|  |  |
| --- | --- |
|  | |
| Bouncing Ball Applet | |
| **Updated** | **10.28.2018 2:44 PM** |

# **Problem 1:**

Write an applet that bounces a ball inside its graphics area. The ball must start in a random location, move in a random direction, and must bounce when it reaches the edge of the graphics area. (This does not require threads).

# **Approach**

The approach I took to the question was to firstly create a ball class which would allow me to then create a Ball object within the Applet. I decided to use java.util.Random generator a random location, move in a random direction variables. I then initialized a ball object. For the animation I used a timer to run at scheduled intervals. The movement of the ball was calculated using delta of x and y for next position and repaint the screen. I used a buffer to and update to reduce the screen flicker.

NOTE: I encountered some issues with having to resize the scree before the ball would appear correctly and I was unable to test in the browser of my laptop for some reason so I hope the HTML is OK

# **OneBall Design**

Create a Ball object

initiate a timer for animation

setup a Random generator

Set the initial horizontal position of the ball.

Set the initial vertical position of the ball.

The ball moves dx pixels vertically

The ball moves dy pixels vertically

Set the radius of the ball

Set the color of the ball

Create a new ball object

Create timer

Run at scheduled time interval

Move Ball

Calculate if bounce ball off the walls

Create off-screen image buffer

Clear background

Draw Ball

Copy the buffer to the screen

Repaint the window background

# **Testing**

The program was run several times with different inputs to ensure the code behaves as expected for each possible execution scenario including the input of invalid and out of bound parameters.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Test** | **Expected Result** | **Actual Result** |
| 1 | Run Applet in Eclipse | Applet runs in Eclipse | **Pass** |
| 2 | Run Applet in browser | Applet runs in Browser | **Fail – environmental issues** |
| 3 | One ball is created in applet | Ball is created | **Pass** |
| 4 | The ball bounces off the boundary walls | Ball bounces off the boundary walls | **Pass** |
|  |  |  |  |

# **Source Code – OneBall.java**

|  |
| --- |
| /\* mmcmahon\_wk7\_Assignment\_OneBall.java  \* Write an Applet that bounces a ball inside its graphics area.  \* <html>  \* <applet code = "OneBall.class" width = "400" height = "400">  \* </applet>  \* </html>  \*/  import java.util.\*;  import java.awt.\*;  import java.applet.\*;  import java.util.Random;  public class OneBall extends Applet  {  private static final long serialVersionUID = -1623236972156601802L;  ABall ball;  Timer timer; //timer for animation  TimerTask task;  Image image; //  Graphics offscreen; //Graphics offscreen  private final Random generator = new Random();  Dimension d; //Window dimensions    @Override  public void init()  {  int x = generator.nextInt(400); // Set the initial horizontal position of the ball.  int y = generator.nextInt(400); // Set the initial vertical position of the ball.  int dx = generator.nextInt(5)+1; // The ball moves dx pixels vertically  int dy = generator.nextInt(5)+1; // The ball moves dy pixels vertically  int radius = 15; // Set the radius of the ball  Color color = Color.red; //Set the color of the ball  ball=new ABall(x,y,radius,dx,dy,color); //Create a new ball object  timer= new Timer(); //create timer  }  public void start()  {  //timer run() every timer.schedule interval  task= new TimerTask() {  public void run() {  move();  }  };  // timer start and run every .10 secs  timer.schedule( task , 0, 10);  }  public void move()  {    // bounce ball off the walls    if ((ball.getX() < ball.getRadius()) || (ball.getX() > 400 - ball.getRadius())){  ball.setDX(ball.getDX()\*-1);  }  if ((ball.getY() < ball.getRadius()) || (ball.getY() > 400 - ball.getRadius())){  ball.setDY(ball.getDY()\*-1);  }    //move ball  ball.setX(ball.getX()+ball.getDX());  ball.setY(ball.getY()+ball.getDY());    repaint();  }  @Override  public void update(Graphics g)  {  super.paint(g);  // create off-screen image buffer  if (image == null) {  image = createImage(400, 400);  offscreen = image.getGraphics();  }  //clear background  offscreen.setColor(Color.BLACK);  offscreen.fillRect(0,0,400, 400);  //draw ball  offscreen.setColor(ball.getBallColor());  offscreen.fillOval(ball.getX()-ball.getRadius(), ball.getY()-ball.getRadius(), ball.getRadius()\*2, ball.getRadius()\*2);  // copy the off-screen image to the screen  g.drawImage(image, 0, 0, this);  }    //repaint the window background  public void paint(Graphics g) {  update(g);  }  } |

# **Problem 2:**

Write a second applet that bounces multiple balls; the number is specified by a parameter in the HTML file. Assign different colours to different balls (you can have 10 unique colours and re-use colours after that). The balls can move independently, passing through each other. Each should have its own thread.

# **Approach**

I use the OneBall.java file as the template and again utilized the ABall.java class. I created a Ball array to hold the balls and created threads to hold each ball. A second array is created to hold the 10 ball colours. These are implement using several loops.

NOTE: I had some issues running applet in my browser on the laptop I had this weekend so hope it works OK – Hard coded in default of numBalls = 10;

# **MultiBall Design**

Declare the thread

remove number when adding HTML

initialize an array for the balls

create an array of ball colours

counter for the colors used

setup a Random generator

New Ball object

New Ball colour

Loop until set Number of balls

Set the initial horizontal position of the ball.

Set the initial vertical position of the ball.

The ball moves dx pixels vertically

The ball moves dy pixels vertically

Set the radius of the ball

Set the color of the ball

Create ball

increase counter

Start Thread

Move Ball

Calculate if bounce ball off the walls

Thread Sleep

Repaint

Create off-screen image buffer

Clear background

Draw Ball

Copy the buffer to the screen

Repaint the window background

# **Testing**

The program was run several times with different inputs to ensure the code behaves as expected for each possible execution scenario including the input of invalid and out of bound parameters.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Test** | **Expected Result** | **Actual Result** |
| 1 | Run Applet in Eclipse | Applet runs in Eclipse | **Pass** |
| 2 | Run Applet in browser | Applet runs in Browser | **Fail – environmental issues** |
| 3 | multiple balls are created in applet | Balls are created | **Pass** |
| 4 | The balls bounces off the boundary walls | Balls bounces off the boundary walls | **Pass** |
| 5 | The balls are created in different colours | Balls are created in different colours | **Pass** |
| 6 | The balls can move independently | Balls can move independently | **Pass** |
| 7 | The balls can pass through each other | Balls can pass through each other | **Pass** |

# **Source Code – MultiBall.java**

|  |
| --- |
| /\* mmcmahon\_wk7\_Assignment\_MultiBall.java  \*Applet bounces multiple balls as specified by a parameter in the HTML file and utilizes multiple colors.  \* <html>  \* <applet code = "MultiBall.class" width = "400" height = "400">  \* <param name=numBalls value="10">  \* </applet>  \* </html>  \*/  **import** java.awt.\*;  **import** java.applet.\*;  **import** java.util.Random;  **public** **class** MultiBall **extends** Applet **implements** Runnable{  **private** **static** **final** **long** ***serialVersionUID*** = -4483729616618053641L;  Thread thread = **null**; // Declare the thread    **int** numBalls = 10; //remove number when adding HTML    ABall balls[]; //initialize an array for the balls  //create an array of ball colours  **private** Color Colors[] = {Color.***GRAY***, Color.***BLUE***, Color.***ORANGE***, Color.***CYAN***, Color.***WHITE***, Color.***MAGENTA***, Color.***GREEN***, Color.***PINK***, Color.***YELLOW***, Color.***RED***};  Color ballColor[];  **int** c; // counter for the colors used    Dimension d; //Window dimensions  Graphics offscreen;  Image image;    **private** **final** Random generator = **new** Random(); // initilize the random generator  @Override  **public** **void** init() {  balls = **new** ABall[numBalls]; //New Ball  ballColor = **new** Color[numBalls]; // New Ball colour  // Loop to create each ball object up to numBalls  **for** (**int** i = 0; i < numBalls; i++) {  **if** (c >= 10) {  c = 0;  }// end if  **int** x = generator.nextInt(400);  **int** y = generator.nextInt(400);  **int** dx = generator.nextInt(5)+1;  **int** dy = generator.nextInt(5)+1;  **int** radius = 15;  Color color = Colors[c];  balls[i] = **new** ABall(x,y,radius,dx,dy,color);  c++;  }// end for  }  **public** **void** start(){  **if**(thread == **null**){  thread = **new** Thread(**this**);  thread.start();  }// end if  }// end method start    @Override  **public** **void** run(){  **while**(**true**){  **for** (**int** j=0; j<numBalls; j++){  **if** (balls[j].getX() < balls[j].getRadius() || balls[j].getX() > 400- balls[j].getRadius()) {  balls[j].setDX(balls[j].getDX()\*-1);  }  **if** (balls[j].getY() < balls[j].getRadius() || balls[j].getY() > 400 - balls[j].getRadius()) {  balls[j].setDY(balls[j].getDY()\*-1);  }  //move ball  balls[j].setX(balls[j].getX()+balls[j].getDX());  balls[j].setY(balls[j].getY()+balls[j].getDY());  }// end for  **try**{  Thread.*sleep*(20);  }// end try  **catch**(InterruptedException except){  }// end catch  repaint();  }// end while  }// end method run  @Override  **public** **void** update(Graphics g){  // create off-screen image buffer  **if** (image == **null**) {  image = createImage(400, 400);  offscreen = image.getGraphics();  }  //clear background  offscreen.setColor(Color.***BLACK***);  offscreen.fillRect(0,0,400,400);  **for** (**int** k=0; k<numBalls; k++){  //draw ball  offscreen.setColor(balls[k].getBallColor());  offscreen.fillOval(balls[k].getX()-balls[k].getRadius(), balls[k].getY()-balls[k].getRadius(), balls[k].getRadius()\*2, balls[k].getRadius()\*2);  g.drawImage(image, 0, 0, **this**);  }// end for  }// end method paint  // repaint the window background  @Override  **public** **void** paint(Graphics g) {  **super**.paint(g);  update(g);  }  } |

# **Problem 3:**

Write a third applet, this time with code so that the balls bounce off each other. A collision will occur between two balls when the distance between their centres is less than the sum of their radii. (Note that this will require synchronization between threads.)

# **Approach**

Unfortunately, I completely ran out of time while working on this third applet so I had to rely somewhat on a working template (based on: Java by Example - a Bouncing Balls applet. (2016). <http://ftp://ftp.sdu.edu.tr/pub/java/javatutor/j_tutor18.html>) which I adapted to my needs instead of building entirely from my previous templates. Consequently, this is applet is a work in progress.

* Does not use my ABall Class
* Does not dynamically create balls – hard coded several balls
* Does not dynamically add colours

# **CollisionBalls Design (Draft)**

Declare the thread

initialize an array for the balls

create an array of ball colours

counter for the colors used

setup a Randon generator

New Ball object

New Ball colour

Loop until set Number of balls

Set the initial horizontal position of the ball.

Set the initial vertical position of the ball.

The ball moves dx pixels vertically

The ball moves dy pixels vertically

Set the radius of the ball

Set the color of the ball

Create ball

increase counter

Start Thread

Move Ball

Calculate if ball bounces off other balls

Calculate if ball bounces off the walls

Thread Sleep

Repaint

Create off-screen image buffer

Clear background

Draw Ball

Copy the buffer to the screen

Repaint the window background

# **Testing**

The program was run several times with different inputs to ensure the code behaves as expected for each possible execution scenario including the input of invalid and out of bound parameters.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Test** | **Expected Result** | **Actual Result** |
| 1 | Run Applet in Eclipse | Applet runs in Eclipse | **Pass** |
| 2 | Run Applet in browser | Applet runs in Browser | **Fail – environmental issues** |
| 3 | multiple balls are created in applet | Balls are created | **Pass** |
| 4 | The balls bounces off the boundary walls | Balls bounces off the boundary walls | **Pass** |
| 5 | The balls are created in different colours | Balls are created in different colours | **Pass** |
| 6 | The balls can move independently | Balls can move independently | **Pass** |
| 7 | The balls bounce off of each other | Balls bounce off of each other | **Pass** |

# **Source Code – CollisionBalls.java**

|  |
| --- |
| /\* mmcmahon\_wk7\_Assignment\_CollisionBalls.java  \*Applet code so that the balls bounce off each other.  \*Applet is based on: Java by Example - a Bouncing Balls applet. (2016).  \*Retrieved 9 October 2016, from http://ftp://ftp.sdu.edu.tr/pub/java/javatutor/j\_tutor18.html  \*  \* <html>  \* <applet code = "CollisionBalls.class" width = "400" height = "400">  \* <param name=numBalls value="5">  \* </applet>  \* </html>  \*/  **import** java.awt.\*;  **import** java.applet.\*;  **import** java.util.Random;  **class** CollideBall  {  **int** width, height;  **public** **static** **final** **int** ***diameter***=30;  **double** x, y, dx, dy, coll\_x, coll\_y;  **boolean** collide;  Color color;  Graphics g;  //the constructor  **public** CollideBall(**int** w, **int** h, **int** x, **int** y, **double** dx, **double** dy, Color c)  {  width=w;  height=h;  **this**.x=x;  **this**.y=y;  **this**.dx=dx;  **this**.dy=dy;  color=c;  }  **public** **double** getCenterX() {**return** x+***diameter***/2;}  **public** **double** getCenterY() {**return** y+***diameter***/2;}  **public** **void** move()  {  **if** (collide)  {  **double** xvect=coll\_x-getCenterX();  **double** yvect=coll\_y-getCenterY();  **if**((dx>0 && xvect>0) || (dx<0 && xvect<0))  dx=-dx;  **if**((dy>0 && yvect>0) || (dy<0 && yvect<0))  dy=-dy;  collide=**false**;  }  x+=dx;  y+=dy;  // bounce ball off the boundry walls  **if**(x<0 || x>width-***diameter***)  {  dx=-dx;  x+=dx;  }  **if**(y<0 || y>height-***diameter***)  {  dy=-dy;  y+=dy;  }  }  **public** **void** hit(CollideBall b)  {  **if**(!collide)  {  coll\_x=b.getCenterX();  coll\_y=b.getCenterY();  collide=**true**;  }  }  **public** **void** paint(Graphics gr)  {  g=gr;  g.setColor(color);  g.fillOval((**int**)x,(**int**)y,***diameter***,***diameter***);  }  }  **public** **class** CollisionBalls **extends** Applet **implements** Runnable{  **private** **static** **final** **long** ***serialVersionUID*** = 491852199048520152L;  Thread thread;  Image image;  Graphics offscreen;    **int** numBalls = 10; //remove number when adding HTML  ABall balls[];    **private** Color Colors[] = {Color.***GRAY***, Color.***BLUE***, Color.***ORANGE***, Color.***CYAN***, Color.***WHITE***, Color.***MAGENTA***, Color.***GREEN***, Color.***PINK***, Color.***YELLOW***, Color.***RED***};  Color ballColor[];  **int** c; // count the colors used  CollideBall ball[];  //how many balls?  **static** **final** **int** ***MAX***=5;  @Override  **public** **void** init()  {  image=createImage(getWidth(),getHeight());  offscreen=image.getGraphics();  ball=**new** CollideBall[***MAX***];  **int** w=getWidth();  **int** h=getHeight();  //shortcut balls with start coordinates, increment values  ball[0]=**new** CollideBall(w,h,50,20,1.5,2.0,Color.***orange***);  ball[1]=**new** CollideBall(w,h,60,100,2.0,-3.0,Color.***red***);  ball[2]=**new** CollideBall(w,h,15,70,-2.0,-2.5,Color.***pink***);  ball[3]=**new** CollideBall(w,h,150,60,-2.7,-2.0,Color.***cyan***);  ball[4]=**new** CollideBall(w,h,210,30,2.2,-3.5,Color.***magenta***);  }  @Override  **public** **void** start()  {  **if** (thread == **null**)  {  thread = **new** Thread (**this**);  thread.start();  }  }  @Override  **public** **void** run()  {  **while**(**true**)  {  //Thread sleeps for 20 milliseconds  **try** {Thread.*sleep*(20);}  **catch** (Exception e) { }  **for**(**int** i=0;i<***MAX***;i++)  ball[i].move();  handleCollision();  repaint();  }  }  **boolean** collide(CollideBall b1, CollideBall b2)  {  **double** wx=b1.getCenterX()-b2.getCenterX();  **double** wy=b1.getCenterY()-b2.getCenterY();  //the distance between the centers two colliding balls  **double** distance=Math.*sqrt*(wx\*wx+wy\*wy);  **if**(distance<CollideBall.***diameter***)  {  **return** **true**;  }  **return** **false**;  }  **private** **void** handleCollision()  {  //we iterate through all the balls, checking for collisions  **for**(**int** i=0;i<***MAX***;i++)  **for**(**int** j=0;j<***MAX***;j++)  {  **if**(i!=j)  {  **if**(collide(ball[i], ball[j]))  {  ball[i].hit(ball[j]);  ball[j].hit(ball[i]);  }  }  }  }  @Override  **public** **void** update(Graphics g)  {  paint(g);  }  @Override  **public** **void** paint(Graphics g)  {  //draw background  offscreen.setColor(Color.***BLACK***);  offscreen.fillRect(0,0,getWidth(),getHeight());  //paint the balls  **for**(**int** i=0;i<***MAX***;i++)  ball[i].paint(offscreen);  g.drawImage (image,0,0, **this**);  }  } |

# **References:**

*66.Java Intro -Chapter09.Introduction to Applets-Lesson6.One Bouncing Ball.*. (2016). *YouTube*. Retrieved 9 October 2016, from <https://www.youtube.com/watch?v=WZ7_qTYnYzY>

*Animating applets using threads*. (2016). *Di.ase.md*. Retrieved 9 October 2016, from <http://www.di.ase.md/~aursu/AppletsAnimationThreads.html>

*Applet, Adding Multiple Bouncing Ball(s) & Making Them Bounce Off - Java | Dream.In.Code*. (2016). *Dreamincode.net*. Retrieved 9 October 2016, from <http://www.dreamincode.net/forums/topic/230450-applet-adding-multiple-bouncing-balls-making-them-bounce-off-each/>

Bodnar, J. (2016). *Java games collision detection*. *Zetcode.com*. Retrieved 9 October 2016, from <http://zetcode.com/tutorials/javagamestutorial/collision/>

*Bouncing ball animation in Applet*. (2010). *All about Web and Cloud*. Retrieved 9 October 2016, from <http://www.jitendrazaa.com/blog/java/mouse-handling-in-applets-using-this/>

*Bouncy Ball Using Applet - Applet Tutorials*. (2016). *YouTube*. Retrieved 9 October 2016, from <https://www.youtube.com/watch?v=TAJh8zAjR5Q>

*Help rid ball trailing colors in balls bounce*. (2009). *Forums.devx.com*. Retrieved 9 October 2016, from <http://forums.devx.com/showthread.php?171933-Help-rid-ball-trailing-colors-in-balls-bounce>

*Java by Example - a Bouncing Balls applet*. (2016). Retrieved 9 October 2016, from <http://ftp://ftp.sdu.edu.tr/pub/java/javatutor/j_tutor18.html>

*MovingBall.java*. (2016). *Math.hws.edu*. Retrieved 9 October 2016, from <http://math.hws.edu/eck/cs124/javanotes3/source/MovingBall.java>

*Putting Math In The Context Of Game Programming Part V- 2D Collisions - Game Programming Tutorials | Dream.In.Code*. (2016). *Dreamincode.net*. Retrieved 9 October 2016, from <http://www.dreamincode.net/forums/topic/229613-putting-math-in-the-context-of-game-programming-part-v-2d-collisions/>

*The world of Bouncing Balls - An introduction to Java Game Programming*. (2016). *Ntu.edu.sg*. Retrieved 9 October 2016, from <http://www.ntu.edu.sg/home/ehchua/programming/java/j8a_gameintro-bouncingballs.html>