Skriftlig eksamen i
Datalogi og Administrativ Datalogi

Modul 1

Vinter 1997/98

Opgavesættet består af 5 opgaver, der ved bedømmelsen tillægges følgende vægte:

Opgave 1	28%
Opgave 2	12%
Opgave 3	25%
Opgave 4	15%
Opgave 5	20%

Alle sædvanlige hjælpemidler er tilladt. Der må ikke medbringes datamaskine. I tilfælde af unøjagtigheder i opgaveteksterne forventes det, at deltagerne selv præciserer besvarelsernes forudsætninger.

Opgavesættet består af en forside, 10 paginerede sider samt bilag på 7 paginerede sider. Kontroller at din kopi er fuldstændig.

Opgave 1: Binære søgetræer (28%)

Nedenstående Javaklasse `BinarySearchTree` kan benyttes til lagring og genfinding af poster ved hjælp af binære søgetræer. For nemheds skyld består en post kun af en nøgle, og nøglen er et tegn (`char`).

```
public class BinarySearchTree {
    public void insert(char key)
      { root = insert(key, root); }

    public boolean contains(char key)
      { return contains(key, root); }

    private Node root;

    private Node insert(char key, Node t) {
        if (t == null)
            t = new Node(key);
        else if (key < t.key)
            t.left = insert(key, t.left);
        else
            t.right = insert(key, t.right);
        return t;
    }

    private boolean contains(char key, Node t) {
     /* Koden ikke medtaget. Se spørgsmål 1.3. */
    }
}

class Node {
    Node(char key) { this.key = key; }

    char key;
    Node left, right;
}
```

Den offentlige metode `insert` benyttes til indsættelse af poster i søgetræet.

Nøglerne E M T I R O G L A (i nævnte rækkefølge) ønskes indsat i et tomt binært søgetræ.

Spørgsmål 1.1 Programmér en kodestump i Java, der opretter et tomt binært søgetræ og indsætter disse nøgler i træet.

Spørgsmål 1.2 Tegn det søgetræ, der derved fremkommer (gerne efter hver indsættelse).

Spørgsmål 1.3 Programmér kroppen i den private metode `contains`. Metoden skal returnere `true`, hvis søgetræet med rod `t` indeholder en post med nøglen `key`. Ellers skal den returnere `false`. Programmér såvel en rekursiv som en ikke-rekursiv udgave.

Nedenfor er vist implementationen af en metode `remove`, der kan benyttes til at slette en post med en given nøgle.

```
public void remove(char key)
  { root = remove(key, root); }

private Node remove(char key, Node t) {
    if (t == null)
        return null;
    if (key < t.key)
        t.left = remove(key, t.left);
    else if (key > t.key)
        t.right = remove(key, t.right);
    else
        t = join(t.left, t.right);
    return t;
}

private Node join(Node t1, Node t2) {
    if (t2 == null)
        return t1;
    if (t2.left == null) {
        t2.left = t1;
        return t2;
    }
    Node p = t2,
         t = t2.left;
    while (t.left != null) {
        p = t;
        t = p.left;
    }
    p.left = t.right;
    t.left = t1;
    t.right = t2;
    return t;
}
```

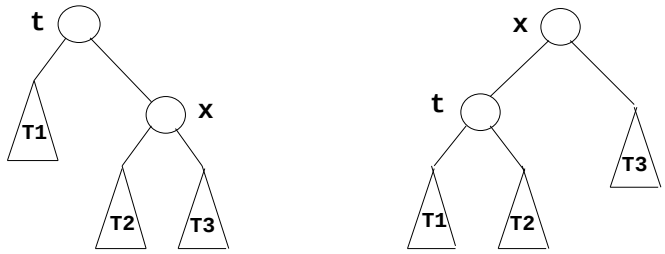
Spørgsmål 1.4 Forklar, hvad metoden `join` udretter, og hvad returværdien betyder.

Spørgsmål 1.5 Programmér en metode `printSorted`, der kan benyttes til at udskrive et givet søgetræs nøgler i stigende orden.

Nedenstående metode foretager en såkaldt *venstrerotation* af et ikke-tomt træ, der har `t` som rod.

```
Node rotateLeft(Node t) {
    Node x = t.right;
    t.right = x.left;
    x.left = t;
    return x;
}
```

Resultatet af et kald kan anskueliggøres ved hjælp af følgende figur.



Spørgsmål 1.6 Programmér en tilsvarende metode til højrerotation, `rotateRight`. Anskueliggør resultatet af et kald ved hjælp af en figur som den, der er vist ovenfor.

Nedenfor ses en alternativ metode, `insert`, til indsættelse i et binært søgetræ. Metoden anvender sig af `rotateLeft` og `rotateRight`.

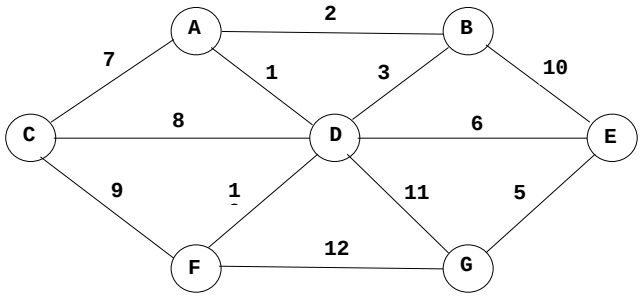
```
private Node insert(char key, Node t) {
    if (t == null)
        return new Node(key);
    if (key < t.key) {
        t.left = insert(key, t.left);
        return rotateRight(t);
    }
    t.right = insert(key, t.right);
    return rotateLeft(t);
}
```

Ved hjælp af denne metode indsættes nøglerne `E M T I R O G L A` (i nævnte rækkefølge) i et tomt binært søgetræ.

Spørgsmål 1.7 Tegn det søgetræ, der derved fremkommer (gerne efter hver indsættelse).

Opgave 2: Minimale udspændende træer (12%)

Betragt nedenstående ikke-orienterede, vægtede graf



hvor bogstaverne betegner knuder, og tallene betegner vægte på kanterne.

Spørgsmål 2.1 Angiv en nabomatrixrepræsentation (eng. *adjacency matrix representation*) og en nabolisterepræsentation (eng. *adjacency list representation*) for grafen. Grafens vægte bør indgå i begge repræsentationer.

Spørgsmål 2.2 Tegn et minimalt udspændende træ for grafen.

Spørgsmål 2.3: Angiv kanterne i træet i den rækkefølge, de bestemmes, når Dijkstras metode (priority-first search) anvendes.

Spørgsmål 2.4: Angiv kanterne i træet i den rækkefølge, de bestemmes, når Kruskals metode anvendes.

Opgave 3: OOA-modellering (25%)

Forestil dig at du indgår i den ene af to projektgrupper, der skal udvikle et system til en radiostation. Radiostationen er organiseret i en række redaktioner, der hver har ansvaret for et program. Hver redaktion laver en eller flere udsendelser om ugen, og vi antager at der intet skal gemmes om en udsendelse, når den er sendt. Systemet skal støtte de enkelte redaktioner og således bruges til planlægning (hvilke elementer skal indgå i hver udsendelse, og hvor lang tid afsættes til hver), produktion (optagelse og redigering), afvikling (programmet udsendes). Din projektgruppe har nu ansvaret for at modellere planlægningen og produktion, mens afvikling varetages af den anden projektgruppe. Systemet skal ikke bruges til administration af de ansatte.

En redaktion består af en producer, en vært, en eller flere journalister (nogle er freelancere) og en teknikker. Planlægning foregår i redaktionslokalet, mens produktion og afvikling foregår i studierne.

En udsendelse består af en række elementer, som hver består af et af nedenstående typer indslag omgivet af værtens speak:
- musik,
- journalistiske indslag (interview eller en tekst som journalisten læser op),
- værtens del (interview med en gæst i studiet).

Musikken skal kunne overføres elektronisk fra en musikdatabase inden udsendelsen starter. Vi antager her at længden af alle elementer er fastlagt inden programmet afvikles, men under planlægning og produktion kan nye elementer tilføjes eller et element kan holdes i reserve eller tages ud af programmet.

Producerens opgave består i at planlægge og beslutte hvilke journalistiske indslag en udsendelse skal indeholde (hvert indslag gives en titel) og hvor lang tid der afsættes til hvert indslag. Produceren får desuden af værten titel og længde på de elementer, som denne har ansvar for (se nedenfor). Produceren bruger i dag et A4 ark til at holde styr på hvilke elementer der planlægges at skulle indgå i udsendelsen, og hvor lang tid der afsættes til hvert element. Dette A4 ark kaldes producerens manus. Produceren fordeler indslagene blandt journalisterne og de planlagte elementer sættes på manus. Produceren kan, helt frem til udsendelsen starter, beslutte at tage et hvilket som helst element midlertidigt ud af manus (det holdes i reserve) eller helt ud af manus. Inden udsendelsen starter har produceren besluttet hvilke elementer der skal med i udsendelsen.

Værtens opgave er at lave to speaks (kaldet henholdsvis oplæg og nedlæg) til hver type indslag, samt at gennemføre interview med en gæst i studiet, mens udsendelsen kører. Endelig vælger værten også den musik der skal spilles, og planlægger tid for det enkelte musikstykke og den samlede tid der skal spilles musik i udsendelsen. Dette kan dog omgøres af produceren (se ovenfor).

Hver journalist producerer et indslag, som enten kan være et interview eller en tekst som journalisten læser op. Disse indslag er optaget, redigeret og ligger klar når udsendelsen starter.

I samarbejde med radiostationen og den anden projektgruppe er I kommet frem til følgende foreløbige systemdefinition udtrykt i BATOFF:

Betingelser: Edb-systemet skal benyttes af redaktioner, der har fast producer og vært, men skiftende journalister, herunder også freelancere, der ikke kan forventes at være fortrolige med systemet.

Anvendelsesområde: Redaktioner, der har ansvar for en eller flere udsendelser om ugen bestående af musik, journalistiske indslag, værtens speak og interview.

Teknologi: Et client/server system, der skal kunne indeholde såvel tekst som lydfiler. Der skal være optage- og redigeringsudstyr såvel i redaktionslokaler som i studier, og de skal være forbundet i et netværk som radiostationens musikdatabase også er forbundet til.

Objektsystem: Manus og indslag.

Funktionalitet: Støtte til planlægning, produktion og afvikling i forbindelse med udsendelser.

Filosofi: Et værktøj til de arbejdsopgaver, der er involveret i at lave udsendelser. På sigt skal systemet også kunne udveksle data med stationens administrative systemer.

> **Spørgsmål 3.1** Lav et klassediagram for objektsystemet.
>
> **Spørgsmål 3.2** Lav et tilstandsdiagram for et journalistisk indslag.
>
> **Spørgsmål 3.3** Skitser og begrund det eller de skærmbilleder, som produceren skal bruge til at interagere med systemet.

NB. Hvis du foretager afgrænsninger eller gør dig yderligere forudsætninger end de beskrevne, skal du beskrive disse i opgavebesvarelsen.

Opgave 4: Parameteroverførsel og tilgangsspecifikation (15%)

Man har ofte brug for at ombytte (eng. *to swap*) værdierne af to variable eller to objekters indhold.

Spørgsmål 4.1 Forklar med dine egne ord, hvilke problemer der kan være med at lave en generel swap-metode i Java, givet sprogets mekanismer til parameteroverførsel.

Spørgsmål 4.2 Programmér en generel metode

```
void swap(Swappable o1, Swappable o2),
```

der ombytter indholdet af to objekter; metoden skal senere indlemmes i klassen `modul1.Utilities`. Interface `Swappable` er defineret på næste side. Forklar hvad der udskrives, når `swap` kaldes fra `main`-metoden i klassen `Clock`, også defineret på næste side.

Spørgsmål 4.3 `swap` ombytter objekters indhold/værdi; til tider har vi imidlertid brug for at ombytte værdierne af simple variable. Tag f.eks. flg. Java-sekvens:

```
...
int from10to5 = 10;
int from5to10 = 5;
System.out.println("before: from10to5 is "+
                   from10to5 +"; from5to10 is "+ from5to10);

// some code that will swap the contents of
// variable from10to5 and variable from5to10

System.out.println("after: from10to5 is "+
                   from10to5 +"; from5to10 is "+ from5to10);
...
```

Angiv mindst to forskellige løsninger på dette swap-problem; den ene skal benytte `swap`-metoden fra spørgsmål 4.2.

I Java kan man specificere mere eller mindre restriktiv tilgang til klasser og klassemedlemmer. Som hovedregel vil vi gerne specificere så restriktiv adgang som muligt; derved mindskes antallet af navne, man på et givet tidspunkt skal kende, og derved bevares kontrollen over de enkelte elementer.

Spørgsmål 4.4 Brug `Clock`-klassen, der er vist på næste side, og tilføj eller ændr tilgangsspecifikationerne, således, at adgangen til de enkelte klasser og klassemedlemmer bliver mest mulig restriktiv uden at programafviklingen hindres.

Spørgsmål 4.5 Forklar for hver enkelt specifikation, hvorfor den bør være som foreslået. Herunder: forklar også betydningen af **ikke** at angive adgangsspecifikationen på en klasse hhv. et klassemedlem.

interface Swappable

```
package modul1;

public interface Swappable
{
    public Object getValue();

    public void setValue(Object val);
}
```

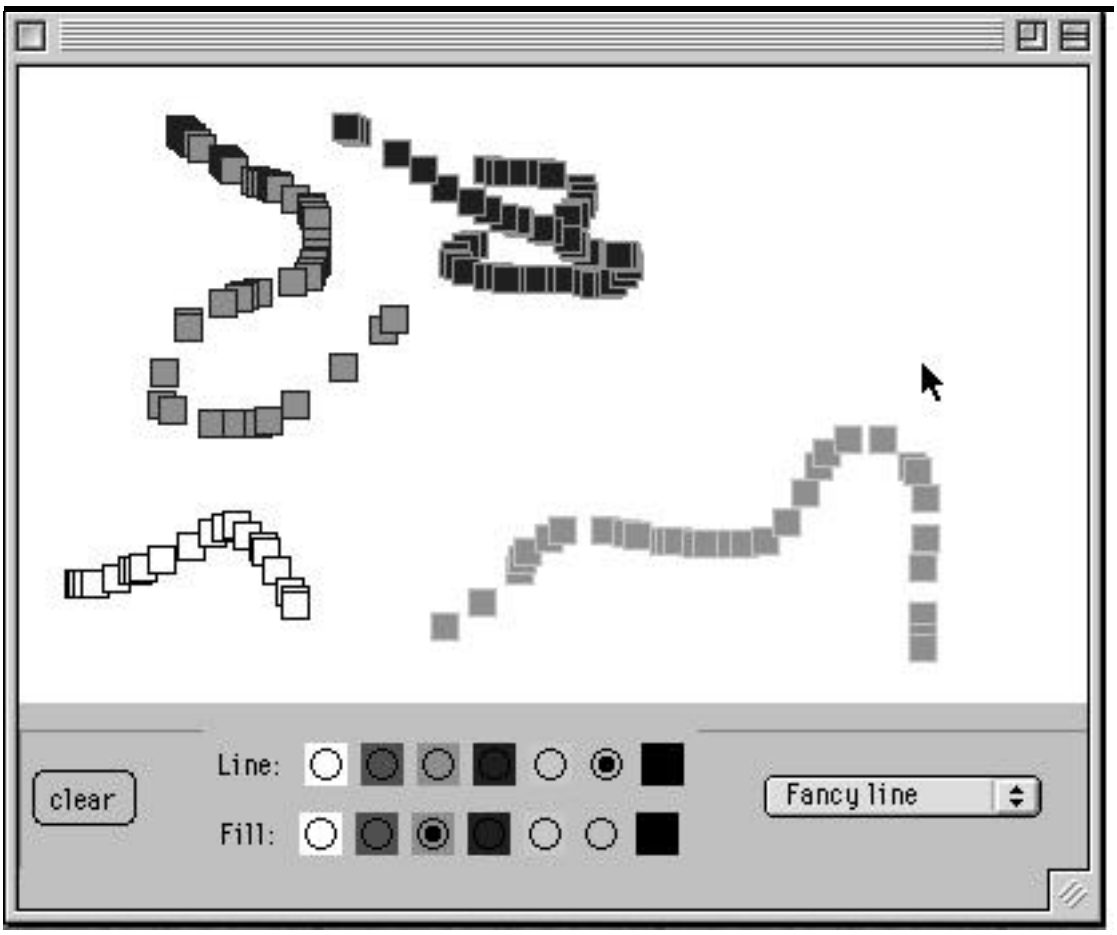
class Clock

```
import modul1.Utilities;
import modul1.Swappable;

class Clock implements Swappable {
    static int counter = 1;
    int serial;
    Integer value;

    Clock() {

        serial = counter++;
        value = new Integer(serial);
    }

    public static void main(String argv[]) {

        Clock c1 = new Clock();
        Clock c2 = new Clock();
        System.out.println("before: c1 is "+c1+"; c2 is "+c2);
        Utilities.swap(c1, c2);
        System.out.println("after: c1 is "+c1+"; c2 is "+c2);
    }

    public Object getValue() { return value; }

    public void setValue(Object val) {value = (Integer) val;}

    public String toString() { return "Clock no. "+serial; }
}
```

Opgave 5: (Objektorienteret programmering) 20%

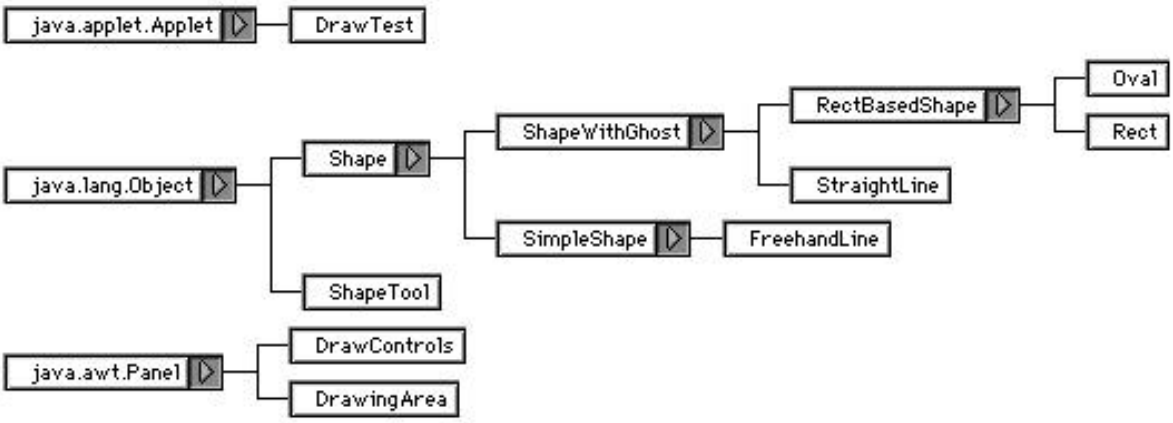
Programmet DrawTest, version F.1 (se bilag 3) skal udbygges; brugerne var meget begejstrede for den seneste facilitet med rektangler og ellipser med separat farveangivelse for kant og fyld.

Nu ønsker brugerne en meget fancy udvidelse af frihåndstegningen ("Freehand line"), hvor man i hver "sampling" tegner en figur - i stedet for (som nu) at tegne et simpelt linjesegment. En "Fancy line", hvor tegnefiguren er en udfyldt firkant, vil f.eks. kunne se ud som vist på figur 1.



Figur 1: Skærmbillede, der viser den ønskede funktionalitet

Som hjælp til din læsning af DrawTest-programmet vises i figur 2 det statiske klassehierarki. Supplér evt. med en skitse af de dynamiske relationer mellem objekterne i DrawTest, herunder hvordan klasserne bruger hinanden (attributter, aggregater, osv.)



Figur 2: Klassehierarkiet i DrawTest F.1

Spørgsmål 5.1 Beskriv ved hjælp af klasse-hierarkiet i figur 2, hvilke ændringer der skal foretages i DrawTest. Angiv hvilke klasser, der skal ændres i DrawTest for at få denne udvidelse til at fungere. For hver klasse beskrives arten af ændring.

Spørgsmål 5.2 Programmér en subklasse inden for Shape-klassehierarkiet, der bl.a. foretager kald af `PlaceholderDrawItem` (eller en subklasse heraf) i hvert samplingspunkt. Du kan gå ud fra `PlaceholderDrawItem`-klassen, der er vist i bilag 2, dvs. en implementation der tegner et udfyldt kvadrat på 10x10 pixels.

Spørgsmål 5.3 Forklar hvad der sker, når brugeren af DrawTest befinder sig i "FreehandLine mode" og trækker musen hen over tegnefladen. Specielt skal du forklare hvad der sker, **fra** vi når til kaldet (DrawTest, bilag 1, linje 67)

```
repaint(rect.x, rect.y, rect.width+1, rect.height+1);
```

i DrawingArea's `handleEvent`-metode (`case` MOUSE_DRAG), og **til** alle aktioner, der igangsættes af dette kald, er færdiggjort.

Spørgsmål 5.4 Ville resultatet have været anderledes, hvis Shape's paint-metode **ikke** var abstract? Antag, at Shape's paint-metode var erklæret som følger:

```
public void paint(Graphics g) {
  System.out.println("This is the paint method of Shape");
}
```

Hvis du mener, resultatet havde været anderledes, bedes du forklare, hvad der så ville ske. Hvis du mener, resultatet ville være uændret, bedes du forklare hvorfor.

Bilag 1: programmet DrawTest

Oversigt over public classes: *Linjenummer*


```
1   // DrawTest is a simple drawing program, allowing the user to draw
2   // with different colors and in different modes.
3
4   public class DrawTest extends Applet {
5
6       public void init() {
7
8           this.setLayout(new BorderLayout(10,10));
9           DrawingArea drawPanel = new DrawingArea();
10          this.add("Center", drawPanel);
11          DrawControls controlPanel = new DrawControls(drawPanel);
12          this.add("South", controlPanel);
13      }
14  }
15
16
17  // DrawControls is the panel that holds
18  // (1) a double set of color checkboxes, one for the line
19  //     and the other for the fill,
20  // (2) the choicelist determining the draw mode, and
21  // (3) a clear button */
22
23  public class DrawControls extends Panel {
24
25      public DrawControls(Component target) { ... }
26
27      public void paint(Graphics g)  { ... }
28
29      public boolean action(Event event, Object arg) {
30      }
31  }
32
33
```

```
34  // DrawingArea is used as the drawing area. It is a panel (not a
35  // Canvas, as would have been a more straight forward  choice).
36
37  public class DrawingArea extends Panel {
38
39      private Vector objBase; // draw objects sampled so far
40      private Shape drawObj;
41      private Color backgroundColor = Color.white;
42      private boolean inTesting = false;
43
44      public DrawingArea()  {
45          objBase = new Vector();
46          setBackground(backgroundColor);
47      }
48
49      // handleEvent is invoked by the system on panel events to
50      // allow us to handle mouse events.
51      // The parameter contains an event as received
52      // from the window system
53
54      public boolean handleEvent(Event anEvent) {
55
56          Rectangle rect;
57
58          switch (anEvent.id) {
59             case Event.MOUSE_DOWN:
60                drawObj = ShapeTool.makeDrawObject(anEvent.x, anEvent.y);
61                return true;
62
63             case Event.MOUSE_DRAG:
64                if (drawObj!=null) {
65
66                   rect = drawObj.sample(anEvent.x, anEvent.y);
67                   repaint(rect.x, rect.y, rect.width+1, rect.height+1);
68                   return true;
69                }
70                else return false;
71
72             case Event.MOUSE_UP:
73                if (drawObj!=null) {
74
75                   rect = drawObj.finishUp(anEvent.x, anEvent.y);
76                   objBase.addElement(drawObj);
77                   drawObj = null; // necessary, otherwise this object
78                               // may be repainted wrongly
79                   repaint(rect.x, rect.y, rect.width+1, rect.height+1);
80                   return true;
81                }
82                else return false;
83
84             case Event.WINDOW_DESTROY:
85                System.exit(0);
86                return true;
87
88             default:
89                return false;
90          }
91      }
92
```

```
93        public void clearAll() {
94
95             objBase.removeAllElements();
96             repaint();
97        }
98
99        public final void update (Graphics g) {
100
101            paint(g);
102       }
103
104       public void paint(Graphics g) {
105
106            // paint the drawing area in the right color
107            Dimension d = size();
108            g.setColor(backgroundColor);
109            g.fillRect(0, 0, d.width, d.height);
110
111            // draw all previously created objects
112            for (int i=0; i < objBase.size(); i++) {
113               Shape shape = (Shape) objBase.elementAt(i);
114               shape.paint(g);
115            }
116
117            // draw the current, unfinished object, if any
118            if (drawObj != null)
119               drawObj.paint(g);
120       }
121   }
122
123
124   // LineSegment is a utility class used by
125   // StraightLine and FreehandLine.
126
127   public class LineSegment  {
128
129       LineSegment(int x1, int y1, int x2, int y2) { ... }
130
131       public Point getStart() { ... }
132
133       public Point getEnd() { ... }
134
135       public void paint(Graphics g) { ... }
136   }
137
138
139   // ShapeTool is an AbstractFactory module
140
141   public class ShapeTool {
142       public static final int STRAIGHTLINE = 0;
143       public static final int FREELINE = 1;
144       public static final int RECTANGLE = 2;
145       public static final int OVAL = 3;
146
147       private static int mode = STRAIGHTLINE;
148       private static Color lineColor = Color.black;
149       private static Color fillColor = null;
150
```

```
151         // setDrawMode lets us shift between drawing modes.
152         // The parameter contains a string describing a new
153         // drawing mode, as requested by the user
154         public static void setDrawMode(String shapeAsText) {
155 
156             if (shapeAsText.equals("Straight line"))
157                mode = STRAIGHTLINE;
158             else
159             if (shapeAsText.equals("Freehand line"))
160                mode = FREELINE;
161             else
162             if (shapeAsText.equals("Rectangle"))
163                mode = RECTANGLE;
164             else
165             if (shapeAsText.equals("Oval"))
166                mode = OVAL;
167             else
168                throw new IllegalArgumentException();
169         }
170 
171         // setLineColor lets us shift between line colors.
172         // The parameter contains the new line color
173         public static void setLineColor(Color selectedColor) {
174 
175             lineColor = selectedColor;
176         }
177 
178         // setFillColor lets us shift between fill colors.
179         // The parameter contains the new fill color
180         public static void setFillColor(Color selectedColor) {
181 
182             fillColor = selectedColor;
183         }
184 
185         // makeDrawObject will create an object of the proper type
186         // determined by the current draw mode. The parameters are
187         // the coordinates of the mouse down
188         public static Shape makeDrawObject(int x, int y) {
189 
190             switch (mode) {
191 
192                case STRAIGHTLINE:
193                   return new StraightLine(x, y, lineColor);
194 
195                case FREELINE:
196                   return new FreehandLine(x, y, lineColor);
197 
198                case RECTANGLE:
199                   return new Rect(x, y, lineColor, fillColor);
200 
201                case OVAL:
202                   return new Oval(x, y, lineColor, fillColor);
203 
204                default:
205                   return null;
206             }
207         }
208     }
209 
210 
```

```
211  // Shape is an abstract super class of all the various shapes
212  // we might think of supporting in DrawTest.
213  // Currently supported drawing modes are divided into two main categories:
214  // (A) simple shapes (e.g., freehand lines) which will be drawn immediately
215  //  at each sampling point (that is, each mouse drag) drawing with simple
216  //  line segments between sampling points.
217  // (B) "ghostly" shapes (e.g., straight line, rectangle and oval) which are
218  //  also drawn at each sampling point but immediately overwritten until
219  //  the sampling is finished (that is, at mouse up). See description of
220  //  ShapeWithGhost for more info.
221  //
222  public abstract class Shape {
223
224      protected Color lineColor, fillColor;
225      protected Rectangle bBox;
226
227      // Constructor for class Shape. The parameters are the current pen
228      // color and the current fill color.
229      public Shape(Color line, Color fill) {
230
231          lineColor = line;
232          fillColor = fill;
233      }
234
235      // sample is invoked at each mouse drag. May be overridden by subclass.
236      // The parameters are the coordinates of the mouse drag
237      public Rectangle sample(int x, int y) { return bBox;}
238
239      // finishUp is invoked at mouse up. May be overridden by subclass.
240      // The parameters are the coordinates of the mouse up
241      public Rectangle finishUp(int x, int y) { return bBox;}
242
243      abstract public void paint(Graphics g);
244  }
245
246
247  // SimpleShape is an abstract super class of the various shape types
248  // that will be drawn immediately at each sampling point (that is, each
249  // mouse drag)
250
251  public abstract class SimpleShape extends Shape {
252
253      protected int noOfSegments;
254      protected Vector strokePath;
255      protected int prevX, prevY;
256
257      // Constructor for class Shape. The parameters are the coordinates
258      // of the mouse down, the current pen color and the current fill color.
259      public SimpleShape(int x, int y, Color line, Color fill) {
260
261          super(line, fill);
262          strokePath = new Vector();
263          prevX = x;
264          prevY = y;
265          bBox = new Rectangle (x, y, 0, 0);
266      }
267  }
268
269
```

```
// FreehandLine is a simple shape that is drawn by connecting all the
// sampling points between mouse down and mouse up

class FreehandLine extends SimpleShape {

    // FreehandLine constructor. The parameters are the coordinates
    // of the mouse down and the pen color
    FreehandLine(int x, int y, Color c) {

        super(x, y, c, null);
    }

    // sample is invoked at each mouse drag. We create a new
    // line segment and stores it in the stroke path
    // The parameters are the coordinates of the mouse down,
    // It returns the boundingbox of the current addition to
    // the drawing
    public Rectangle sample(int x, int y) {

        LineSegment seg = new LineSegment(prevX, prevY, x, y);
        strokePath.addElement(seg);
        bBox.add(x, y);
        prevX = x;
        prevY = y;
        return bBox;
    }

    public void paint(Graphics g) {

        g.setColor(lineColor);
        for (int i=0; i < strokePath.size(); i++) {
           LineSegment seg = (LineSegment) strokePath.elementAt(i);
           seg.paint(g);
        }
    }
}


// ShapeWithGhost is an abstract super class of all those shapes
// that needs to be drawn with a "ghost".
// Currently supported drawing modes are: straight line, rectangle and oval.
// Straight line mode drawing will create straight lines between the location
// of mouse down and the location of mouse up, ignoring the movements of the
// cursor in between.
// Rectangle mode drawing will create a rectangle with the location of mouse
// down and the location of mouse up determining the diagonal
// Oval mode drawing will create an oval within the rectangle described
// diagonally by the location of mouse down and the location of mouse up

public abstract class ShapeWithGhost extends Shape {

    public ShapeWithGhost(int x, int y, Color line, Color fill) { ... }

    abstract public Rectangle sample(int x, int y);

    public Rectangle finishUp(int x, int y) { ... }

    public void paint(Graphics g) { ... }

    abstract protected void paintGhost(Graphics g) ;

    abstract protected void paintReal(Graphics g) ;
}
```

Bilag 2: class PlaceholderDrawItem

```
class PlaceholderDrawItem  {

    private int nPoints = 5;
    private int xPoints[] = {0, 10, 10, 0, 0};
    private int yPoints[] = {0, 0, 10, 10, 0};
    private int lastX, lastY;
    private Color lineColor;
    private Color fillColor;

    PlaceholderDrawItem(int x, int y, Color line, Color fill) {

        lineColor = line;
        fillColor = fill;
        moveTo(x, y);
    }

    int getWidth() { return 10; }
    int getHeight() { return 10; }

    void moveTo(int x, int y) {

        int xDelta = x-lastX;
        int yDelta = y-lastY;
        for (int i = 0; i<nPoints; i++) {

           xPoints[i] += xDelta;
           yPoints[i] += yDelta;
        }
        lastX = x; lastY = y;
    }

    public void paint(Graphics g) {

        g.setColor(fillColor);
        g.fillPolygon(xPoints, yPoints, nPoints);

        g.setColor(lineColor);
        g.drawPolygon(xPoints, yPoints, nPoints);
    }
}
```