Beispiel: Eine Choice-Liste ...

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class FirstChoice extends Applet
                         implements ItemListener {
   private Color color = Color.white;
   private Choice colorChooser;
   ...
```

```
public void init() {
   setFont(new Font("Serif", Font.PLAIN, 18));
   colorChooser = new Choice();
   colorChooser.setBackground(Color.white);
   colorChooser.add("white");
   colorChooser.add("red");
   colorChooser.add("green");
   colorChooser.add("blue");
   colorChooser.addItemListener(this);
   add(colorChooser);
   setBackground(Color.orange);
}
...
```

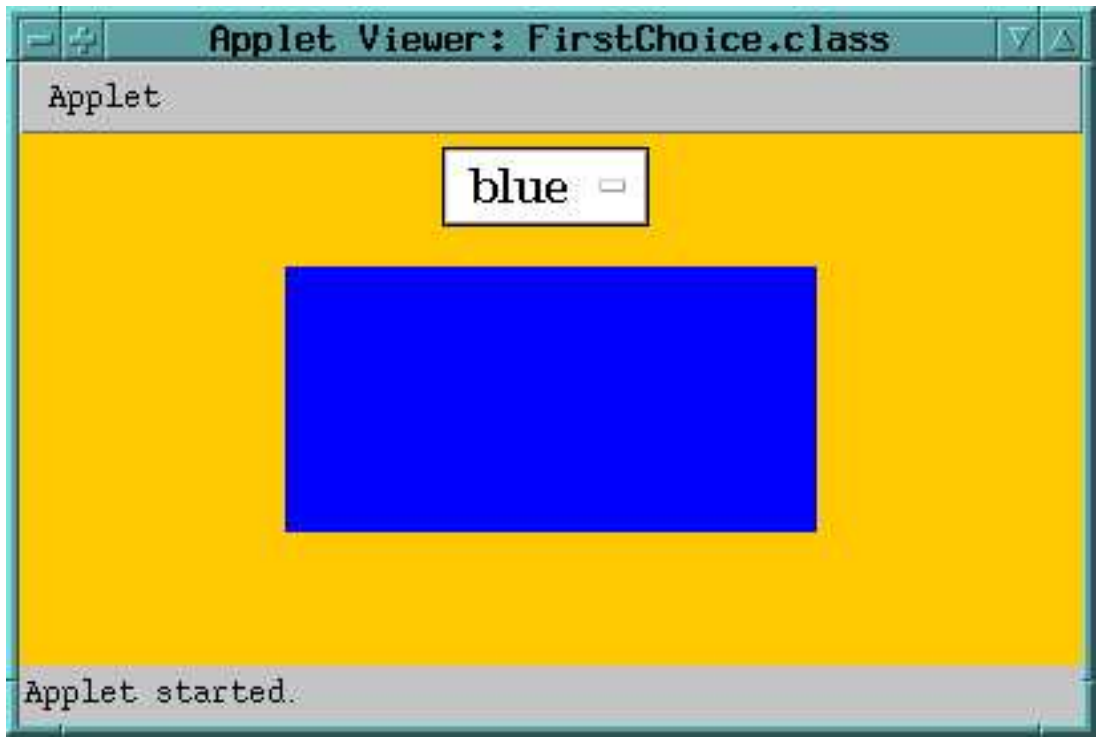
- `public Choice();` legt ein neues `Choice`-Objekt an;
- Zu diesem Objekt können beliebig viele Items hinzugefügt werden. Dazu dient die ObjektMethode:

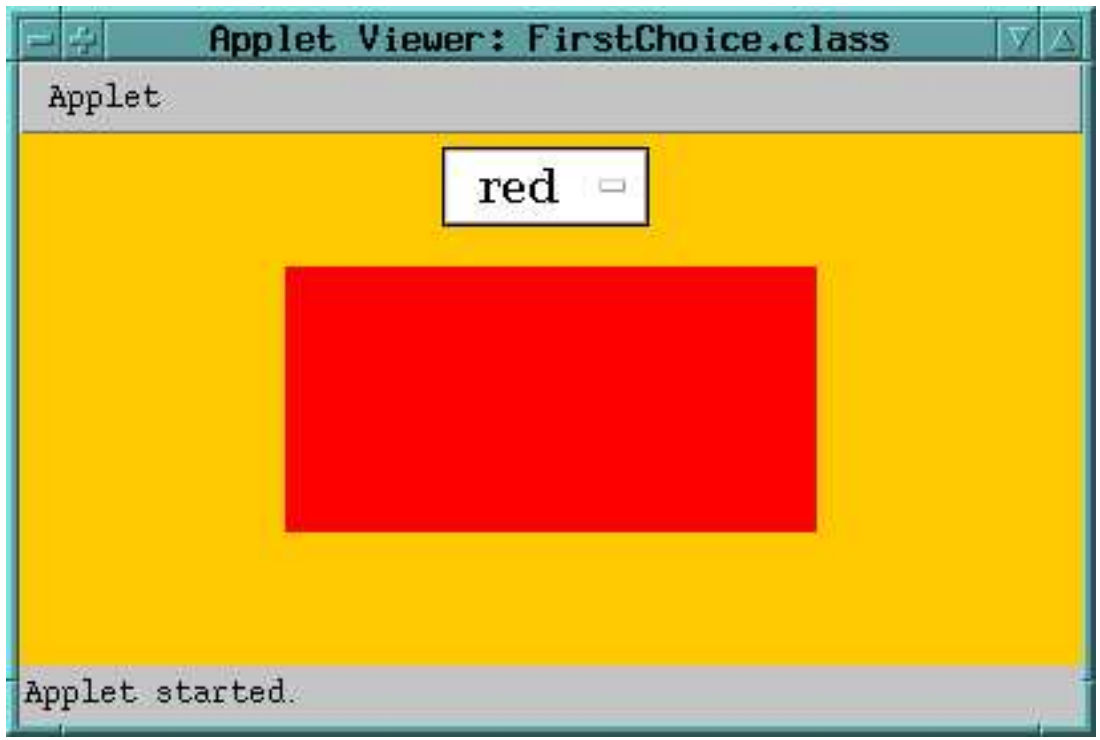
  ```
  public void add(String str);
  ```

- `public void addItemListener(ItemListener listener);` registriert das Objekt `listener` für die erzeugten `ItemEvent`-Objekte.
- `ItemListener` ist ein Interface ähnlich wie `ActionListener`.
- Wieder fügen wir die neue Komponente mithilfe von `void add(Component comp)` dem Applet hinzu.

```
    ...
    public void paint(Graphics g) {
       g.setColor(color);
       g.fillRect(100,50,200,100);
    }
    public void itemStateChanged(ItemEvent e) {
       String s = (String) e.getItem();
       switch(s.charAt(0)) {
          case 'w':   color = Color.white; break;
          case 'r':   color = Color.red; break;
          case 'g':   color = Color.green; break;
          case 'b':   color = Color.blue;
       }
       repaint();
    }
} // end of Applet FirstChoice
```

- Das Interface `ItemListener` verlangt die Implementierung einer Methode `public void itemStateChanged(ItemEvent e);`
- Diese Methode ist für die Behandlung von `ItemEvent`-Objekten zuständig.
- `ItemEvent`-Objekte bieten (u.a. :-) die folgenden Methoden and:
  - `public ItemSelectable getItemSelectable();` — liefert den Selektions-Knopf;
  - `public Object getItem();` — liefert den Text des Items.
- Dann sieht das Ganze so aus:


Applet Viewer: FirstChoice.class
Applet
blue
Applet started.

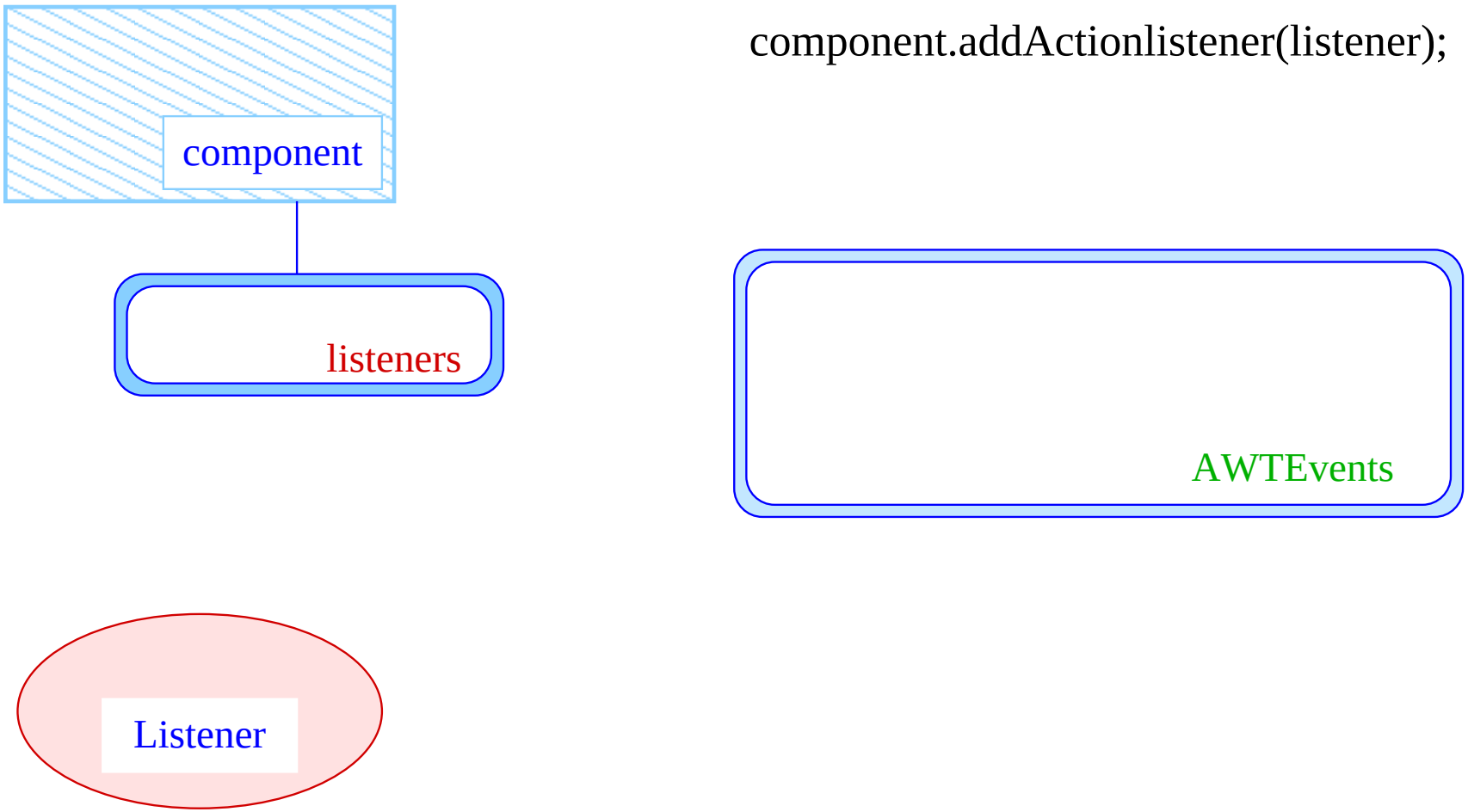

Applet Viewer: FirstChoice.class
Applet
red
Applet started.

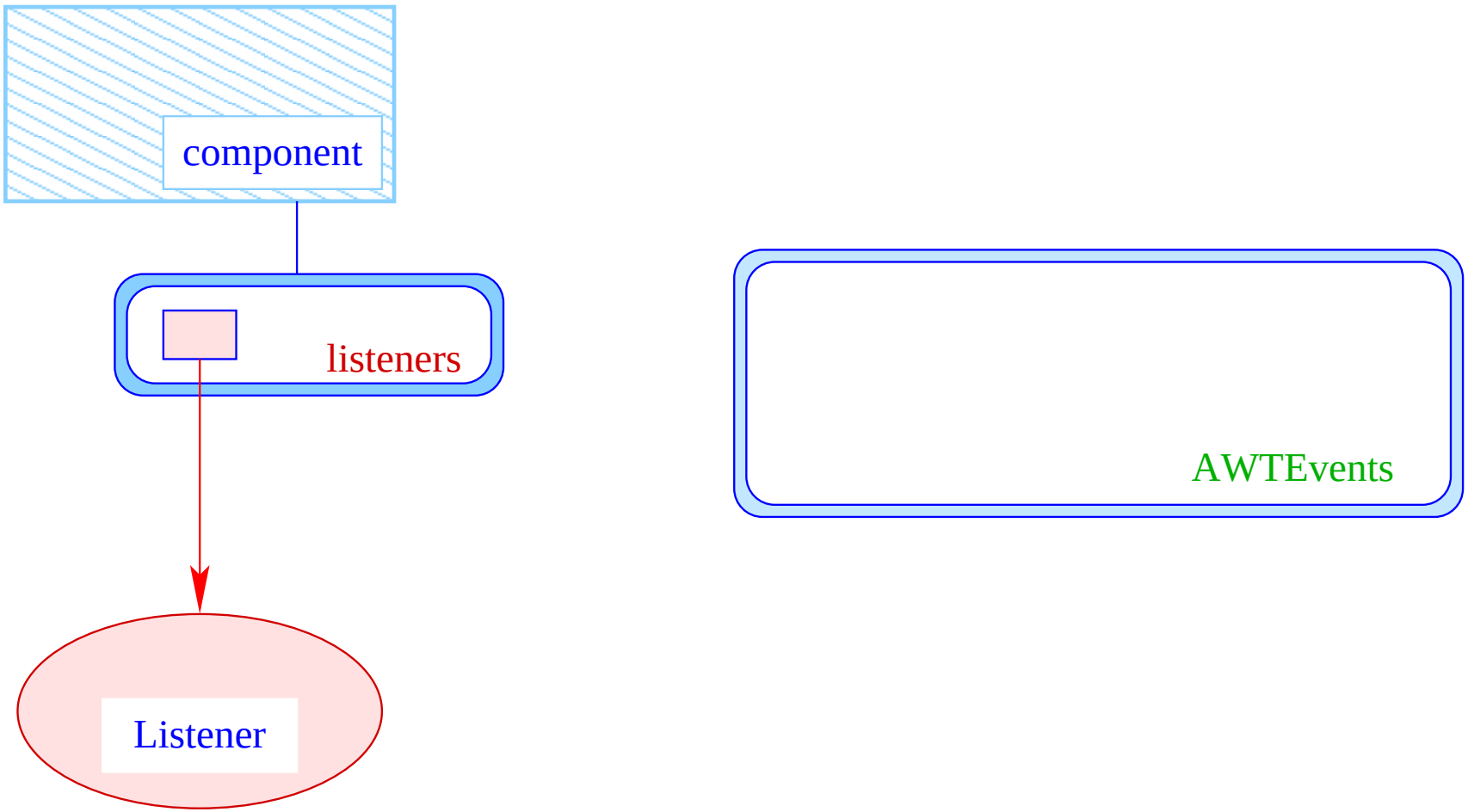
1.3 Ereignisse

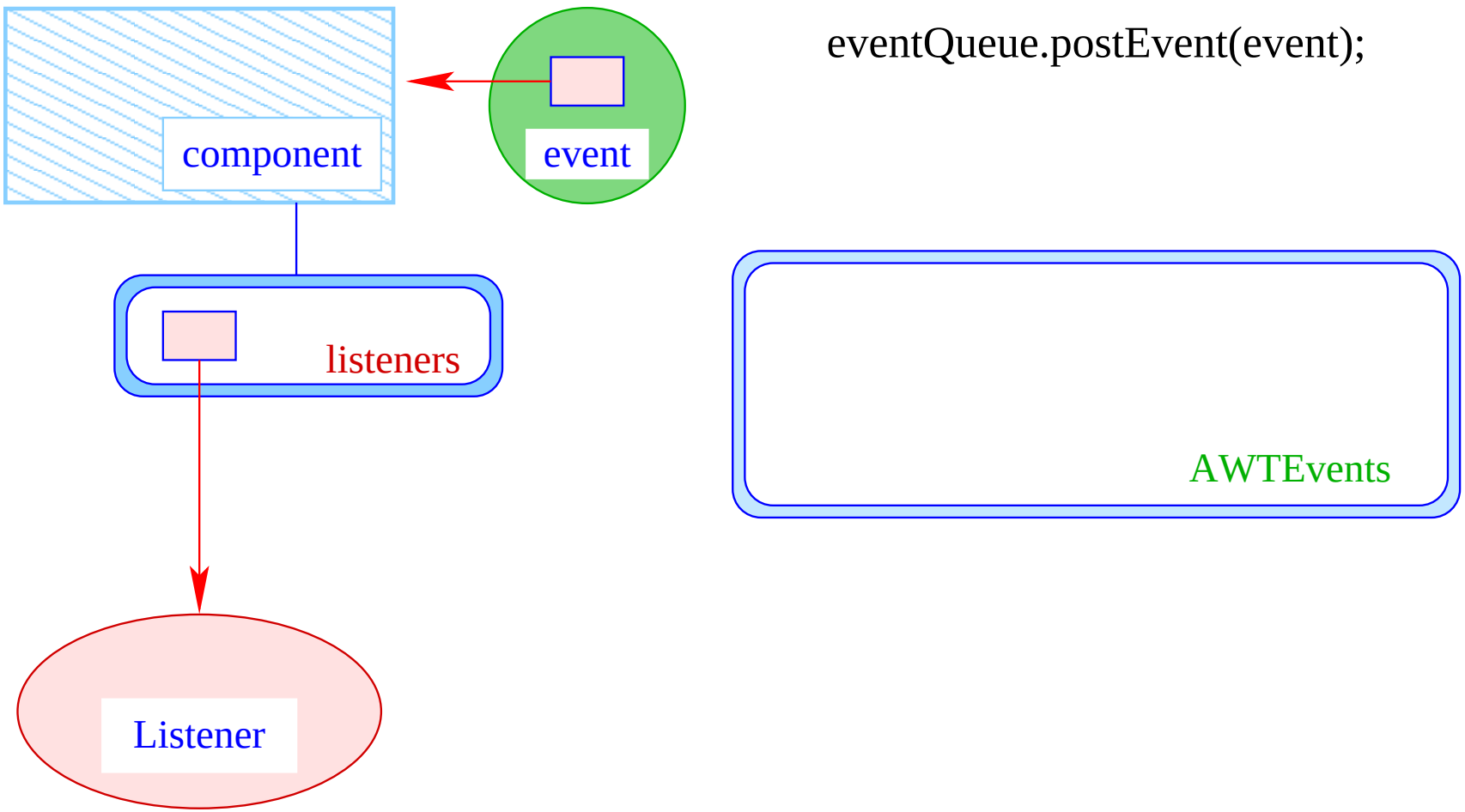
- Komponenten erzeugen Ereignisse;
- Listener-Objekte werden an Komponenten für Ereignis-Klassen registriert;
- Ereignisse werden entsprechend ihrer Herkunft an Listener-Objekte weitergereicht.

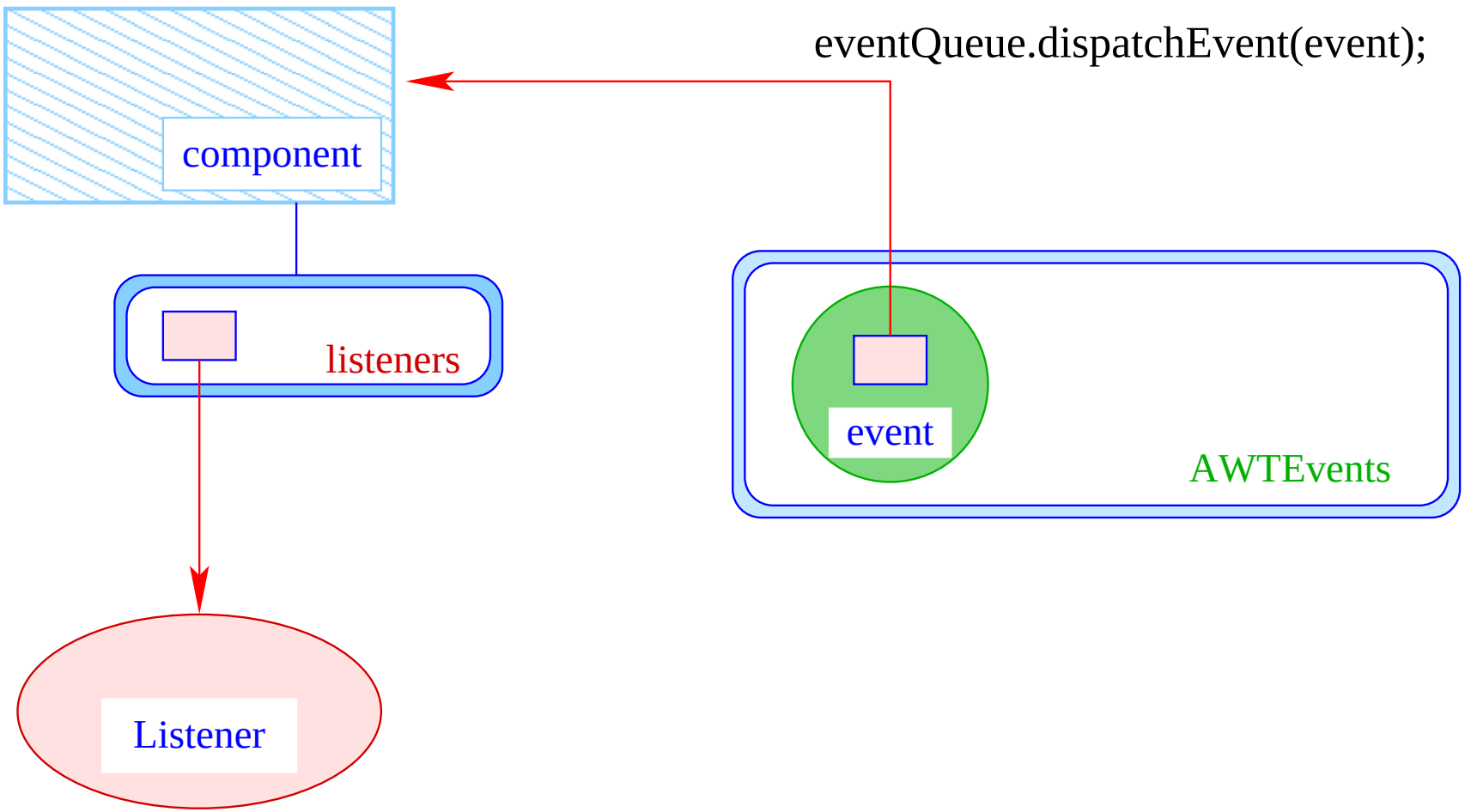
Im Detail:

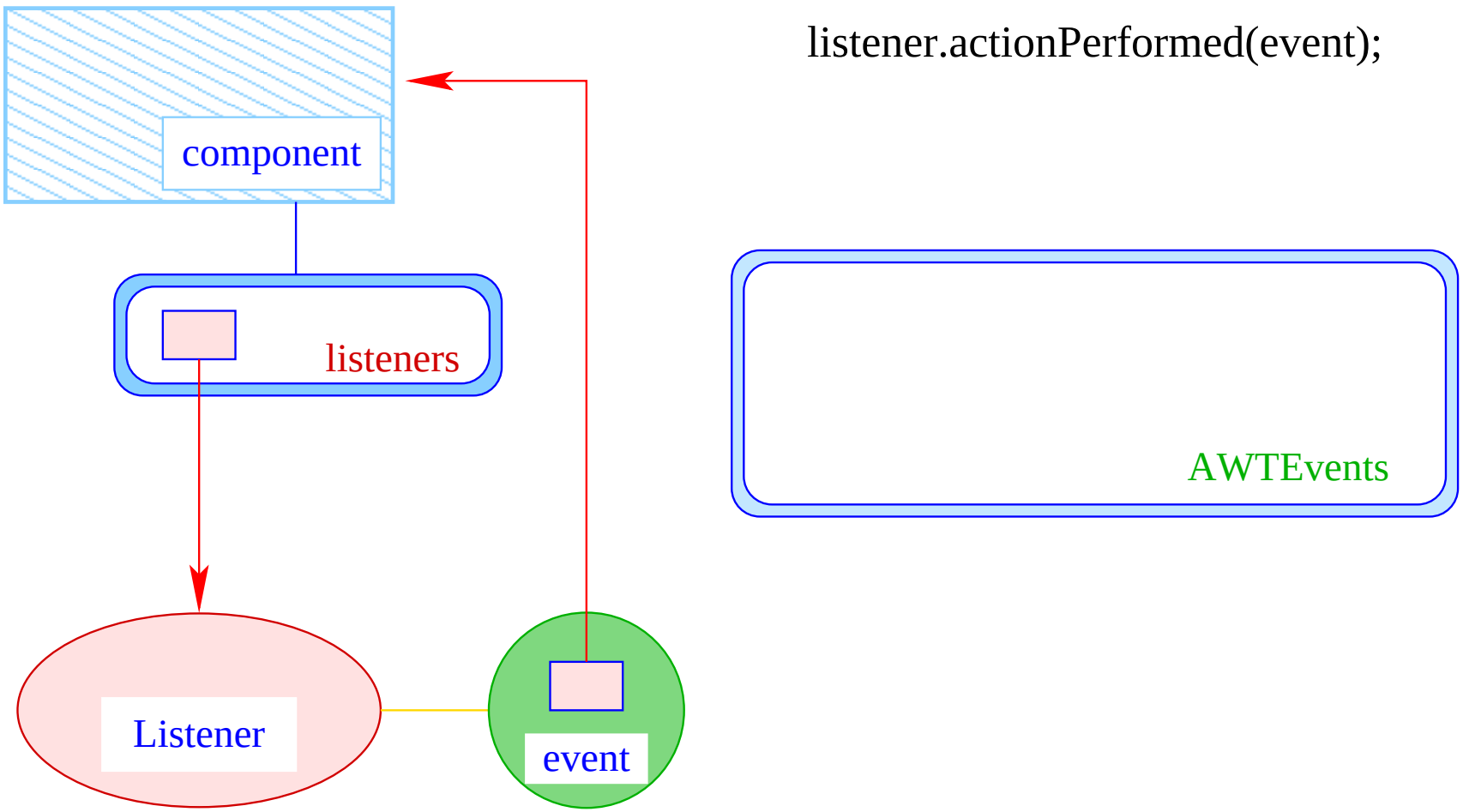
component.addActionlistener(listener);




component
listeners
Listener
AWTEvents


component
event
eventQueue.postEvent(event);
listeners
AWTEvents
Listener


eventQueue.dispatchEvent(event);
component
listeners
event
AWTEvents
Listener


component
listeners
Listener
event
listener.actionPerformed(event);
AWTEvents

- Jedes `AWTEvent`-Objekt verfügt über eine Quelle, d.h. eine Komponente, die dieses Ereignis erzeugte.

  `public` Object `getSource();` (der Klasse `java.util.EventObject`) liefert dieses Objekt.

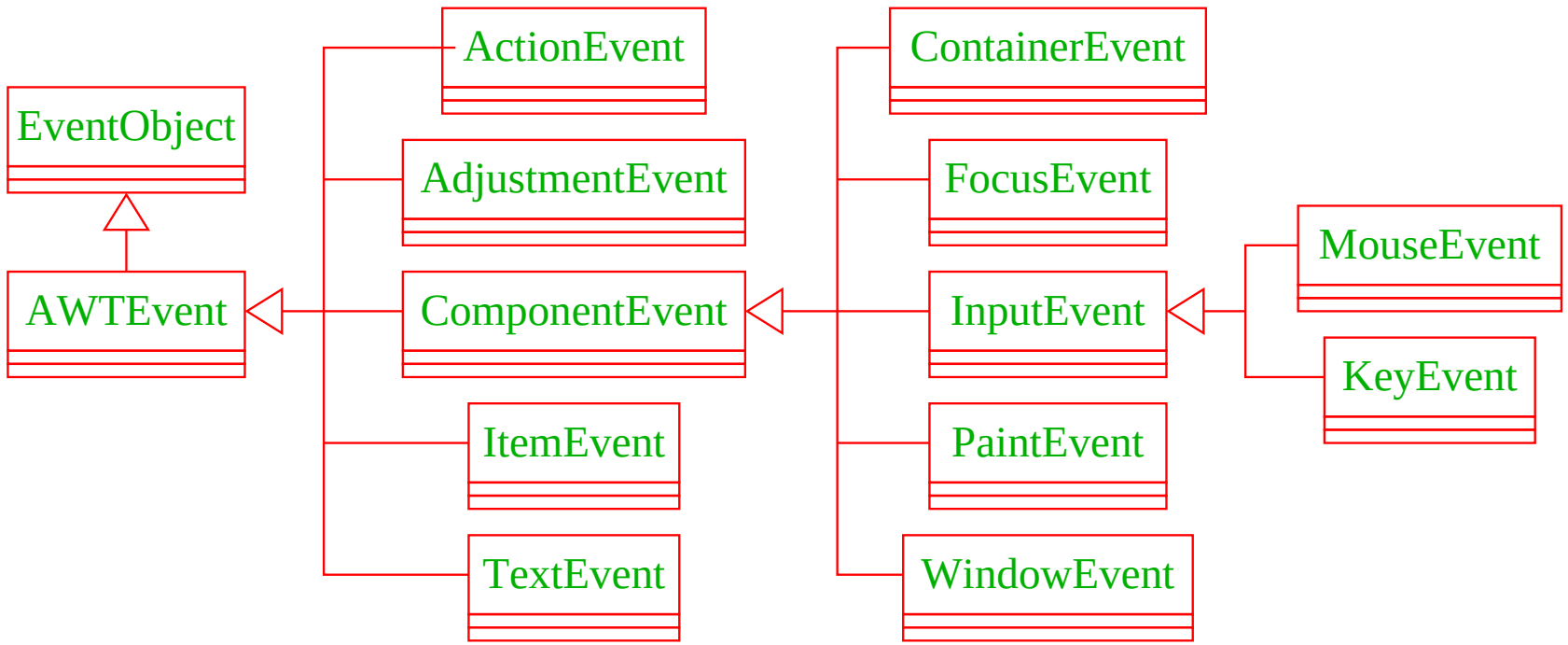
- Gibt es verschiedene Klassen von Komponenten, die Ereignisse der gleichen Klasse erzeugen können, werden diese mit einem geeigneten Interface zusammengefasst.

Beispiele:

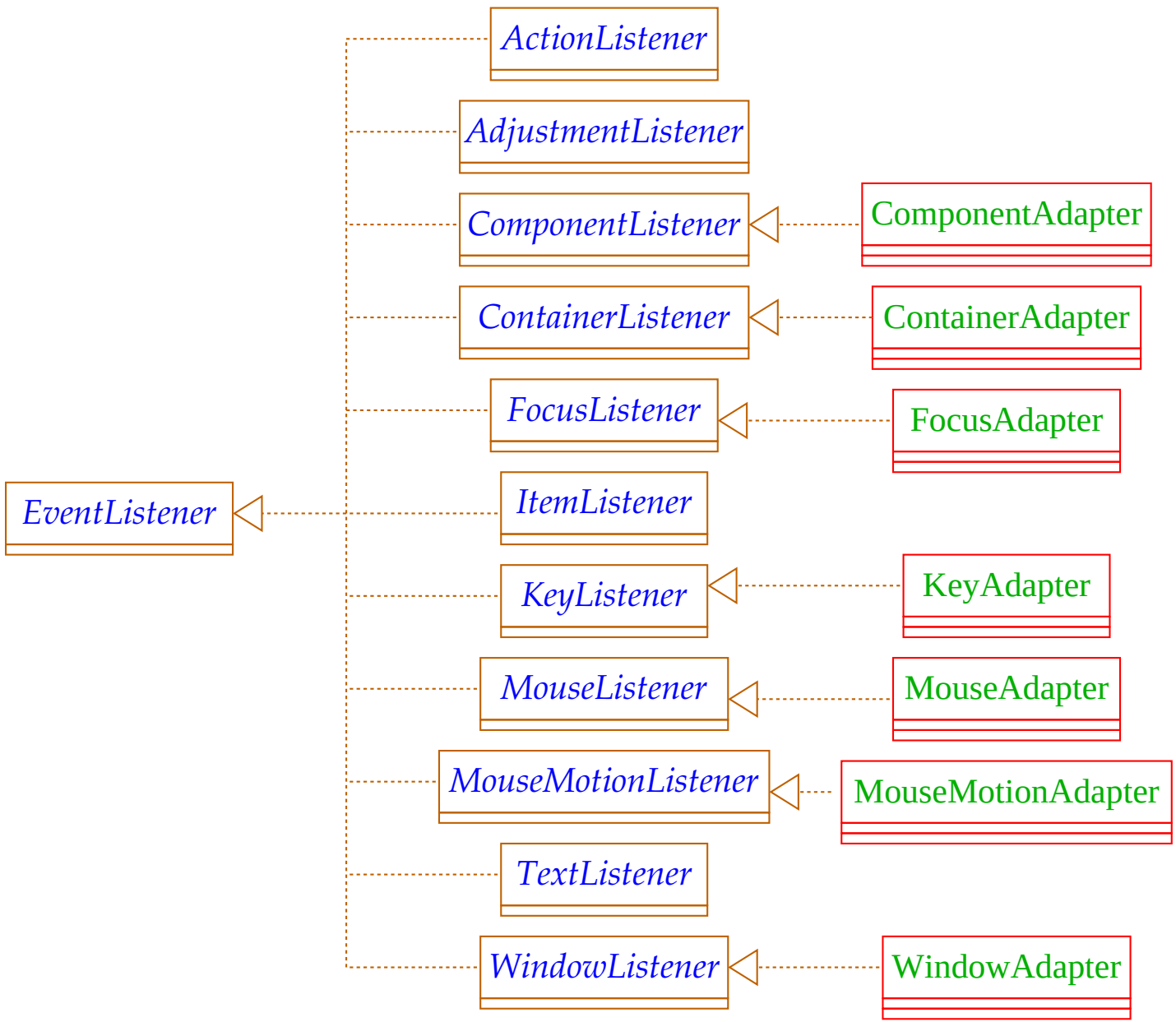
Ereignis-Klasse	Interface	Objekt-Methode
`ItemEvent`	`ItemSelectable`	`ItemSelectable getItemSelectable();`
`AdjustmentEvent`	`Adjustable`	`Adjustable getAdjustable()`

- Eine Komponente kann Ereignisse verschiedener `AWTEvent`-Klassen erzeugen.
- Für jede dieser Klassen können getrennt Listener-Objekte registriert werden ...
- Man unterscheidet zwei Sorten von Ereignissen:
  1. semantische Ereignis-Klassen — wie `ActionEvent` oder `AdjustmentEvent`;
  2. low-level Ereignis-Klassen — wie `WindowEvent` oder `MouseEvent`.

Ein Ausschnitt der Ereignis-Hierarchie ...


EventObject
AWTEvent
ActionEvent
AdjustmentEvent
ComponentEvent
ItemEvent
TextEvent
ContainerEvent
FocusEvent
InputEvent
PaintEvent
WindowEvent
MouseEvent
KeyEvent

- Zu jeder Klasse von Ereignissen gehört ein Interface, das die zuständigen Listener-Objekte implementieren müssen.
- Manche Interfaces verlangen die Implementierung mehrerer Methoden.
- In diesem Fall stellt Java Adapter-Klassen zur Verfügung.
- Die Adapterklasse zu einem Interface implementiert sämtliche geforderten Methoden auf triviale Weise ;-)
- In einer Unterklasse der Adapter-Klasse kann man sich darum darauf beschränken, nur diejenigen Methoden zu implementieren, auf die man Wert legt.


EventListener
ActionListener
AdjustmentListener
ComponentListener
ComponentAdapter
ContainerListener
ContainerAdapter
FocusListener
FocusAdapter
ItemListener
KeyListener
KeyAdapter
MouseListener
MouseAdapter
MouseMotionListener
MouseMotionAdapter
TextListener
WindowListener
WindowAdapter

Beispiel: ein `MouseListener`

- Das Interface `MouseListener` verlangt die Implementierung der Methoden:
  - `void mousePressed(MouseEvent e);`
  - `void mouseReleased(MouseEvent e);`
  - `void mouseEntered(MouseEvent e);`
  - `void mouseExited(MouseEvent e);`
  - `void mouseClicked(MouseEvent e);`
- Diese Methoden werden bei den entsprechenen Maus-Ereignissen der Komponente aufgerufen.
- Unser Beispiel-Applet soll bei jedem Maus-Klick eine kleine grüne Kreisfläche malen ...

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
class MyMouseListener extends MouseAdapter {
   private Graphics gBuff;
   private Applet app;
   public MyMouseListener(Graphics g, Applet a) {
      gBuff = g; app = a;
   }
   public void mouseClicked(MouseEvent e) {
      int x = e.getX(); int y = e.getY();
      gBuff.setColor(Color.green);
      gBuff.fillOval(x-5,y-5,10,10);
      app.repaint();
   }
}  // end of class MyMouseListener
```

- Wir wollen nur die Methode `mouseClicked()` implementieren. Darum definieren wir unsere `MouseListener`-Klasse `MyMouseListener` als Unterklasse der Klasse `MouseAdapter`.
- Die `MouseEvent`-Methoden:

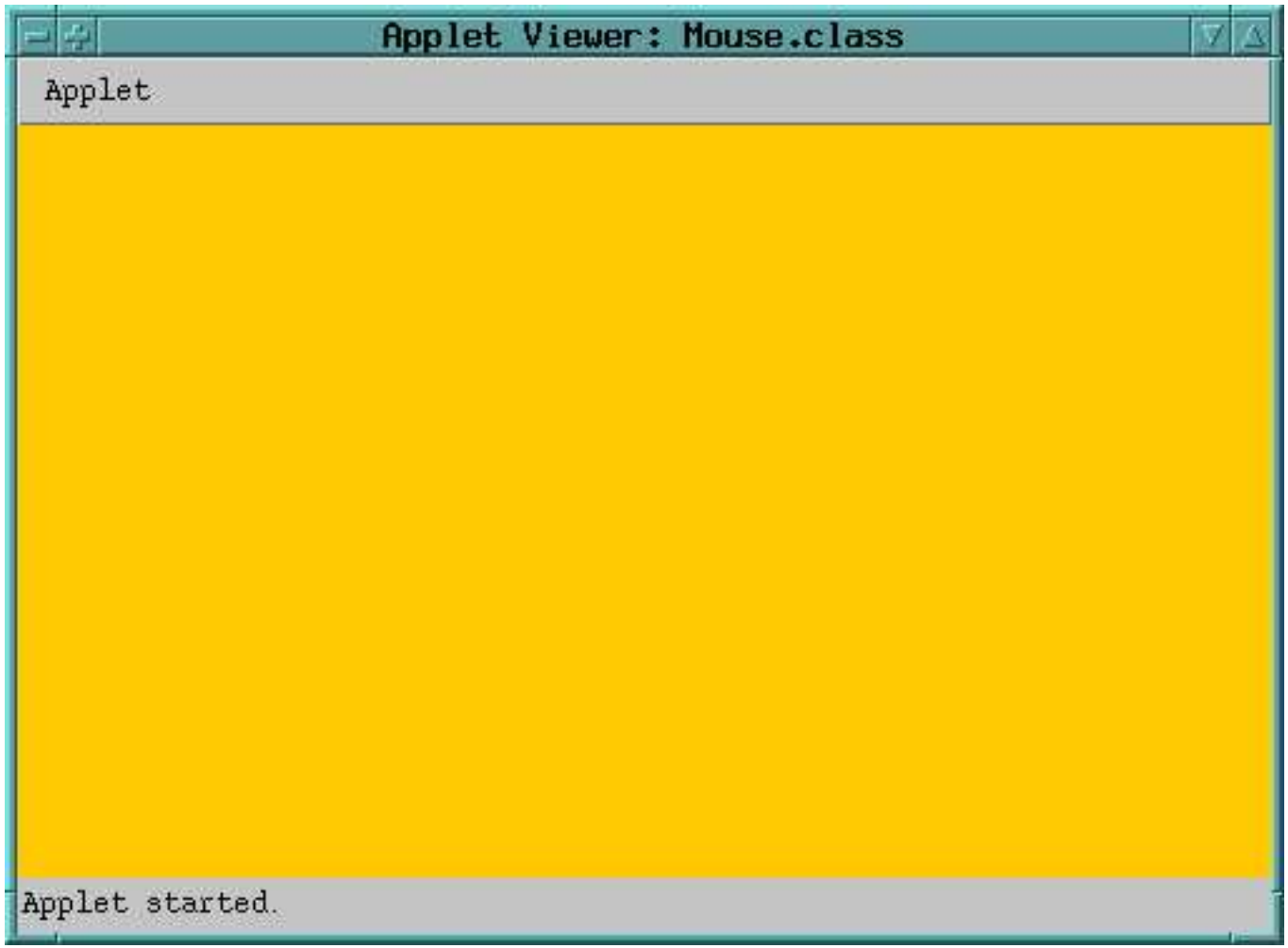
  ```
  public int getX();     public int getY();
  ```

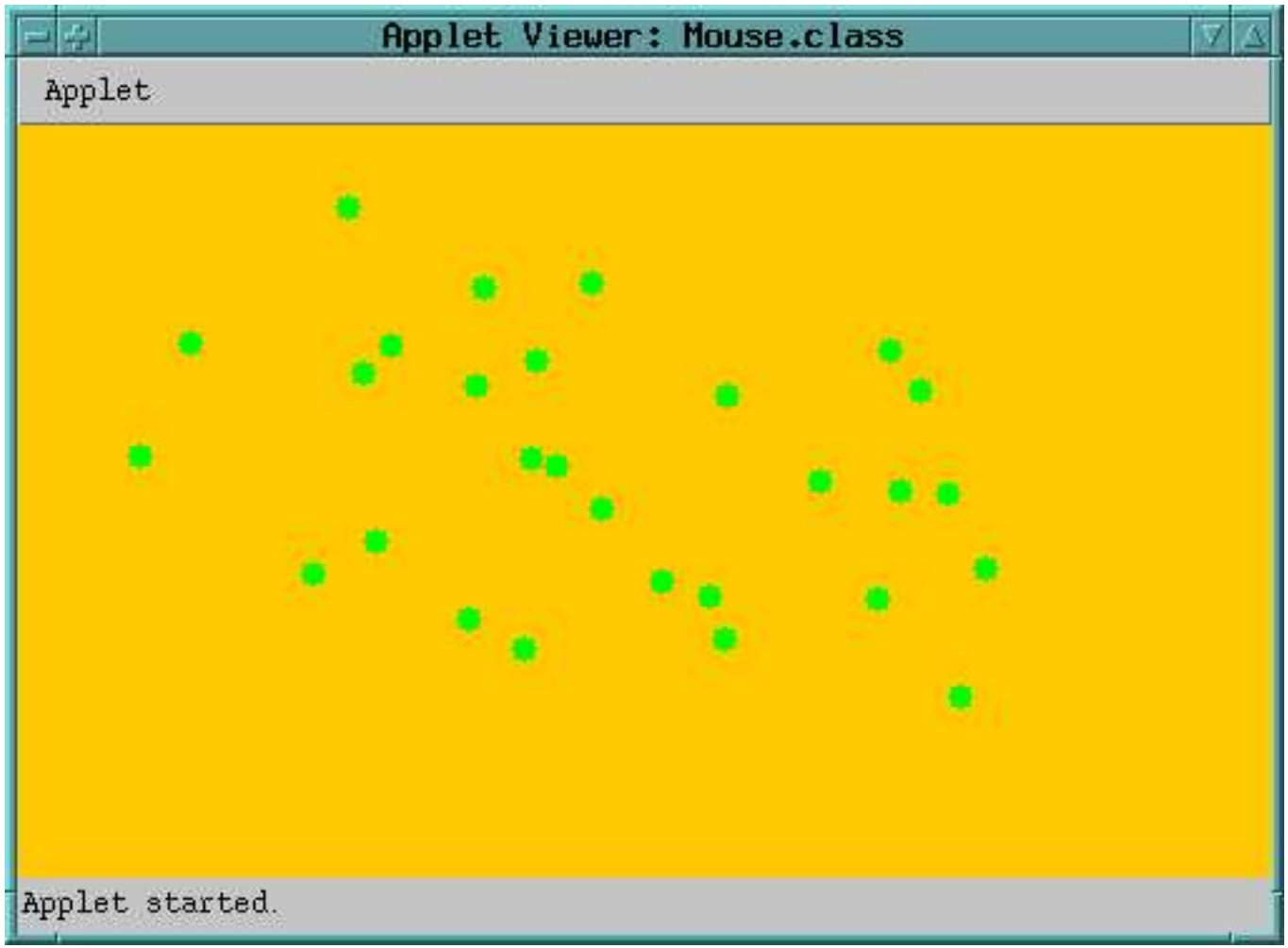
  liefern die Koordinaten, an denen der Mouse-Klick erfolgte ...
- an dieser Stelle malen wir einen gefüllten Kreis in den Puffer.
- Dann rufen wir für das Applet die Methode `repaint()` auf, um die Änderung sichtbar zu machen ...

```
public class Mouse extends Applet  {
   private Image buffer; private Graphics gBuff;
   public void init() {
      buffer = createImage(500,300);
      gBuff = buffer.getGraphics();
      addMouseListener(new MyMouseListener(gBuff,this));
   }
   public void start() {
      gBuff.setColor(Color.orange);
      gBuff.fillRect(0,0,500,300);
   }
   public void paint(Graphics page) {
      page.drawImage(buffer,0,0,500,300,this);
   }
} // end of class Mouse
```

- Die Methode `init()` legt den Puffer an, in dem die kleinen grünen Scheiben gemalt werden. Dann erzeugt sie ein `MyMouseListener`-Objekt und registriert es als `MouseListener` des Applets.
- Die Methode `start()` malt den Puffer orange.
- Die Methode `paint()` überträgt den Puffer auf die Applet-fläche.

... der Effekt:

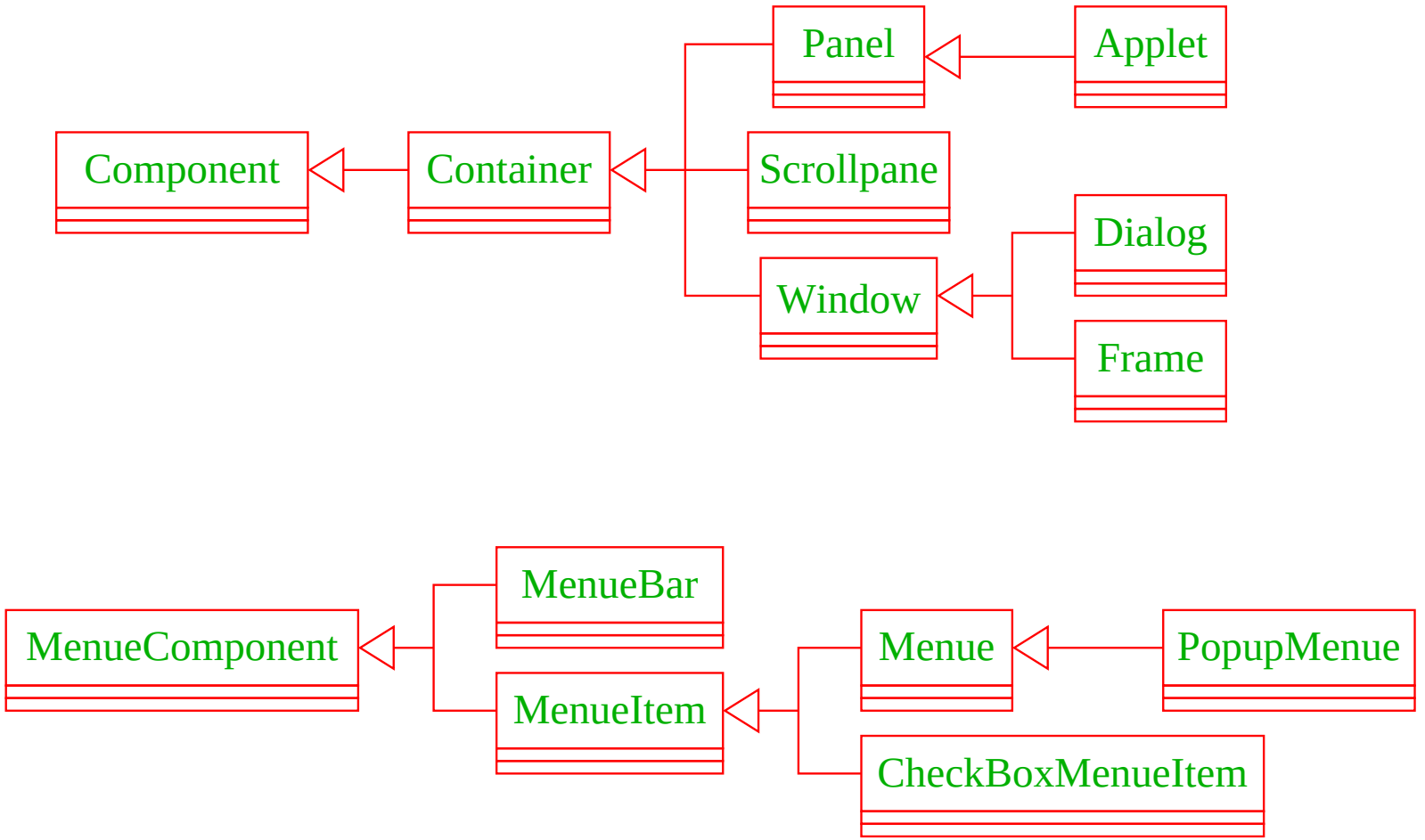

Applet Viewer: Mouse.class
Applet
Applet started.


Applet Viewer: Mouse.class
Applet
Applet started.

1.4 Schachtelung von Komponenten

- Komponenten, die andere Komponenten aufnehmen können, heißen Container.
- Der LayoutManager des Containers bestimmt, wie Komponenten innerhalb eines Containers angeordnet werden.

Ein Ausschnitt der `Container`-Hierarchie:


Component
Container
Panel
Applet
Scrollpane
Window
Dialog
Frame
MenueComponent
MenueBar
MenueItem
Menue
PopupMenue
CheckBoxMenueItem

Container: Abstrakte Oberklasse aller Komponenten, die andere als Bestandteil enthalten können.

Panel: Konkrete `Container`-Klasse zum Gruppieren von Komponenten.

Applet: Unterklasse von `Panel` für das Internet.

Window: Ein nackter `Container` zur Benutzung in normalen Programmen. Kein Rand, kein Titel, keine Menue-Leiste.

Frame: Ein `Window` mit Rand und Titel-Zeile. Unterstützt die Benutzung von Menues.

Dialog: Spezielles `Window`, das sämtlichen sonstigen Benutzer-Input blockieren kann, bis das `Dialog`-Fenster geschlossen wurde.

Beispiel: Das `Basic Applet` als `Frame`

- Statt der Klasse `Applet` benutzen wir die (Ober-)Klasse `Panel`.

  Der Grund: `Applet` ist eine Unterklasse – allerdings mit zusätzlichen Multimedia-Features, über die `Panel` nicht verfügt – wie z.B. Bilder aus dem internet zu laden.

  Indem wir nur `Panel`-Methoden zulassen, garantieren wir, dass die Extra-Features nicht benutzt werden.

  Da wir nur auf eine Fläche malen wollen, würde (hier) auch ein `Canvas`-Objekt reichen.

- Das `Panel`-Objekt passen wir in einen `Frame` ein.

- Ein `Frame`-Objekt ist normalerweise unsichtbar. Um es sichtbar zu machen, rufen wir `public void setVisible(boolean b)` mit `b==true` auf.

```
import java.awt.*;
class BasicPanel extends Panel {
   public BasicPanel() {
      setBackground(Color.orange);
   }
   public void paint(Graphics g) {
      g.setColor(Color.red);
      g.fillRect(50,50,200,100);
      g.setColor(Color.blue);
      g.fillRect(100,100,200,100);
      g.setColor(Color.green);
      g.fillRect(150,150,200,100);
   }
} // end of class BasicPanel
...
```

- Was in den Methoden `init()` bzw. `start()` passierte, erfolgt nun in den Konstruktoren des `Panel`-Objekts ...
- Der Methode `destroy()` entspricht die Methode `public void finalize();` die aufgerufen wird, wenn das Objekt freigegeben wird (deren Existenz wir bisher verschwiegen haben :-).
- Die `paint()`-Methode entspricht derjenigen des Applets und wird entsprechend automatisch aufgerufen, wenn die Fläche neu bemalt werden soll.

```
...
public class Basic extends Frame {
   public Basic(int x, int y) {
      setLocation(x,y);
      setSize(500,300);
      add(new BasicPanel());
   }
   public static void main(String[] args) {
      (new Basic(50,50)).setVisible(true);
      (new Basic(600,600)).setVisible(true);
   }
}  // end of class Basic
```