Välikoe / 20.3

Vastaa neljään (4) tehtävään. Jos vastaat 5:een, 4 huonointa arvostellaan. Kunkin tehtävän vastaus eri konseptille.

1. Pöytätesti

Pöytätestaa seuraava ohjelma. (6p)

```
Tutki ohjelman toimintaa pöytätestillä käyttäen ohessa olevaa
pöytätestilomaketta.
- Merkitse harmaaksi ne alueet, jolloin muuttujaa ei ole olemassa.
- Ruutuun merkintä vain jos muuttujan arvo muuttuu tai olio muuttuu
  roskaksi.
- Merkitse iso R-kirjain kun olio muuttuu roskaksi.
- Merkitse rivillä "* = viite" * jokaisen muuttujan päälle joka on
  viitemuuttuja.
- N1 tarkoittaa ensimmäisenä new:llä kekoon luotua oliota.
  N2 toista jne.
- Käytä &-merkkiä olioviitteisiin. (esim. &N3 viittaa N3:een)
```

```
/* 01 */ public class Esivaalit {
/* 02 */
/* 03 */    public static class Ehdokas {
/* 04 */          private int aania;
/* 05 */          public static boolean valittu;
/* 06 */
/* 07 */          public Ehdokas(int aanestetty) {
/* 08 */                aania = ++aanestetty;
/* 09 */                valittu = !valittu;
/* 10 */          }
/* 11 */
/* 12 */          public void julista() {
/* 13 */                StringBuffer vaalilause =
                           new StringBuffer("Ääniä on");
/* 14 */                vaalilause.append(" ");
/* 15 */                System.out.println(vaalilause.toString()
                           + aania--);
/* 16 */          }
/* 17 */
/* 18 */          public void vaittele(Ehdokas vv, boolean voitolla) {
/* 19 */                if (voitolla)
/* 20 */                      julista();
/* 21 */                else
/* 22 */                      vv.julista();
/* 23 */          }
/* 24 */    }
/* 25 */
/* 26 */    public static void media(Ehdokas eka, Ehdokas toka) {
/* 27 */          Ehdokas.valittu = !Ehdokas.valittu;
/* 28 */    }
/* 29 */
/* 30 */    public static void media(String toka, String eka) {
```

```
/* 31 */          Ehdokas.valittu = Ehdokas.valittu;
/* 32 */    }
/* 33 */
/* 34 */    public static void main(String[] args) {
/* 35 */          Ehdokas eka = new Ehdokas(1000);
/* 36 */          Ehdokas toka = eka;
/* 37 */          toka.julista();
/* 38 */          eka = new Ehdokas(50);
/* 39 */          toka.vaittele(eka, false);
/* 40 */          media(toka, eka);
/* 41 */          eka = null;
/* 42 */          toka.julista();
/* 43 */    }
/* 44 */ }
```

2. Testien teko

a) Kirjoita alla olevaan luokkaan `toString`-metodi, jonka avulla testaaminen helpottuu. (2p)

b) Testaa alla olevasta luokasta 2 metodia ja sen sisäluokasta 1 metodi (ComTest tai JUnit4). (3p)

c) Pitäisikö sisäluokan olla `static`? Perustelu. (1p)

```
import java.io.*;
import fi.jyu.mit.ohj2.*;
import java.util.StringTokenizer;
import java.util.SortedMap;
import java.util.TreeMap;
import java.util.Set;
import java.util.Iterator;

/**
 * Luokka sanojen esiintymismäärien laskemiseksi.  Toteutettu
 * TreeMap-luokalla ja erillisellä Laskuri-luokalla.
 * @author Vesa Lappalainen
 * @version 1.0, 16.03.2003
 */
public class SanatSortedMap {

  public class Laskuri {
    private int arvo;
    public Laskuri(int arvo) { this.arvo = arvo; }
    public void lisaa() { arvo++; }
    public String toString() { return ""+arvo; }
  }


  SortedMap alkiot = new TreeMap();

  public void lisaa(String s) {
    Laskuri sana = (Laskuri)alkiot.get(s);
    if ( sana != null ) {
      sana.lisaa();
```

```
        return;
      }
      sana = new Laskuri(1);
      alkiot.put(s,sana);
    }

    public void tulosta(OutputStream os) {
      PrintStream out = Tiedosto.getPrintStream(os);
      Set entrySet = alkiot.entrySet();
      for (Iterator i=entrySet.iterator(); i.hasNext(); )
        out.println(i.next());
    }


    public void kasitteleRivi(String rivi) {
      String sana;
      StringTokenizer st = new StringTokenizer(rivi," ,();.:[]{}+-");
      while ( st.hasMoreTokens() ) {
        sana = st.nextToken();
        lisaa(sana);
      }
    }

    public void lueTiedosto(String tied) throws IOException {
      BufferedReader fi = Tiedosto.avaa_lukemista_varten(tied);
      if ( fi == null )
        throw new FileNotFoundException("Ei löydy: " + tied);

      try {
        String rivi;
        while ( ( rivi = fi.readLine() ) != null  ) {
          kasitteleRivi(rivi);
        }
      } finally {
        fi.close();
      }
    }
  }
```

3. Algoritmit

Lue alla olevat algoritmit "`kissa`" ja "`koira`". Nämä algoritmit tuottavat saman tuloksen eri tavalla laskien. Perustele vastauksesi.

a) mikä on ratkaisevin ero näiden toteutuksien välillä? (2p)

b) Arvioi algoritmeja luettavuuden kannalta. (2p)

c) algoritmisen kompleksisuuden (eli suorituksen tehokkuuden) kannalta. (2p)

bonus: mitä algoritmit laskevat (1p)

```
public static int kissa (final int n) throws ArithmeticException {
        if ( n < 0 )
          throw new ArithmeticException("Oltava positiivinen");
```

```
        if ( n < 2 ) return n;
        return kissa(n-2) + kissa(n-1);
    }

    public static int koira (final int n) throws ArithmeticException {
        if ( n < 0 )
          throw new ArithmeticException("Oltava positiivinen");
        if ( n == 0 ) return 0;
        int nykyinen = 1;
        int vanha = 0;

        for (int i=1; i<n; i++) {
                int summa = nykyinen + vanha;
                vanha = nykyinen;
                nykyinen = summa;
        }

        return nykyinen;
    }
```

4. Tiedostot

a) Erota `cutLongLines` -aliohjelmasta itse toiminnallisuus omaksi aliohjelmakseen. (2p)

b) Kerro lyhyesti mitä hyötyä ohjelmalogiikan erottelusta erillisiin aliohjelmiin on. (2p)

c) Miten muuttaisit ohjelmaa vielä siten, että sille voitaisiin tuoda parametrina operaatio, joka halutaan tehdä luetuille riveille? (2p)

bonus: Mitä muita toiminnallisia osia aliohjelmasta voisi vielä erotella ja miksi? (1p)

```
package teht4;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.PrintWriter;

import fi.jyu.mit.ohj2.Tiedosto;

public class FileFormatter {

    public static void cutLongLines(final String inFile, final String
                                     outFile, final int lineLen)
                                             throws IOException {
        BufferedReader in = null;
        PrintWriter out = null;
        try {
            in = Tiedosto.avaa_lukemista_varten(inFile);
            if (in == null)
                throw new IOException("Failed to open file: " + inFile);
            out = Tiedosto.avaa_kirjoittamista_varten(outFile);
            if (out == null)
               throw new IOException("Failed to open file for writing: "
                         + outFile);
            String line;
```

```
                while ((line = in.readLine()) != null) {
                    int beginIx = 0;
                    int fullChunks = line.length() / lineLen + 1;
                    for (int i = 1; i < fullChunks; i++) {
                        int endIx = beginIx + lineLen;
                        out.println(line.substring(beginIx, endIx));
                        beginIx = endIx;
                    }
                    out.println(line.substring(beginIx));
                }
            } finally {
                if (out != null)
                    out.close();
                if (in != null)
                    in.close();
            }
        }
    }
```

5. Matriisin generointi

Eräässä demotehtävässä tehtiin aliohjelma `matriisinSuurin`. Aliohjelma voi epäonnistua mm. jos suurin on jonkin rivin alussa tai lopussa. Ja toki jos suurin on jossakin keskellä. Helpointa testaaminen olisi jos olisi tapa tuottaa kaikki mahdolliset tietyn kokoiset matriisit, jossa suurin on kaikissa mahdollisissa matriisin paikoissa.

a) Piirrä 6 erilaista 2x3 matriisia (2 riviä, 3 saraketta), joissa kaikissa suurin alkio on eri paikassa. (1p)

b) Toteuta (Javalla) tapa tuottaa kaikki tietyn kokoiset matriisit, jossa suurin on ensimmäisellä kutsulla 1. rivin 1. sarakkeessa, 2. kutsulla 1. rivin 2. sarakkeessa ja lopulta m*n:llä kutsulla n:n rivin m:n sarakkeessa. (4p)

c) Kirjoita tekemääsi toteutusta käyttäen ”täydellinen” testi `matriisinSuurin`-aliohjelmalle kun matriisin koko on 3x4. (1p)