15. Ereignisse (Events)

Java-Beispiel:

EyesApplet.java

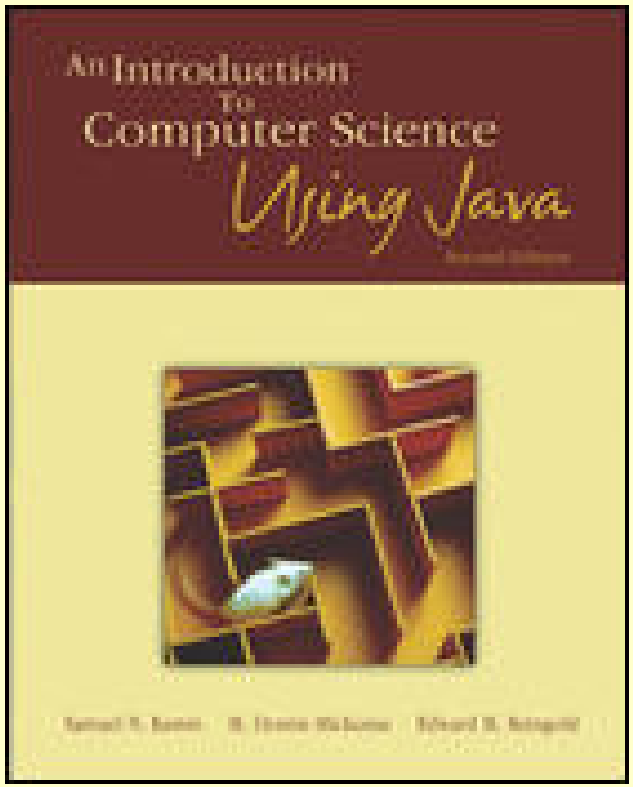
Eyes.java

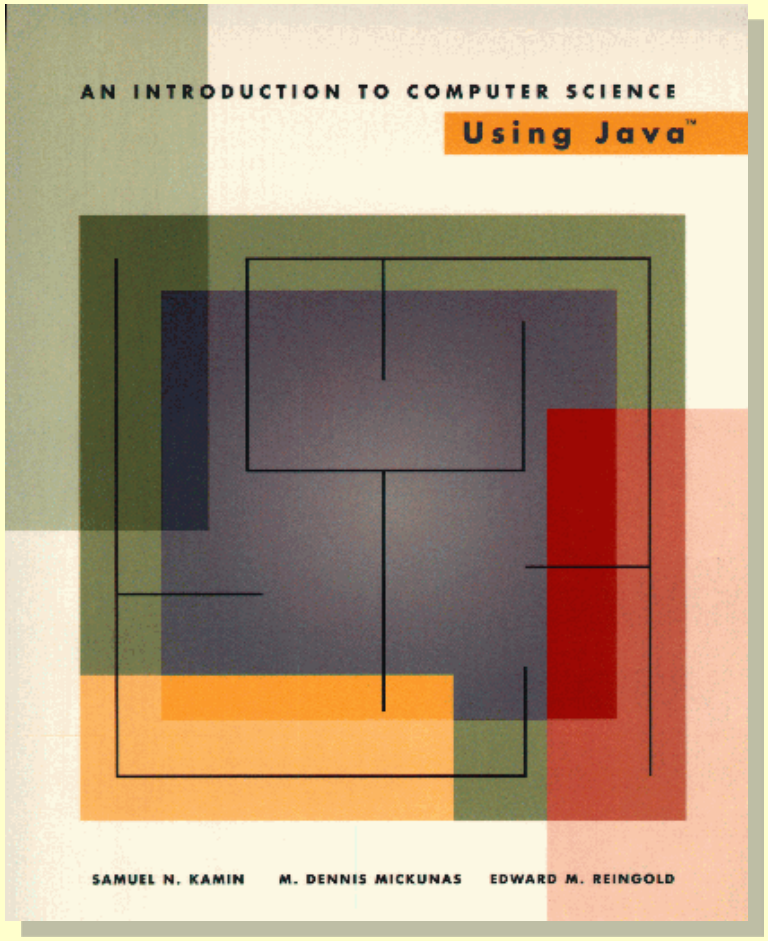
Schwerpunkte

- Ereignisbehandlung (Event Handling)
- Vom Problem zum Programm
- SW-Architektur:
  Programme mit graphischer Nutzeroberfläche
- Ein Beispiel zu Events, Applets und Graphics
- Ereignisse: erzeugen und behandeln
  ... ist so ähnlich wie Ausnahmen erzeugen und behandeln
- API-Klassen:
  - MouseMotionListener
  - MouseEvent
  - Applet
  - Graphics

Java-Programmbeispiel: Quellen

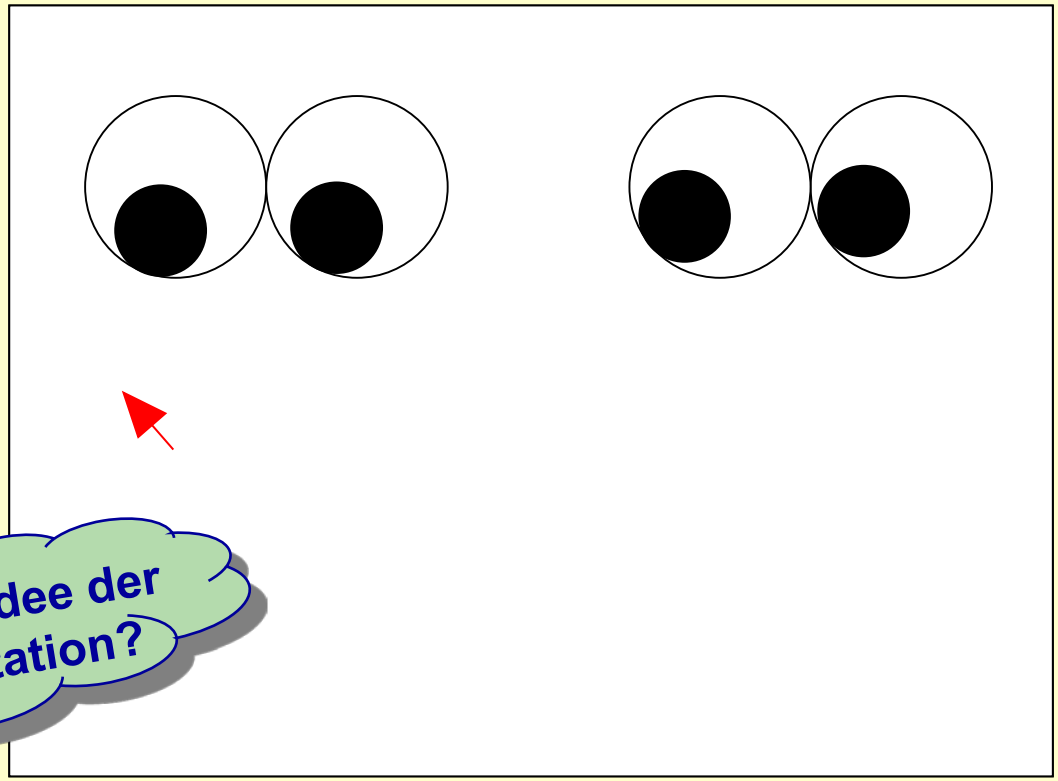
Basierend auf einer Idee von S.N. Kamin, M.D. Mickunas, E.M. Reingold: „An introduction to computer science – Using Java“, McGraw-Hill, 1st und 2nd Edition, 1998, 2002





Aufgabe

Man entwickle ein Applet, in dem Augenpaare dem Cursor (der von der Maus bewegt wird) folgen.



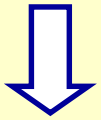
Inhaltliche Idee der Implementation?

Cursor bewegt (Ereignis) → Position des Cursors abfragen → Pupillen neu zeichnen

Ereignis

- **Besonderes Vorkommnis außerhalb des Programms …**
  (Hardware)

  - Tastatur betätigt (z. B. Enter-Taste)
  - Maus bewegt / Taste betätigt
  - graphisches Nutzerinterface bedient
    (z.B. Button gedrückt)



- **… löst bestimmte Reaktion des Programms aus.**

  - Ausgabe des Grad-Celsius-Wertes (**`TempApplet.java`**)
  - Veränderte Augenpaare (**`Eyes.java, EyesApplet.java`**)
  - Graphisches Objekt erscheint o. ä.

Ereignisgesteuerte Programmausführung

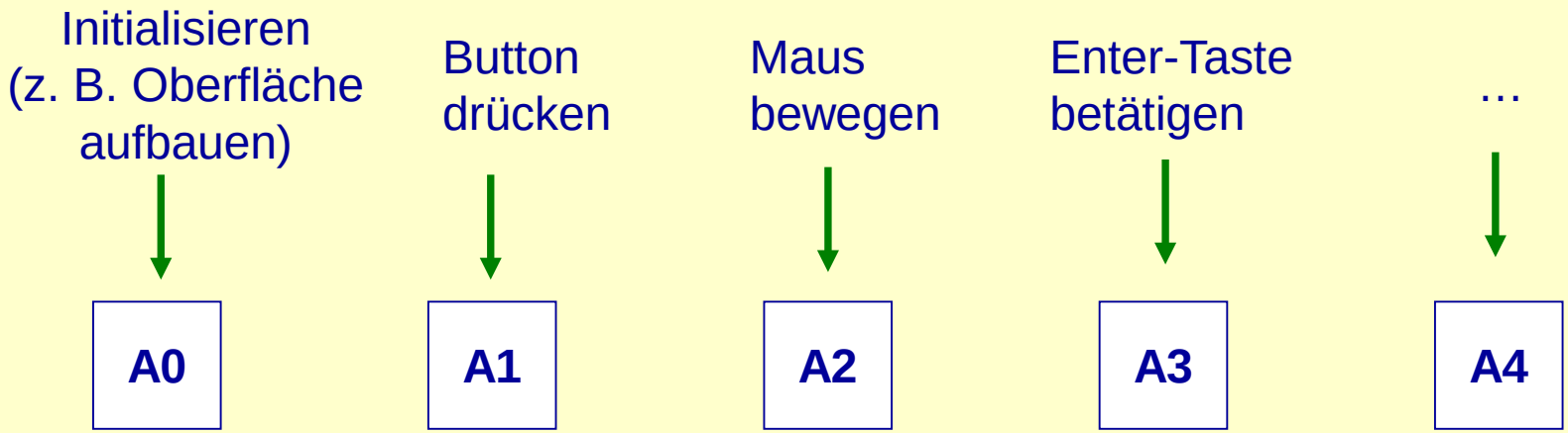
„Normale“ Programmsteuerung:

mit main() oder init() gestartet und beendet

(z.B. Quicksort, Maus im Labyrinth)

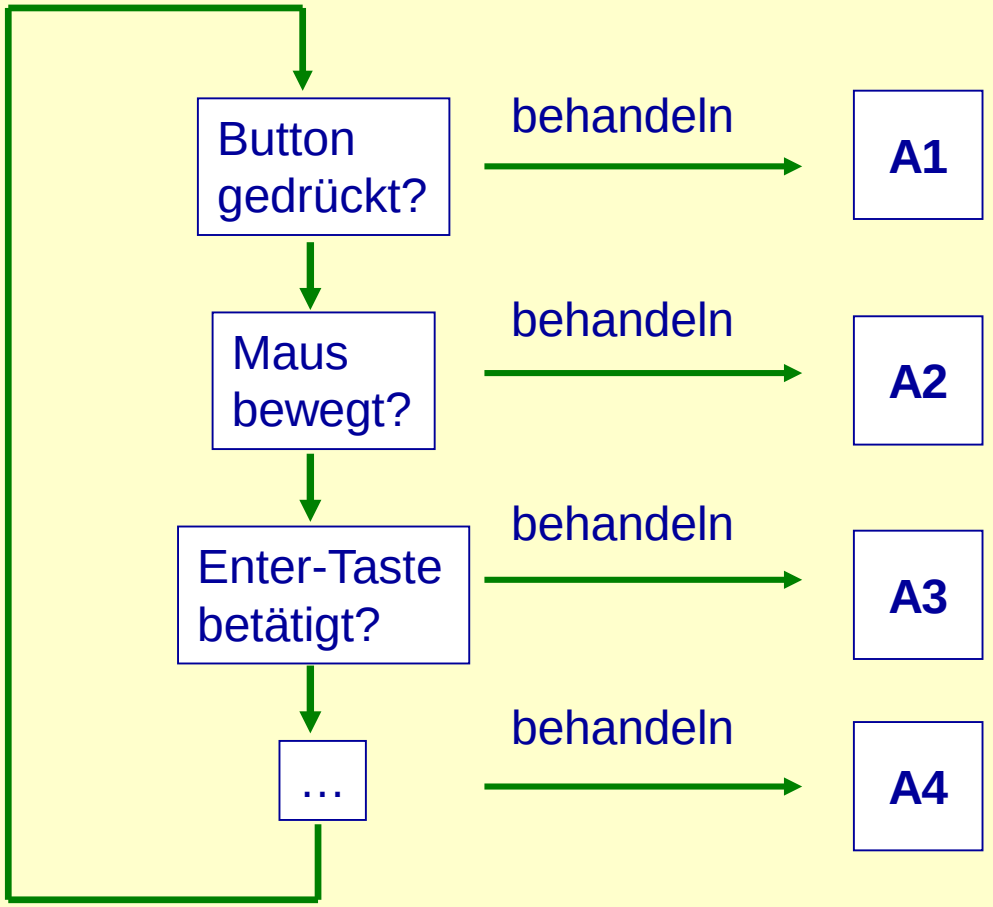
Ereignisgesteuerte Programmsteuerung:

Programmausführung: ***viele*** „kleine“ Algorithmen werden aktiviert



Ereignisgesteuertes Programm: Ereignisschleife

JVM steuert die Ereignisschleife: beobachtet, ob Ereignis auftritt



Zur Anforderungsdefinition

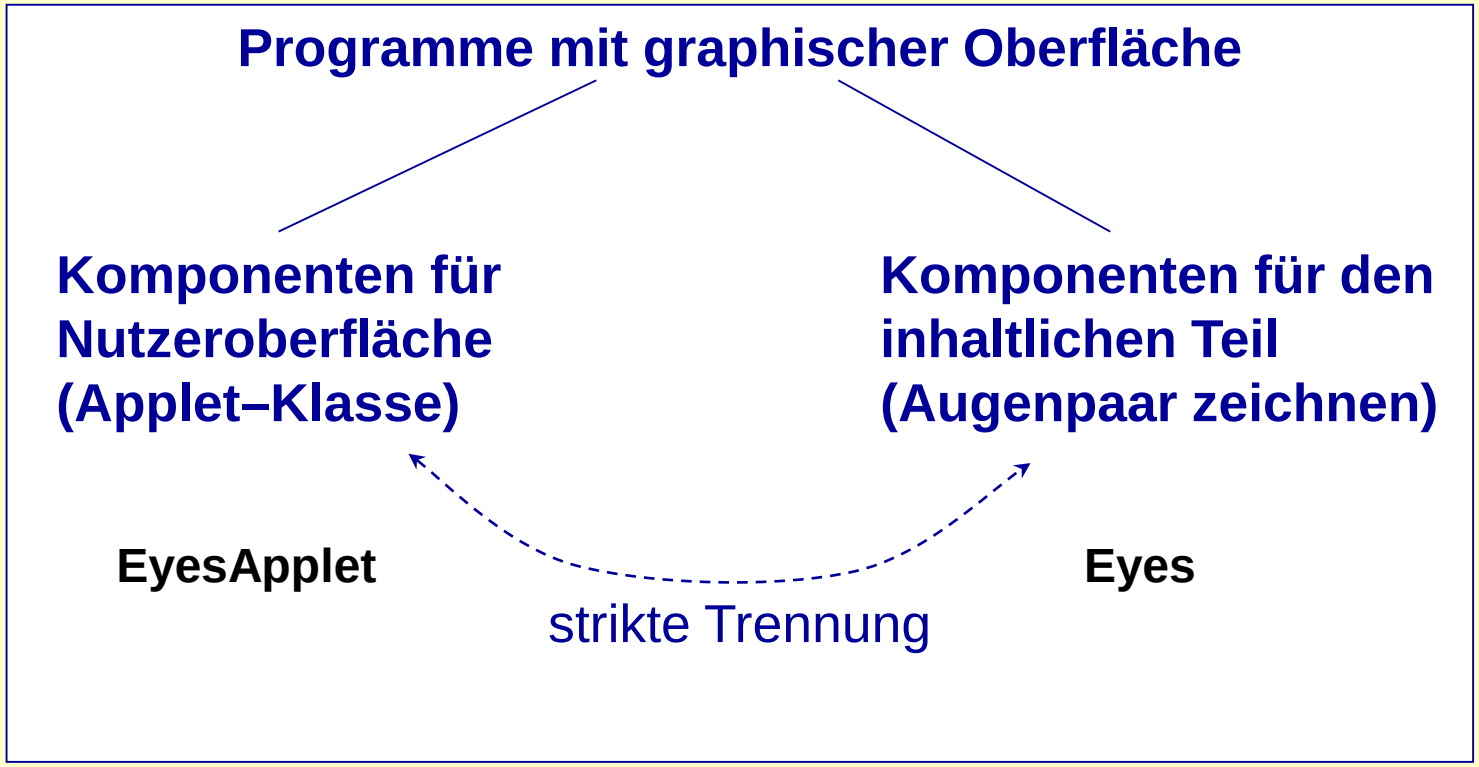
Man entwickle ein Applet, in dem Augenpaare dem Cursor (der von der Maus bewegt wird) folgen.

Zu klärende Probleme:

- Anzahl Augenpaare *)
- Platzierung der Augen *)
- Abmessungen für Pupille und Augapfel *)
- In welchen zeitlichen Abständen soll die Blickrichtung verändert werden?
  (laufend/kontinuierlich: in kleinster möglicher Zeiteinheit)

*) Feste Werte nach Programmstart, aber durch Konstanten im Programm leicht modifizierbar

Software-Architektur: Grundprinzip

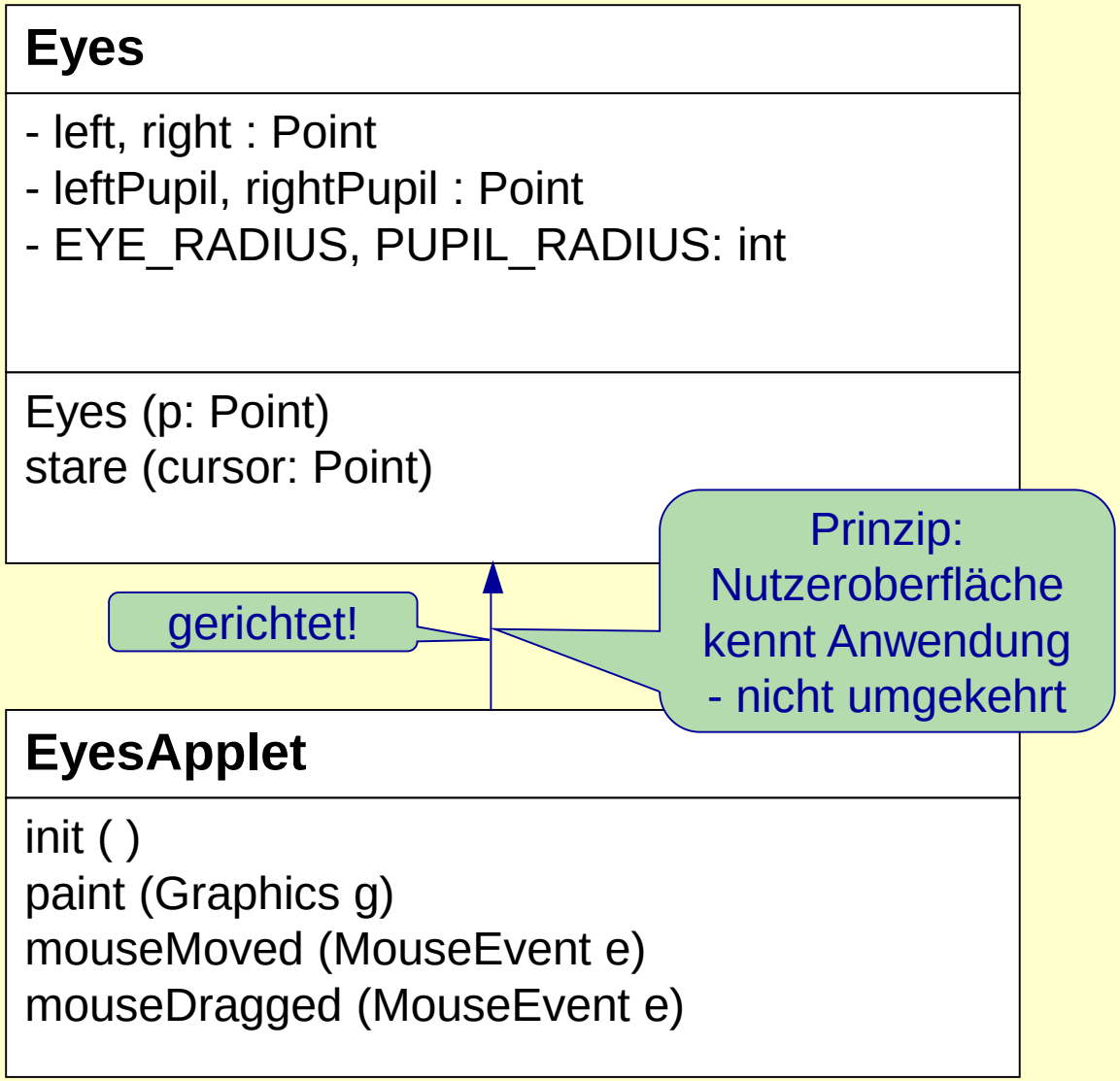


Architekturfehler: Vermischung beider Komponenten

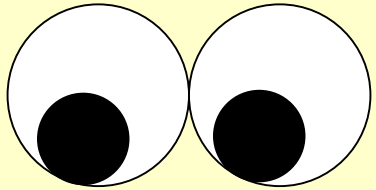
Software–Architektur: UML

Augen-paar zeichnen

Nutzer-oberfläche



Klasse 'E y e s'



Zum Zeichnen eines Augenpaares

Zustand

Eyes

- left, right : Point
- leftPupil, rightPupil : Point
- EYE_RADIUS, PUPIL_RADIUS: int

Rechtes / linkes Auge:
jeweils Mittelpunkt von Augapfel und Pupille sowie Radius

Verhalten

Eyes (p: Point)

Erzeuge Augenpaar mit Position p (Anfangszustand)

stare (cursor: Point)

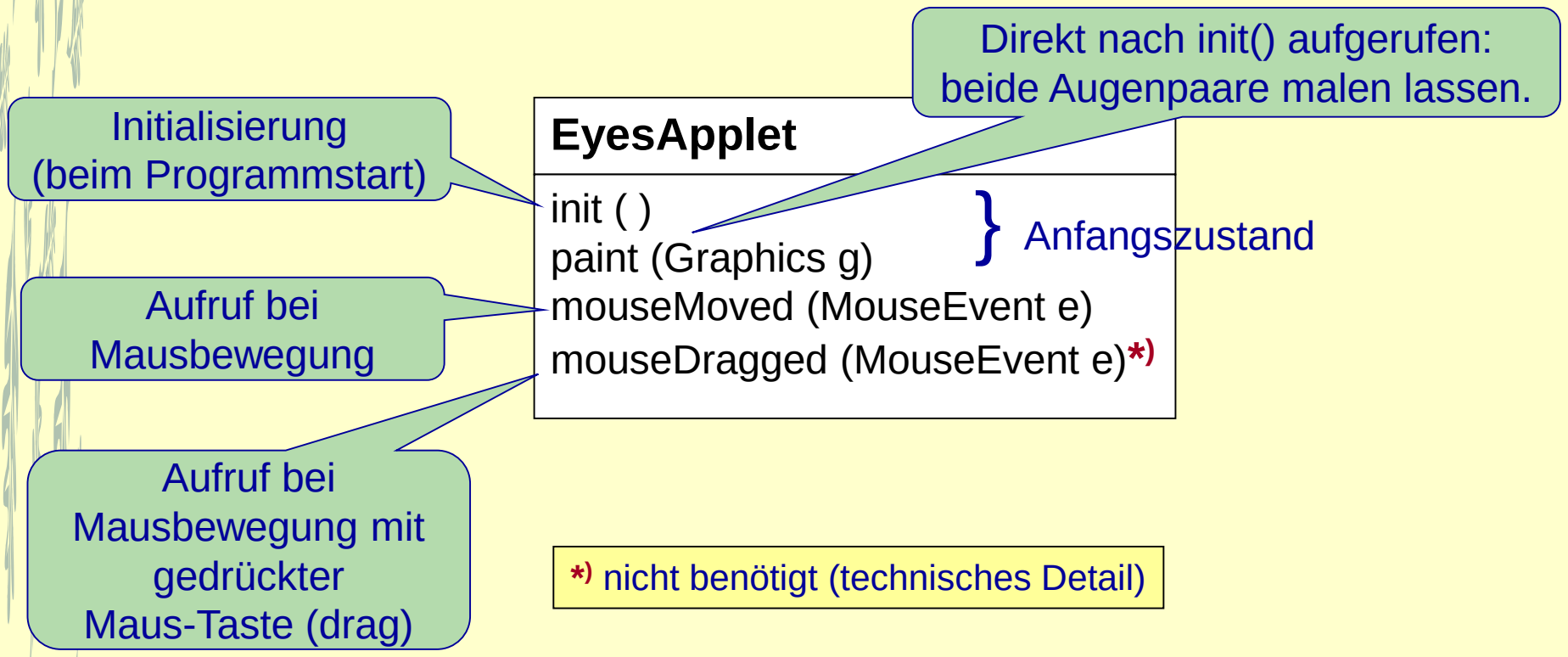
Zeichne Augenpaar in Richtung 'cursor'
(stare = (an)starren)

Welche Zustands-änderung durch stare?

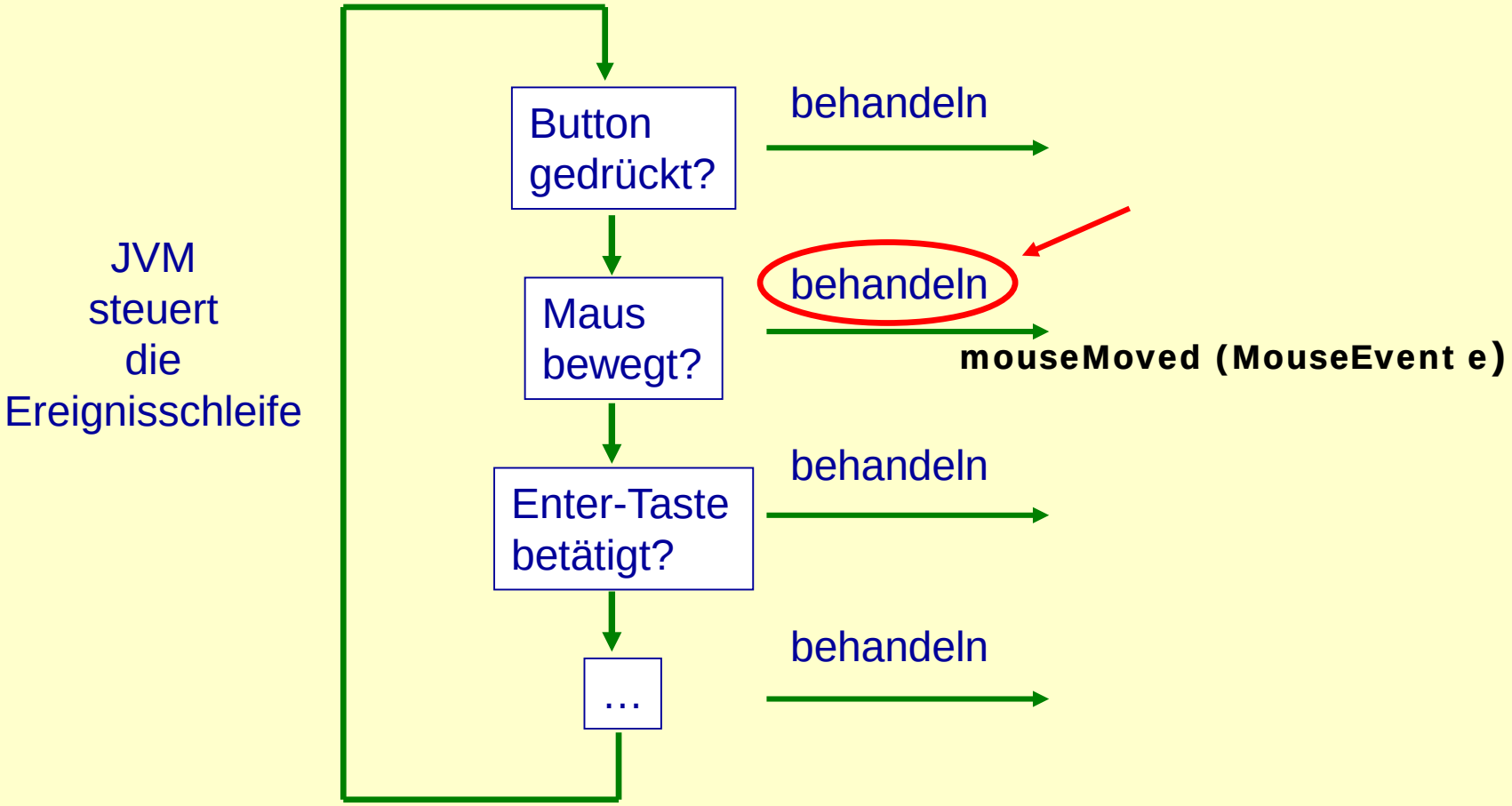
Volle Form: **`stare(g: Graphics, cursor:Point)`**

Klasse ‚EyesApplet'

- Grundlagen:
  - Ereignisbehandlung (Klasse MouseMotionListener)
  - Applets (Klasse Applet)
- Aufruf aller Methoden durch Interpreter
  - nicht durch das Programm!



Ereignisgesteuertes Programm: Ereignisschleife



EyesApplet.java

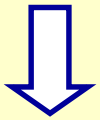
```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class EyesApplet extends Applet
              implements MouseMotionListener {
  Point cursor;
  Eyes e1, e2;

  public void init () {
    // Register the Listener.
    addMouseMotionListener(this);
    setSize(500,400);
    setBackground(Color.LIGHT_GRAY);
    e1 = new Eyes(new Point (63,30));
      // center of one eye
    e2 = new Eyes(new Point (437,30));
      // center of the other
    cursor = new Point(250, 2000);
      // initial cursor
  }

  public void paint(Graphics g) {
    e1.stare(g, cursor);
    e2.stare(g, cursor);
  }

  public void mouseMoved (MouseEvent e) {
    cursor = e.getPoint();
    repaint();
    }

  public void mouseDragged (MouseEvent e) {}
}
```

Eyes.java

```
public class Eyes {

  private Point left, right, leftPupil, rightPupil;
  private final int EYE_RADIUS = 30,PUPIL_RADIUS = 10;

  public Eyes(Point c) {
    left = new Point(c.x-EYE_RADIUS-3, c.y);
    right = new Point(c.x+EYE_RADIUS+3, c.y);
  }

  private void fillCircle (Graphics g,
                           Point center, int radius) {
    g.fillOval(center.x-radius, center.y-radius,
               2*radius, 2*radius);
  }

  public void stare (Graphics g, Point cursor) {

    // Draw the white eyes
    g.setColor(Color.WHITE);
    fillCircle(g, left, EYE_RADIUS);
    fillCircle(g, right, EYE_RADIUS);

    // Draw the pupils
    g.setColor(Color.black);
    leftPupil = compute (cursor, left);
    fillCircle(g, leftPupil, PUPIL_RADIUS);
    rightPupil = compute (cursor, right);
    fillCircle(g, rightPupil, PUPIL_RADIUS);
  }

  private Point compute (Point cursor, Point eye) {
    double d = Math.sqrt((cursor.x-eye.x)*(cursor.x-eye.x)
                         + (cursor.y-eye.y)*(cursor.y-eye.y));
    int r = EYE_RADIUS - PUPIL_RADIUS;
    return new Point (eye.x + (int)((cursor.x-eye.x)*r/d),
                      eye.y + (int)((cursor.y-eye.y)*r/d));
  }
}
```

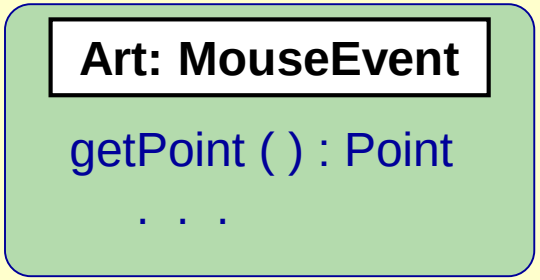
Ereignisbehandlung (Event Handling)

- **Besondere externe Vorkommnisse außerhalb des Programms: Signale der Umgebung an Programm**
  - Tastatur betätigt (z. B. Enter-Taste)
  - Maus bewegt / Taste betätigt
  - graphisches Nutzerinterface bedient
    (z.B. Button gedrückt)



- **... führen zur Bildung eines Ereignis-Objekts.**
  - Objekt einer Event-Klasse

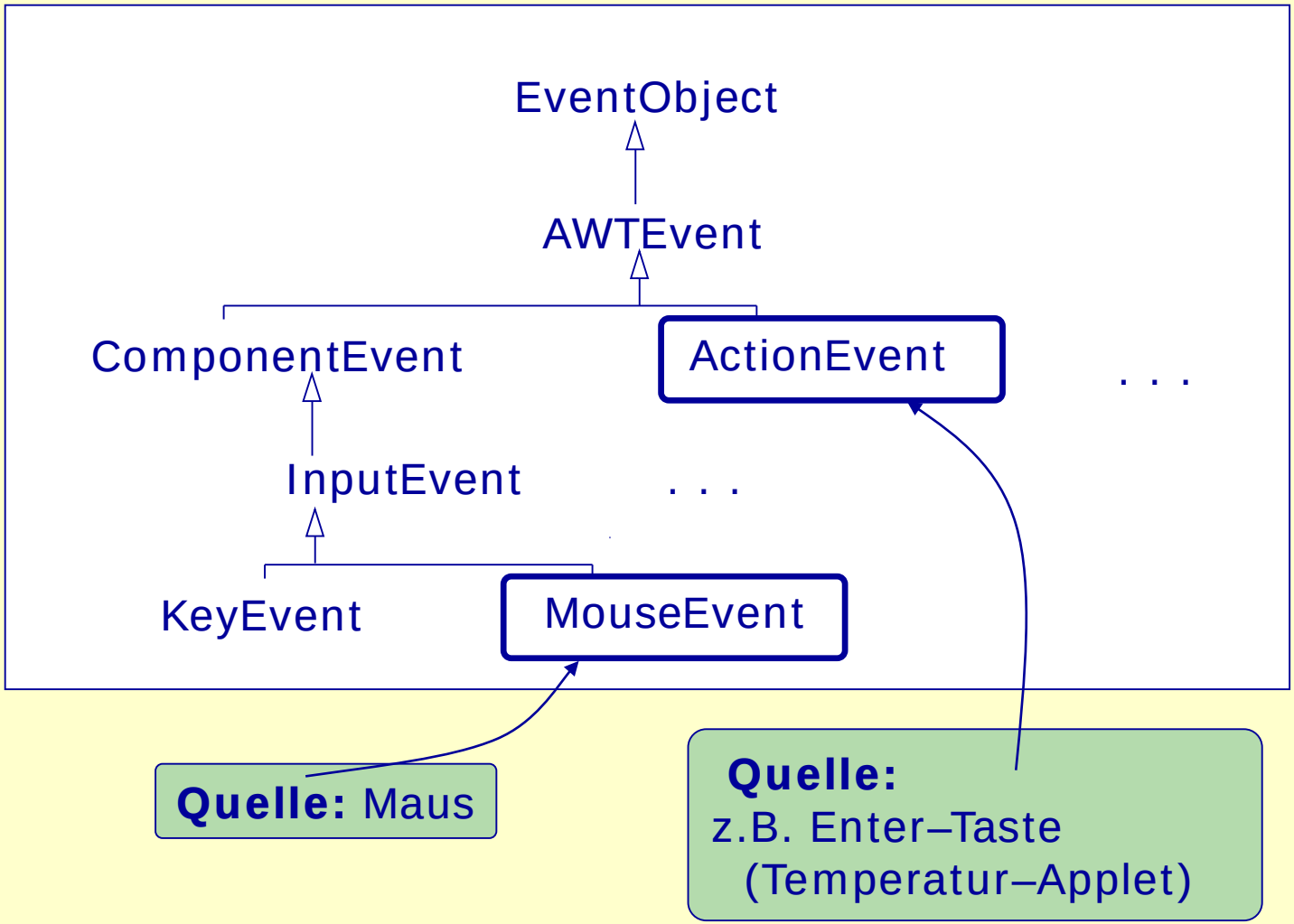
  (vgl. Ausnahme-Objekte erzeugt bei Laufzeitfehlern)



vgl. Ausnahmebehandlung (exceptions)

Vererbungshierarchie der Ereignisklassen

(Auszug aus dem Java-API)



API-Klasse ‚MouseEvent'

java.awt.event
Class MouseEvent

java.lang.Object
  └ java.util.EventObject
    └ java.awt.AWTEvent
      └ java.awt.event.ComponentEvent
        └ java.awt.event.InputEvent
          └ **java.awt.event.MouseEvent**

All Implemented Interfaces:
Serializable

Direct Known Subclasses:
MenuDragMouseEvent, MouseWheelEvent

```
public class MouseEvent
extends InputEvent
```

An event which indicates that a mouse action occurred in a component. A mouse action is considered to occur in a particular component if and only if the mouse cursor is over the unobscured part of the component's bounds when the action happens. Component bounds can be obscured by the visible component's children or by a menu or by a top-level window. This event is used both for mouse events (click, enter, exit) and mouse motion events (moves and drags).

An event which indicates that a mouse action occurred in a component…

Field Summary

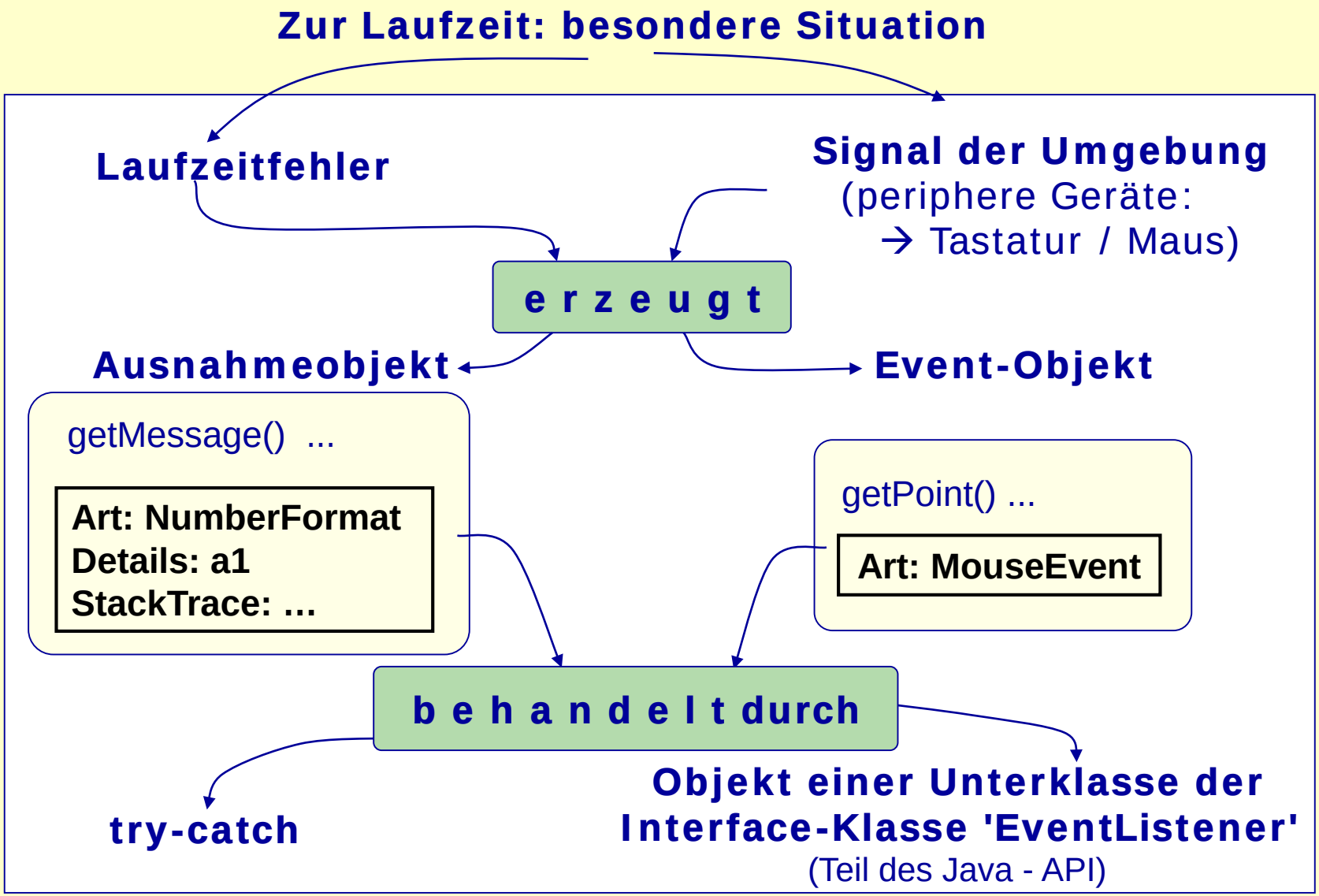
static int	**BUTTON1** Indicates mouse button #1; used by getButton().
static int	**BUTTON2** Indicates mouse button #2; used by getButton…
static int	**BUTTON3** Indicates mouse button #3… by getButton().
static int	**MOUSE_CLICKED** The "mouse clicked" event.
static int	**MOUSE_DRAGGED** The "mouse dragged" event.
static int	**MOUSE_ENTERED** The "mouse entered" event.
static int	**MOUSE_EXITED** The "mouse exited" event.
static int	**MOUSE_FIRST** The first number in the range of ids used for… events.
static int	**MOUSE_LAST** The last number in the range of ids used for mouse even…
static int	**MOUSE_MOVED** The "mouse moved" event.
static int	**MOUSE_PRESSED** The "mouse pressed" event.
static int	**MOUSE_RELEASED** The "mouse released" event.

`static int MOUSE_CLICKED`
The "mouse clicked" event.

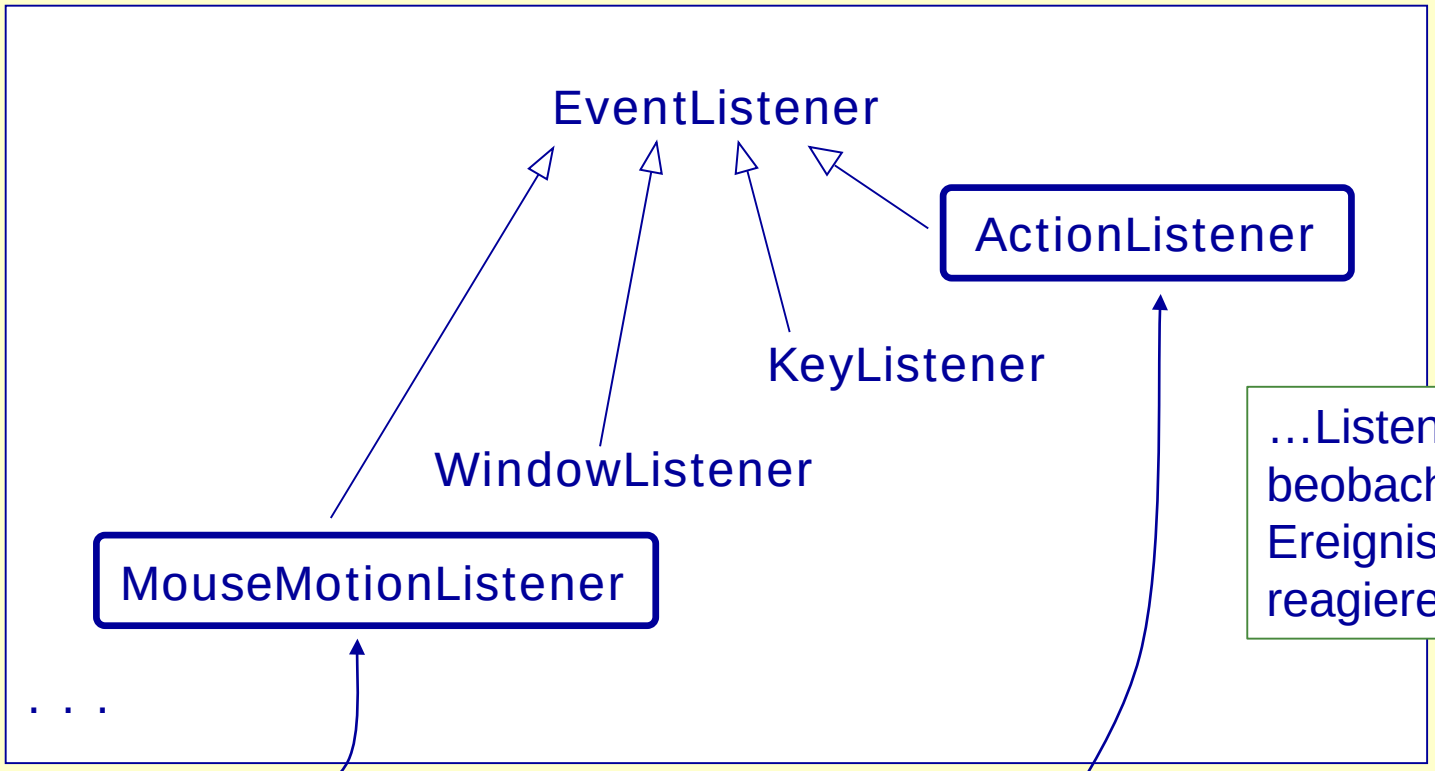
`static int MOUSE_MOVED`
The "mouse moved" event.

`Point getPoint()`
Returns the x,y position (curser) of the event relative to the source component.

Analogie: Ausnahmen - Ereignisse (Events)



(Interface-)Komponenten zur Behandlung von Ereignissen: EventListener-Hierarchie



…Listener: beobachten Ereignisse und reagieren

behandelt: MouseMotionEvents

behandelt: ActionEvents
(vgl. Temperatur–Applet)

… durch festgelegte Methodenaufrufe.

API-Klasse (Interface) ‚MouseMotionListener'

java.awt.event

Interface MouseMotionListener

public interface **MouseMotionListener**
extends EventListener

The listener interface for receiving mouse motion events on a component. (For clicks and other mouse events, use the MouseListener.)

Method Summary

void	**mouseDragged**(MouseEvent e) Invoked when a mouse button is pressed
void	**mouseMoved**(MouseEvent e) Invoked when the mouse cursor has been

The listener interface for receiving mouse motion events on a component. (For clicks and other mouse events, use MouseListener.)

Method Detail

mouseDragged

void **mouseDragged**(MouseEvent e)
Invoked when a mouse button is pressed on a component and then dragged. MOUSE_DRAGGED events will continue to be delivere
where the drag originated until the mouse button is released (regardless of whether the mouse position is within the bounds of th

Due to platform-dependent Drag&Drop implementations, MOUSE_DRAGGED events may not be delivered during a native Drag&D

`void mouseDragged(MouseEvent e)`
Invoked when a mouse button is pressed on a component and then dragged.

Ereignis übergeben

mouseMoved

void **mouseMoved**(MouseEvent e)
Invoked when the mouse cursor has been moved

`void mouseMoved(MouseEvent e)`
Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

Auswertung von MouseEvents durch MouseMotionListener

EyesApplet.java

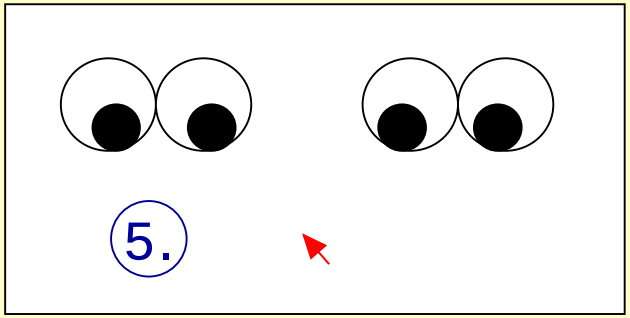
```
class EyesApplet extends Applet
   implements MouseMotionListener {

   public void mouseMoved (MouseEvent e){
      cursor = e.getPoint();
      repaint();
   }
  ...
}
```

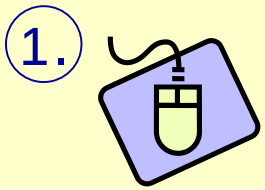
Aufrufreihenfolge bei Maus-Bewegung

- → Objekt e vom Typ MouseEvent gebildet;
  e enthält Informationen über Position des Cursors.
- → Aufruf mouseMoved(e)
- → neue Position der Maus abgefragt: e.getPoint
- → repaint(): ruft paint() erneut auf – Augen jetzt neu gemalt

Technik und Ablauf der Ereignisbehandlung

EyesApplet.java





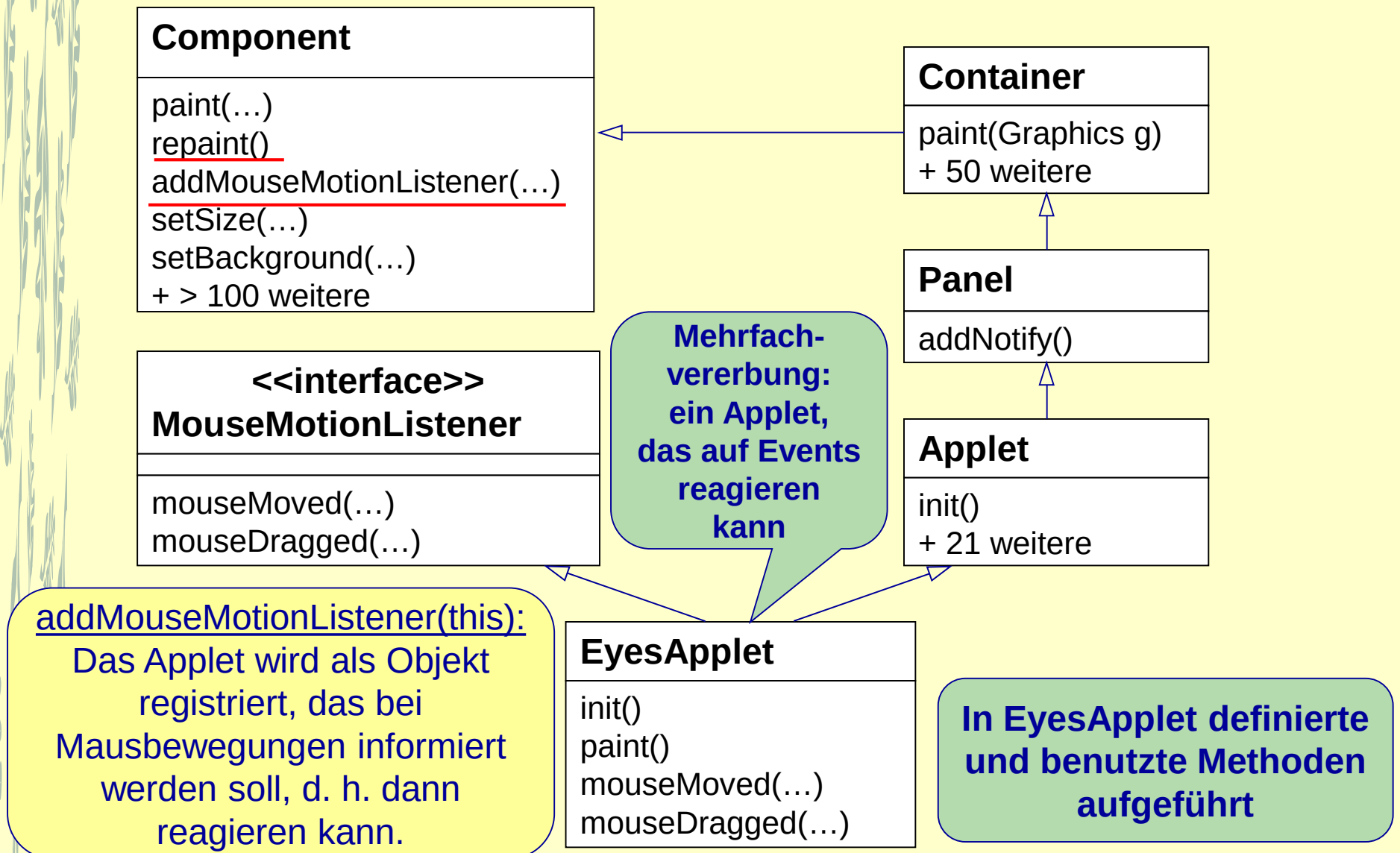
1. Hardware schickt Signal an Betriebssystem
2. Java-VM-Maschine empfängt Signal vom Betriebssytem
3. Java-VM-Maschine erzeugt Ereignis-Objekt e
4. Java-VM-Maschine sendet Botschaft an Objekt der Ereignisbehandlung (d.h. durch Aufruf einer Methode): z.B.

```
mouseMoved(e)
```

5. Ereignisbehandlungsmethode wertet Ereignis-Objekt aus und reagiert, z.B.

```
e.getPoint(); repaint();
```

SW-Architektur: erweiterte Sicht (API)



EyesApplet.java

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

public class EyesApplet extends Applet
              implements MouseMotionListener {
  Point cursor;
  Eyes e1, e2;

  public void init () {
    // Register the Listener.
    addMouseMotionListener(this);
    setSize(500,400);
    setBackground(Color.LIGHT_GRAY);
    e1 = new Eyes(new Point (63,30));
      // center of one eye
    e2 = new Eyes(new Point (437,30));
      // center of the other
    cursor = new Point(250, 2000);
      // initial cursor
  }

  public void paint(Graphics g) {
    e1.stare(g, cursor);
    e2.stare(g, cursor);
  }

  public void mouseMoved (MouseEvent e) {
    cursor = e.getPoint();
    repaint();
    }

  public void mouseDragged (MouseEvent e) {}
}
```

Eyes.java

```
public class Eyes {

  private Point left, right, leftPupil, rightPupil;
  private final int EYE_RADIUS = 30,PUPIL_RADIUS = 10;

  public Eyes(Point c) {
    left = new Point(c.x-EYE_RADIUS-3, c.y);
    right = new Point(c.x+EYE_RADIUS+3, c.y);
  }

  private void fillCircle (Graphics g,
                           Point center, int radius) {
    g.fillOval(center.x-radius, center.y-radius,
               2*radius, 2*radius);
  }

  public void stare (Graphics g, Point cursor) {

    // Draw the white eyes
    g.setColor(Color.WHITE);
    fillCircle(g, left, EYE_RADIUS);
    fillCircle(g, right, EYE_RADIUS);

    // Draw the pupils
    g.setColor(Color.black);
    leftPupil = compute (cursor, left);
    fillCircle(g, leftPupil, PUPIL_RADIUS);
    rightPupil = compute (cursor, right);
    fillCircle(g, rightPupil, PUPIL_RADIUS);
  }

  private Point compute (Point cursor, Point eye) {
    double d = Math.sqrt((cursor.x-eye.x)*(cursor.x-eye.x)
                         + (cursor.y-eye.y)*(cursor.y-eye.y));
    int r = EYE_RADIUS - PUPIL_RADIUS;
    return new Point (eye.x + (int)((cursor.x-eye.x)*r/d),
                      eye.y + (int)((cursor.y-eye.y)*r/d));
  }
}
```

Klasse 'Eyes': das Problem ?

(für das Zeichnen des Auges)

Cursor

Klasse 'Eyes': das Problem ?

(für das Zeichnen des Auges)

das Problem:

geg.: Augapfel + Cursorposition

ges.: Wo liegt die Pupille ?

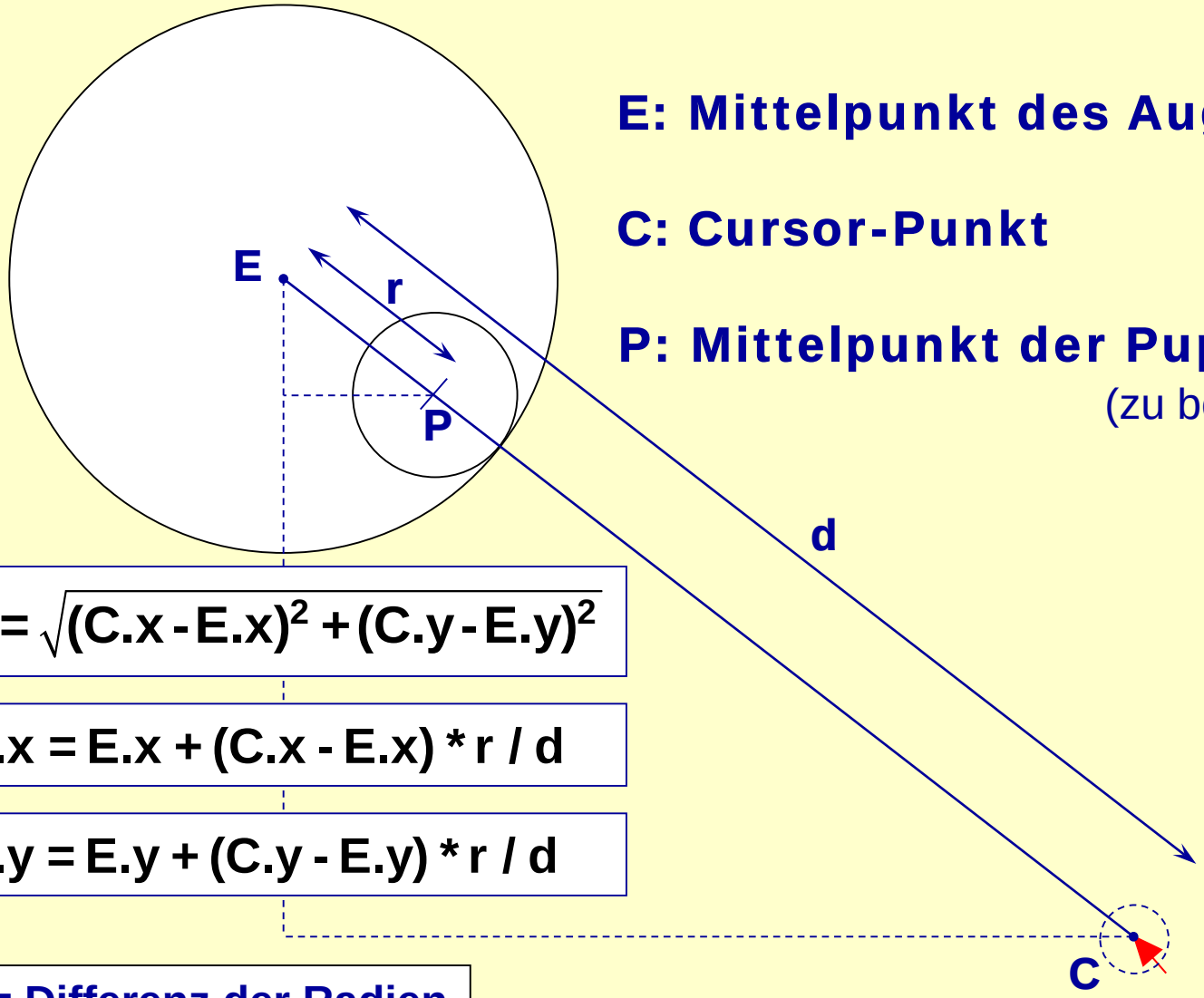
Mittelpunkt

→Satz des Pythagoras

→Ähnliche Dreiecke

Cursor

Klasse 'Eyes': das Problem ?



E: Mittelpunkt des Auges (Eye)

C: Cursor-Punkt

P: Mittelpunkt der Pupille
(zu berechnen)

$$d = \sqrt{(C.x - E.x)^2 + (C.y - E.y)^2}$$

$$P.x = E.x + (C.x - E.x) * r / d$$

$$P.y = E.y + (C.y - E.y) * r / d$$

r = Differenz der Radien

EyesApplet.java

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

public class EyesApplet extends Applet
              implements MouseMotionListener {
  Point cursor;
  Eyes e1, e2;

  public void init () {
    // Register the Listener.
    addMouseMotionListener(this);
    setSize(500,400);
    setBackground(Color.LIGHT_GRAY);
    e1 = new Eyes(new Point (63,30));
      // center of one eye
    e2 = new Eyes(new Point (437,30));
      // center of the other
    cursor = new Point(250, 2000);
      // initial cursor
  }

  public void paint(Graphics g) {
    e1.stare(g, cursor);
    e2.stare(g, cursor);
  }

  public void mouseMoved (MouseEvent e) {
    cursor = e.getPoint();
    repaint();
    }

  public void mouseDragged (MouseEvent e) {}
}
```

Eyes.java

```
public class Eyes {

  private Point left, right, leftPupil, rightPupil;
  private final int EYE_RADIUS = 30,PUPIL_RADIUS = 10;

  public Eyes(Point c) {
    left = new Point(c.x-EYE_RADIUS-3, c.y);
    right = new Point(c.x+EYE_RADIUS+3, c.y);
  }

  private void fillCircle (Graphics g,
                           Point center, int radius) {
    g.fillOval(center.x-radius, center.y-radius,
               2*radius, 2*radius);
  }

  public void stare (Graphics g, Point cursor) {

    // Draw the white eyes
    g.setColor(Color.WHITE);
    fillCircle(g, left, EYE_RADIUS);
    fillCircle(g, right, EYE_RADIUS);

    // Draw the pupils
    g.setColor(Color.black);
    leftPupil = compute (cursor, left);
    fillCircle(g, leftPupil, PUPIL_RADIUS);
    rightPupil = compute (cursor, right);
    fillCircle(g, rightPupil, PUPIL_RADIUS);
  }

  private Point compute (Point cursor, Point eye) {
    double d = Math.sqrt((cursor.x-eye.x)*(cursor.x-eye.x)
                         + (cursor.y-eye.y)*(cursor.y-eye.y));
    int r = EYE_RADIUS - PUPIL_RADIUS;
    return new Point (eye.x + (int)((cursor.x-eye.x)*r/d),
                      eye.y + (int)((cursor.y-eye.y)*r/d));
  }
}
```

stare und compute

Eyes.java

Malt Augenpaar neu:
- Alte Position der Pupille: schwarz → weiß
- Pupille blickt in Richtung ‚cursor' (schwarz)

```
public void stare (Graphics g, Point cursor)
```

Umsetzung der geometrischen Grundlagen:
Berechnet Punkt P (Mittelpunkt der Pupille)

```
private Point compute (Point cursor, Point eye)
```

fillCircle : Kreis mit 'radius' um 'center'-Punkt

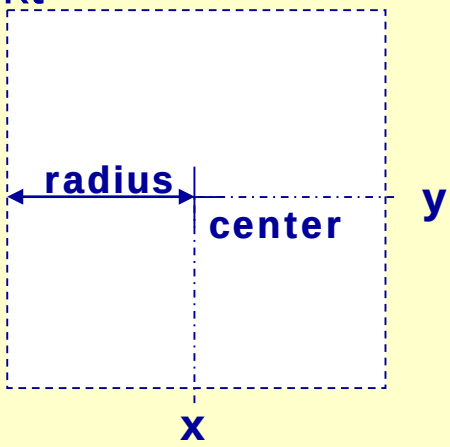
Eyes.java

```
private void fillCircle (Graphics g,
                         Point center,
                         int radius)
{
      g.fillOval(center.x-radius,
                 center.y-radius,
                 2*radius, 2* radius);
}
```

malt Kreis in Quadrat (Oval in Rechteck)

linker oberer Eckpunkt

Länge und Breite des Rechtecks



- Zu jedem Applet generiert JVM ein Graphics-Objekt g zum Malen im Applet

API-Klasse ‚Graphics'

java.awt
Class Graphics

java.lang.Object
└ java.awt.Graphics

public abstract class **Graphics**
extends Object

The Graphics class is the abstract base class for all g

The Graphics class is the abstract base class for all graphics contexts that allow an application to draw onto components ...

Constructor Summary

protected	**Graphics**() Constructs a new Graphics obj

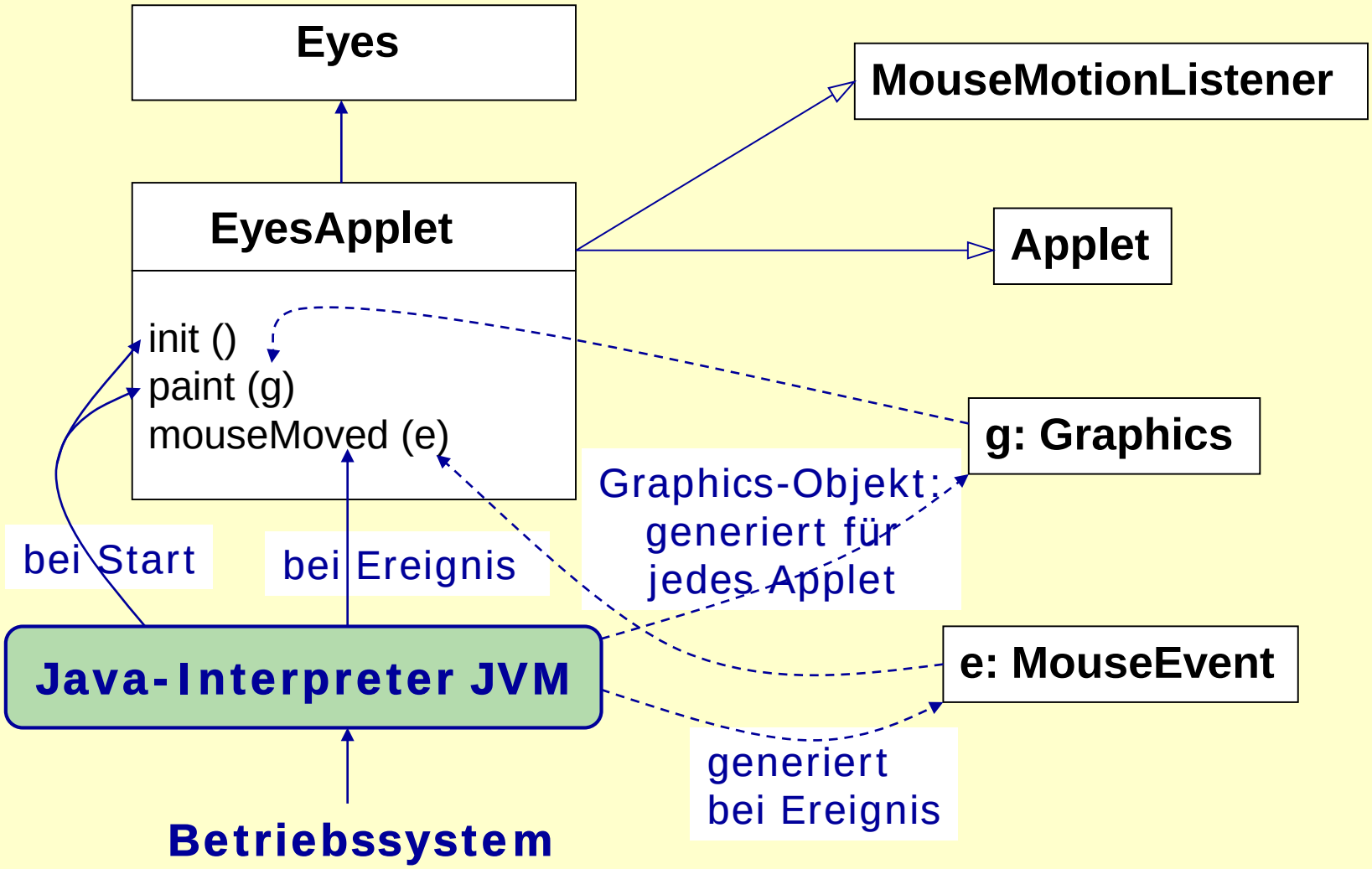
Method Summary

abstract void	**clearRect**(int x, int y, int width, int height) Clears the specified rectangle by filling it with the background color of the current drawing surface.
abstract void	**clipRect**(int x, int y, int width, int height) Intersects the current clip with the specified rectangle.
abstract void	**copyArea**(int x, int y, int width, int height, int dx, int dy) Copies an area of the component by a distance specified by dx and dy.
abstract Graphics	**create**() Creates a new Graphics object that is a copy of this Graphics object.
Graphics	**create**(int x, int y, int width, int height) Creates a new Graphics object based on this Graphics object, but with a
abstract boolean	**drawImage**(Image img, int x, int y, int width, int height, ImageObserver Draws as much of the specified image as has already been scaled to fit inside the spe
abstract void	**drawLine**(int x1, int y1, int x2, int y2) Draws a line, using the current color, between the points (x1, y1) and graphics context's coordinate system.
abstract void	**drawOval**(int x, int y, int width, int height) Draws the outline of an oval.
abstract void	**drawPolygon**(int[] xPoints, int[] yPoints, Draws a closed polygon defined by arrays
void	**drawPolygon**(Polygon p) Draws the outline of a polygon d by the specifie
abstract void	**drawPolyline**(int[] xP s, int[] yPoints, in Draws a sequence of connected lines defined by array
void	**drawRect**(int x, int y, int width, int height) Draws the outline of the specified rectangle.
abstract void	**drawRoundRect**(int x, int y, int width, int he Draws an outlined round-cornered rectangle using this g
abstract void	**drawString**(AttributedCharacterIterator iterator, int x, in Draws the text given by the specified iterator, using this graphics nt color.

`void drawRect(...)`
Draws the outline of the specified rectangle.

`abstract void fillOval(int x, int y, int width, int heigth)`
Fills an oval bounded by the specific rectangle with the current color

Dynamische Sicht auf 'EyesApplet'



Klasse EyesApplet: Aufruf der Methoden

EyesApplet.java

```
public class EyesApplet extends Applet
                  implements MouseMotionListener {

   Point cursor;
   Eyes e1, e2;

   public void init () {...}

   public void paint(Graphics g) {
        e1.stare(g, cursor);
        e2.stare(g, cursor);
   }

   public void mouseMoved (MouseEvent e) {
        cursor = e.getPoint();
        repaint();
   }

   public void mouseDragged (MouseEvent e)  {...}
```

Applet: initialisiert und registriert als MouseMotionListener

direkt nach **`init()`** aufgerufen

automatischer Aufruf bei Maus-Bewegung

liefert physische Mausposition

- Zu jedem Applet generiert JVM ein Graphics-Objekt g zum Malen im Applet
- Alle Methoden: durch JVM-Maschine aufgerufen

Mögliche Aufgaben zur Vertiefung

0. Programm anwenden und vollständig verstehen (Java-API)
1. Platzierung der Augen verändern
2. Andere Abmessungen: Größe von Pupille und Augapfel
3. Statt zwei Augenpaaren: vier …
4. Dynamische Erzeugung von neuen Augenpaaren (Maustasten)
5. Erscheinungsbild neu: Augenlid
6. Auf Maustastendruck: Augenlider schließen sich