.Practical Exercise 12 – Timers

This prac demonstrates how a Timer object can be used to create a simple game involving a ball moving around the screen.

# Task 1 - One ball

Copy the following code samples into three Java classes (e.g. *OneBall*, *Court*, *Ball*) in the same project folder. You should see a green ball bouncing around inside a rectangle – but something is wrong.

Examine the code carefully and make the following changes:

* double the size of the ball
* halve the speed of the ball
* make the ball start in the centre of the court, heading towards the top-right
* fix the program so that the ball bounces off all the walls correctly
* modify the Ball class to include the colour of the ball as one of its properties and use this to make the ball blue

**import** java.awt.\*;

**import** javax.swing.\*;

**import** java.util.Timer;

**import** java.util.TimerTask;

**class** MyCanvas **extends** JPanel

{

Court court;

**int** left = 10;

**int** right = 400;

**int** top = 10;

**int** bottom = 300;

Ball ball1;

**int** initBallX = 50;

**int** initBallY = 50;

**int** initBallXSpeed = 4;

**int** initBallYSpeed = 4;

**int** initBallSize = 20;

Timer timer1;

**int** initialDelay = 10;

**int** refreshRate = 10;

**public** **void** init()

{

setSize(right+left,bottom+top);

court = **new** Court(left, right, top, bottom);

ball1 = **new** Ball(initBallX, initBallY, initBallXSpeed, initBallYSpeed, initBallSize);

timer1 = **ne**imer();

// initial delay and refresh rate are in milliseconds

timer1.sche dule(**new** AnimationTask(), initialDelay, refreshRate);

}

**private** **class** AnimationTask **extends** TimerTask

{

**public** **void** run() // contains code that you want to run repeatedly

{

ball1.move(court);

repaint();

}

}

**public** **void** paint(Graphics g)

{

paintComponent(g);

g.setColor(Color.***black***);

court.draw(g);

g.setColor(Color.***green***);

ball1.draw(g);

}

}

**public** **class** OneBall

{

**public** **static** **void** main(String[] a)

{

MyCanvas myCanvas = **new** MyCanvas();

myCanvas.init();

JFrame window = **new** JFrame();

window.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

window.setBounds(20, 20, 620, 620);

window.getContentPane().add(myCanvas);

window.setVisible(**true**);

}

}

**import** java.awt.Graphics;

**public** **class** Court

{

**private** **int** left, right, top, bottom;

**public** Court(**int** left, **int** right, **int** top, **int** bottom)

{

**this**.left = left;

**this**.right = right;

**this**.top = top;

**this**.bottom = bottom;

}

**public** **int** getLeft()

{

**return** left;

}

**public** **int** getRight()

{

**return** right;

}

**public** **int** getTop()

{

**return** top;

}

**public** **int** getBottom()

{

**return** bottom;

}

**public** **void** draw(Graphics g)

{

g.drawRect(left, top, right-left, bottom-top);

}

}

**import** java.awt.Graphics;

**public** **class** Ball

{

**private** **int** x, y, xSpeed, ySpeed, size;

**public** Ball(**int** x, **int** y, **int** xSpeed, **int** ySpeed, **int** size)

{

**this**.x = x;

**this**.y = y;

**this**.xSpeed = xSpeed;

**this**.ySpeed = ySpeed;

**this**.size = size;

}

**public** **void** setCoordinates(**int** x, **int** y)

{

**this**.x = x;

**this**.y = y;

}

**public** **int** getX()

{

**return** x;

}

**public** **int** getY()

{

**return** y;

}

**public** **void** move(Court c)

{

//move the ball according to its speed

x = x + xSpeed; // update the ball's x-position

y = y + ySpeed; // update the ball's y-position

//if the ball hits any side of the court, change its direction

**if** ( x < c.getLeft() || x >= c.getRight() )

{

xSpeed = -xSpeed;

}

**if** ( y < c.getTop() || y >= c.getBottom() )

{

ySpeed = -ySpeed;

}

}

**public** **void** draw(Graphics g)

{

g.fillOval(x, y, size, size);

}

}

# Task 2 - Ball and Paddle

Your challenge is to make a mouse-controlled paddle to your program that can swat the bouncing ball.

You will make use of the Paddle class, the mouseListener, and the mouseMotionListener.

Make the following changes:

1. Copy *OneBall* into a new class, e.g. *BallAndPaddle*.

2. Copy and paste the following code into a new class called Paddle in the same project folder.

**import** java.awt.Graphics;

**import** java.awt.Rectangle;

**import** java.awt.geom.Ellipse2D;

**public** **class** Paddle

{

**private** **int** x, y, size;

**public** Paddle(**int** x, **int** y, **int** size)

{

**this**.x = x;

**this**.y = y;

**this**.size = size;

}

**public** **void** setCoordinates(**int** x, **int** y)

{

**this**.x = x;

**this**.y = y;

}

**public** **int** getX()

{

**return** x;

}

**public** **int** getY()

{

**return** y;

}

**public** **void** draw(Graphics g)

{

g.fillRect(x, y, size, size);

}

**public** **boolean** intersects(Ball b)

{

Rectangle r = **new** Rectangle (x,y,size,size);

Ellipse2D.Double e = **new** Ellipse2D.Double(b.getX(),b.getY(),b.getSize(),b.getSize());

**if** (e.intersects(r))

**return** **true**;

**else**

**return** **false**;

}

}

3. You will notice an error with the getSize() method call. Fix this by creating a getSize() method in the Ball class that returns the size of the ball.

4. To incorporate the paddle, add the follow code fragments in the appropriate locations…

**public** **class** MyCanvas **extends** JPanel **implements** MouseListener, MouseMotionListener

{

…

Paddle paddle;

**int** initPaddleX = 100;

**int** initPaddleY = 100;

**int** initPaddleSize = 20;

…

**public** **void** init()

{

…

paddle = **new** Paddle(initPaddleX, initPaddleY, initPaddleSize);

addMouseListener(**this**);

addMouseMotionListener(**this**);

…

}

**public** **void** paint(Graphics g)

{

…

g.setColor(Color.***red***);

paddle.draw(g);

…

}

@Override

**public** **void** mouseMoved(MouseEvent e) {

paddle.setCoordinates(e.getX(),e.getY());

repaint();

}

@Override

**public** **void** mouseClicked(MouseEvent e) {

paddle.setCoordinates(e.getX(),e.getY());

**if** (paddle.intersects(ball))

{

// System.out.println("Hit!"); // This might be helpful

timer1.cancel();

}

repaint();

}

…

5. You should now be able to swat the ball (i.e. freeze it). Run the program and check.

6. With the ball size set to 20, it can be a bit tricky to hit. One problem is that the mouse has to be still when you click on it. Why? How could you fix this?

# Task 3 - Two balls

Copy your code from *BallAndPaddle* and copy it into a new class e.g. *TwoBalls*.

Using your existing Ball class, create a second ball with properties different to the first ball, and animate it using a second timer.

Modify your code so that both balls can be stopped independently with the one swatter.

# Task 4 - Fine tuning (bonus task)

Once you’ve got all the basics sorted, there are a few small adjustments you could make to further refine this game…

* How close do you have to be to swat the ball at the moment? How could you adjust the collision detection to make it a bit easier?
* How could you adjust your code so that paddle is centred on the tip of the mouse pointer, rather than the top-left corner?

# Double Buffering

*Note 8/8/19: This approach used to work back when we were working with Applets. I haven't had a chance to test this with the current approach, so you may need to mess around with a bit and/or do some research of your own to get it working.*

To avoid a possible problem with flickering, you can use something called Double Buffering.

In the top of the class, paste the follow lines of code:

**private** Image dbImage; //double buffering

**private** Graphics dbg; //double buffering

After the init() method, paste in the following method:

//Double Buffering Method

**public** **void** update (Graphics g)

{

**if** (dbImage == **null**)

{

dbImage = createImage (**this**.getSize().width, **this**.getSize().height);

dbg = dbImage.getGraphics();

}

dbg.setColor (getBackground ());

dbg.fillRect (0, 0, **this**.getSize().width, **this**.getSize().height);

dbg.setColor (getForeground());

paint(dbg);

g.drawImage(dbImage, 0, 0, **this**);

}