Tentamen Programmeringsteknik I 2011-12-13

Skrivtid: 1400-1700
Hjälpmedel: Java-bok

Tänk på följande

- Det finns en referensbok (Java) hos tentavakten som du får gå fram och läsa men inte ta tillbaka till bänken.
- Skriv läsligt! Använd *inte* rödpenna!
- Skriv bara på framsidan av varje papper.
- Börja alltid ny uppgift på nytt papper.
- Lägg uppgifterna i ordning. Skriv uppgiftsnummer och pin-kod (eller namn om du saknar sådan) på alla papper. Skriv inte längst upp i vänstra hörnet - det går inte att läsa där efter sammanhäftning.
- Fyll i försättssidan fullständigt.
- Det är principer och idéer som är viktiga. Skriv så att du övertygar examinatorn om att du har förstått dessa även om detaljer kan vara felaktiga.
- Programkod skall vara läslig dvs den skall vara vettigt strukturerad och indenterad. Namn på variabler, metoder, klasser etc skall vara beskrivande men kan ändå hållas ganska korta.
- Betygsgränser: 15 ger säkert 3, 18 ger säkert 4, 21 ger säkert 5.

Lycka till!

Tom

Uppgifter

1. Klassen `Measurements` representerar en följd av mätvärden

```
public class Measurements {
  private double[] m;   // Mätvärden
  private int n;        // Aktuellt antal värden

  public Measurements() {
    this(10);
  }

  public Measurements(int size) {
      ...
  }

  public String toString() {
     ...
  }

  public void add(double v) {
      m[n] = v;
      n++;
  }

  public double mean() {
    ...
  }

  public double max() {
     ...
  }

  public Measurements smooth() {
    ...
  }

  // Testprogram
  public static void main(String[] args) {
    Measurements ms = new Measurements();
    for (int i=0; i<10; i++) {
      ms.add(Math.random());
    }
    System.out.println("Original     : " + ms);
    System.out.println("Mean value   : " + ms.mean());
    System.out.println("Maximal value: " + String.format("%4.2f", ms.max()));
    System.out.println("Smooth       : " + ms.smooth());

    ms.add(42.);  // Ger fel
  }
}
```

   Resultat vid körning:

```
> run Measurements
Original     : [ 0,11  0,76  0,45  0,13  0,64  0,90  0,60  0,68  0,89  0,73 ]
Mean value   : 0.589112126781489
Maximal value: 0,90
Smooth       : [ 0,11  0,44  0,45  0,41  0,56  0,72  0,73  0,72  0,77  0,73 ]
java.lang.ArrayIndexOutOfBoundsException: 10
at Measurements.add(Measurements.java:23)
at Measurements.main(Measurements.java:86)
```

   a) Skriv klar konstruktorn `public Measurements(int n)` som skapar ett objekt med plats för n mätvärden.

   b) Skriv klar metoden `public double mean()` som beräknar och returnerar medelvärdet av de lagrade värden.

c) Skriv klar metoden `public double max()` som beräknar och returnerar det största av de lagrade värdena.

d) Skriv klar `toString`-metoden. Se `main`-metoden och körexemplet för specifikation av användning och resultat. Ledning: Studera `main`-metoden för att se hur man kan åstadkomma exakt två decimaler när man konverterar en double till en sträng.

e) Skriv klar metoden `public Measurements smooth()`. Metoden skall skapa och returnera ett nytt `Measurements`-objekt med lika många värden som i originalobjektet. Värdena på första och sista plats skall vara lika. För övriga värden gäller att värdet på plats i skall vara *medelvärdet* av värdena på plats $i-1$, i, $i+1$ i originalobjektet.

f) Med den givna koden kommer programmet avbrytas om man försöker lagra fler värden än det antal man givit i konstruktorn (se körexemplet). Modifiera programmet så att det *går* att lagra fler värden än man angivit i konstruktorn. Vad, var och hur skall man ändra? Anmärkning: uppgiften går att lösa på mer än ett sätt.

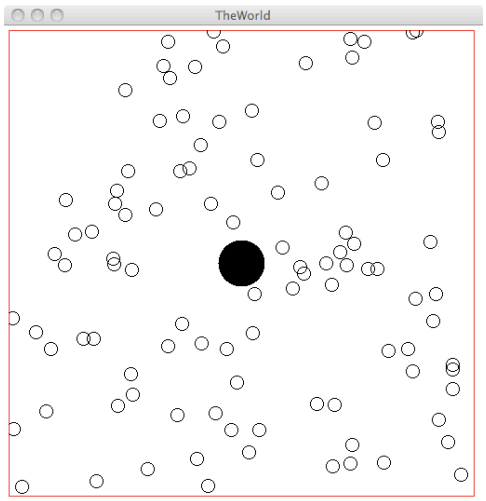
2. I en simulering av ett trafiksystem behöver man en klass `TrafficLight` för att representera trafikljus. Andra klasser i systemet kommer att vara *filer*, *fordon*, *rondeller* mm.

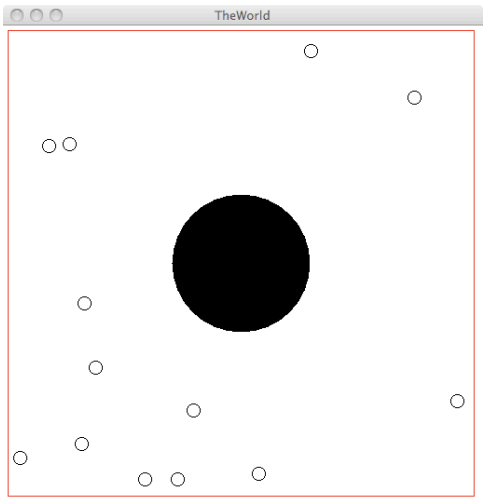
   Simuleringen skall göras med en tidsstegning.

   Designa och implementera klassen `TrafficLight`!

   Anm: Det räcker med att skriva koden men om du känner dig osäker på detaljer kan du presentera designen med hjälp av klassdiagram.

3. Nedan finns ett program som simulerar partiklar som rör sig i en kvadratisk 2-dimensionell låda. Lådan har sina hörn i punkten (0,0) och (1,1). Partiklarna studsar mot väggarna men, för enkelhets skull, inte mot varandra. Mitt i lådan finns ett svart hål som alltså har centrum i punkten (0.5, 0.5). När en partikel nuddar det svarta hålet förintas partikeln och hålet area växer med partikelns area. Allteftersom tiden går minskar således antal partiklar och det svarta hålet växer. Bilden nedan till vänster visar hur det ser ut när programmet startar och bilden till höger hur det ser ut efter 400 tidssteg.





Programmet består av 4 klasser: `Vector`, `Particle`, `Box` och `Gui`. Klassen `Vector` används för att representera partiklarnas position och hastighet. Klassen `Gui` sätter upp systemet, driver simuleringen och ritar ögonblicksbilder. Av utrymmesskäl är vissa delar av koden (t ex alla import-satser) utelämnade

a) Skriv färdigt metoden `copy` i klassen `Vector`! Varför vill man ha en sådan metod?

b) I klassen `Particle` finns en metod `move` som flyttar partikeln ett steg med hastighetsvektorn och hanterar studs mot väggarna. Om partikeln nuddar det svarta hålet skall den dödförklaras och överlåta sin yta till det svarta hålet. Skriv klart metoden!

c) Klassen `Box` har en metod `sweep` som tar bort alla dödförklarade objekt ur partikelsamlingen. Skriv klart metoden! Varför kan metoden `move` i ovanstående deluppgift ombesörja detta?

Anm: I koden finns konstruktioner och begrepp som inte ingått i kursen (bl a grafikhanteringen) men man behöver inte kunna eller förstå dessa för att lösa uppgifterna.

```
public class Vector {
  private double x;
  private double y;

  public Vector(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public String toString() {
    return "<" + x + ", " + y + ">";
  }

  /** Modify this vector by adding v */
  public void add(Vector v) {
    x += v.x;
    y += v.y;
  }

  /** Returns the distance from this vector (point) to vector (point) v */
  public double distance(Vector v) {
    return Math.sqrt((x-v.x)*(x-v.x) + (y-v.y)*(y-v.y));
  }

  public double getx() {
    return x;
  }

  public double gety() {
    return y;
  }

  public void setx(double x) {
   this.x = x;
  }

  public void sety(double y) {
    this.y = y;
  }

  /** Creates and returns a copy of this vector */
  public Vector copy() {
     ...
  }

  /** Creates a vector of specified length pointing in a random direction */
  public static Vector randomVector(double len) {
    double angle = Math.random()*2.*Math.PI;
    return new Vector(Math.cos(angle)*len, Math.sin(angle)*len);
  }
}
```

```
public class Particle {
  private Vector c;     // Center
  private Vector v;     // Velocity
  private double r;     // Radius
  private boolean dead;

  public Particle(Vector c, Vector v, double r) {
    this.c = c.copy();
    this.v = v.copy();
    this.r = r;
    dead = false;
  }

  public String toString() {
    return "P(" + c + ", " + v + ", " + r + ")";
  }

  public double getRadius() {
    return r;
  }

  public boolean isDead() {
    return dead;
  }

  public void move() {
    c.add(v);  // Add the velocity vector to the position

    // Handle collision with the black hole
      ... Uppgift!

    // Handle boundary bouncing
    if (c.getx() < r && v.getx() < 0) { // left boundary
      v.setx(-v.getx());
    }

    if (c.getx() > 1.-r && v.getx() > 0) { // right boundary
      v.setx(-v.getx());
    }

    if (c.gety() < r && v.gety() < 0) { // upper boundary
      v.sety(-v.gety());
    }

    if (c.gety() > 1.-r && v.gety()>0) { // lower boundary
      v.sety(-v.gety());
    }
  }

  public void paintComponent(Graphics g) {
    int ix = (int)(c.getx()*Box.getsize());
    int iy = (int)(c.gety()*Box.getsize());
    int ir = (int)(r*Box.getsize());
    g.drawOval(ix- ir, iy-ir, 2*ir, 2*ir);
  }
}
```

```
public class Box extends JPanel {

  public static double deathRadius; // Black hole radius
  public static int size; // Drawing size of the box

  ArrayList<Particle> a; // The collection of particles

  public Box(int sz, double dr) {
    setPreferredSize(new Dimension(sz+5, sz+5));
    deathRadius = dr;
    size = sz;
    a = new ArrayList<Particle>();
  }

  public static double getDeathRadius() {
    return deathRadius;
  }

  public static int getsize() {
    return size;
  }

  /** Feeds the black hole with one particle */
  public static void feed(Particle p) {
    double a = (deathRadius*deathRadius +
                p.getRadius()*p.getRadius())*Math.PI;
    deathRadius = Math.sqrt(a/Math.PI);
  }

  public void add(Particle p) {
    a.add(p);
  }

  /** Advance all particles one time step */
  public void step() {
    for (Particle p : a)
      p.move();
    sweep(); // Remove all dead particles
  }

  /** Removes all dead particles from the arraylist */
  private void sweep() {
     ... Uppgift!
  }

  /** Fills the box with random particles */
  public void fill(int n, double r) {
    for (int i=0; i<n; i++) {
      Vector p = new Vector(Math.random(), Math.random());
      Vector v = Vector.randomVector(0.05*Math.random());
      add(new Particle(p, v, r));
    }
  }

  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.setColor(Color.RED);
    g.drawRect(0,0,size,size);
    g.setColor(Color.BLACK);
    int dr = (int)(deathRadius*size);
    g.fillOval(size/2-dr, size/2-dr, 2*dr, 2*dr); // Black hole
    for(Particle p : a) {
      p.paintComponent(g); // Individual particles
    }
  }
}
```

```
public class Gui extends JFrame {

  public static void main(String[] args) {
    JFrame frame = new JFrame("TheWorld");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(600, 600);
    frame.setBackground(Color.WHITE);
    frame.setLayout(new FlowLayout());
    Box b = new Box(500,0.05);
    b.fill(100, 0.015);
    frame.add(b);
    frame.pack();
    frame.setVisible(true);
    JOptionPane.showMessageDialog(null, "Start simulation?");

    for (int i=0; i<500; i++) {
      try {
        b.step();
        b.repaint();
        Thread.sleep(50);

      } catch(InterruptedException e) {}
    }
    JOptionPane.showMessageDialog(null, "Simulation ended");
    frame.dispose();
  }
}
```