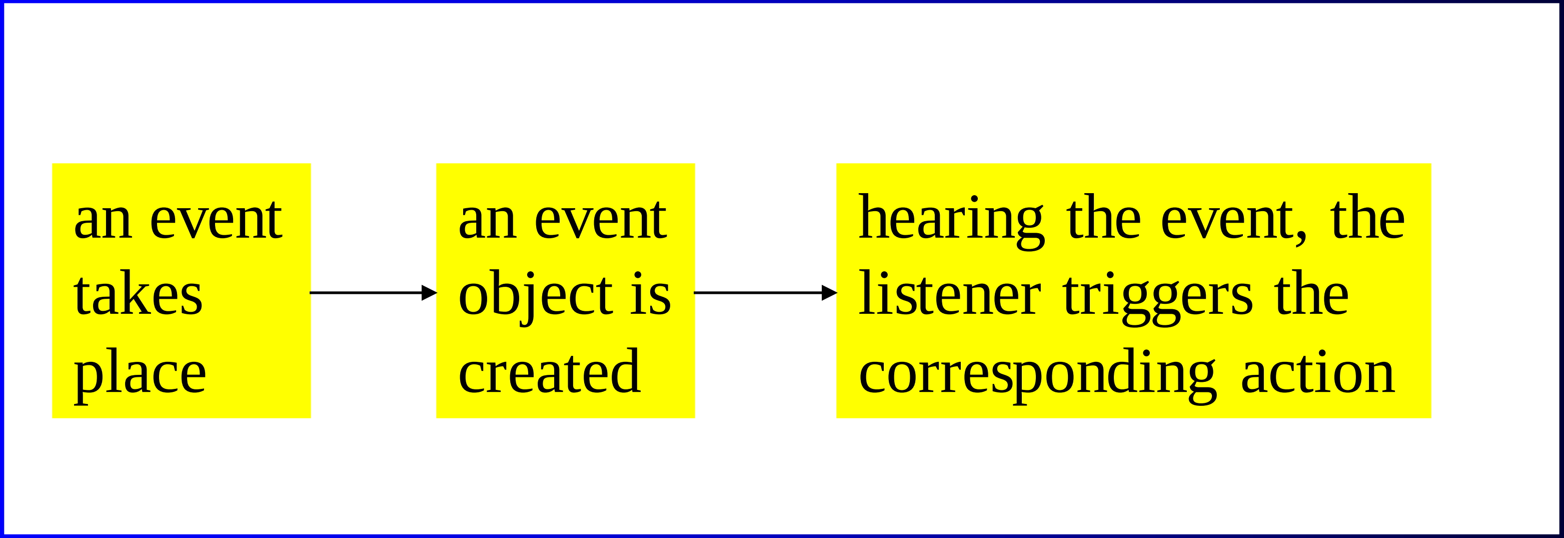
GUI Programming

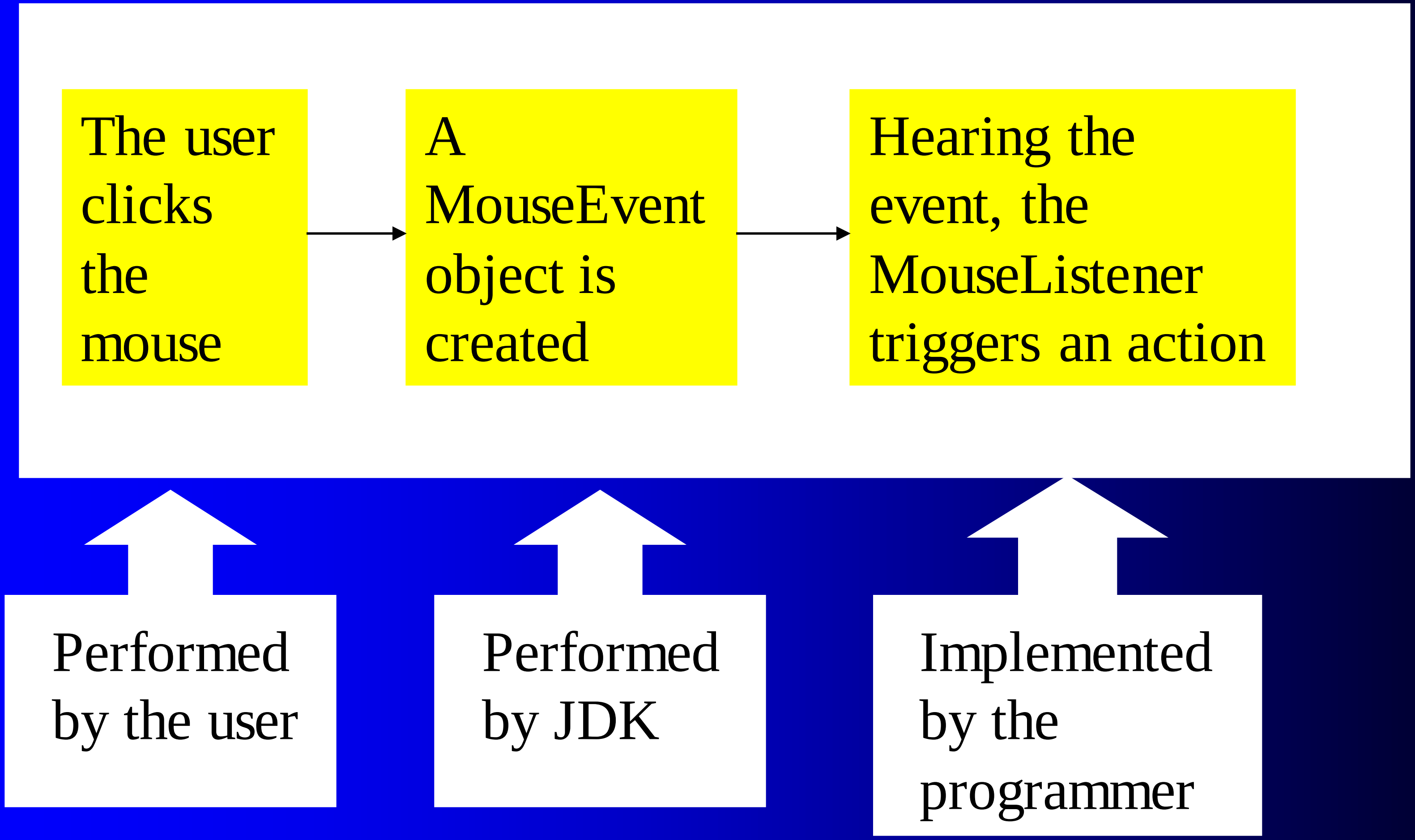
- Reading: Savitch, Chapter 12

Event Driven Programming

- Event driven programming is led by events. Its model is:


an event takes place
an event object is created
hearing the event, the listener triggers the corresponding action

Example


The user clicks the mouse
A MouseEvent object is created
Hearing the event, the MouseListener triggers an action
Performed by the user
Performed by JDK
Implemented by the programmer

- There are different types of events, and therefore different types of listeners.

Example

KeyEvent, MouseEvent, ComponentEvent, WindowEvent etc.

KeyListener, MouseListener, ComponentListener and WindowListener etc.

Example

```
// MouseSpy.java
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
/*The listener prints out the listener method name and
    the x- and y-coordinate of the mouse position.*/
public class MouseSpy implements MouseListener
{
   public void mousePressed(MouseEvent event)
   {
      System.out.println("Mouse pressed. x = "
         + event.getX() + " y = " + event.getY());
   }
```

```
public void mouseReleased(MouseEvent event)
{
  System.out.println("Mouse released. x = "
    + event.getX() + " y = " + event.getY());
}

public void mouseClicked(MouseEvent event)
{
  System.out.println("Mouse clicked. x = "
    + event.getX() + " y = " + event.getY());
}
```

```
    public void mouseEntered(MouseEvent event)
    {
       System.out.println("Mouse entered. x = "
          + event.getX() + " y = " + event.getY());
    }

    public void mouseExited(MouseEvent event)
    {
       System.out.println("Mouse exited. x = "
          + event.getX() + " y = " + event.getY());
    }
 }
```

```
// MouseSpyApplet.java
import java.applet.Applet;
/* This applet installs a mouse spy. Try to generate the
    five mouse event types by moving and clicking the
    mouse.*/
public class MouseSpyApplet extends Applet
{
   public MouseSpyApplet()
   {
      MouseSpy listener = new MouseSpy();
      addMouseListener(listener);
   }
}
```

GUI Tools

- AWT (Abstract Window Toolkit) and Swing are pre-defined library packages in API for GUI programming.

- Swing is developed based on AWT. Most of components in Swing are lightweight (*ie*. they are platform independent.)

- The key elements of a Java graphical user interface are:
  - GUI components
  - layout managers
  - event processing

- GUI components are the screen elements that a user manipulates with the mouse and keyboard.

  For example, text fields, buttons, frames, windows etc.

- <u>Layout managers</u> govern how the components appear on the screen.

- <u>Events</u> signal important user actions.

  For example, a mouse click, a mouse move, a key press etc.

Event Driven Programming Model

- GUI program is event driven programming.

Event Driven Programming Model

<u>Event driven programming always follows this model</u>

(1) Choose the events you want to listen for (the events you want to act upon).

(2) Define your own listener class by extending it either from the listener interface or the adapter class.

(3) In your listener class, overwrite the inherited methods.

(4) Add the listener.

Example

```
//MouseApplet.java
import java.applet.Applet;
import java.awt.*;
Import java.awt.event.*;
/* Let the user move a rectangle by clicking the mouse.*/
public class MouseApplet extends Applet {
   private Rectangle box;
   private static final int BOX_X = 100;
   private static final int BOX_Y = 100;
   private static final int BOX_WIDTH = 20;
   private static final int BOX_HEIGHT = 30;
```

```
public MouseApplet() {
    // the rectangle that the paint method draws
    box = new Rectangle(BOX_X, BOX_Y,
        BOX_WIDTH, BOX_HEIGHT);
    // add mouse press listener
    class MousePressListener implements MouseListener {
        public void mousePressed(MouseEvent event)
        {
            int x = event.getX();
            int y = event.getY();
            box.setLocation(x, y);
            repaint();
        }
```

```
     // do-nothing methods
          public void mouseReleased(MouseEvent event) {}
          public void mouseClicked(MouseEvent event) {}
          public void mouseEntered(MouseEvent event) {}
          public void mouseExited(MouseEvent event) {}
        }
        MouseListener listener = new MousePressListener();
        addMouseListener(listener);
      }
      public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        g2.draw(box);
      }
    }
```

Some notes

- paint() and repaint()

  paint() is called when we want to display the contents of a component on the monitor. repaint() is called when the contents has been modified, and so need to be re-displayed. In our program, we always implement the paint() method, but always call the repaint() method. paint() cannot be called directly by a programmer.

- There is an inner class MousePressListener defined inside the constructor of MouseApplet. This is a common practice in GUI programming – to implement the listener as an inner class.

- The MousePressListener class defines the event the program will listen for and the corresponding action it will take. It only listens to an event. That is mousePressed.
- For any other events (such as mouseReleased, mouseClicked, mouseEntered, mouseExited), it doesn’t care.

- MousePressedListener extends the interface MouseListener, and implements all the methods defined in MouseListener (even though it is only interested in one of the methods, and doesn’t care if the mouse is clicked, moved etc). This is because when a class implements an interface, it has to implement all the abstract methods in the interface.

- MouseListener (defined in API) has the following structure

```
public interface MouseListener {
    public void mousePressed(MouseEvent event);
            //Called when the mouse is pressed.
    public void mouseReleased(MouseEvent event);
    public void mouseClicked(MouseEvent event);
    public void mouseEntered(MouseEvent event);
    public void mouseExited(MouseEvent event);
    }
```

- The MouseAdapter class (defined in API) implements MouseListener, and implements all its methods as empty.

```
public class MouseAdapter implements MouseListener ...{
    ... ...
    public void mousePressed(MouseEvent event) {}
    public void mouseReleased(MouseEvent event) { }
    public void mouseClicked(MouseEvent event) { }
    public void mouseEntered(MouseEvent event) { }
    public void mouseExited(MouseEvent event) { }
}
```

- The benefit of having MouseAdapter is that MouseAdapter is a class, and so when extending from it, we don't have to implement all its methods. Therefore the MousePressListener class can also be implemented as follows.

```
class MousePressListener extends MouseAdapter {
        public void mousePressed(MouseEvent event)
        {
           int x = event.getX();
           int y = event.getY();
           box.setLocation(x, y);
           repaint();
        }
}
```

- Similar to the MouseListener and MouseAdapter pair, there are other interface and class pairs such as KeyListener and KeyAdapter, WindowListener and WindowAdaptor, ContainerListener and ContainerAdapter etc.

Class Exercise

- Modify the previous example, so that the MousePressListener is not implemented as an inner class.