CHALMERS TEKNISKA HÖGSKOLA
Datavetenskap TDA546

Tentamen i Objektorienterad programmering E

Tisdagen 11 januari 2011, 8.30 – 12.30.
Jourhavande lärare: Pelle Evensen, tel 0708–482879.

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) Definiera en funktion

   ```
   public static int count(int n, int[] a)
   ```

   som returnerar antalet gånger heltalet `n` förekommer som element i fältet `a`. (4 p)

   (b) Skriv en klass `Uppgift1` som inkluderar funktionen `count` från (a) samt en `main`-rutin som testar `count`. Testet ska göras genom att `n` följt av ett antal tal som ska utgöra elementen i `a` läses från kommandoraden och resultatet av att anropa `count` skrivs ut. (4 p)

   Exempel på körningar:

   ```
   > java Uppgift1 1 2 9 1 1 7
   1 förekommer 2 gånger
   > java Uppgift1 3 3 4 5
   3 förekommer 1 gånger
   >
   ```

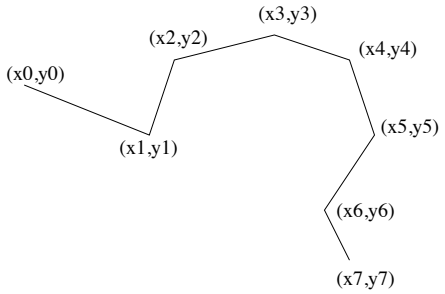
2. Betrakta följande två klasser:

```
public class Point {

    private int x, y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {return x;}
    public int getY() {return y;}
}

public class Uppgift2 {

    public static void proc(Point p, Point q) {
        p = q;
    }
```

```
    public static void main(String[] args) {
        Point p1 = new Point(1,2);
        Point p2 = new Point(3,4);
        Point p3 = p1;
        proc(p1,p2);
        System.out.println(p1.getX());
        System.out.println(p2.getX());
        System.out.println(p3.getX());
    }
}
```

Vilken utskrift fås när programmet `Uppgift2` körs? Illustrera med en principskiss med minnesceller och pekare just innan programmet avslutas. (4 p)

3. Ett program behöver hantera "vägar" i planet som i nedanstående exempel:



Vägen bestäms i exemplet av punkterna (x0,y0),(x1,y1),...(x7,y7); antalet punkter i en väg är förstås inte alltid åtta.

Definiera en klass `Path` vars instanser representerar en sådan väg. Som hjälpklass får du använda klassen `Point` från uppgift 2. Konstrueraren i `Path` ska ta en punkt (startpunkten) som argument. Vidare ska det finnas en metod `addPoint`, som också tar en punkt som argument och lägger till denna punkt som ny slutpunkt. Slutligen ska det finnas en metod `getPoints` som returnerar hela vägen, dvs alla punkterna, i någon form. Du får själv välja resultattyp; det kan vara en lista eller ett fält eller kanske något annat. (10 p)

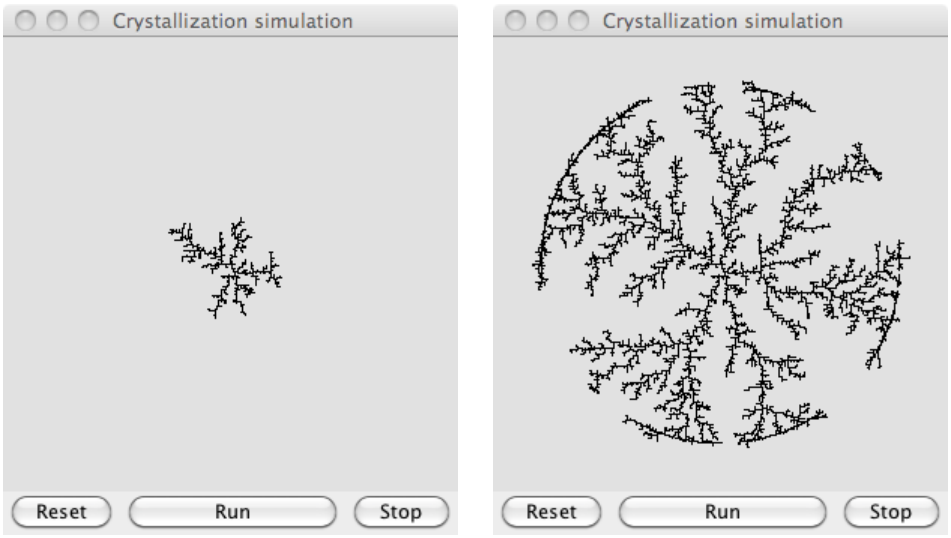
4. Följande `interface` beskriver vad ett `ImageFilter` är:

```
public interface ImageFilter {
    public String getMenuName();
    public void apply(int[][][] src, int[][][] dest);
}
```

Interfacet används i ett bildbehandlingsprogram; de två fälten `src` och `dest` representerar bägge en bild. Elementet `src[x][y][c]` betecknar intensiteten av färgen `c` i bildpunkten med koordinater (x,y). Här står c=0 för rött, c=1 för grönt och c=2 för blått. Ett filter filtrerar bilden i `src` på något sätt och lämnar resultatet i `dest`.

Ett filter som ger den ungefärliga effekten att se bilden genom en frostad glasskiva får man genom att låta intensiteterna i punkten (x,y) i `dest` tas från en annan, närliggande punkt i `src`; denna punkt väljs slumpmässigt så att dess x- och y-koordinater ligger högst fem bildpunkter från x resp y. Implementera detta filter. (8 p)

5. Vi betraktar ett program som simulerar kristallbildning i en joniserad lösning. Inga kunskaper om jonisering eller kristallbildning behövs för att förstå och lösa problemet. Två skärmdumpar av programmets gränssnitt under exekvering är följande:



Den växande kristallen är svart och består av ett stort antal partiklar, var och en endast en pixel stor. Kristallen växer genom en simulerad process där en partikel släpps i en slumpmässigt vald punkt på en cirkelperiferi, som man kan ana i den högra bilden. Partikeln startar då en slumpvandring, där den om och om igen flyttar sig ett steg åt vänster, höger, uppåt eller nedåt tills endera av två saker inträffar:

- Partikeln hamnar i en punkt (x, y) som är granne till kristallen, dvs någon av punkterna till höger, vänster, ovanför eller nedanför ingår redan i kristallen. Partikeln "fastnar" då på kristallen, som därmed växer genom att punkten (x, y) också kommer att ingå.
- Partikeln hamnar vid kanten på bilden; den anses då försvinna och bidrar inte till kristallens tillväxt.

Från början består kristallen av en enda partikel, i fönstrets mitt. När man trycker på Run-knappen startar simuleringen och man kan se kristallen växa. I bilden till vänster har simuleringen inte hunnit så långt, medan man i den högra bilden är nära dess slut; när hela startcirkeln ingår i kristallen växer kristallen inte mer. Slutligen noterar vi att Stop-knappen stoppar simuleringen, medan Reset-knappen stoppar simuleringen och återställer programmet till utgångsläget, där kristallen består av en enda partikel mitt i bilden.

Programmet består av fyra klasser:

- En modellklass `CrystalModel`, vars instanser lagrar information om kristallen. Klassen har metoder för att uppdatera klassen genom att släppa en ny partikel och låta den vandra, för att återställa till ursprungsläget samt för att avläsa tillståndet.
- En vyklass `CrystalWorld`, en subklass till `JPanel`, som används för att visualisera kristallen. Här omdefinieras `paintComponent`.
- En vyklass `CrystalView`, också en subklass till `JPanel`, som placerar kristallvyn och de tre knapparna lämpligt i förhållande till varandra samt definierar lyssnare till knapparna och den `Timer` som sköter animeringen.
- En huvudklass `Main`, som innehåller en `main`-rutin. Denna skapar ett modellobjekt, en vy och ett fönster samt placerar vyn i fönstret och gör detta synligt på skärmen.

De tre förstnämnda klasserna ges på följande sidor. Dina uppgifter är nu följande:

(a) Koden i metoden `actionPerformed` i klassen `ResetListener` (i `CrystalView`) har tagits bort. Lägg till den igen. (3 p)

(b) Skriv klassen `Main`. (3 p)

(c) Modifiera programmet så att längst ner till vänster i panelen med kristallen skrivs ett heltal, antalet partiklar i kristallen. Detta antal är från början 1 och växer med ett varje gång en ny partikel fastnar, så att man kan hela tiden kan se antalet partiklar i kristallen. Du behöver bara beskriva de ändringar som görs i de olika klasserna. (4 p)

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

public class CrystalWorld extends JPanel {

    private CrystalModel model;
    private final int size;

    public CrystalWorld(CrystalModel model) {
        this.model = model;
        size = model.getWidth();
        setPreferredSize(new Dimension(size,size));
        setBackground(new Color(225,225,225));
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.BLACK);
        for(int i=0; i<size; i++) {
            for (int j=0; j<size; j++) {
                if (model.getCell(i,j)) {
                    g.fillRect(i,j,1,1);
                }
            }
        }
    }
}
```

```
public class CrystalModel {

    private int width, mid;
    private boolean[][] crystal;
    private double rad;

    public CrystalModel(int r) {
        width = 2*r + 1;
        mid = r;
        rad = 0.8 * r;
        reset();
    }

    public int getWidth() {return width;}
    public boolean getCell (int x, int y) {return crystal[x][y];}

    public void reset() {
        crystal = new boolean[width][width];
        crystal[mid][mid] = true;
    }

    public void step() {
        double angle = Math.random() * 2 * Math.PI;
        int x = mid + (int)(rad * Math.cos(angle));
        int y = mid + (int)(rad * Math.sin(angle));
        while (true) {
            if (x == 0 || y == 0 || x == width-1 || y == width-1) return;
            if (crystalNeighbor(x,y)) {
                if (!crystal[x][y]) {
                    crystal[x][y] = true;
                }
                return;
            }
            switch ((int)(Math.random() * 4)) {
                case 0: x--; break;
                case 1: x++; break;
                case 2: y--; break;
                case 3: y++; break;
            }
        }
    }

    public void step(int k) {
        for (int i=0; i<k; i++) step();
    }

    private boolean crystalNeighbor(int x, int y) {
        return crystal[x-1][y] || crystal[x+1][y] || crystal[x][y-1] || crystal[x][y+1];
    }
}
```

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

public class CrystalView extends JPanel {

    private CrystalModel model;
    private CrystalWorld world;
    private Timer timer;

    public CrystalView(CrystalModel model) {
        this.model = model;
        world = new CrystalWorld(model);
        JButton resetButton = new JButton("Reset");
        JButton runButton = new JButton("Run");
        JButton stopButton = new JButton("Stop");
        setLayout(new BorderLayout());
        add(world,BorderLayout.NORTH);
        add(resetButton,BorderLayout.WEST);
        add(runButton,BorderLayout.CENTER);
        add(stopButton,BorderLayout.EAST);
        timer = new Timer(20,new StepListener());
        resetButton.addActionListener(new ResetListener());
        runButton.addActionListener(new RunListener());
        stopButton.addActionListener(new StopListener());
    }
    private class StepListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            model.step(200);
            world.repaint();
        }
    }
    private class ResetListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
        // To be completed by you.
        }
    }
    private class RunListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            timer.start();
        }
    }
    private class StopListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            timer.stop();
        }
    }
}
```