DD1339 Introduktion till datalogi 2013/2014

Uppgift nummer: 7

Namn: Marcus Larsson

Grupp nummer: 5

Övningsledare: Marcus Dicander

Betyg: ..... Datum: ..... Rättad av: .....

Exercise 5.62-5.67

Kod är bifogad i slutet på detta dokument.

Exercise 5.63

En HashMap är onödig i syftet eftersom vi inte behöver ha en primärnyckel för varje boll. Det duger bra med ett index eftersom vi ska iterera genom alla bollar ändå.

Ett HashSet känns överflödigt med funktionaliteter av hash. ArrayList fungerar väldigt bra för syftet då vi ska ha en lista av bollar där vi inte bryr oss om vilken som är vilken utan itererar bara genom alla och gör samma föränding på alla.

Enligt en studie på javacodegeeks[http://www.javacodegeeks.com/2010/08/java-best-practices-vector-arraylist.html] så är iteration genom en ArrayList dessutom markant snabbare än genom HashSet och därför är ArrayList bra för detta. Nu när man inte har så många bollar, så märker man dock ingen skillnad.

Exercise 5.67

Om man ökar GRAVITY så blir det som att gravitationen ökar. Bollen faller snabbare ned och studsar inte upp lika mycket.

Om man minskar GRAVITY så faller bollen långsammare och den studsar högre.

Om man sätter gravitationen till negativ så flyger bollen uppåt och kommer inte tillbaka då det inte finns något tak.

Exercise 5.68

public static final double TOLERANCE = 0,001;

private static final int PASSMARK = 40;

public static final char = 'h';

Exercise 5.69

Dem använder sig av konstanter för att definiera att det endast är 5 fält som läses in per inlägg i loggen. Om man då lägger till ett fält i slutet på loggen utan att ändra koden så kan man få ArrayIndexOutOfBoundsException. Detta kan på ett sätt vara bra för att försäkra att loggfilen ser ut på rätt sätt. Men det förhindrar också om det skulle ligga mer saker i loggen än bara det man vill hämta.

Konstanter används också för att bestämma på vilket index programmet hämtar årtal, månad och dag etc. När programmet läser in en fil som bara består av text kan den då veta att det första "ordet" eller gruppen av bokstäver är årtalet, nästa månad etc. När den nått index 4 (det femte ordet) så börjar den om igen på 0.

Ett annat problem är om det står text i ett av fälten istället för ett tal. Då får man InputMissmatchException.

Användandet av konstanter beror som sagt på helt vad man vill åstadkomma. Genom att använda på detta sätt kan det uppstå fel som gör att programmet krashar, men den kommer inte visa fel data. Därför är det på ett sätt bra och på ett annat sätt dåligt. Man skulle kunna hantera felen och skicka ett felmeddelande till skärmen som användaren förstår, eller i detta fall kanske skicka genom något annat medie till administratören vad som är fel på logfilen.

Att ha denna typ av data i konstanter tycker jag är bra, det innebär att man vet att det ändras endast på den raden i koden om hur programmet läser loggen.

Exercise 5.70

Förutsatt att man sätter årtalet till ett konstant årtal behöver man endast ändra på vilka index month, day, hour och minute är. Och sedan byta NUMBER_OF_FIELDS till 4. Då läser den bara 4 grupper av siffror innan den börjar om för månad igen.

I setWhen() behöver man då inte ändra något om årtalet är satt fast från början. Men om man vill ta bort årtal helt så måste man ta bort det även här. Men då går det inte längre ihop med metoden set() i java.util.Calendar,eftersom den skulle tolka det som att minuter togs bort.

Exercise 5.71

```
/**
 * Write a description of class NameGenerator here.
 *
 * @author Marcus Larsson
 * @version 2013.11.15
 */
public class NameGenerator
{
   /**
    * This method will take 3 first letters in last name + 2 first letters in first name, that will be your Star Wars fist name.
    * Your Star Wars last name will be 2 first letters in your mothers maiden name + 3 first letters in your city of birth.
    * @param firstName Enter your first name.
    * @param lastName Enter your last name
    * @param motherMaiden Enter your mothers maiden name
    * @
    */
   public String generateStarWarsName(String fistName, String lastName, String motherMaiden, String birthCity)
   {
     String starwarsName = "";
     starwarsName=lastName.substring(0,3)+ fistName.substring(0,2)+" "+ motherMaiden.substring(0,2)+birthCity.substring(0,3);;

     return starwarsName;
   }
}
```

Exercise 5.72

En String går inte att modifiera.

Man måste istället skapa en ny String och tilldela den nya strängen det som returneras från metoden substring.

Korrekt:

```
public void printUpper(String s){
                String result = s.toUpperCase();
                System.out.println(result);
}
```

Exercise 5.73

Variabler av primitiva typer lagrar värdet i variabeln medan referensvariabler till objekt bara lagrar en adress till objektet. Tilldilning av dessa sker på olika sätt.

Exempel, om a är av typen int och lagrar vädet 3. Sedan skriver man int i1=a; Det som händer då är att värdet som lagras i a kopieras till l1, inte adressen till var värdet 3 lagras för a. Det innebär att man får bara en kopia i en annan del av minnet, och ändrar man då på l1 så ändras inte a.

I exemplet i uppgiften så vid anropet swap(a,b) så tilldelas alltså parametern i1 värdet i a, men inte referensen till a. Därför när ändringarna görs så utförs dem bara på i1 och i2 och rör aldrig a och b.

Exercise Frågor från webben

Tidskomplexitet på bubblesort

Algoritmen går som minst igenom hela vektorn 2 gånger.

Antal iterationer för varje loop är n-1 eftersom den går från index 0 till n-2.

Detta innebär antalet operationer blir

2(n-1) = 2n-2

Men i värsta fall är listan sorterad baklänges. exempelvis om man ska sortera 5,4,3,2,1 i stigande ordning istället så utgör det värsta fallet. Detta innebär att 2n-2 utförs n-1 gånger. Eftersom första gången lyckas den flytta största elementet sist. Nästa gång näst största till näst sist osv. i mitt exempel, när den gjort detta för 5,4,3,2, så står redan 1 först och den kommer då fram till i den andra loopen att listan är sorterad.

Alltså är värsta fall: (n-1)(2n-2) = $2n^2$-2n-2n+2

Tidskomlexitet(1 sekund = 10^6 microsekunder.)

Uppgiften löstes genom att sätta funktionen till 10^6 och lösa ut n. Då får man ut värdet för 1 sekund. Viktigt att tänka på är att man inte kan avrunda uppåt då man inte hinner med fler än det värdet man får ut. Sedan upprepas samma steg men istället för 10^6 för en sekund blir en minut $60*10^6$ osv..

T(n)	1 sekund	1 minut	1 timme	1 dag	1 år
log(n)	$2^{1000000}$	$2^{60000000}$	$2^{3600000000}$	$2^{86400000000}$	$2^{31536000000000}$
n	10^6	$6*10^7$	$36*10^8$	$864*10^8$	$31536*10^9$
n log(n)	62746	2801417	133378058	2755147513	797633893349
n^2	1000	7745	60000	293938	5615692
n^3	100	391	1532	4420	31593
2^n	19	25	31	36	44
n!	9	11	12	13	16

Kod till Bouncing balls

BallDemo

```
import java.awt.Color;
import java.util.ArrayList;
```

```
import java.util.Random;

/**
 * Class BallDemo - a short demonstration showing animation with the
 * Canvas class.
 *
 * @author Michael Kölling and David J. Barnes
 * @author Marcus Larson (modified code 2013.11.11)
 * @version 2013.11.11
 */

public class BallDemo
{
   private Canvas myCanvas;
   private ArrayList<BouncingBall> balls;
   private ArrayList<BoxBall> boxBalls;
   private Random random;

   /**
    * Create a BallDemo object. Creates a fresh canvas and makes it visible.
    */
   public BallDemo()
   {
      myCanvas = new Canvas("Ball Demo", 600, 500);
      balls = new ArrayList<BouncingBall>();
      boxBalls = new ArrayList<BoxBall>();
      random = new Random();
   }

   /**
    * Simulate entered number of bouncing balls
    * @param numOfBalls Enter the number of balls you would like to simulate.
    */
   public void bounce(int numOfBalls)
   {
      int ground = 400;   // position of the ground line


      myCanvas.setVisible(true);

      // draw the ground
      myCanvas.drawLine(50, ground, 550, ground);

      // crate and show the balls
      for (int i=0;i<numOfBalls;i++){
         BouncingBall ball = new BouncingBall(random.nextInt(250)+50, random.nextInt(180)+20,
random.nextInt(15)+8, new Color(random.nextInt(256),random.nextInt(256),random.nextInt(256)), ground,
myCanvas);
         balls.add(ball);
         ball.draw();
      }

      // make them bounce
      boolean finished =  false;
      while(!finished) {
         myCanvas.wait(50);          // small delay
         for(BouncingBall ball:balls){
```

```
            ball.move();
            // stop once ball has travelled a certain distance on x axis
            if(ball.getXPosition() >= 550) {
               finished = true;
            }
         }

      }
   }

   /**
    * Simulate entered number of balls bouncing inside a box.
    * @param numOfBalls Enter the number of balls you would like to simulate.
    */
   public void boxBounce(int numOfBalls){

      int top = 90;
      int ground = 300;
      int leftWall = 100;
      int rightWall = 400;
      //draw the box
      myCanvas.drawLine(leftWall, top, rightWall, top);
      myCanvas.drawLine(leftWall, ground, rightWall, ground);
      myCanvas.drawLine(leftWall, top, leftWall, ground);
      myCanvas.drawLine(rightWall, top, rightWall, ground);
      for (int i=0;i<numOfBalls;i++){
         int diameter = random.nextInt(10)+8;
         BoxBall ball = new BoxBall(random.nextInt(rightWall-leftWall-diameter)+leftWall+1, random.nextInt(ground-
top-diameter)+top+1, diameter, new Color(random.nextInt(256),random.nextInt(256),random.nextInt(256)),
ground,top,leftWall,rightWall, myCanvas);
         boxBalls.add(ball);
         ball.draw();
      }
      // make them bounce
      boolean finished =  false;
      while(!finished) {
         myCanvas.wait(50);          // small delay
         for(BoxBall ball:boxBalls){
            ball.move();
            // will never stop.
         }

      }
   }
}
```

BoxBall

```
import java.awt.*;
import java.awt.geom.*;
import java.util.Random;

/**
 * Class BoxBall - a graphical ball that observes the effect of gravity. The ball
 * has the ability to move. Details of movement are determined by the ball itself. It
 * will fall downwards, accelerating with time due to the effect of gravity, and bounce
 * upward again when hitting the ground.
 *
```

```
 * This movement can be initiated by repeated calls to the "move" method.
 *
 * @author Marcus Larsson (modified BouncingBall by Michael Kölling (mik), David J. Barnes and Bruce Quig)
 *
 *
 * @version 2013.11.11
 */

public class BoxBall
{
   private Ellipse2D.Double circle;
   private Color color;
   private int diameter;
   private int xPosition;
   private int yPosition;
   private final int groundPosition;      // y position of ground
   private final int topPosition;
   private final int rightWallPosition;
   private final int leftWallPosition;
   private Canvas canvas;
   private Random random;
   private int ySpeed, xSpeed;

   /**
    * Constructor for objects of class BoxBall
    *
    * @param xPos  the horizontal coordinate of the ball
    * @param yPos  the vertical coordinate of the ball
    * @param ballDiameter  the diameter (in pixels) of the ball
    * @param ballColor  the color of the ball
    * @param groundPos  the position of the ground (where the wall will bounce)
    * @param drawingCanvas  the canvas to draw this ball on
    */
   public BoxBall(int xPos, int yPos, int ballDiameter, Color ballColor,
                  int groundPos, int topPos, int leftWall, int rightWall, Canvas drawingCanvas)
   {
      random = new Random();
      xPosition = xPos;
      yPosition = yPos;
      color = ballColor;
      diameter = ballDiameter;
      groundPosition = groundPos;
      topPosition = topPos;
      rightWallPosition = rightWall;
      leftWallPosition = leftWall;
      canvas = drawingCanvas;
      ySpeed = random.nextInt(41)-20;
      xSpeed = random.nextInt(41)-20;
   }

   /**
    * Draw this ball at its current position onto the canvas.
    **/
   public void draw()
   {
      canvas.setForegroundColor(color);
      canvas.fillCircle(xPosition, yPosition, diameter);
```

```
    }

    /**
     * Erase this ball at its current position.
     **/
    public void erase()
    {
        canvas.eraseCircle(xPosition, yPosition, diameter);
    }

    /**
     * Move this ball according to its position and speed and redraw.
     **/
    public void move()
    {
        // remove from canvas at the current position
        erase();

        // compute new position
        yPosition += ySpeed;
        xPosition += xSpeed;

        // check if it has hit the ground
        if(yPosition >= (groundPosition - diameter) && ySpeed > 0) {
            yPosition = (int)(groundPosition - diameter);
            ySpeed = -ySpeed;
        }
        // check if it has hit the top
        if(yPosition <= (topPosition) && ySpeed < 0) {
            yPosition = (int)(topPosition)+1;
            ySpeed = -ySpeed;
        }
        // check if it has hit the right wall
        if(xPosition >= (rightWallPosition - diameter) && xSpeed > 0) {
            xPosition = (int)(rightWallPosition - diameter);
            xSpeed = -xSpeed;
        }
        // check if it has hit the left wall
        if(xPosition <= (leftWallPosition) && xSpeed < 0) {
            xPosition = (int)(leftWallPosition)+1; //+1 is to fix graphics bug, when turning on left wall it looked like 1 pixel
was erased from the wall.
            xSpeed = -xSpeed;
        }

        // draw again at new position
        draw();
    }

    /**
     * return the horizontal position of this ball
     */
    public int getXPosition()
    {
        return xPosition;
    }

    /**
```

```
     * return the vertical position of this ball
     */
    public int getYPosition()
    {
        return yPosition;
    }

    /**
     * @return Returns the diamater of this ball.
     */
    public int getDiameter(){
        return diameter;
    }

    /**
     * @return The vertical speed
     */
    public int getYSpeed(){
        return ySpeed;
    }
    /**
     * @return The horisontal speed
     */
    public int getXSpeed(){
        return xSpeed;
    }
}
```