Laboration 1 – GUI-programmering

Syfte

Erbjuder studenterna en möjlighet att lära sig *grunderna i gränssnittsprogrammering i Java*.

Genomförande

Genomförs *individuellt* eller i grupp om *2 personer*.

Uppskattad tidsåtgång

16-40 timmar effektiv arbetstid per gruppmedlem, beroende på förkunskaper.

Informations- och bakgrundsmaterial

- Föreläsnings- och övningsanteckningar, ev. annat utdelat material.
- *The Swing Tutorial*, http://java.sun.com/docs/books/tutorial/uiswing/
- *2D Graphics Tutorial*, http://java.sun.com/docs/books/tutorial/2d/
- http://www.kidpad.org/ för inspiration.
- Google!

Uppgift

Konstruera en riteditor med två varianter av gränssnittet. Funktionaliteten kan vara samma. Prova dessa på ett par kompisar och redovisa vad de tycker är för- och nackdelar med varianterna.

Redovisning

Redovisas per e-post till Marin Berglund mabe02@kth.se och Yngve Sundblad y@kth.se senast **måndag 18 februari kl. 23.59**. Följande ska finnas med i redovisningen:

1. Studenternas (-ens) namn och personnummer.
2. Ett ZIP-arkiv med *källkod*, en *färdigkompilerad version* av programmet, samt *instruktioner för hur man startar det*.
3. En kort text om utvärderingen med kompisar.

Förslag på uppdelning (om ni jobbar i par)

Designa verktyget tillsammans genom att rita en skiss. Diskutera vad som skulle vara ett ”användarvänligt” gränssnitt. Genomför uppgifterna sida vid sida och hjälp varandra! Alternativt skriv några av klasserna var (men var noga med att bestämma vilket interface era klasser ska ha först så att det blir enkelt att integrera ihop dem).

Detaljbeskrivning

I konventionella riteditorer finns oftast en palett med olika verktyg. För att rita, selektera eller sudda väljs ett verktyg ur paletten. Därefter befinner sig editorn i en speciell mod för en kortare eller längre tid. Om man t.ex. väljer ellips i paletten, kanske man ritar ellipsen genom att definiera en rektangel på ritytan. Därefter återgår editorn till selektionsmod.

Du ska nu konstruera en lite annorlunda riteditor i vilken man kan placera minst två olika sorters ritverktyg på ritytan (t.ex. pensel, raderverktyg, kopieringsverktyg, ellipsritare, splineritare, etc.).

Det skall vara möjligt att ha flera "instanser" av en viss typ av verktyg samtidigt på ritytan. Om ett av verktygen är en penna, t.ex., ska man kunna ha tre olika sådana pennor "liggande på pappret". Varje instans skall kunna ställas in på sitt eget sätt (färg, tjocklek linjen, osv.). För att rita ska man kunna ”ta tag” i pennan med musen och rita. När man ”släpper” pennan ska den ligga kvar på sin plats tills man tar upp den igen.

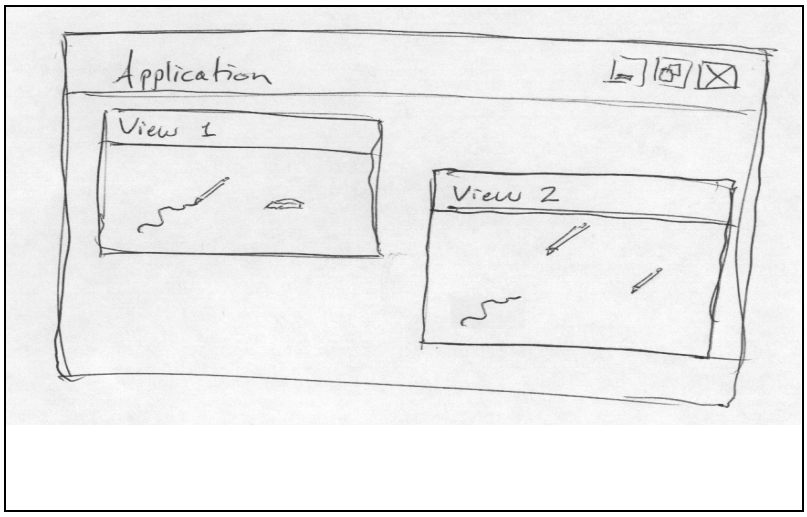
Det skall också vara möjligt att öppna flera fönster mot samma rityta. Om ett objekt läggs till, tas bort eller på annat sätt manipuleras så skall det efter genomförd "manipulation" avspeglas i alla fönster. Du får själv välja om verktygsuppsättningen ska vara densamma i alla fönster eller om det ska gå att ha olika uppsättningar i olika fönster.

Hur man skapar nya verktygsinstanser och hur man ritar med (och ställer in) verktygen får du själv fundera ut, men försök att göra mekanismerna så direktmanipulativa som möjligt.

Om du känner dig osäker på hur du ska bygga ritprogrammet kan du använda dig av tipsen nedan. Men tänk på att du inte *måste* göra som det föreslås – tänk gärna ut bättre alternativ!

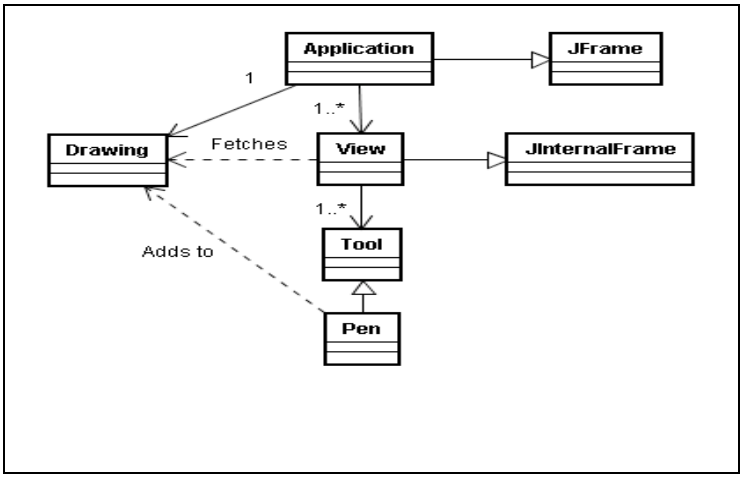
Tips

Här är en skiss på hur ritprogrammet skulle kunna se ut:



Vi har ett huvudfönster (”Application”) som visar två vyer av ritytan (”View 1” och ”View 2”). I den första vyn i skissen ritar användaren med en penna och strecket dyker upp i den andra vyn samtidigt. Det finns en verktygsuppsättning i varje vy, vilket gör att vyerna måste ha kontroller för att lägga till nya (och kanske ta bort?) verktyg, samt att ändra egenskaper för de verktyg som finns på ritytan. (Visas inte i skissen.) ”Application” måste också ha en kontroll för att lägga till nya vyer, förslagsvis i form av en meny.

Ett sätt att implementera skissen är att utgå från en klass för huvudfönstret, `Application` i UML-diagrammet nedan. Vi har en `View`-klass som visualiserar ritytan, med verktygen ovanpå. `Application` har en lista med `View`-instanser, som vardera är ett `JInternalFrame`-fönster.



`Drawing` är en representation av den bild som finns på ritytan. Det finns många sätt att representera en bild, men i det här fallet kanske det är enklast att lagra den som en uppsättning listor med Java2D-primitiver (`Line2D`, `Rectangle2D`, osv.).

Modellens gränssnitt mot `View`-klassen skulle då kunna vara något i stil med

```
public class Drawing {
  …
  void fetchDrawing(Graphics2D graphics) {
    …
  }
}
```

`fetchDrawing()` ritar alltså in alla primitiverna i den givna `Graphics2D`-instansen.

Ett annat alternativ är att använda ett bildobjekt, förslagsvis BufferedImage, som representation. Du kan då enkelt manipulera bilden genom att använda dess metod `getGraphics()` och använda Graphics2D-objektet som metoden returnerar. Varje vy kan sedan enkelt rita ut bilden på sin yta med hjälp av metoden `drawImage()` när vyn ska ritas om.

Ritverktyg representeras av basklassen `Tool`. Varje `View` har en lista med `Tool`-instanser, och varje `Tool`-instans har en `draw()`-metod som `View` kan anropa för att verktyget ska ritas ut. Om nu `View`-klassen också ”lyssnar” på t.ex. mus-events (genom att implementera `MouseListener`; visas inte i UML-diagrammet ovan) kan dessa events också skickas vidare till den `Tool`-instans som är aktiv för närvarande.

Varje enskild typ av verktyg implementeras i en subklass till `Tool` och varje verktyg uppdaterar `Drawing` när de används.

Hur man kan representera ett ritverktyg som GIF-bild

Här följer ett exempel som visar hur man kan få en given färg i en GIF-bild att bli genomskinlig så att det går att använda GIF-bilden som en ”ikon” för ett ritverktyg. Vi behöver en hjälpklass, `Transparency`, som har en metod `makeColorTransparent()`. Denna metod tar en bild som indata, gör den valda färgen genomskinlig, och returnerar resultatet.

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

public class Transparency {
  public static Image makeColorTransparent(Image im,
                                           final Color color) {
    ImageFilter filter = new RGBImageFilter() {
      public int markerRGB = color.getRGB() | 0xFF000000;
      public final int filterRGB(int x, int y, int rgb) {
        if ( ( rgb | 0xFF000000 ) == markerRGB ) {
          return 0x00FFFFFF & rgb;
        } else {
          return rgb;
        }
      }
    };
    ImageProducer ip = new FilteredImageSource(im.getSource(), filter);
    return Toolkit.getDefaultToolkit().createImage(ip);
  }
}

public class Pen public Tool implements ImageObserver {
  public Pen() {
    ImageIcon iic = new ImageIcon("c:/pen.gif");
    img = Transparency.makeColorTransparent(iic.getImage(),
                                            new Color(0).white);
  }

  public void draw(Graphics2D graphics) {
    graphics.drawImage(img,AffineTransform.getTranslateInstance(x, y),this);
  }

  public boolean imageUpdate(Image img, int infoflags, int x, int y,
    int width, int height) {
    return true;
  }
  …
  private int x;        // On-screen position
  private int y;        // On-screen position
  private Image img;    // Pen icon
}
```

Om du har tid och lust…

- Lägg till fler typer av verktyg!
- Gör så att man kan ladda och spara bilder man ritat!
- Använd Photoshop för att skapa extra snygga ikoner för dina verktyg!
- Om du har representerat bilden som en uppsättning Java2D-primitiver: Försök introducera lite enkel animation genom att skapa en `Thread` och låt denna modifiera primitivernas kontrollpunkter!