Applet Tutorial – Simple Timer & Movement

- In our first applet tutorial, you made a scene with a house and some other elements
- Now we are going to introduce a simple timer so that we can make our objects move around the screen

Task 1: Create a New Project (AppletAnimation)

- Create a new project called "Last, First – AppletAnimation
- Use this one project for each class in this tutorial

Task 2: Create a Class (Stationary)

- Within your project 'AppletAnimation' create a new class and call it "Stationary"
- You will need these three import statements above the `public class` statement

```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
```

- Now edit the class declaration to include the words `extends Applet implements Runnable`

```
public class Stationary extends Applet implements Runnable{
```

- In order for our applet to do something, we will need the `init()` and `paint()` method that we used previously. They belong inside the class open and close brackets {}

```
public void init()
	{

	}

public void paint(Graphics g)
	{

	}
```

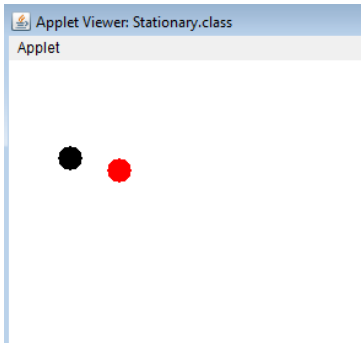
- Now that we want to add animation, we also need to include a method called `run()`

```
 public void run()
{
}
```

- Run your class at this point to make sure that it opens a small applet window without crashing!

- Now it is time to add in a red dot and a black dot. We will start them off at (80,80) and (40,70). Add the following code to your pain method:

```
public void paint(Graphics g)
{

    this.setSize(500,500);
    //Draw the red ball at (80,80) with radius 20
    g.setColor(Color.RED);
    g.fillOval(80, 80, 20, 20);

    //Draw the black ball at (40,70) with radius 20
    g.setColor(Color.BLACK);
    g.fillOval(40, 70, 20, 20);



}
```



- Run your program at this step to make sure the two dots show up.

- In order to eventually make the dots move around the screen, we will need to be able to change their x- and y-coordinates. Instead of "hard coding" the coordinates as (80,80) and (40,70) we are going to use variables (redX, redY) for the red dot, and (blackX, blackY) for the black dot.

- Make four integer variables – these are called **GLOBAL VARIABLES** because they are not inside a method. Global variables are typically considered poor programming, but for our timer to work properly we will accept them as necessary in this case.

```
//Global variables to hold the x and y coordinates of the balls
int redX, redY, blackX, blackY;
//Global variables to hold each radius;
int redRadius, blackRadius;
```

Global variables belong above all methods – directly underneath the class declaration

- Now that we have variables set aside, we will initialize (set their initial `init()` method like this:

```
public void init()
{
    //initial values for x,y coordinates
    redX = 80;
    redY = 80;

    blackX = 40;
    blackY = 70;

    //initial values for radius
    redRadius = 20;
    blackRadius = 20;

}
```

- Finally we will change the values of the arguments for `g.fillOval` from hard coded numbers to our new variables.

```
g.fillOval(redX, redY, redRadius, redRadius);
g.fillOval(blackX, blackY, blackRadius, blackRadius);
```

- Now run this program, which admittedly will be pretty boring with two dots that don't yet move!

Task 2: Create a 2nd Class in the same project (LeftAndRight)

- Rather than start from scratch, you can make a copy of your "Stationary" class by right clicking it in the package explorer and choosing copy as shown in Image1
    - Paste the copy into your project by right clicking on the "src" folder and choosing Paste (Image2). Rename the class to "LeftAndRight" when prompted.

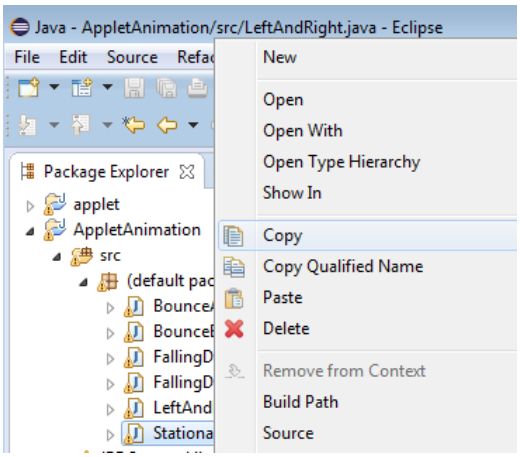


Image1

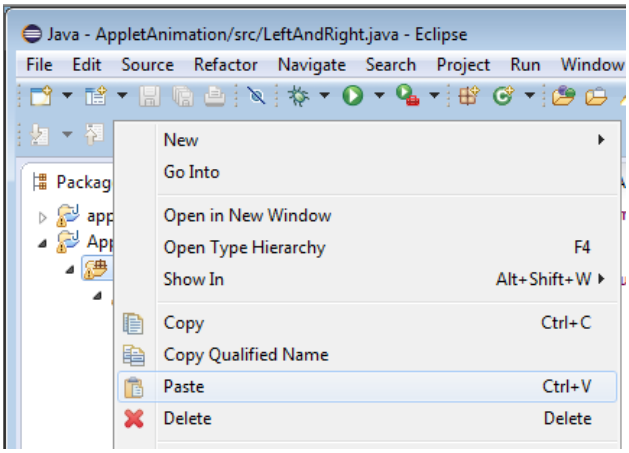


Image2

A) Start a timer – at the very bottom of your `init()` method, add this code to start a timer

```
//necessary for animation - start the timer
Thread t=new Thread(this);
t.start();
```

B) Run the timer – edit your run() method to look like this:

```
// repaint every time the timer ticks
 public void run()
 {
     while(true) // always
       {
           try
           {


              Thread.sleep(20);
              repaint();
           }
           catch(Exception e){}
       }
 }
```

- We want to actually see that the timer is ticking and the scene is actually redrawing, so we will add a print statement at the end of your paint method.

```
System.out.println("Timer is running!");
```

- Run your program and see that the timer is actually working!

C) Change the positions of the dots – Finally!!

- Add this code to move the dots to the right, place it as shown in the `run()` method. (The 1 and 2 represent how far we want the dots to move each tick, so the black dot will move faster)

```
//move the dots
redX = redX + 1;
blackY = blackY + 2;

Thread.sleep(20);
repaint();
```

- Let's keep track of the coordinates of each dot by changing our print statement to:

```
System.out.println("Red ("+ redX + ", "+ redY + ")");
System.out.println("Black ("+ blackX + ", "+ blackY + ")");
```

- Run your program to check out how it works!

D) Now let's make the dots bounce when they hit the walls!

- In order to change direction, we need to add a value to the x-coordinate to move right, and subtract to move left. This is kind of complicated, so we will need to make two new variables to keep track of the velocity of each dot (POSITIVE velocity will be movement to the right or down, and NEGATIVE velocity will be movement to the left or up).
- Add these new global variables (up at the top with the other global variables

```
int redVelX, redVelY;
int blackVelX, blackVelY;
```

- And initialize them in your init method (again, we will make the black dot move faster)

```
//set the velocities of the two dots
redVelX = 1;
blackVelY = 2;
```

- Now change the hard coded movement to reflect our new variables

```
try
{
      //move the dots
      redX = redX + redVelX;
      blackY = blackY + blackVelY;

      Thread.sleep(20);
      repaint();
}
catch(Exception  e){}
```

- Run your program – should work the same as before!

- Now we are going to deal with hitting the walls! If the hit a wall, multiple the velocity by -1 to make the direction change

```
try
 {
      //move the dots
      redX = redX + redVelX;
      blackY = blackY + blackVelY;

      // did they hit a wall?
      // if so, change their direction

      if ((redX >= 500)||(redX<=0)){
            redVelX = redVelX * -1;
      }

      if ((blackY >= 500)|| (blackY <= 0)) {
            blackVelY = blackVelY * -1;
      }

      Thread.sleep(20);
      repaint();
 }
catch(Exception  e){}
```

Programming Exercises to Try:

1) Make a new class in the same project and call it BouncingDots. These dots will move diagonally until they hit the walls, and then change direction.

- You will need both x- and y-velocities for each dot

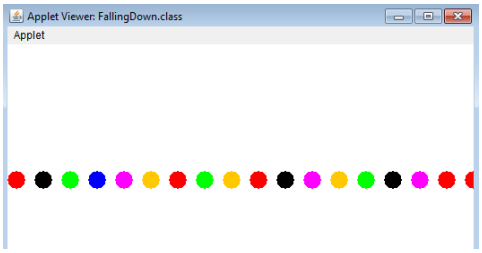
```
//set the velocities of the two balls
redVelX = 2;
redVelY = 1;
blackVelX = -1;
blackVelY = -2;
```

- You will need to check all four walls:

```
// did they hit a wall?
// if so, change their direction

if ((redX >= 500)||(redX<=0)){
    redVelX = redVelX * -1;

}
if ((redY >= 500)|| (redY <= 0)) {
    redVelY = redVelY * -1;

}

if ((blackX >= 500)||(blackX<=0)){
    blackVelX = blackVelX * -1;

}
if ((blackY >= 500)|| (blackY <= 0)) {
    blackVelY = blackVelY * -1;

}
```

2) Make a new class in the same project and call it ManyDots. Add at least 5 more dots with many colours and different sizes, and have them bounce around.

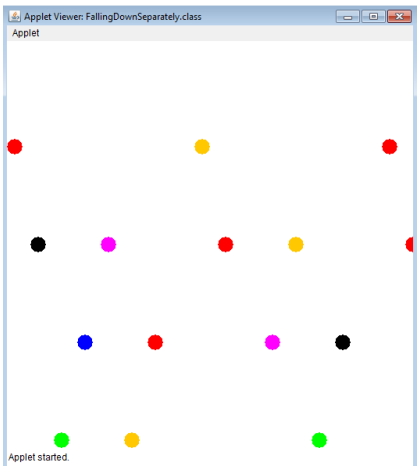
3) Make a new class in the same project and call it FallingDots. It should have many different coloured dots along the top of the page, and they all will fall together down to the bottom.



Hints: draw many different ovals with offset x coordinates like this (x and y are global integers that change on the timer):

```
g.setColor(Color.RED);
g.fillOval(x,y,radius, radius);
g.setColor(Color.BLACK);
g.fillOval(x+30,y,radius, radius);
g.setColor(Color.GREEN);
g.fillOval(x+60,y,radius, radius);
g.setColor(Color.BLUE);
g.fillOval(x+90,y,radius, radius);
g.setColor(Color.MAGENTA);
```

4) Make a new class in the same project and call it FallingDifferentSpeeds. It should have many dots that fall at different speeds down to the bottom.



Hints: Have a few different y-coordinate options and move them at different speeds

```
g.fillOval(x,y,radius, radius);
g.setColor(Color.BLACK);
g.fillOval(x+30,a,radius, radius);
g.setColor(Color.GREEN);
g.fillOval(x+60,c,radius, radius);
g.setColor(Color.BLUE);
g.fillOval(x+90,b,radius, radius);
g.setColor(Color.MAGENTA);
g.fillOval(x+120,a,radius, radius);
```

```
try
   {
    //move the dots down with a few different speed options
    y++;
    a+=2;
    b+=3;
    c+=4;
    d+=5;

      Thread.sleep(20);
      repaint();
   }
```