# Lecture 36

- Covers
  - Event handling
  - Mouse events
  - Animation with the Timer class (extension)

# Event-driven programming

- Event-driven programming (i.e. writing programs to respond to events) is quite different from the kind of programming that we have done so far
- We will first describe Java's approach to event-driven programming, and then illustrate it with mouse events

# Events and event handling

- Certain actions performed by a user can generate events
- For example, when you click the mouse on an applet's screen, the system generates an event
- Such events can be ignored or responded to by the program
- The process of responding to events is known as event handling

# Java's delegation approach

- Java's approach to event handling is based on what is known as the *delegation model*
- In this approach, associated with each event are the event source and an event listener
- The *event source* is the object (e.g. an applet) on which the event is generated
- The *event listener* is the object that responds to the event

# Events

- The event source must *register* the event listener in order to *delegate* the task of handling the event to the listener
- The event listener is an instance of a class that implements a *listener interface*
- In Java, the term *interface* is also used to denote a collection of abstract methods to capture a set of behaviours
- When we want to use these methods in a class, the class must *implement* this interface and *define* versions of these abstract methods

# Events

- Upon the occurrence of an event, an Event object, which contains information about the event, is generated and passed to the event listener
- The event listener can use the information in the event object to handle the event

# Events

- Mouse events (clicked, pressed, released, enter a component, leaving a component) are processed by MouseListener objects
- A MouseListener object is an instance of a class that implements the MouseListener *interface*

# MouseListener

- The MouseListener interface has the following methods

```
public void mouseClicked(MouseEvent event)
public void mousePressed(MouseEvent event)
public void mouseReleased(MouseEvent event)
public void mouseEntered(MouseEvent event)
public void mouseExited(MouseEvent event)
```

# MouseEvent

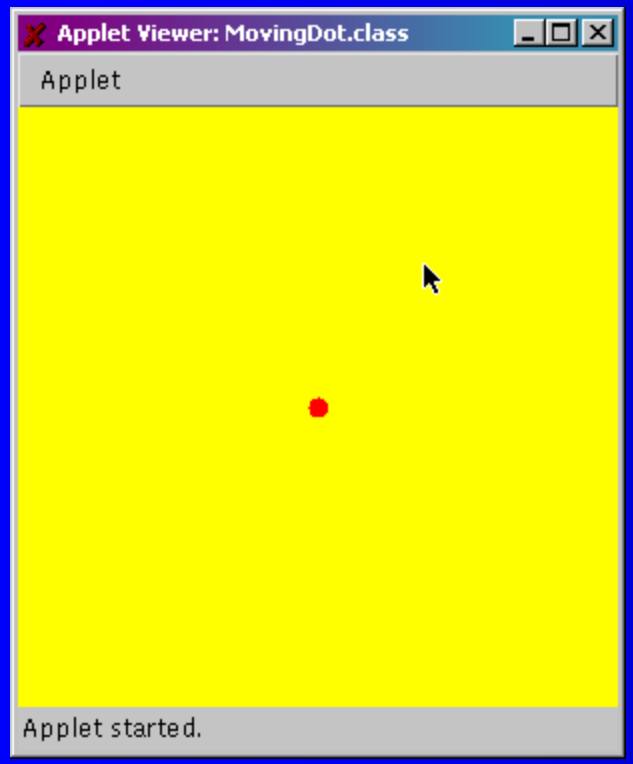
- The MouseEvent class has a number of methods
- The most commonly used are

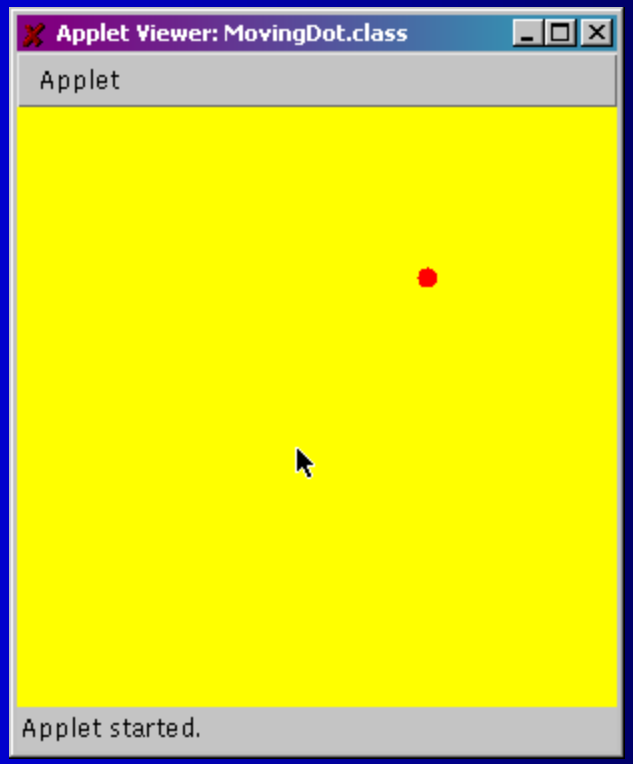
```
public int getX( )
public int getY( )
```

- getX( ) and getY( ) return the x- and y-coordinates of the mouse's position when the event occurs

# Moving dot example

- An applet with a red dot at the centre initially
- When we click the mouse on the applet, the red dot will move to where the mouse is



# The design

- In this program, the applet is the source of mouse events
- What about the mouse listener?
- Basically, we have two choices
  - We can create a new class to be the mouse listener
  - Or, we can let the applet itself be the mouse listener
- Let's choose the second option, i.e. the applet itself will implement the listener interface

# Moving dot example code

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class MovingDot extends Applet implements MouseListener
{
    private int x, y;                    // the dot's coordinates

    public void init( )
    {
        setSize(300, 300);
        setBackground(Color.yellow);
        setForeground(Color.red);
        x = 150;
        y = 150;
        addMouseListener(this);  // register itself as the mouse listener
    }
```

# Moving dot example code

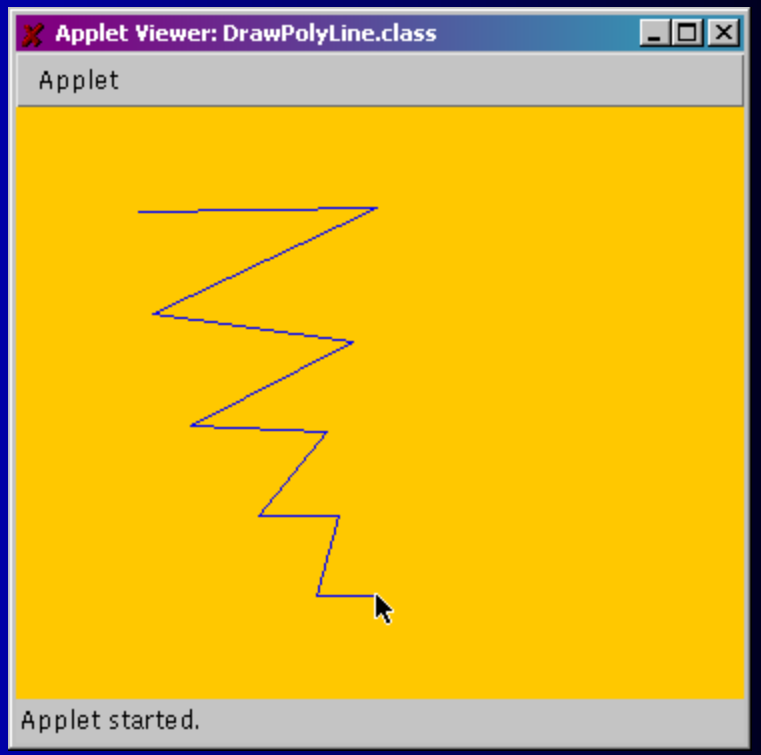
```
public void paint(Graphics g)
{
    // display the dot
    g.fillOval(x-5, y-5, 10, 10);
}

// implements the mouse listener interface to respond to
// mouse clicked event
public void mouseClicked(MouseEvent event)
{
    // set x, y to the mouse's position
    x = event.getX( );
    y = event.getY( );

    // call repaint to redisplay the applet (repaint will call paint)
    repaint( );
}
```

# Moving dot example code

```
    public void mousePressed(MouseEvent event)  { }
    public void mouseReleased(MouseEvent event)  { }
    public void mouseEntered(MouseEvent event)  { }
    public void mouseExited(MouseEvent event)  { }
}
```

# Polyline example

- Write an applet that draws lines from the place the mouse was last clicked to the new mouse click location
- A polyline is drawn



# Polyline example code

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class DrawPolyLine extends Applet implements MouseListener
{
    private int[ ] xCoords;
    private int[ ] yCoords;
    private int numberPoints;

    public void init( )
    {
        setSize(300, 300);
        xCoords = new int[1000];
        yCoords = new int[1000];
        numberPoints = 0;
        setBackground(Color.orange);
        setForeground(Color.blue);
        addMouseListener(this);  // register itself as the mouse listener
    }
```

# Polyline example code

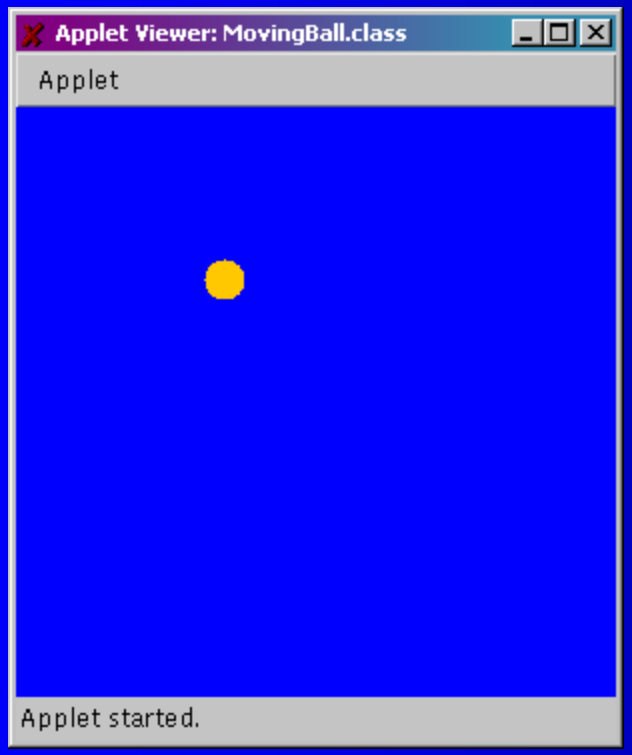
```
public void paint(Graphics g)
{
    g.drawPolyline(xCoords, yCoords, numberPoints);
}

public void mouseClicked(MouseEvent event)
{
    xCoords[numberPoints] = event.getX( );
    yCoords[numberPoints] = event.getY( );
    ++numberPoints;
    repaint( );
}
public void mousePressed(MouseEvent event) { }
public void mouseReleased(MouseEvent event) { }
public void mouseEntered(MouseEvent event) { }
public void mouseExited(MouseEvent event) { }
}
```

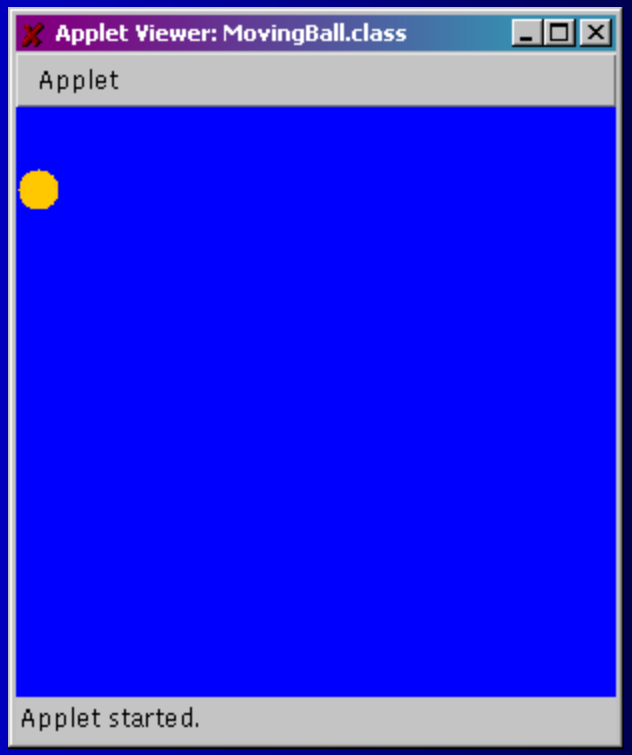
# Animations

- Sequence of images/drawings
- By changing the image frequently, it appears that the image is moving
- Java has a Timer class that creates regular events (every specified number of milliseconds)
- We can implement an ActionListener to respond to those events

# Animation example

- Write an applet in which a ball moves around the screen, changing direction when it comes to an edge



# Animation example

```
public class MovingBall extends Applet implements ActionListener
{
    private int xc;
    private int yc;
    private int size;
    private Timer timer;
    private int xDirection = 3;
    private int yDirection = 3;
    private int delay = 50;

    public void init( )
    {
        xc = 10;
        yc = 10;
        size = 20;
        setBackground(Color.blue);
        setForeground(Color.orange);
        timer = new Timer(delay, this);
    }
```

# Animation example

```
public void start( )
{
    timer.start( );
}

public void stop( )
{
    timer.stop( );
}

public void paint(Graphics g)
{
    g.fillOval(xc, yc, size, size);
}
```

# Animation example

```
public void actionPerformed (ActionEvent event)
{
    xc = xc + xDirection;
    yc = yc + yDirection;

    if (xc <= 0 || xc >= getWidth( ) - size)
    {
        xDirection = -xDirection;
    }
    if (yc <= 0 || yc >= getHeight( ) - size)
    {
        yDirection = -yDirection;
    }
    repaint( );
  }
}
```

# Next lecture

- Traditional Java Console Input