Exercises 5.62-5.67

The **BallDemo** class.

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

/**
 * Class BallDemo - a short demonstration showing animation with the
 * Canvas class.
 *
 * @author Michael Ã¶Klling                balls.getClass(i).draw(); and David J. Barnes
 * @version 2011.07.31
 */
import java.util.HashSet;

public class BallDemo
{
    private Canvas myCanvas;

    private ArrayList<BouncingBall> balls = new ArrayList<BouncingBall>();
    private ArrayList<BoxBall> balls2 = new ArrayList<>();

    /**
     * Create a BallDemo object. Creates a fresh canvas and makes it visible.
     */
    public BallDemo()
    {
        myCanvas = new Canvas(”Ball Demo”, 600, 500);
    }


    /**
     * Simulate two bouncing balls
     */
    public void bounce(int amountOfBalls)
    {
        int ground = 400;   // position of the ground line

        myCanvas.setVisible(true);

        // draw the ground         BoxBall a = new BoxBall(1,2,16, Color.BLUE, ground, 10, 20, myCanvas);
        myCanvas.drawLine(50, ground, 550, ground);

        for(int i = 0; i < amountOfBalls; i++)
        {
            balls.add(new BouncingBall(randInt(0,550) , randInt(0,ground), 16, Color.BLUE, ground, myCanvas));
            balls.get(i).draw();
        }

        // make them bounce
        boolean finished =  false;
        while(!finished) {

            Iterator it = balls.iterator();

            while(it.hasNext())
            {
                ((BouncingBall)it.next()).move();
            }
            myCanvas.wait(50);           // small delay

            // stop once ball has travelled a certain distance on x axis

            Iterator it2 = balls.iterator();

            while(it2.hasNext() && !finished)
            {
                if(((BouncingBall)it2.next()).getXPosition() >= 550)
                {
                    finished=true;
                }
            }
        }

    }

    public void BoxBounce(int noOfBalls)
    {
        myCanvas.setVisible(true);
        myCanvas.setForegroundColor(Color.GREEN);
        myCanvas.setBackgroundColor(Color.GREEN);

        myCanvas.fillRectangle(10, 20 , 300, 200);



        for(int i = 0; i < noOfBalls; i++)
        {
            balls2.add(new BoxBall(randInt(30,300) , randInt(56,200), 16, RandomColour(), 0, 10,20, myCanvas));
            balls2.get(i).setSpeed(randInt(1, 10));
            balls2.get(i).draw();
        }

        // make them bounce
        boolean finished =  false;
        while(true) {

            Iterator it = balls2.iterator();

            while(it.hasNext())
            {
                ((BoxBall)it.next()).move();
            }
            myCanvas.wait(50);           // small delay
```

```java

            // stop once ball has travelled a certain distance on x axis

            Iterator it2 = balls2.iterator();

            while(it2.hasNext())
            {
                BoxBall current = (BoxBall)it2.next();
                if( current.getYPosition() >= 200 ||
                    current.getYPosition() <= 56)
                {
                    current.setSpeed(current.getSpeed() * -1);
                }
            }
        }
    }

    public Color RandomColour()
    {
        Random rm = new Random();

        float r = rm.nextFloat();
        float g = rm.nextFloat();
        float b = rm.nextFloat();

        return new Color(r,g,b);

    }


    private int randInt(int min, int max)
    {

        // NOTE: Usually this should be a field rather than a method
        // variable so that it is not re-seeded every call.
        Random rand = new Random();

        // nextInt is normally exclusive of the top value,
        // so add 1 to make it inclusive
        int randomNum = rand.nextInt((max - min) + 1) + min;

        return randomNum;
    }
}
```

The **BoxBall** class.

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

/**
 * Class BouncingBall - 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 Michael Ã¶Klling (mik)
 * @author David J. Barnes
 * @author Bruce Quig
 *
 * @version 2011.07.31
 */

public class BoxBall
{
    private static final int GRAVITY = 3;  // effect of gravity

    private int ballDegradation = 2;
    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 Canvas canvas;
    private int ySpeed = 1;                // initial downward speed

    int x,y;

    /**
     * Constructor for objects of class BouncingBall
     *
     * @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 x, int y, Canvas drawingCanvas)
    {
        xPosition = xPos;
        yPosition = yPos;
        color = ballColor;
        diameter = ballDiameter;
        groundPosition = groundPos;
        canvas = drawingCanvas;
        this.x = x;
        this.y = y;
    }

    /**
     * Draw this ball at its current position onto the canvas.
     **/
    public void draw()
```

```java
    {
        canvas.setForegroundColor(color);
        canvas.fillCircle(xPosition-x, yPosition-y, diameter);
    }

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

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

        yPosition += ySpeed;


        // check if it has hit the ground

        // 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;
    }

    public void setSpeed(int value)
    {
        ySpeed = value;
    }
    public int getSpeed()
    {
        return ySpeed;
    }
}
```

Exercise 5.68

```
public final double tolerance = 0.001;
private final int passMark = 40;
public final char helpChar = ’h’;
```

Exercise 5.69

The constants in the **LogEntry** class are used to shape the structure of a general log entry. For instance, in the fields below.

```
private static final int YEAR = 0, MONTH = 1, DAY = 2,
                         HOUR = 3, MINUTE = 4;
private static final int NUMBER_OF_FIELDS = 5;
```

They assume that each log entry, at any time in the future, will have this structure. If log files keep to have this structure, this can be a good way of using the *constant* functionality as it hides unnecessary information from the user. However, sometimes, the structure of a log entry can change; in that case, it’s better to allow the user to specify these as optional parameters.

Exercise 5.70

We would not need to change that much in the **LogEntry** class to make sure that it understands the new format. We only have to change the fields and the constructor. Yes, it’s good to use named constants as it facilitates this sort of procedure.

Exercise 5.71

```
public class NameGenerator
{
    public static String generateStarWarsName(String firstName, String
        lastName, String mothersMaidenName, String homeTown)
    {
        String swFirstName = lastName.substring(0,3) +
            firstName.substring(0,2).toLowerCase();
        String swLastName = mothersMaidenName.substring(0,2) +
            homeTown.substring(0,3).toLowerCase();


        return swFirstName + ” ” + swLastName;
    }
}
```

Exercise 5.72

The book’s version does not work because the **String** class is *immutable*. The *toUpperCase* method does not modify the object, but rather a new *String* object is returned.

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

Exercise 5.73

Int is an *immutable* object so we cannot change the value once it is declared (unless we assign a new value to it). Int is a value type and when **a** and **b** are passed into **swap** method, we only send the value of them, not the reference. Therefore, the changes do not affect the **a** and **b** outside the method.

Exercise 'bubble sort'

The algorithm will have to perform approx. n^2 for an array of n elements. However, two cases can be considered: when the array is sorted, and when it is unsorted.

1. When it is sorted, for example $\{1, 2, 3, 4\}$, we need $2x$ operations, where $x = n - 2$. That is, the array has to contain at least two elements.
2. When it is unsorted, for example $\{4, 3, 2, 1\}$, we need $2x^2$ operations, where $x = n - 2$. This type of unsorted array will have the most number of operations as the algorithm is linear and works from left to right.

Exercise 'time vs. n'

$T(n)$	1 second	1 minute	1 hour	1 day	1 year
$\log(n)$	$2^{10^6} \approx 9.9 \times 10^{301029}$	$2^{6\times10^7}$	$2^{3.6\times10^9}$	$2^{8.6\times10^{10}}$	$2^{3.2\times10^{13}}$
n	10^6	6×10^7	3.6×10^9	8.6×10^{10}	3.2×10^{13}
$n\log(n)$	6.2×10^4	2.8×10^6	1.3×10^8	6.0×10^{10}	8.1×10^{11}
n^2	$\sqrt{10^6} \approx 3.2 \times 10^3$	7.7×10^3	6×10^4	3.0×10^5	5.7×10^6
n^3	$\sqrt[3]{10^6} \approx 2.2 \times 10^2$	4.0×10^2	1.5×10^3	4.4×10^3	3.2×10^4
2^n	$\log(10^6) \approx 20$	≈ 26	≈ 32	≈ 36	45
$n!$	$\lessapprox 10$	$\lessapprox 11$	$\lessapprox 13$	$\lessapprox 14$	16