Tentamen 2006-0823
Objekt-orienterad programmering med Java, 1DL100
Sommarkurs och distanskurs

Uppsala Universitet
Instutitionen för informationsteknologi
Avdelningen för datalogi

Kursansvarig: Sven-Olof Nyström

May 2, 2007

Läs igenom följande instruktioner noggrant:

- Skrivtiden är fem timmar.
- Inga extra hjälpmedel är tillåtna.
- Skriv endast en lösning per blad och skriv namn på varje blad.
- Skriv inte på baksidan av bladen.
- Hänvisa inte mellan uppgifterna.
- Använd inte rödpenna.
- Oläsbara eller otydliga lösningar ger noll poäng per uppgift.
- Om tvetydigheter uppstår, gör intelligenta antaganden och ange dem.
- Om du har svårt att komma ihåg exakt hur någon metod eller klass i Javas standardbibliotek ser ut och fungerar, ange noggrant ditt antagande.

Maxpoäng är 40. För godkänd krävs vanligtvis 20 poäng och för väl godkänd 30 poäng.

Läs igenom hela tentan innan du börjar. Om du fastnar på en uppgift, gå vidare till nästa—uppgifterna är inte nödvändigtvis ordnade i svårighetsgrad. Om en uppgift består av flera deluppgifter och du fastnar på en av dem, gå vidare till nästa deluppgift—det kan hända att efterföljande deluppgifter kan lösas utan det fullständiga svaret på tidigare uppgifter.

1. Godis (8p)

Anta att godisaffären säljer fem slag av godis:

- lösgodis för 4kr per hekto,
- fint lösgodis för 8kr per hekto,
- inte så fint lösgodis för 1.5kr per hekto,
- påse med gelehallon för 11 kr per styck och
- enstaka lakritssnören för 0.9 kronor per styck.

(a) Definiera en abstrakt klass `Godis`. Den ska ha abstrakta metoder `pris()` och `vikt()`.

(b) Definiera lämpliga klasser för de olika typerna av godis. De ska ärva klassen `Godis` och definiera metoderna `pris()` och `vikt()`. (Ge bilar och bilpåsar lämplig vikt.) Tänk noggrant igenom hur de olika klassernas konstruktorer ska se ut.

(c) Skriv klassen Godispåse. Den ska definiera följande metoder:

- `void add( Godis g)` — placera godis i påsen
- `double vikt()` — returnera totalvikt
- `double pris()` — returnera sammanlagt pris

Vikt och pris ska beräknas vid metodanropet.

(d) Din implementation bör vara skriven så att det går att lägga till en ny typ av godis utan att göra ingrepp i den kod som du skrivit. Beskriv hur detta ska göras.

(e) Ge ett testprogram som provkör klassdefinitionen ovan genom att skapa en godispåse, fylla den med olika typer av godis och beräkna totalvikt och pris.

Om du vill får du anta att en godispåse inte kan lagra mer än ett visst antal objekt.

2. Referenser (6p)

Betrakta följande program

```
import java.util.*;

class Test {
    public static void main(String args[])  {
        int val;
        List<String> x, y;

        val = 10;
        x = new ArrayList<String>();
        x.add("äpplen");

        y = new ArrayList<String>();
        y.add("päron");

        System.out.println("val = " + val);
        System.out.println("x = " + x);
        System.out.println("y = " + y);
        System.out.println("");

        System.out.println("anropar m1");
```

```
        m1(val, x, y);
        System.out.println("återvänt från m1");
        System.out.println("");

        System.out.println("val = " + val);
        System.out.println("x = " + x);
        System.out.println("y = " + y);
  }

    public static void m1(int a, List<String> r1,
                          List<String> r2)  {
        System.out.println("i metoden m1...");
        a = 0;
        r1 = null;  //1
        r2.add(" smakar bra");
        System.out.println("a = " + a);
        System.out.println("r1 = " + r1);
        System.out.println("r2 = " + r2);
    }
}
```

(a) Beskriv i stora drag vad programmet gör.

(b) Beskriv vad programmet gör i detalj och vad det skriver ut.
(Tips: Om variabeln `l` är bunden till ett objekt av typen `ArrayList` som innehåller strängarna `"Hej"` och `"Hopp"` ger `System.out.println(l)` utskriften `[Hej, Hopp]`).

(c) Vad händer om en metod returnerar ett objekt?

(d) Finns det någon situation när ett objekt kopieras?

3. Hej hopp (6p)

Anta att vi har följande klassdefinitioner.

```
class Nisse {
    void go () {
        System.out.println("Hej");
    }
}

class Kalle extends Nisse {
    void go () {
        System.out.println("Hopp");
    }
}

class Olle extends Nisse {
   void go () {
        System.out.println("Hit");
    }

    void stop () {
        System.out.println("Dit");
    }
}
```

Betrakta följande kodfragment. Vilka accepteras av kompilatorn? Vilka ger fel vid körning? Vilka kan både kompileras och köras utan problem? Dina svar ska motiveras (kort).

(a)
```
Nisse a = new Kalle();
a.go();
```

(b)
```
Olle a = new Nisse();
a.go();
```

(c)
```
Olle a = new Kalle();
a.go();
```

(d)
```
Nisse a = (Nisse)new Olle();
a.stop();
```

(e)
```
Olle a = (Olle)(Nisse)new Kalle();
a.stop()
```

4. OOP (8p)

Hur ska ett välskrivet objektorienterat program se ut? Utgå från följande punkter.

(a) Val av klasser.

(b) Utformning av klassdefinitioner.

(c) Utformning av enskilda metoder.

(d) Användning av arv.

(e) Duplicerad kod.

(f) Utformning av programmet i stort.

Kan du komma på någon aspekt som ej täcks av punkterna ovan?

5. Samlingar (6p)

Skriv en metod som tar ett antal samlingar av strängar och skriver en lista av de strängar som förekommer i två eller tre av samlingarna.

Du bör använda collection classes, men oavsett vilken datastruktur du väljer för att representera indata, ange hur samlingarna representeras.

Ange också hur metoden är tänkt att anropas.

6. Scribble (6p)

(a) Betrakta programmet på nästa sida. Vad gör det?

(b) En brist hos programmet är att bilden inte är beständig. Hur kommer det sig? Skissa på hur programmet kan modifieras så att bilden ej försvinner om fönstret tillfälligt döljs.

(c) Anta att man vill lägga till möjlighet att rita andra figurer. Skissa på hur programmet måste ändras för att tillåta detta.

(d) Anta att man vill kunna redigera en bild genom att ta bort utritade linjer. Skissa på hur programmet måste ändras för att tillåta detta.

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

public class Scribble extends JFrame {

    public Scribble() {
        setTitle("Scribble Pad");
        canvas = new ScribbleCanvas();
        canvas.setBackground(Color.white);
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(canvas, BorderLayout.CENTER);
        addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    System.exit(0);
                }
            });
    }

    public static void main(String[] args) {
        int width = 600;
        int height = 400;
        JFrame frame = new Scribble();
        frame.setSize(width, height);
        Dimension screenSize =
            Toolkit.getDefaultToolkit().getScreenSize();
        frame.setLocation(screenSize.width/2 - width/2,
                          screenSize.height/2 - height/2);
        frame.show();
    }

    protected ScribbleCanvas canvas;

}


class ScribbleCanvas extends JPanel {

    public ScribbleCanvas() {
        listener = new ScribbleCanvasListener(this);
        addMouseListener((MouseListener) listener);
        addMouseMotionListener((MouseMotionListener) listener);

    }

    protected EventListener listener;
    protected boolean mouseButtonDown = false;
    protected int x, y;

}
```

```
class ScribbleCanvasListener
    implements MouseListener, MouseMotionListener {

    public ScribbleCanvasListener(ScribbleCanvas canvas) {
        this.canvas = canvas;
    }

    public void mousePressed(MouseEvent e) {
        Point p = e.getPoint();
        canvas.mouseButtonDown = true;
        canvas.x = p.x;
        canvas.y = p.y;
    }

    public void mouseReleased(MouseEvent e) {
        canvas.mouseButtonDown = false;
    }

    public void mouseDragged(MouseEvent e) {
        Point p = e.getPoint();
        if (canvas.mouseButtonDown) {
            canvas.getGraphics().drawLine(canvas.x, canvas.y,
                                          p.x, p.y);
            canvas.x = p.x;
            canvas.y = p.y;
        }
    }

    public void mouseClicked(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    public void mouseMoved(MouseEvent e) {}

    protected ScribbleCanvas canvas;

}
```

LYCKA TILL!

Sven-Olof

Appendix: något om samlingar och iteratorer

Metoder i gränssnittet Collection:

- add(obj): lägg in obj
- addAll(coll): lägg in alla objekt i coll
- clear(): tar bort samtliga objekt
- contains(obj): returnerar sant om obj finns i samlingen
- containsAll(coll): returnerar sant om alla objekt i coll finns i samlingen
- equals(coll): returnerar sant om coll och aktuell samling är lika.
- isEmpty(): returnerar sant om samlingen är tom
- iterator(): returnerar en Iterator som kan användas för att loopa genom samlingen
- remove(obj): tar bort obj ur samlingen
- removeAll(coll): tar bort objekten i coll ur samlingen
- retainAll(coll): tar bort alla objekt utom de som finns i coll ur samlingen
- size(): returnerar antal element i samlingen
- toArray(): returnerar en array med alla element i samlingen

Klasser som implementerar Collection:

- ArrayList (implementerar List)
- LinkedList (implementerar List)
- Vector (implementerar List)
- TreeSet (sorterad mängd, implementerar SortedSet)
- HashSet (mängd, implementerar Set)

Metoder i iteratorer:

- hasNext(): returnerar sant om det finns fler element
- next(): returnerar nästa element, flyttar ”pekaren”
- remove(): tar bort aktuellt element

Exempel på hur man använder iteratorer:

```
static void loop (Collection c) {
       for(Iterator i = c.iterator(); i.hasNext(); ) {
            Object obj = i.next();
            System.out.println( obj );
       }
   }
```

Metoder i gränsnittet Map (som implementeras av HashMap och TreeMap):

- int size() - returnerar antal (nyckel-värde)-par.
- void clear() - tar bort alla par i avbildningen.
- Object put(Object nyckel, Object värde) - lägger in (nyckel,värde)-paret, ersätter det gamla (det med samma nyckel) om det finns.
- Object get(Object nyckel) - returnerar värdet för nyckeln nyckel eller null om nyckel ej finns.
- Set entrySet() - returnerar ett Set med varje (nyckel-värde)-par i form av ett Map.Entry-element (se vidare nedan i loop-exemplet).
- Set keySet() - returnerar ett Set med bara nycklarna.
- Collection values() returnerar en samling med bara värdena.

Loop genom avbildning (mappen i exemplet) med utskrift av nyckel och värde:

```
Set mapAsSet = mappen.entrySet();
for (Iterator i = mapAsSet.iterator(); i.hasNext();) {
  Map.Entry me = (Map.Entry) i.next();
    System.out.println(me.getKey() + ": " + me.getValue());
}
```