fi.jyu.mit.graphics ja ComTest

Vesa Lappalainen
Tietotekniikan laitos
Jyväskylän yliopisto
vesal@jyu.fi

Jonne Itkonen
Tietotekniikan laitos
Jyväskylän yliopisto
ji@jyu.fi

26. lokakuuta 2008

1 Johdanto

Ohjelmoinnin opetuksessa opiskelijoiden mielenkiinnon ylläpito on monesti kovin haasteellista. Perusasioiden opettaminen on hyvä tehdä yksinkertaisilla esimerkkiohjelmilla, mutta nykyisiä opiskelijoita eivät pelkät kuvaruudulle tulostuvat tekstit ja numerosarjat paljon jaksa motivoida. Syykin on ymmärrettävä, kun katsoo millaisia ovat ohjelmat, joita he itse käyttävät. Ne ovat täynnä visuaalista ilotulitusta ja rikkaita äänimaisemia.

Tällaisen multimediaspektaakkelin ohjelmointi aivan ruohonjuuritasolta lähtien on työlästä. On kuitenkin kehitetty järjestelmiä, esimerkiksi Alice[1], jotka mahdollistavat kolmiulotteisten interaktiivisten ohjelmien yksinkertaisen toteutuksen. Monesti kuitenkin riittää vähempikin graafisuus, ja tätä varten on kehitetty fi.jyu.mit.graphics.

fi.jyu.mit.graphics on Java-ohjelmointikielellä kehitetty kirjasto, jonka avulla voidaan yksinkertaisesti tehdä kaksi- ja kolmiulotteisia graafisia sovelluksia. Kirjastoa on käytetty Jyväskylän yliopistossa ohjelmoinnin ensimmäisellä kurssilla ohjelmoinnin peruskäsitteitä opetettaessa. Opiskelijat ovat ottaneet kirjaston innolla vastaan.

Kohta peruskäsitteiden koostamisen opittuaan on opiskelijoiden hyvä oppia testaamisen merkitys. Näin he oppivat ymmärtämään hyödyn, mikä testaamisesta on ohjelmiston kehitykselle jo sen varhaisessa vaiheessa, eikä testauksesta muodostu pakollista rasitetta, joka vain pitää tehdä projektin lopussa. Opiskelija oppii suunnittelemaan ohjelmia testaten.[3]

Testien kirjoittaminen Javalla, vaikka JUnit[2] olisikin käytössä, on työlästä. Testit usein jäävät tällöin myös unhoon. ComTest-työkalu auttaa testien kirjoittamisessa, mutta nostaa myös testit esille, sillä ne dokumentoituvat mukaan rajapintadokumentaatioon esimerkkeinä rajapinnan käytöstä.

2 fi.jyu.mit.graphics

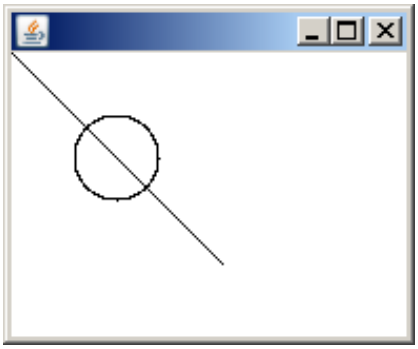
fi.jyu.mit.graphics, myöhemmin *graphics*, on Java-kirjasto yksinkertaisten kaksi- ja kolmiulotteisten ohjelmien tekemiseen. Kirjasto on kehitetty Jyväskylän yliopiston tietotekniikan laitoksella. Kirjasto esimerkkeineen ja lähdekoodeineen on julkisesti saatavilla osoitteessa `https://trac.cc.jyu.fi/projects/ohj1/wiki/graphics`.

2.1 Kuvaus

Alla on yksinkertainen esimerkkiohjelma graphics-kirjastolla tehtynä.

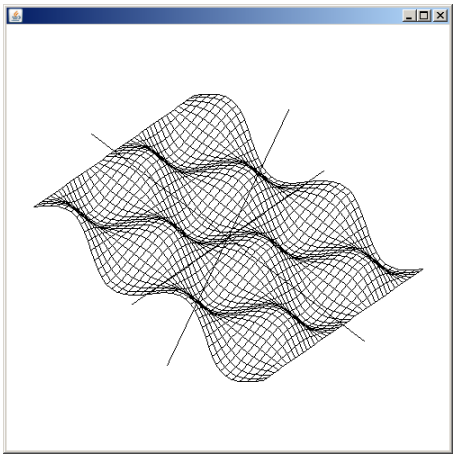
```
import fi.jyu.mit.graphics.EasyWindow;
/**
 * Yksinkertaisin esimerkki kirjaston käytöstä.
 * Piirtää nurkasta alkavan viivan ja ympyrän.
 * @author vesal
 */
public class SimpleGraphics {
    public static void main(String[] args) {
        EasyWindow window = new EasyWindow();
        window.addLine(0,0,100,100);
        window.addCircle(50,50, 20);
        window.showWindow();
    }
}
```

Ohjelma luo ikkunan, ja piirtää siihen ympyrän ja tämän lävistävän viivan. Kuva 1 näyttää ikkunan sisältöineen. Jo näin pienestä ohjelmasta näkyy graphics-kirjaston idea pyrkiä tiiviiseen ja selkeään grafiikan ohjelmoitavuuteen. Alustaminen ja reagoiminen tapahtumiin on pyritty tekemään mahdollisimman näkymättömäksi.



Kuva 1: Ruutukaappaus SimpleGraphics-ohjelman ikkunasta.

Myös monimutkaisempi, kolmiulotteinen piirto onnistuu, kuten kuvassa 2 on tehty.



Kuva 2: Monimutkaisempi, kolmiulotteinen kuva.

2.2 Luentokäyttö

Luennoilla graphics-kirjastoa on käytetty ohjelmoinin perusrakenteiden, kuten ehtolauseiden, silmukoiden, aliohjelmien, funktioiden ja muuttujien käytön opetteluun. Aliohjelmien[1] käyttöä voidaan motivoida vaikka seuraavasti. Esitellään opiskelijoille ohjelma, jolla voi piirtää kaksi lumiukkoa.

```
import fi.jyu.mit.graphics.EasyWindow;

/**
 * Piirretään lumiukkoja "tyhmällä" tavalla
 * @author vesal
 * @version 14.9.2008
 */
public class Lumiukot0 {
    public static void main(String[] args) {
      EasyWindow window = new EasyWindow();

      window.addCircle(20,80−2∗15−20−10,10);
      window.addCircle(20,80−15−20,15);
      window.addCircle(20,80,20);

      window.addCircle(70,90−2∗15−20−10,10);
      window.addCircle(70,90−15−20,15);
      window.addCircle(70,90,20);

      window.showWindow();
    }
}
```

Opiskelijoiden huomio kannattaa kiinnittää esimerkissä käskyjen peräkkäisyyteen ja kahteen samanlaiseen käskysarjaan. Näiden avulla saadaan havainnollistettua aliohjelman tarvetta.

Tämän jälkeen opiskelijoita pyydetään piirtämään useampia lumiukkoja, ja siirtämällä piirtolauseet aliohjelman

```
public static void
lumiukko(EasyWindow window, double x, double y)
```

ja muuttaen pääohjelma kutsumaan luotua aliohjelmaa. Kun lumiukon piirtokoodi on siirretty aliohjelmaan, on opeteltu yksinkertainen muuttujien käyttö parametrien käytön yhteydessä. Tätä voidaan vielä tarkentaa siirtämällä piirrossa esiintyviä laskutoimituksia kuvaavampien muuttujien arvoiksi, tai erillisiksi aliohjelmiksi tai funktioiksi.

3 ComTest

Omien ohjelmien ja demovastausten testien kirjoittamista helpottamaan on tehty ComTest-työkalu. ComTest mahdollistaa testien kirjoittamisen JavaDoc-dokumentaation joukkoon, jolloin testit voivat toimia myös esimerkkeinä aliohjelman käytöstä. ComTest generoi testikuvauksista tavallisia JUnit-testejä[2]. ComTestille on tehty myös liitännäinen Eclipse-kehitysympäristöön, joka helpottaa testien kirjoittamista ja ajamista. ComTest on saatavilla osoitteessa `https://trac.cc.jyu.fi/projects/comtest`.

3.1 Kuvaus

Alla on esimerkki yksinkertaisesta testistä ComTestillä toteutettuna:

1. Vaikka peruskursseillamme käytetty ohjelmointikieli Java ei varsinaisia aliohjelmia sisälläkään, käytetään Javan luokkametodeita ohjelmoinnin alkeisopetuksessa hyvin aliohjelmamaisesti. Tästä syystä käytämme alussa termiä *aliohjelma* termien *metodi* tai *luokkametodi* sijaan.

```
public class PieninJakaja {
//#STATICIMPORT

  /**
   * Funktiolla etsitään luvun pienin jakaja
   * @param n tutkittava luku
   * @return luvun pienin jakaja.  1 jos luku on alkuluku
   * @example
   * <pre name="test">
   *  pieninJakaja(1) === 1;
   *  pieninJakaja(2) === 1;
   *  pieninJakaja(3) === 1;
   *  pieninJakaja(4) === 2;
   *  pieninJakaja(5) === 1;
   *  pieninJakaja(6) === 2;
   * </pre>
   */
  public static int pieninJakaja(int n) {
    for (int i=2; i<=n/2; i++)
      if ( n % i == 0 ) return i;
    return 1;
  }

  public static void main(String[] args) {
    int n = 25;
    int jakaja = pieninJakaja(n);
    System.out.println(jakaja);
  }
}
```

Ohjelmasta nähdään, kuinka ComTest-testit kirjoitetaan JavaDoc-kommentiin *example*-tagiin. HTML-muotoilun mukainen *pre*-tagi mahdollistaa ComTest-testejä sisältävän Java-lähdekoodin API-dokumentaation tuottamisen ilman ComTestiä. Attribuutilla *name* voidaan halutessa määritellä testimoduulille nimi. ComTest-työkalulla muodostetaan annetusta testistä JUnit-testi, joka ajetaan JUnit-työkaluilla. ComTest-liitännäinen Eclipse-kehitysympäristöön mahdollistaa testien generoinnin ja ajamisen automatisoinnin.

Testien kirjoittamiseen käytetään ComTestin omaa makrokieltä, joka on hyvin paljolit Javaa höystettynä testien kirjoittamista helpottavalla syntaktisella kuorrutuksella. Esimerkiksi ylläoleva testi voitaisiin myös kirjoittaa taulukkomuodossa seuraavasti:

```
  /**
   * Funktiolla etsitään luvun pienin jakaja
   * @param n tutkittava luku
   * @return luvun pienin jakaja.  1 jos luku on alkuluku
   * @example
   * <pre name="test">
   *  pieninJakaja($arg) === $result;
   *
   *  ------------------
   *   $arg | $result
   *  ------------------
   *    1  |   1
   *    2  |   1
   *    3  |   1
   *    4  |   2
   *    5  |   1
   *    6  |   2
   * </pre>
   */
```

Muuttujat *arg* ja *pre* korvataan taulukon vastaavien sarakkeiden arvoilla, ja näin generoituneet testilauseet käännetään JUnit-testiksi.

Seuraavassa on testi metodille *add*, joka lisää lukujen summaa, minimiä ja maksimia taltioivaan *Counter*-luokkaan uuden luvun. Testi näyttää, kuinka ComTest-testeissä voidaan käyttää muuttujia, joille annetaan arvo alla olevassa taulukossa.

```
/**
```

```
 * Adds i to current counter if not allready too many added.
 * @param i value to add
 * @example
 * <pre name="testCounterAddTable">
 * Counter cnt = new Counter(3);
 * cnt.add($add); cnt.getCount() === $count;
 * cnt.getSum() === $sum;
 * cnt.getMax() === $max; cnt.getMin() === $min;
 *
 * ------------------------------
 *  $add | $count | $sum  | $max | $min
 * ------------------------------
 *    -  |   0    |  0    |   0  |   0  // after creation
 *    1  |   1    |  1    |   1  |   1
 *    2  |   2    |  1+2  |   2  |   1
 *    3  |   3    | 1+2+3 |   3  |   1
 *    4  |   3    |  6    |   3  |   1  // MaxN exceeded
 * ======================================
 *    5  |   1    |  5    |   5  |   5
 *    2  |   2    |  5+2  |   5  |   2
 *    3  |   3    | 5+2+3 |   5  |   2
 * ======================================
 *   -1  |   1    |  -1   |  -1  |  -1
 *    2  |   2    | -1+2  |   2  |  -1
 *    9  |   3    | -1+2+9|   9  |  -1
 *
 * </pre>
 */
public void add(int i) {
  // ...
}
```

Osoitteesta `http://kurssit.it.jyu.fi/ITKP102/2008s/luento/` löytyy 14. luennon luentovideoista esimerkki Hirsipuu-pelin toteutuksesta ComTestiä apuna käyttäen.

3.2 Luentokäyttö

Testaus osana ohjelmiston kehitystä tulee mukaan luennoille jo kurssin varhaisessa vaiheessa, sekä opetuksessa, että demoissa. Opetuksessa esitetyille aliohjelmille tehdään ensin testit, sitten vasta toteutus. Kun ComTestin käyttö on opittu, toivotaan ComTestin käyttöä opiskelijoilta myös demoja palauttaessa. Demoissa opiskelija toistaa opetuksessa käytettyä tyyliä tekemällä ensin tehtävän vaatimille aliohjelmille tyhjät rungot, eli aliohjelmat, jotka eivät toteuta vielä haluttua toiminnallisuutta. Tyhjälle aliohjelmalle tehdään sitten JavaDoc-kommentti, johon aletaan kirjoittamaan testiä. Testien kirjoittamisen jälkeen aletaan tekemään varsinaisen toiminnallisuuden toteutusta. Toinen vaihtoehto on kirjoittaa testiä ja toiminnallisuutta rinnan.

Yksi mahdollisuus olisi antaa opiskelijoille valmiina testit, joilla he voisivat vastauksensa testata. Tähän ei kuitenkaan ole lähdetty, sillä sopivien ja riittävän kattavien testien keksiminen on tärkeä osa ohjelmoinnin oppimista. Toki voisi kokeilla demoille tehtyä palautusautomaattia, johon olisi tallennettu tehtäville vakiotestit. Tämä toki vaatisi kurinalaisempaa nimeämistä ja rakennetta pakettien, luokkien ja metodien suhteen.

Toivomme testaavan ohjelmoinnin tavan säilyvän mukana opiskelijoilla peruskursseista valmistumiseen ja aina työelämään saakka.

Viitteet

[1] Matthew Conway et al.: “Alice: lessons learned from building a 3D system for novices”, teoksessa “CHI ’00: Proceedings of the SIGCHI conference on Human factors in computing systems”, (ss. 486–493), New York, NY, USA: ACM, 2000, ISBN 1-58113-216-6.

[2] “JUnit testing framework”, http://www.junit.org/.

[3] Michael Wick, Daniel Stevenson ja Paul Wagner: “Using testing and JUnit across the curriculum”, *SIGCSE Bull.*, 37(1), ss. 236–240, 2005, ISSN 0097-8418.