillOval(MARGIN+selCol*GRID_SIZE,MARGIN+selRow*GRID_SIZE,
                       CIRCLE_SIZE,CIRCLE_SIZE);
        }
    }

    private class MyListener extends MouseAdapter {
        public void mouseClicked(MouseEvent e) {
            // Överhoppas här
        }
    }
}
```

Tillståndsvariabeln `hasSelection` håller reda på om någon kula är vald; i så fall finns rad och kolonn i `selCol` och `selRow`. Detta framgår också i `paintComponent`, där en vit kula ritas endast om `hasSelection` är `true`.

Muslyssnarens metod `mouseClicked`, som anropas av Java när användaren klickar med musen och som måste se till att dessa variabler har rätt värden, har överhoppats.

- En huvudklass med en `main`-rutin, som skapar ett modellobjekt, ett vyobjekt och ett fönster samt placerar vyn i fönstret, packar detta och gör det synligt. Överhoppas.

Dina uppgifter är nu:

(a) Definiera metoden `move` i `SolitaireModel`. (4 p)

(b) Definiera metoden `mouseClicked` i `SolitaireView`. (4 p)

6. Vi återknyter till klassen `Horse` i uppgift 3. Observera dock att man kan lösa denna uppgift även om man inte löst uppgift 3.

   Vi vill utveckla en klass `HorseLib` som innehåller funktioner och subrutiner relaterade till klassen `Horse`. Din uppgift är att påbörja detta genom att definiera en subrutin

   ```
   public static void printDescendants(Horse h)
   ```

   som kan skriva ut namnet på en häst och alla dess ättlingar (dvs barn, barnbarn, barnbarnsbarn ...). Här är ett exempel på utskrift:

   ```
   Brunte
       Arvid
           Tina
           Svarten
               Karolina
           Sverker
       Blanka
       Lotta
           Rosa
   ```

   Detta är resultatet av anrop av `printDescendants(h)` där hästen `h` heter Brunte och har tre barn: Arvid, Blanka och Lotta. Arvid i sin tur har tre barn (Tina, Svarten och Sverker) samt ett barnbarn (Karolina, barn till Svarten). Blanka har inga barn, medan Lotta har barnet Rosa. Utskriften ska se ut som ovan, dvs en häst per rad och med varje generation indragen fyra positioner. (6 p)