Pavel Tsvyatkov · Stepan Burlachenko

04 | 18 | 2019 · WH C 579

09 | Bouncing Balls

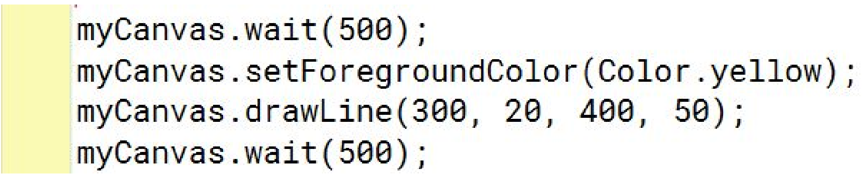
Index

Assignment

Task 1

Install the project in the lab. Experiment with canvas operations by making changes to the drawDemo method in BallDemo. Draw some more lines, shapes and text.

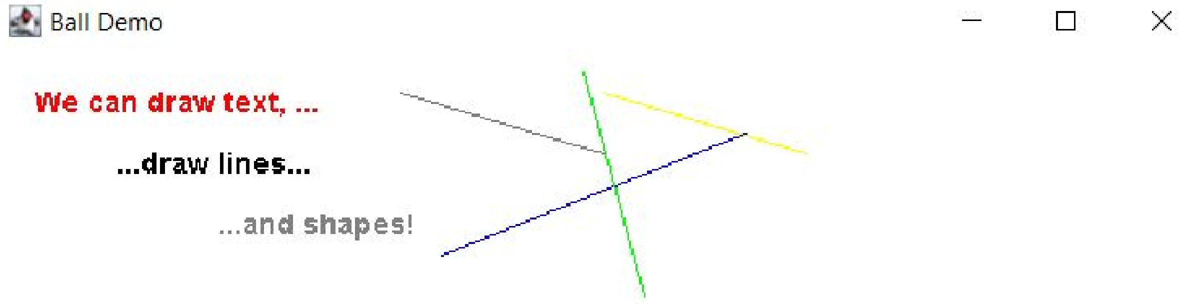
Before we started with the first exercise, we talked a bit about the pre-lab and what we have tried so far. Then we downloaded and opened the Bouncing Balls project in BlueJ.
First we decided to draw a line and wrote the following piece of code.



```
myCanvas.wait(500);
myCanvas.setForegroundColor(Color.yellow);
myCanvas.drawLine(300, 20, 400, 50);
myCanvas.wait(500);
```

Screenshot 1: drawing a yellow line

We wanted to draw a line with different color from what we currently had on the canvas so we used the setForegroundColor and set the color to yellow. We knew that drawLine() takes four parameters for the X and Y positions and we played with it a bit to see where the line will be drawn based on our input. We also set a small delay before and after execution. We tested it and we saw the yellow line on the canvas.



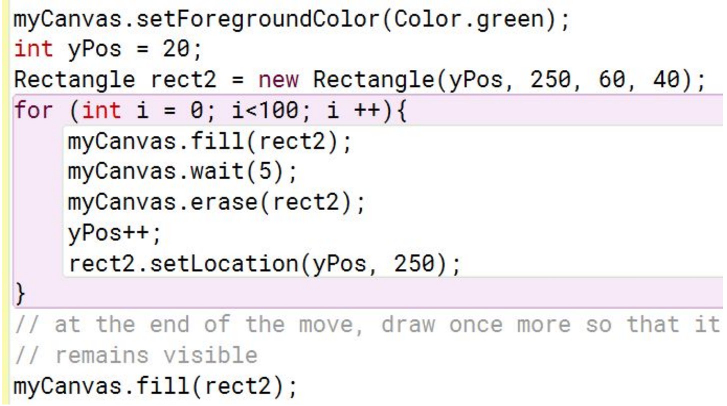
We can draw text, ...

...draw lines...

...and shapes!

Screenshot 2: result of drawing a yellow line

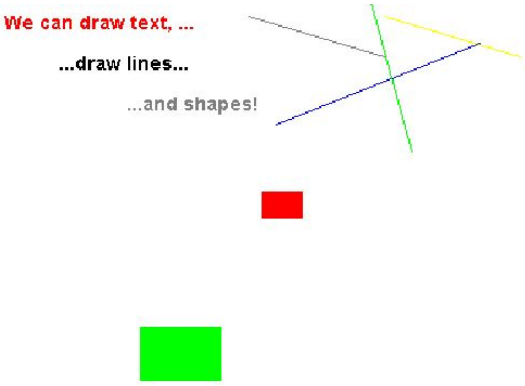
After that we wanted to draw a circle, so we looked in the Java API, but didn't find anything about Circle. We also took a look at Shape Interface, but there wasn't anything as well. We checked the Canvas class in the Bouncing Balls project and saw that there is a fillCircle method, but we weren't sure how it worked. So we decided to draw another rectangle and wrote the following code.



```
myCanvas.setForegroundColor(Color.green);
int yPos = 20;
Rectangle rect2 = new Rectangle(yPos, 250, 60, 40);
for (int i = 0; i<100; i ++){
    myCanvas.fill(rect2);
    myCanvas.wait(5);
    myCanvas.erase(rect2);
    yPos++;
    rect2.setLocation(yPos, 250);
}
// at the end of the move, draw once more so that it
// remains visible
myCanvas.fill(rect2);
```

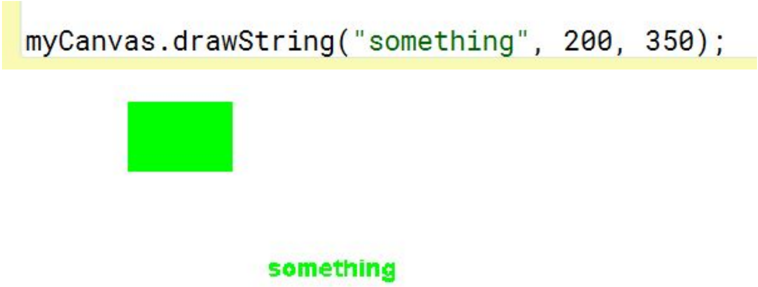
Screenshot 3: drawing another rectangle

We set the color to green and initialized a new variable named yPos. We then created the new rectangle "rect2" and wanted to make it bigger than the one we had already. We checked that a rectangle takes four ints as a parameter, where the first two are for the X and Y position and the last two are for the width and height of the rectangle. Then we wrote a for loop to make the rectangle move and in the end used the fill method so that it stays visible. We tested it and it worked exactly as we wanted it to.



Screenshot 4: result of drawing another rectangle

Up to this point one of the main things we were forgetting was to write "myCanvas." before a method and we were wondering for a few seconds why it doesn't work. In the end we wrote the following to draw a String under our figures and it worked.



```
myCanvas.drawString("something", 200, 350);
```

Screenshot 5: drawing another string and seeing the result

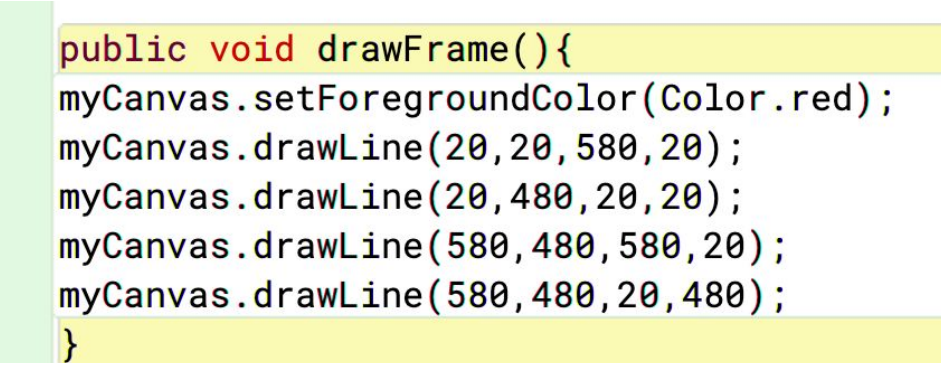
[15 min]

Task 2

Draw a frame around the canvas by drawing a rectangle 20 pixels inside the window borders. Draw four lines to make the rectangle. Put this functionality into a method called drawFrame in the BallDemo class.

At first, we were a little confused about the task, because we were not sure if the 20 pixels stated in the task were the thickness of the frame or how deep into the canvas it had to be placed. Our next challenge was figuring out the default size of the canvas. For some reason, we thought it was stated in the Canvas class and there we found a line saying “this(title, 300, 300, Color.white);”. This brought us to the idea that the canvas is 300x300 and from there we started playing with a new method to draw a frame.

Upon testing, we realized that the assumption of the 300x300 default size was entirely wrong and we had to search the code for the correct default size again. It was right in the Ball Demo class: 600x500. Based on that, we modified the method. First, it set the color for the frame, then it drew 4 separate lines: each 20 pixels away from the border.



```
public void drawFrame(){
myCanvas.setForegroundColor(Color.red);
myCanvas.drawLine(20,20,580,20);
myCanvas.drawLine(20,480,20,20);
myCanvas.drawLine(580,480,580,20);
myCanvas.drawLine(580,480,20,480);
}
```

Screenshot 6: drawing four lines to make up a frame

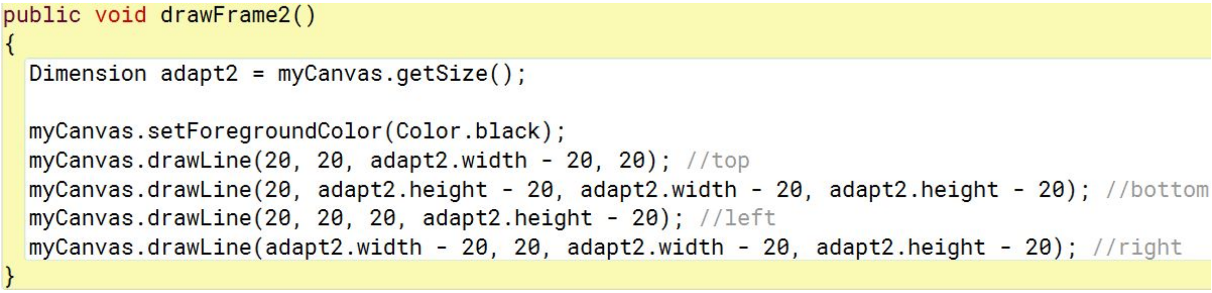
In the end, we got a nice square with red borders 20 pixels away from the canvas rim.

[15 min]

Task 3

Improve your drawFrame method to adapt automatically to the current canvas's size (that is, do not hard-code the size of the canvas into this method. To do this, you need to find out how to make use of an object of class Dimension. (Hint: check the API) Test it by manually resizing the canvas and calling drawFrame again.

We got to this exercise in the lab and got stuck. We looked at the Java API about the class Dimension, but we weren’t sure how to use it. We agreed to look at this exercise at home and continue working on it. After checking the Canvas class again we found a method called setVisible, which was using "Dimension" size = canvas.getSize(). Then we read the Java API about Dimension again and we got a better understanding of what we had to do. Our first approach was to create a Dimension object that gets the size of myCanvas and now with the help of it we could work with the width and height of any size canvas we create. We decided to draw four lines again as we did in the previous exercise, but instead of hard coding it, this time we used the dimension object we created to refer to the starting and ending point of the lines. In this way we were always getting the result we wanted.

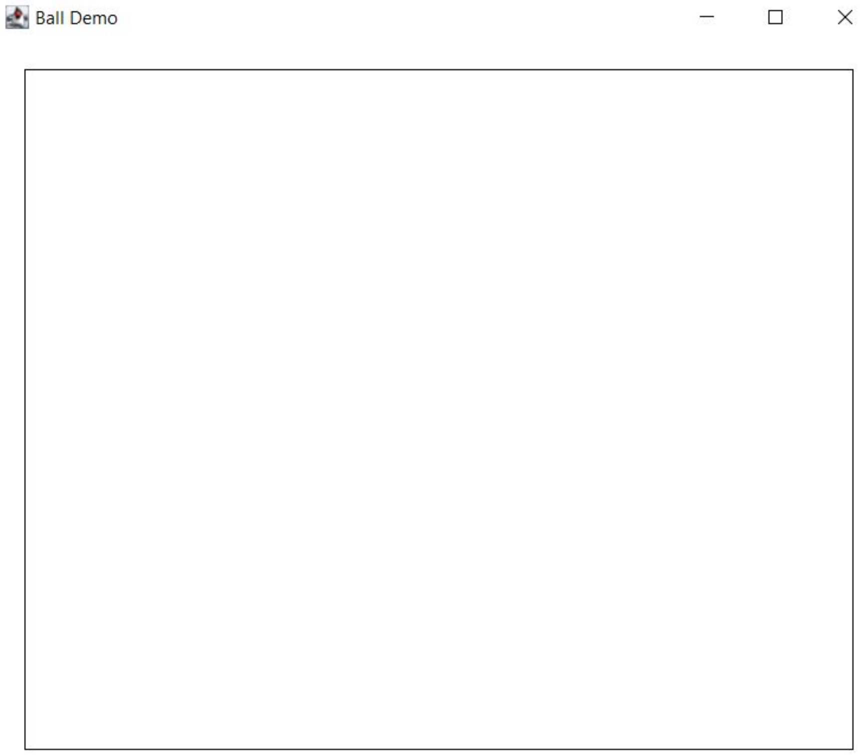


```
public void drawFrame2()
{
    Dimension adapt2 = myCanvas.getSize();

    myCanvas.setForegroundColor(Color.black);
    myCanvas.drawLine(20, 20, adapt2.width - 20, 20); //top
    myCanvas.drawLine(20, adapt2.height - 20, adapt2.width - 20, adapt2.height - 20); //bottom
    myCanvas.drawLine(20, 20, 20, adapt2.height - 20); //left
    myCanvas.drawLine(adapt2.width - 20, 20, adapt2.width - 20, adapt2.height - 20); //right
}
```

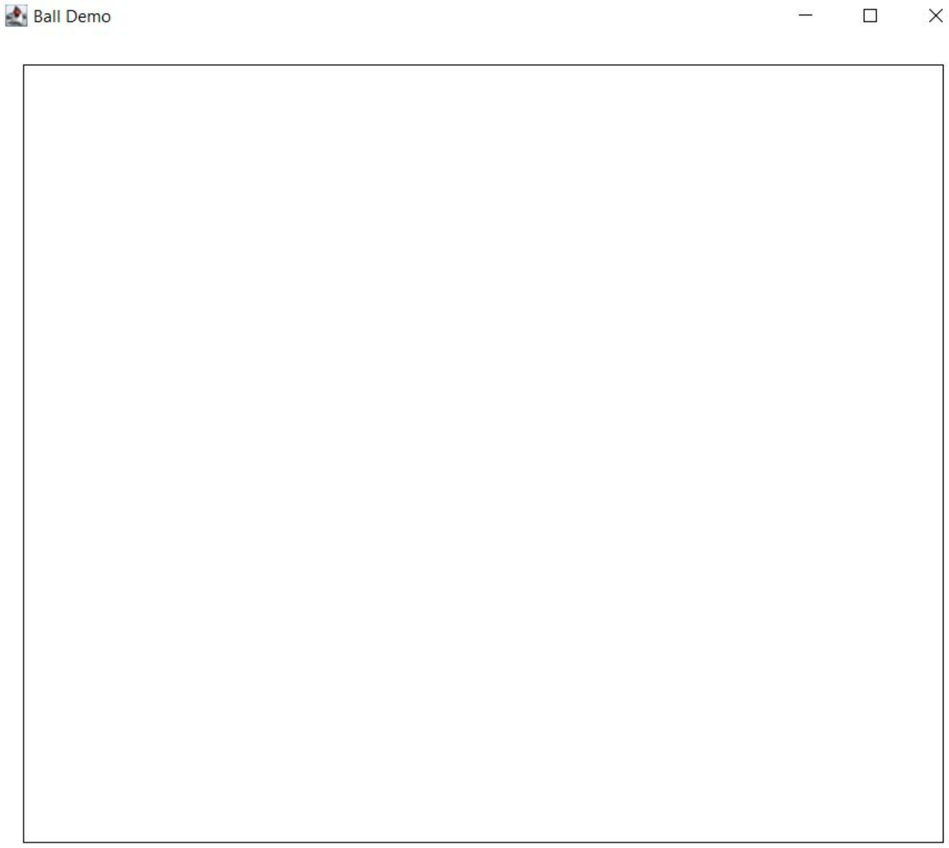
Screenshot 7: using the Dimension class to create a frame

Then, we went on to test this by changing the canvas size manually and checking if the four lines align well to form a border anytime.



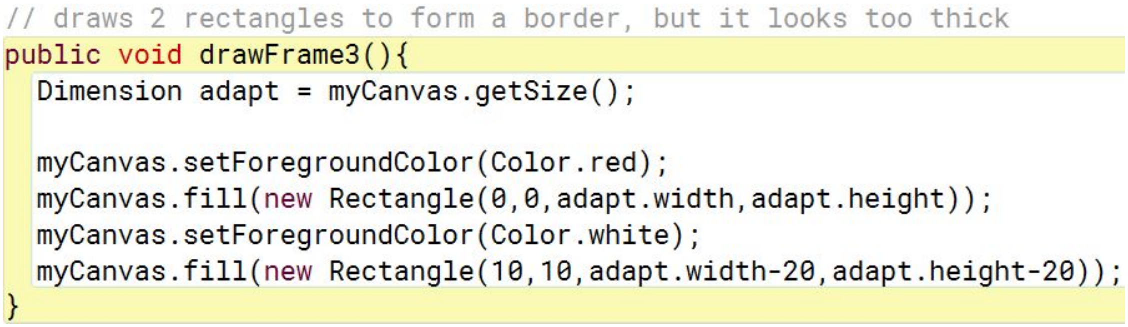
Screenshot 8:
a 600x500 Canvas

We also tested it with bigger canvas size and it was working well.



Screenshot 9: a 700x600 canvas

We also had another idea and decided to try something different to see if it works. We decided to create two rectangles with different colors that fill the canvas and make it look like we have a border. It sounded like a good idea to try so we wrote the following piece of code.

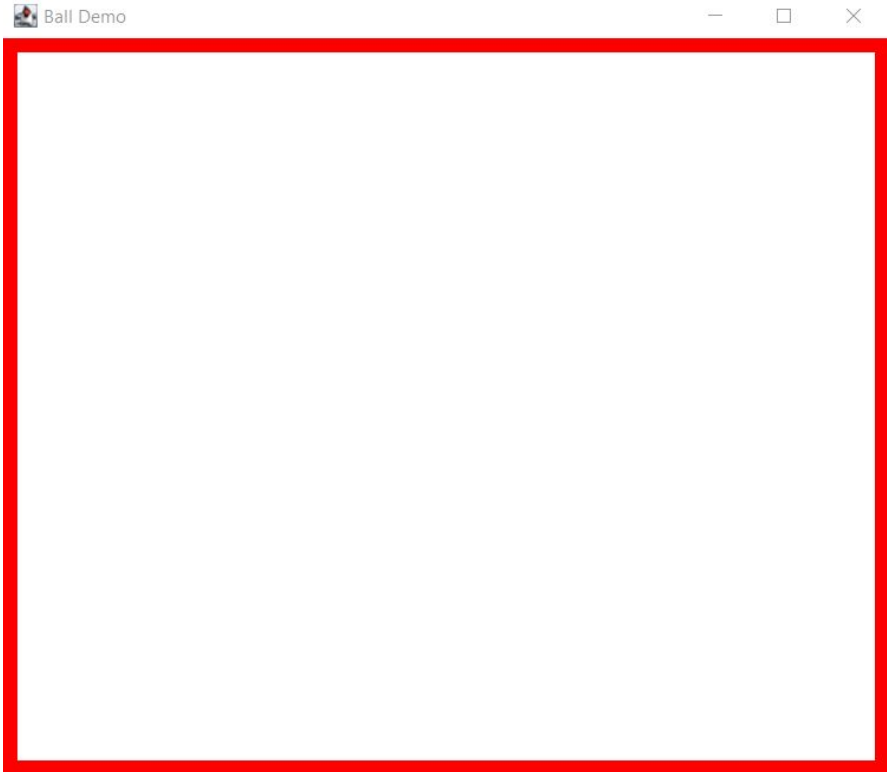


```
// draws 2 rectangles to form a border, but it looks too thick
public void drawFrame3(){
  Dimension adapt = myCanvas.getSize();

  myCanvas.setForegroundColor(Color.red);
  myCanvas.fill(new Rectangle(0,0,adapt.width,adapt.height));
  myCanvas.setForegroundColor(Color.white);
  myCanvas.fill(new Rectangle(10,10,adapt.width-20,adapt.height-20));
}
```

Screenshot 10: two rectangles placed onto each other

However, when we tested it we didn’t quite get the result we wanted and we weren’t sure how to fix that, so we just sticked with our previous method that worked well. But at least we got to try a different approach to solve this exercise.



Screenshot 11: creating two rectangles to imitate a frame.

[30 min]

Task 4

Change the method bounce to let the user choose how many balls should be bouncing. Use a collection to store the balls so that you can deal with one, three, or 75 balls at once. The balls should initially be placed in a row along the top of the canvas. But what kind of a collection? We know about ArrayList, HashMap, and HashSet. Explain in your report why you chose the collection you did.

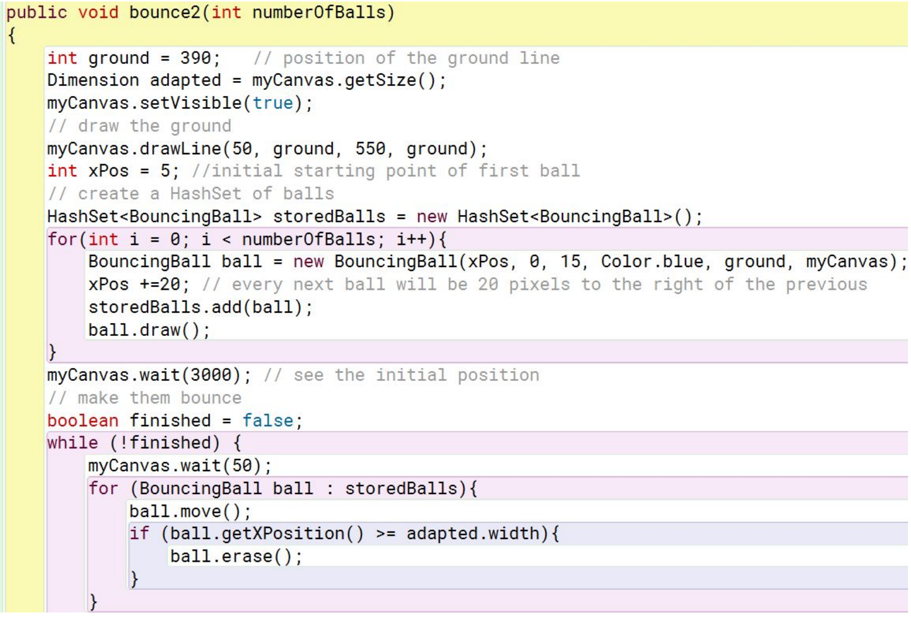
First, we looked at the bounce method that was already in the project to understand how the whole thing is built there. We played around with it for a while to make sure we understand the code well and we know what exactly each line of the code does. After a while, we saw that the position of the ground line is set by initializing a local variable called "ground", the canvas is made visible, the ground line is drawn, two balls are created one after another. What is of interest for us there was the parameters of these two new balls. After we had played around with the parameters for a while and changed them, we figured out the parameters are for the following things: the coordinates of the starting point on the x- and y-axises, the ball diameter, the color, the position of the ground to bounce off and finally the canvas where this all is happening on. Then, for the bouncing of these two balls, there is a while loop with a move method invoked on both balls (defined in the BouncingBall class). Then there is an if statement which checks for the position of the balls and when they have traveled a certain distance (550px) on the X-axis, the while loop then ends and the balls stop bouncing. Then they get erased immediately afterwards (again, using a method defined in the BouncingBall Class).

When it came to choosing which collection type to use, we decided to use a HashSet. We thought about it and agreed that we didn’t need to know the order in which the balls are placed and we didn’t need to be able to retrieve a certain ball object, which is what an ArrayList collection can do. We also thought that we did not need key-value pair of a HashMap for this collection.

In the improved bounce2 method, we again set the ground line, use an object of the Dimension class to get the current size of the Canvas to later use it for erasing the balls when they travel a certain distance, we make the canvas visible and then draw the ground line. We also initialized a new local variable "xPos" with the idea to use it as the starting point of the row, in which the balls will be placed one after another, in our for loop. Next step is creating a HashSet collection named storedBalls. The following for-each loop creates and then draws a collection of balls of the pre-set color and in the pre-set location. This is where we use the xPos value to move each following ball 20 pixels away from the previous one along the X-axis to create a row of balls. Then, we include the wait method to make sure the row is really created and the balls are NOT positioned unevenly.

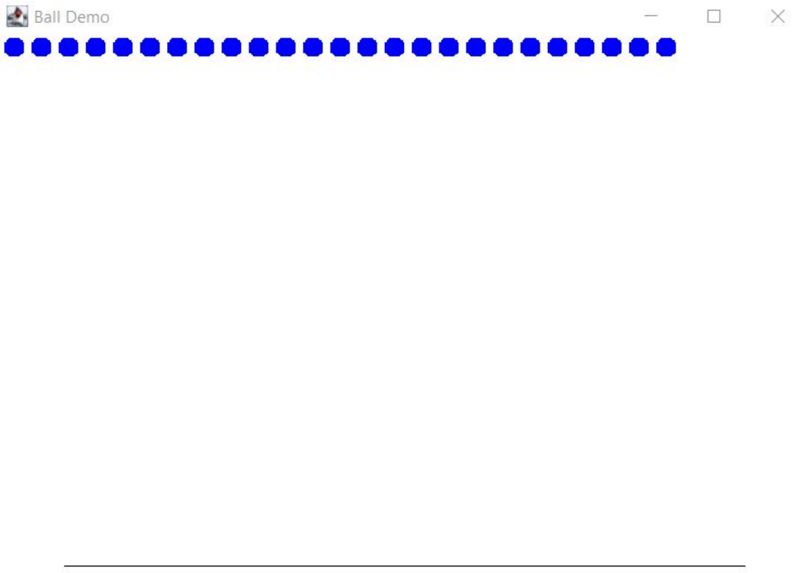
Finally, to make the created row of balls bounce, we use a for-each loop with pretty much the same setup as in the original method.

Obviously, using the HashSet required us importing this class to the top of our BallDemo class code.



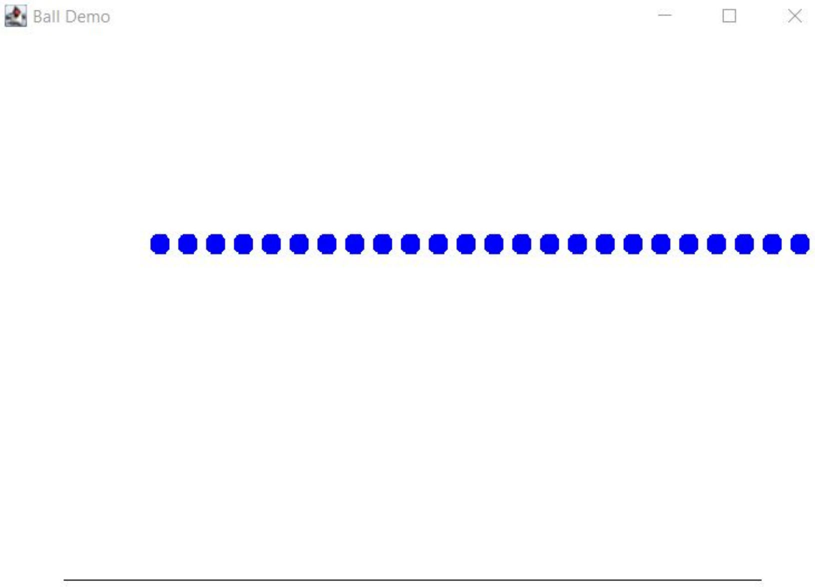
```
public void bounce2(int numberOfBalls)
{
    int ground = 390;   // position of the ground line
    Dimension adapted = myCanvas.getSize();
    myCanvas.setVisible(true);
    // draw the ground
    myCanvas.drawLine(50, ground, 550, ground);
    int xPos = 5; //initial starting point of first ball
    // create a HashSet of balls
    HashSet<BouncingBall> storedBalls = new HashSet<BouncingBall>();
    for(int i = 0; i < numberOfBalls; i++){
        BouncingBall ball = new BouncingBall(xPos, 0, 15, Color.blue, ground, myCanvas);
        xPos +=20; // every next ball will be 20 pixels to the right of the previous
        storedBalls.add(ball);
        ball.draw();
    }
    myCanvas.wait(3000); // see the initial position
    // make them bounce
    boolean finished = false;
    while (!finished) {
        myCanvas.wait(50);
        for (BouncingBall ball : storedBalls){
            ball.move();
            if (ball.getXPosition() >= adapted.width){
                ball.erase();
            }
        }
```

Screenshot 12: bounce2 method with a row of balls bouncing



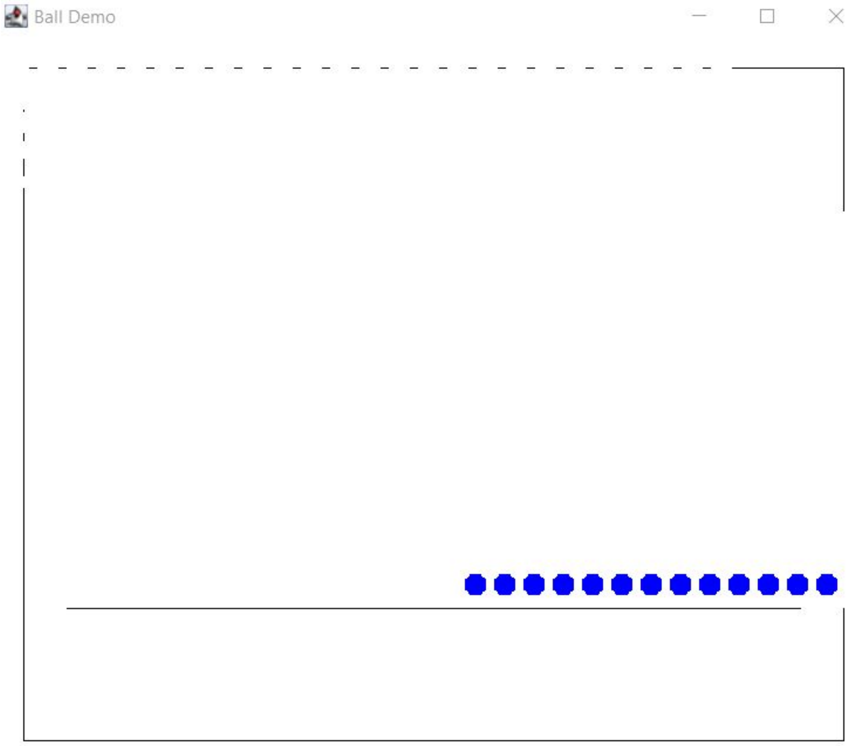
We tested our method and it was working well. The balls were being placed at the top of the canvas.

Screenshot 13: bounce2 method with all balls aligned at the top of the canvas



After the initial wait time we set, the balls started bouncing to the ground line and got erased in the end. It was interesting for us to play around with it, because we could change the size of the balls and even make all of them being different size, by changing the parameter to increase every time we loop.

Screenshot 14: bounce2 method showing the balls bouncing and disappearing



We also observed what happened when we had our frame drawn on the canvas. When a ball passes through the line, the place it went through was getting erased.

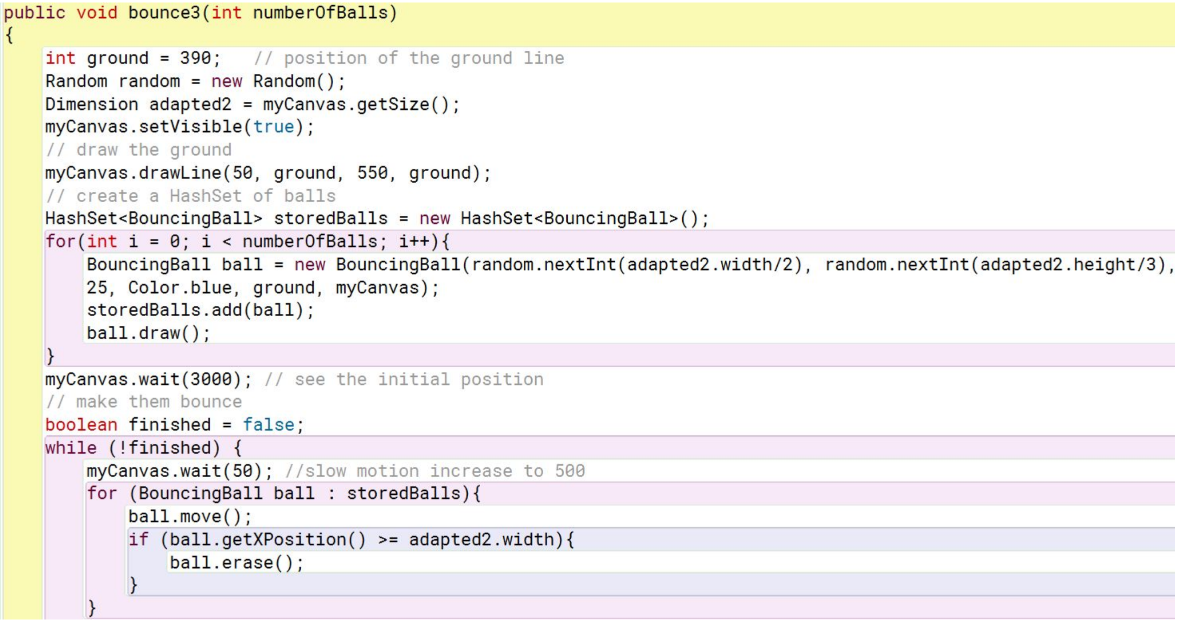
Screenshot 15: the balls erase the frame after bouncing

[25 min]

Task 5

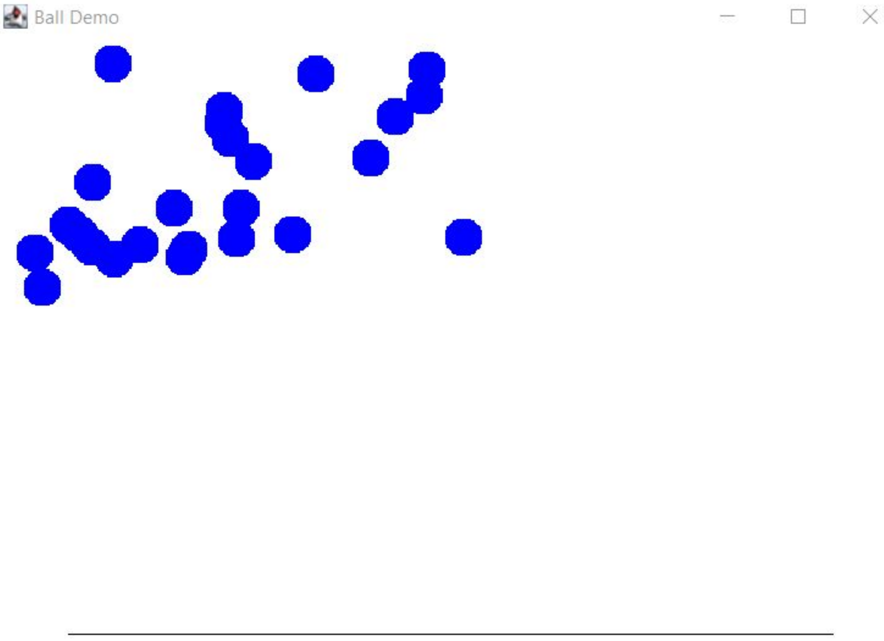
Change the method bounce to place the balls randomly anywhere in the top half of the screen.

To place the balls randomly, we had to import the Random class to the BallDemo class code. Then, we copied the same code from the Bounce2 method and did some slight modifications in the line creating the collection of balls. Their starting horizontal and vertical points have to be different for each loop adding a new ball to the collection. For calculating the starting position points, we used the nextInt(int bound) method of the Random class which we found in the Java API.It was generating a random number between 0 and the parameter we give as an input, so we decided it would be suitable in our case. Then to have all the balls placed somewhere close to the left upper corner of the canvas, for both X- and Y-positions we used the width and the height of the canvas and divided it by 2 and 3 correspondingly.



```
public void bounce3(int numberOfBalls)
{
    int ground = 390;   // position of the ground line
    Random random = new Random();
    Dimension adapted2 = myCanvas.getSize();
    myCanvas.setVisible(true);
    // draw the ground
    myCanvas.drawLine(50, ground, 550, ground);
    // create a HashSet of balls
    HashSet<BouncingBall> storedBalls = new HashSet<BouncingBall>();
    for(int i = 0; i < numberOfBalls; i++){
        BouncingBall ball = new BouncingBall(random.nextInt(adapted2.width/2), random.nextInt(adapted2.height/3),
        25, Color.blue, ground, myCanvas);
        storedBalls.add(ball);
        ball.draw();
    }
    myCanvas.wait(3000); // see the initial position
    // make them bounce
    boolean finished = false;
    while (!finished) {
        myCanvas.wait(50); //slow motion increase to 500
        for (BouncingBall ball : storedBalls){
            ball.move();
            if (ball.getXPosition() >= adapted2.width){
                ball.erase();
            }
        }
```

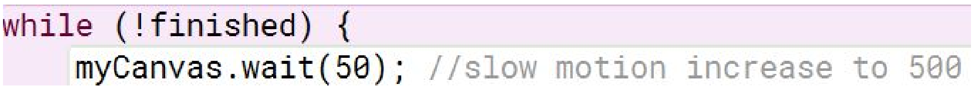
Screenshot 16: bounce3 method with the balls placed randomly



We tested our bounce3 method and it was working well. All of the balls were initially positioned in the top half of the canvas.

Screenshot 17: initial position of the balls after calling bounce3 method

Something funny we found while playing around was changing the wait in our while loop to half a second. Then the balls were moving as if we put them in slow motion.



```
while (!finished) {
    myCanvas.wait(50); //slow motion increase to 500
```

[20 min]

Evaluation

Pavel

This exercise reminded me a bit of our very first experience with BlueJ where we were drawing shapes. However, this time we were working with the code. I learned how to also draw lines and text and set them to have different color. I also learned how to draw a border using the drawLine method only and make the border adapt to the size of the canvas automatically. It was interesting to learn how to do it, because we were also working with the Java API and discovering new things. I also learned how to store the balls in a collection and make them bounce. Working on the code and finding solutions to the exercises made me think how to go step by step to solve something and it was a nice learning experience.

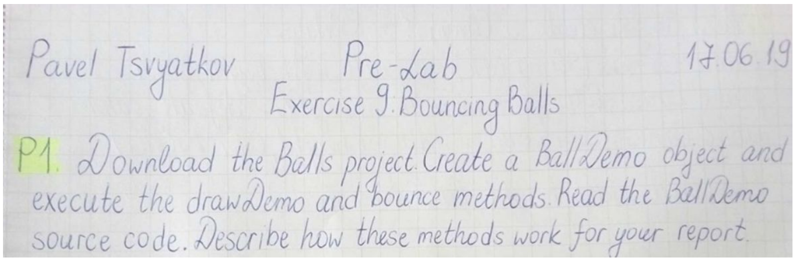
Stepan

This exercise reminded me of Lab 0 as well because of the shapes. Although the tasks were quite challenging and demanded looking up information in the API, I enjoyed doing it because of the visual aspect of the task. Everything we did, we could immediately see on the canvas. Getting instant feedback is always more stimulating.
This exercise made me revise which collection type work best in which scenario, learn different methods of new classes: Dimenstion,

Appendix

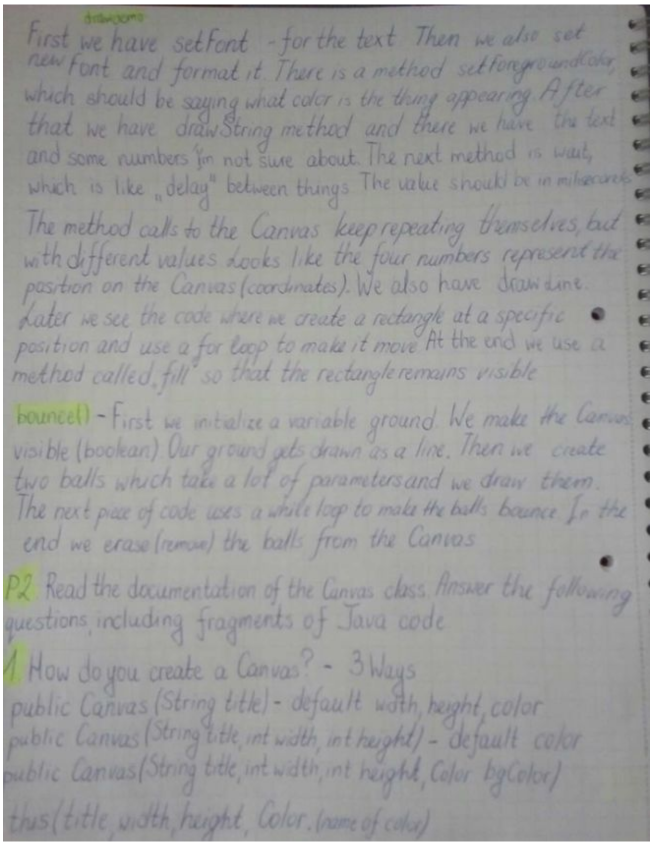
Pre Lab

Pavel



Pavel Tsvyatkov   Pre-Lab   17.06.19
Exercise 9. Bouncing Balls

P1. Download the Balls project. Create a BallDemo object and execute the drawDemo and bounce methods. Read the BallDemo source code. Describe how these methods work for your report.



drawDemo

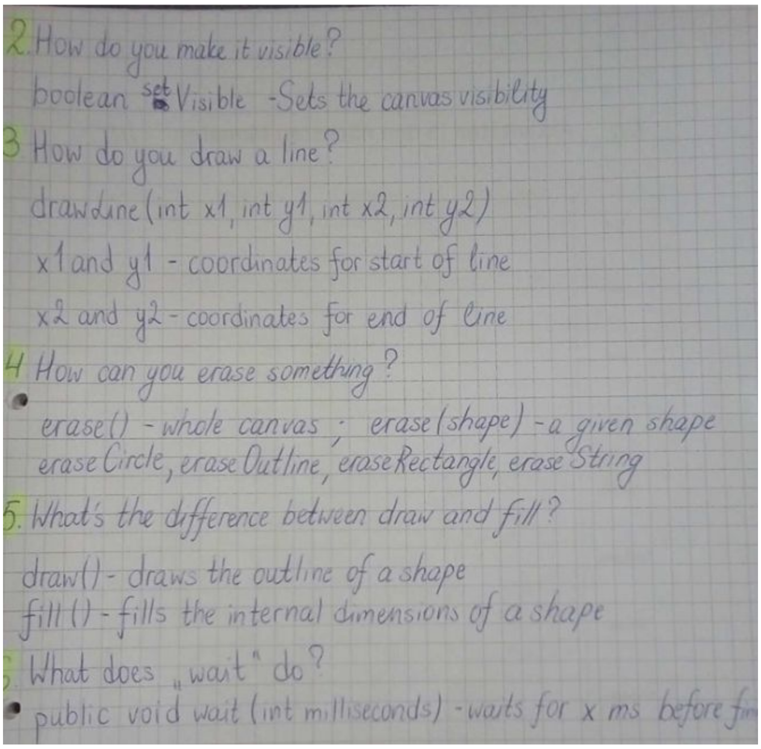
First we have setFont - for the text Then we also set new Font and format it. There is a method setForegroundColor, which should be saying what color is the thing appearing. After that we have drawString method and there we have the text and some numbers I'm not sure about. The next method is wait, which is like „delay" between things The value should be in miliseconds

The method calls to the Canvas keep repeating themselves, but with different values Looks like the four numbers represent the position on the Canvas (coordinates). We also have drawLine. Later we see the code where we create a rectangle at a specific position and use a for loop to make it move At the end we use a method called „fill" so that the rectangle remains visible

bounce() - First we initialize a variable ground. We make the Canvas visible (boolean). Our ground gets drawn as a line. Then we create two balls which take a lot of parameters and we draw them. The next piece of code uses a while loop to make the balls bounce. In the end we erase (remove) the balls from the Canvas

P2 Read the documentation of the Canvas class. Answer the following questions, including fragments of Java code

1 How do you create a Canvas? - 3 Ways
public Canvas (String title) - default width, height, color
public Canvas (String title, int width, int height) - default color
public Canvas(String title, int width, int height, Color bgColor)
this(title, width, height, Color. (name of color)



2. How do you make it visible?
boolean setVisible -Sets the canvas visibility

3 How do you draw a line?
drawLine(int x1, int y1, int x2, int y2)
x1 and y1 - coordinates for start of line
x2 and y2 - coordinates for end of line

4 How can you erase something?
erase() - whole canvas ; erase(shape) - a given shape
eraseCircle, eraseOutline, eraseRectangle, eraseString

5. What's the difference between draw and fill?
draw() - draws the outline of a shape
fill() - fills the internal dimensions of a shape

6. What does „wait" do?
public void wait (int milliseconds) - waits for x ms before fin

Stepan

P1. Download the Balls project. Create a BallDemo object and execute the drawDemo and bounce methods. Then read the BallDemo source code. Describe, in detail, how these methods work for your report.

drawDemo

First of all, the method does not take any parameters meaning no user input is required and the output of the method is always the same.
All the following statements in the method are methods of the class Canvas and the class Graphics2D. Among them are:

- setFont (Canvas)
- setForegroundColor (Canvas) - drawString (Graphics)
- wait (Canvas)
- drawLine (Graphics)

These are used to write two strings, draw three lines, and write another string.

The second part of the method is creating and moving a rectangle. First, a statement is used to create a rectangle of desired size and put it in the desired position.
Then, the methods of the Graphics2D class as well as the Canvas are used.

- fill -wait

Then, to move the rectangle a sequence of other methods is used.

The erase method of the MethodType class is used to erase all reference types to Object. Then the horizontal position (x) is getting increased by one, this changing the location of the object, finally stopping when doing the for each loop 200 times.

Finally, the fill method is used to draw the rectangle in its finite position.

bounce

Again, this method does not take any parameters meaning the outcome does not depend on the user.
First, we set the position of the ground. Then we use the method of the Component class to make the canvas visible: setVisible.

The drawLine method of the Graphics class. The line drawn starts at 50 and ends at 550 (the x-axe). The y-axe is determined by the ground initiated at 400.
Next, two instances of the Bouncing Balls class are created and they are assigned a color and an initial position.

Finally, they are made to bounce. The final action is the erasing method of the MethodType class to erase all references types to Object.

P2. Read the documentation of the Canvas class. Then answer the following questions in writing, including fragments of Java code:

1. How do you create a Canvas?
Creating a canvas requires four parameters: title, width, height and bgColor.

2. How do you make it visible? The method to use for this is setVisible.

```
public void setVisible(boolean visible)
```

3. How do you draw a line?
The drawLine method has to be used here. To draw a line, this method has to have 4 parameters responsible for the start and the end of both x- and y-lines.

```
public void drawLine(int x1, int y1, int x2, int y2)
```

4. How can you erase something?
To erase an object, one has to use the erase the method and the object to be erased as the parameter.

```
public void erase(java.awt.Shape shape)
```

5. What is the difference between draw and fill?
The definition of fill - Fill the internal dimensions of a given shape with the current foreground color of the canvas.
The definition of draw - Draws an object onto the canvas.

6. What does wait do?

Waits for a specified number of milliseconds before finishing. This provides an easy way to specify a small delay which can be used when producing animations.

```
public void wait(int milliseconds)
```

Final Code

BallDemo

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


/**
 * Class BallDemo - provides two short demonstrations showing how to use the
 * Canvas class.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 2008.03.30
 */

public class BallDemo
{
    private Canvas myCanvas;

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

    /**
     * Demonstrate some of the drawing operations that are
     * available on a Canvas object.
     */
    public void drawDemo()
    {
        myCanvas.setFont(new Font("helvetica", Font.BOLD, 14));
        myCanvas.setForegroundColor(Color.red);

        myCanvas.drawString("We can draw text, ...", 20, 30);
        myCanvas.wait(1000);

        myCanvas.setForegroundColor(Color.black);
        myCanvas.drawString("...draw lines...", 60, 60);
        myCanvas.wait(500);
        myCanvas.setForegroundColor(Color.gray);
        myCanvas.drawLine(200, 20, 300, 50);
        myCanvas.wait(500);
        myCanvas.setForegroundColor(Color.blue);
        myCanvas.drawLine(220, 100, 370, 40);
        myCanvas.wait(500);
        myCanvas.setForegroundColor(Color.green);
        myCanvas.drawLine(290, 10, 320, 120);
        myCanvas.wait(1000);
        myCanvas.setForegroundColor(Color.gray);
        myCanvas.drawString("...and shapes!", 110, 90);

        myCanvas.wait(500);
        myCanvas.setForegroundColor(Color.yellow);
        myCanvas.drawLine(300, 20, 400, 50);
        myCanvas.wait(500);

        myCanvas.setForegroundColor(Color.red);

        // the shape to draw and move
        int xPos = 10;
        Rectangle rect = new Rectangle(xPos, 150, 30, 20);

        // move the rectangle across the screen
        for(int i = 0; i < 200; i ++) {
            myCanvas.fill(rect);
            myCanvas.wait(10);
            myCanvas.erase(rect);
            xPos++;
            rect.setLocation(xPos, 150);
        }

        myCanvas.setForegroundColor(Color.red);
        myCanvas.fill(rect);

        myCanvas.setForegroundColor(Color.green);
        int yPos = 20;
        Rectangle rect2 = new Rectangle(yPos, 250, 60, 40);
        for (int i = 0; i<100; i ++){
            myCanvas.fill(rect2);
            myCanvas.wait(5);
            myCanvas.erase(rect2);
            yPos++;
            rect2.setLocation(yPos, 250);
        }
        // at the end of the move, draw once more so that it
        // remains visible
        myCanvas.fill(rect2);

        myCanvas.drawString("something", 200, 350);
    }

    /**
     * Simulate two bouncing balls
     */
    public void bounce()
    {
        int ground = 500;   // position of the ground line

        myCanvas.setVisible(true);

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

        // crate and show the balls
        BouncingBall ball = new BouncingBall(350, 50, 33, Color.blue, ground, myCanvas);
        ball.draw();
        BouncingBall ball2 = new BouncingBall(370, 80, 43, Color.red, ground, myCanvas);
        ball2.draw();

        // make them bounce
        boolean finished =  false;
        while(!finished) {
            myCanvas.wait(50);           // small delay
            ball.move();
            ball2.move();
            // stop once ball has travelled a certain distance on x axis
            if(ball.getXPosition() >= 550 && ball2.getXPosition() >= 550) {
                finished = true;
            }
        }
        ball.erase();
        ball2.erase();
    }

     public void drawFrame()
     {
      myCanvas.getSize();

      myCanvas.setForegroundColor(Color.red);
      myCanvas.drawLine(20,20,580,20);
      myCanvas.drawLine(20,480,20,20);
      myCanvas.drawLine(580,480,580,20);
      myCanvas.drawLine(580,480,20,480);
    }

    // draws 2 rectangles to form a border, but it looks too thick
    public void drawFrame3(){
      Dimension adapt = myCanvas.getSize();

      myCanvas.setForegroundColor(Color.red);
      myCanvas.fill(new Rectangle(0,0,adapt.width,adapt.height));
      myCanvas.setForegroundColor(Color.white);
      myCanvas.fill(new Rectangle(10,10,adapt.width-20,adapt.height-20));
    }
```

```
  public void drawFrame2()
  {
    Dimension adapt2 = myCanvas.getSize();

    myCanvas.setForegroundColor(Color.black);
    myCanvas.drawLine(20, 20, adapt2.width - 20, 20); //top
    myCanvas.drawLine(20, adapt2.height - 20, adapt2.width - 20, adapt2.height - 20);
//bottom
    myCanvas.drawLine(20, 20, 20, adapt2.height - 20); //left
    myCanvas.drawLine(adapt2.width - 20, 20, adapt2.width - 20, adapt2.height - 20); //right
  }

  public void bounce2(int numberOfBalls)
  {
     int ground = 390;   // position of the ground line
     Dimension adapted = myCanvas.getSize();

     myCanvas.setVisible(true);

     // draw the ground
     myCanvas.drawLine(50, ground, 550, ground);
     //initial starting point of first ball
     int xPos = 5;
     // crate a HashSet of balls and show the balls
     HashSet<BouncingBall> storedBalls = new HashSet<BouncingBall>();
     for(int i = 0; i < numberOfBalls; i++){
        BouncingBall ball = new BouncingBall(xPos, 0, 15, Color.blue, ground, myCanvas);
        xPos +=20; // every next ball will be 20 pixels to the right of the previous
        storedBalls.add(ball);
        ball.draw();
     }

     myCanvas.wait(3000); // see the initial position
     // make them bounce
     boolean finished = false;
     while (!finished) {
        myCanvas.wait(50);
        for (BouncingBall ball : storedBalls){
           ball.move();
           if (ball.getXPosition() >= adapted.width){
              ball.erase();
           }
        }
     }
  }

  public void bounce3(int numberOfBalls)
  {
     int ground = 390;   // position of the ground line
```

```
        Random random = new Random();
        Dimension adapted2 = myCanvas.getSize();

        myCanvas.setVisible(true);

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

        // crate a HashSet of balls and show the balls
        HashSet<BouncingBall> storedBalls = new HashSet<BouncingBall>();
        for(int i = 0; i < numberOfBalls; i++){
            BouncingBall ball = new BouncingBall(random.nextInt(adapted2.width/2),
random.nextInt(adapted2.height/3),
            25, Color.blue, ground, myCanvas);
            storedBalls.add(ball);
            ball.draw();
        }

        myCanvas.wait(3000); // see the initial position
        // make them bounce
        boolean finished = false;
        while (!finished) {
            myCanvas.wait(50); //slow motion increase to 500
            for (BouncingBall ball : storedBalls){
                ball.move();
                if (ball.getXPosition() >= adapted2.width){
                    ball.erase();
                }
            }
        }
    }
}
```