UMEÅ UNIVERSITET
Inst. för Datavetenskap

PRAKTISKT PROV I DATAVETENSKAP

OOP I JAVA, 5P

Datum	:	020605
Tid	:	9-13
Hjälpmedel	:	Allt. Kommunikation med andra personer (direkt eller indirekt) är dock inte tillåten, som t ex via mail, mobiltelefon, gemensama kataloger etc.
Antal uppgifter	:	4
Totalpoäng	:	max 28 (halva poängtalet krävs normalt för godkänt)

- Provet består av 4 uppgifter : 1 och 2 samt 2 fritt valda uppgifter bland 3,4,5 & 6.
- Kryssa för de uppgifter du lämnar in.
- Källkod skrivs ut i ett ickeproportionellt typsnitt (t.ex. `courier`).
- Namn, personnummer, användarnamn och sökvägen till filen / filarna skall finnas på all källkod.
- Lösningarna skall vara snyggt och prydligt nedskrivna. Tankegången skall vara lätt att följa. Alla antaganden som inte är uppenbara skall redovisas.

OBS! Kontrollera att din dator fungerar innan du börjar på allvar.

Lycka till!

Uppgift 1 (6 poäng)

På kursens hemsidan hittar du appleten `Dots2` som är en utbyggnad av den `Dots` vi tittat på tidigare. Här kan man klicka många gånger och sätta ut godtyckligt många ”dots” på ritytan. Dessutom ges information om hur många ”dots” som satts ut totalt.

Skriv om appleten så att färgen på texten slumpmässigt varierar mellan färgerna röd, grön, blå, gul och ytterligare någon.

Koden till `Dots2` finns på websidan (och som appendix) :
http://www.cs.umu.se/kurser/TDBA62/VT02/kod0605/Dots2.java/

Uppgift 2 (6 poäng)

I denna uppgift skall du designa (ej implementera) en Java applikation som realiserar en mycket enkel miniräknare. Miniräknaren hanterar bara heltal och endast de fyra räknesätten +, -, * och /.
Det skall finnas knappar för siffrorna och operatorerna och dessutom en "="-knapp som genererar beräkningen. All evaluering sker från vänster till höger, dvs. i den ordning operatorerna knappas in.

Beskriv alla klasser, metoder och attribut som ingår i designen. Du får göra det med hjälp av UML diagram eller som halvfärdiga Java klasser. Alla klasser, metoder och attribut skall kommenteras.

I den följande delen får du lämna in högst två av uppgifterna

Uppgift 3 (6 poäng)

Största gemensamma delare (sgd) är ett begrepp som används t.ex. när man vill förenkla bråk.
Bråket $\frac{25}{10}$ kan förenklas till $\frac{5}{2}$ eftersom 5 är det största tal som delar både 25 och 10.
Ett annat exempel är $\frac{39}{91}$ som kan förenklas till $\frac{3}{7}$ eftersom 13 är det största tal som delar både 39 och 91. Det spelar alltså ingen roll vilket av talen i bråket som är störst.

Givet är följande rekursiva definition på största gemensamma delare (sgd) :

För alla positiva heltal tal *a* och *b* ($a,b > 0$) gäller

$$sgd(a,b) = \begin{cases} b & om \quad a \bmod b = 0 \\ sgd(b, a \bmod b) & annars \end{cases}$$

där mod är resten vid heltalsdivision.

Implementera en rekursiv metod som returnerar sgd(a,b) där a och b ges som parametrar. Implementera även sgd med en iterativ metod.

Gör metoderna statiska och lägg dem i en klass `MatteFunktioner` så att de enkelt kan testas i BlueJ.

Uppgift 4 (8 poäng)

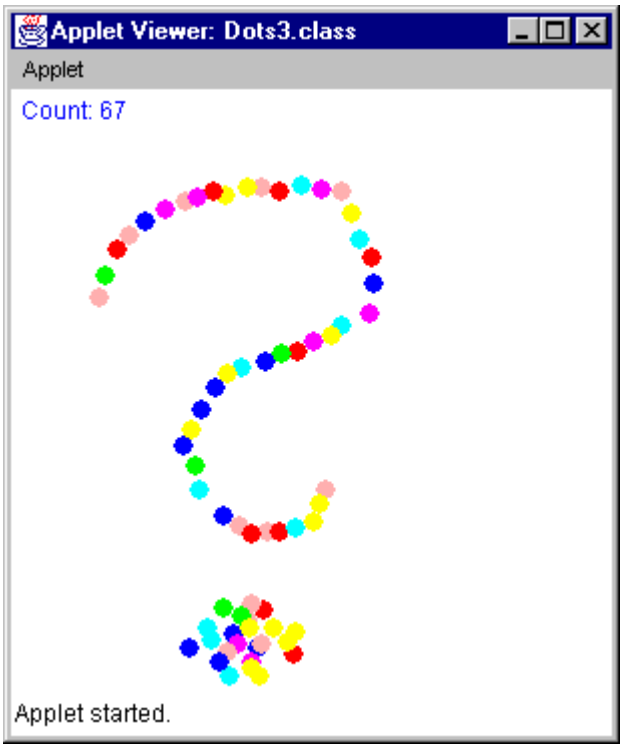
Implementera en metod **`binSearch`** som söker igenom ett sorterat fält med heltal. Sökningen skall göras med binär sökning, dvs. antalet möjliga element skall halveras vid varje kontroll. Detta åstadkomms genom att man jämför med det mittersta elementet och därefter vet om man ska fortsätta leta i den vänstra eller högra halvan av listan.
Metoden skall returnera index på det sökta elementet om elementet finns i fältet och `-1` om det sökta elementet inte finns i fältet.

Följande kod skall fungera ihop med din metod **`binSearch`**:

```
int index;
int[] lista = {…}; //En godtycklig lista med sorterade heltal
int tal = …;  // ett sökt värde

index = binSearch (tal, lista);
if (index >= 0)
  System.out.println ("lista[" + index + "] == " + lista[index]);
else
  System.out.println ("Värdet " + tal + " finns inte i listan");
```

Uppgift 5 (8 poäng)

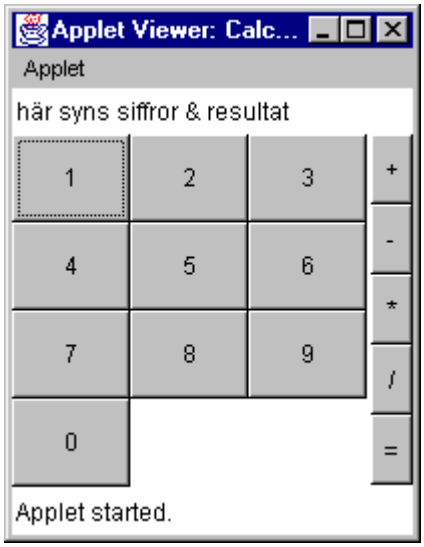


Utgå igen från appleten `Dots2`. I den använder man klassen `Point` för att notera vilken punkt som klickats, dessa lagras sedan eftersom i en `Vector`.
Konstruera en klass `Dot` som härleds från `Point`, och som dessutom känner till sin färg, vilken sätts slumpmässigt när objektet skapas (utnyttja gärna din lösning för texten i uppgift 1). Låt fortfarande ritandet ske i appleten, även om man här skulle kunna argumentera för att en `Dot` borde sköta detta själv.
Kod till Dots2 finns på websidan :

http://www.cs.umu.se/kurser/TDBA62/VT02/kod0605/Dots2.java

Uppgift 6 (8 poäng)

I denna uppgift skall du implementera räknaren som beskrivs i uppgift 2.
Utseendet kan vara enkelt, t.ex. :



```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class Dots2 extends Applet implements MouseListener
{
   private final int APPLET_WIDTH = 300;
   private final int APPLET_HEIGHT = 300;
   private final int RADIUS = 5;

   private Vector pointList;
   private int count;

   //-----------------------------------------------------------------
   //  Creates a Vector object to store the points.
   //-----------------------------------------------------------------
   public void init()
   {
      pointList = new Vector();
      count = 0;

      addMouseListener(this);

      setBackground (Color.white);
      setSize (APPLET_WIDTH, APPLET_HEIGHT);
   }

   //-----------------------------------------------------------------
   //  Draws all of the dots stored in the Vector.
   //-----------------------------------------------------------------
   public void paint (Graphics page)
   {
      page.setColor (Color.blue);

      // Retrieve an iterator for the vector of points
      Iterator pointIterator = pointList.iterator();

      while (pointIterator.hasNext())
      {
         Point drawPoint = (Point) pointIterator.next();
         page.fillOval (drawPoint.x - RADIUS, drawPoint.y - RADIUS,
                        RADIUS * 2, RADIUS * 2);
      }

      page.drawString ("Count: " + count, 5, 15);
   }

   //-----------------------------------------------------------------
   //  Adds the current point to the list of points and redraws the
   //  applet whenever the mouse is pressed.
   //-----------------------------------------------------------------
   public void mousePressed (MouseEvent event)
   {
      pointList.addElement (event.getPoint());
      count++;
      repaint();
   }

   //-----------------------------------------------------------------
   //  Provide empty definitions for unused event methods.
   //-----------------------------------------------------------------
   public void mouseClicked (MouseEvent event) {}
   public void mouseReleased (MouseEvent event) {}
   public void mouseEntered (MouseEvent event) {}
   public void mouseExited (MouseEvent event) {}
}
```