CHALMERS TEKNISKA HÖGSKOLA
Datavetenskap TDA546

Tentamen i Objektorienterad programmering E

Måndagen 8 mars 2010, 8.30 – 12.30.
Jourhavande lärare: Björn von Sydow, tel 1040.

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 swap(int i, int j, int[] a) {
        int tmp = a[i];
        a[i] = a[j];
        a[j] = tmp;
    }

    public static void main(String[] args) {
        int[] a = {3, 7, 9};
        swap(0,2,a);
        for (int i=0; i<a.length; i++)
            System.out.print(a[i] + " ");
        System.out.println();
    }
}
```

   Vilken utskrift fås när programmet körs? Förklara kortfattat, gärna med hjälp av en figur.
   (4 p)

2. I *Game of Life* studeras utvecklingen för en population av celler i en rektangulär värld. En modellklass `LifeModel` i ett program som kan simulera denna utveckling lagrar den aktuella generationen i en tillståndsvariabel med deklarationen

   ```
   private boolean[][] isAlive;
   ```

   Fältet skapas i konstrueraren:

   ```
   public LifeModel (int width, int height) {
      isAlive = new boolean[width][height];
   }
   ```

Cellen i position `(i,j)` är levande om `isAlive[i][j]` är `true`. Förutom de metoder som vi tidigare sett att modellklassen behöver vill vi lägga till ytterligare en metod:

```
public int aliveCells()
```

som returnerar antalet levande celler i den aktuella generationen. Definiera denna metod. Du behöver inte definiera klassen i övrigt. (4 p)

3. En enkel transform av en talföljd $x_0, x_1, x_2, \ldots x_{n-1}$ är löpande medelvärdesbildning: den transformerade följden är $y_0, y_1, y_2, \ldots y_{n-1}$, där y_i är medelvärdet av x_{i-1}, x_i och x_{i+1}, för $i = 1, 2, \ldots n-2$. Det första och sista elementet i resultatet kan inte beräknas med medelvärdesbildning och är desamma som motsvarande element i argumentet.

   Ett program som läser en följd tal på kommandoraden samt beräknar och skriver ut transformen kan som exempel ge följande resultat vid körning:

```
> java Uppgift2 1.5 1.7 2.2 2.1 2.3 1.9
1.5 1.8 2.0 2.2 2.1 1.9
>
```

   Det andra talet i utskriften, 1.8, är beräknat som medelvärdet av 1.5, 1.7 och 2.2, och så vidare.

   (a) Definiera en statisk funktion

   ```
   public static double[] average3(double[] a)
   ```

   som beräknar denna transform. (4 p)

   (b) Skriv ett program `Uppgift2` som testar funktionen `average3` och kan köras på ovanstående sätt. (4 p)

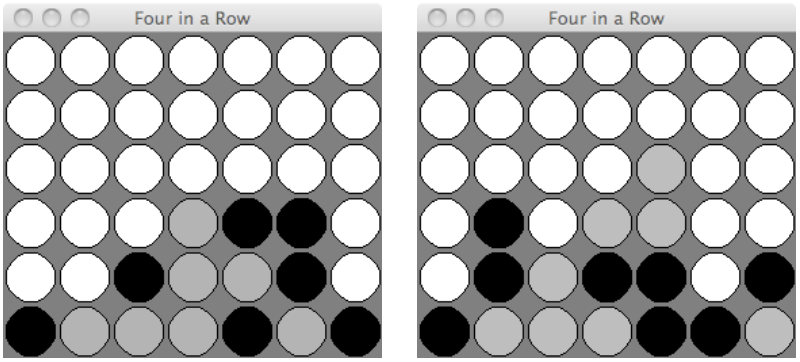
4. I ett Java-program för en webbutik finns bland annat en klass `Product` som lagrar information om produkter. Denna klass innehåller de publika metoderna `String getName()` och `int getPrice()`.

   Dessutom behöver programmet en klass `ShoppingCart`. En instans av denna klass lagrar kundens inköpslista (eller *kundvagn*, som den ofta kallas i svenska webbutiker) under en pågående session. Klassen erbjuder följande metoder:

   - `public void add(Product product, int quantity)`. Lägger till `quantity` enheter av produkten `product` till inköpslistan.
   - `public int totalAmount()`. Returnerar det totala belopp kunden ska betala för valda varor.
   - `public void print(PrintStream out)`. Skriver ut inköpslistan på `out`, med en vara per rad. Formatteringen av raden är inte viktig, men produktnamn, pris och antal valda ska finnas med.

   Definiera klassen `ShoppingCart`. (8 p)

5. *Fyra i rad* är ett spel med två deltagare som har brickor i varsin färg. Man turas om att placera brickor på spelplanen, som är vertikal så att brickorna som placeras i en kolumn ramlar neråt och hamnar så lågt som möjligt. Nedanstående bilder visar två möjliga spellägen (vita cirklar är tomma; grått och svart är här färgerna för de två spelarnas brickor):



Den spelare som står på tur väljer en kolumn; om denne i den vänstra bilden väljer mittkolumnen kommer den nya brickan att hamna omedelbart ovanför de tre gråa brickorna. Målet är att få fyra brickor i rad, horisontellt, vertikalt eller diagonalt. I bilden till höger har spelaren med grå brickor just vunnit med en diagonal rad som börjar nertill i andra kolumnen och växer uppåt åt höger.

I denna uppgift betraktar vi ett program där programmet spelar mot en mänsklig användare. Programmet är ofullständigt i flera avseenden:

- Datorns strategi är inget vidare: kolumnen för nästa bricka väljs på måfå.
- Programmet upptäcker inte när någon spelare har vunnit.

Programmet består av tre klasser; en modellklass, en vyklass och en huvudklass med en `main`-rutin. Dessutom används en uppräkningstyp för att representera tillståndet i en viss position:

```
public enum State {USER, PROGRAM, FREE}
```

Vyklassen är följande:

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

public class FIARView extends JPanel {

    private static final int SIZE = 40;
    private static final int DIAMETER = SIZE-4;
    private FIARModel model;

    public FIARView(FIARModel model) {
        this.model = model;
        setPreferredSize(new Dimension(SIZE*model.getWidth(),
                                       SIZE*model.getHeight()));
        setBackground(Color.GRAY);
        setOpaque(true);
```

```
        addMouseListener(new MyListener());
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        for (int x = 0; x < model.getWidth(); x++) {
            for(int y = 0; y<model.getHeight(); y++) {
                State s = model.getState(x,y);
                g.setColor(color(s));
                g.fillOval(SIZE*x+2, SIZE*(model.getHeight()-1-y)+2,
                           DIAMETER, DIAMETER);
                g.setColor(Color.BLACK);
                g.drawOval(SIZE*x+2, SIZE*(model.getHeight()-1-y)+2,
                           DIAMETER, DIAMETER);
            }
        }
    }

    private class MyListener extends MouseAdapter {
        public void mouseClicked (MouseEvent e) {
        // återstår att definiera
        }
    }

    private static Color color (State s) {
        if (s==State.USER) return Color.YELLOW;
        if (s==State.PROGRAM) return Color.BLUE;
        return Color.WHITE;
    }
}
```

Huvudklassen är

```
import javax.swing.*;

public class FIARMain {

    public static void main(String[] args) {
        FIARModel model = new FIARModel(7, 6);
        FIARView view = new FIARView (model);

        JFrame frm = new JFrame("Four in a Row");
        frm.getContentPane().add(view);
        frm.pack();
        frm.setVisible(true);
    }
}
```

(a) Definiera modellklassen `FIARModel`. Den ska lagra spelets tillstånd och tillhandahålla följande publika metoder:

- `void drop(int x)`. Ett anrop av denna metod placerar spelarens bricka i kolumn `x` och gör därefter ett slumpmässigt drag för programmets räkning. Om

kolumnen `x` redan är full görs ingenting (inte heller programmets drag).

- `State getState(int x, int y)`. Returnerar tillståndet i kolumn `x`, rad `y`. Kolumner numreras från vänster och rader med början längst ner.
- `int getWidth()`. Returnerar antalet kolumner i spelplanen.
- `getHeight()`. Returnerar antalet rader i spelplanen.

(8 p)

(b) Komplettera metoden `mouseClicked` i `FIARView`. Meningen är att användaren ska kunna klicka med musen var som helst i den kolumn han väljer. (2 p)

6. Följande interface kan användas för att hantera funktioner av en reell variabel i Java:

```
public interface Function {
    public double apply(double x);
}
```

Som exempel betraktar vi uppgiften att bestämma ett nollställe till en funktion: Antag att f är kontinuerlig och strikt växande på intervallet $a \leq x \leq b$, samt att $f(a) < 0$ och $f(b) > 0$. Då finns precis en mellanliggande punkt x_0 sådan att $f(x_0) = 0$.

Definiera en statisk funktion

```
public static double findZero(Function f, double a, double b, double eps)
```

som tar som argument en funktion f och ändpunkterna a och b på ett intervall så att ovanstående villkor är uppfyllda, samt en tolerans $\varepsilon > 0$. Funktionen `findZero` ska returnera en approximation `y` av nollstället x_0, sådan att $|\mathtt{y} - x_0| < \varepsilon$.

Skriv sedan ett program som finner det positiva nollstället till $f(x) = x - \cos x$ med ett fel som är högst 10^{-6}. (6 p)

Ledning: `findZero` kan beräkna f:s värde i intervallets mittpunkt och därmed avgöra i vilken halva nollstället ligger.