Grundlagen der Programmierung (GP)

Prof. Dr. H. Schlingloff

Joachim Hänsel

17. 6. 2010

Kapitel 8: Java-Programmierung

8.1 Ereignisbehandlung, Benutzungsschnittstellen

8.2 Graphikprogrammierung

8.1 Ereignisbehandlung

- Benutzerinteraktion per Konsolschnittstelle

  Programm wartet bis Benutzer „return“ eingibt

  in Java:

```
BufferedReader konsole = new BufferedReader(new
InputStreamReader(System.in)) ;
try {n = Integer.parseInt(konsole.readLine());}
  catch (IOException e){}
```

- Benutzerinteraktion per GUI (graphical user interface)

  Viele verschiedene Eingabemöglichkeiten, Ereignisse

- ablaufgesteuerte Programmierung:

  Programm = Folge von Anweisungen (= Methodenaufrufen)

- ereignisgesteuerte Programmierung:

  Programm = Sammlung von Behandlungsroutinen

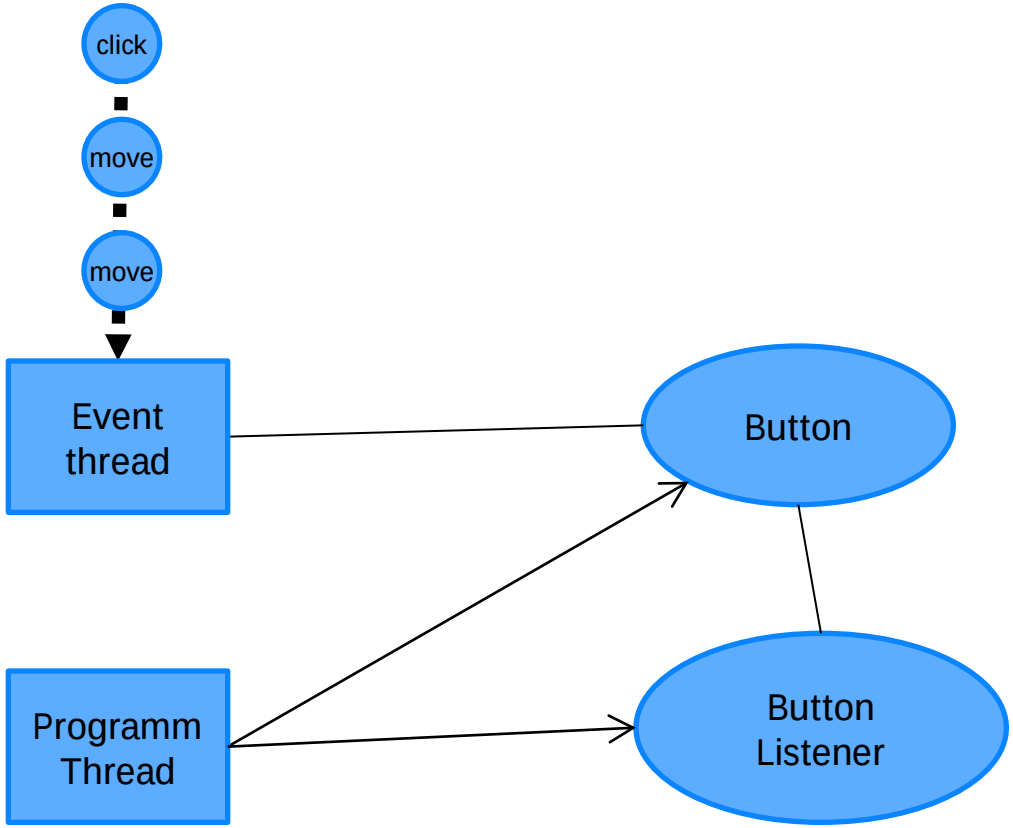
  (= Methoden) für verschiedene Ereignisse

Ereignisse (events)

- Mausklick, Mausziehen, Mausbewegung, Return-Taste, Tastatureingabe, ...
- programmerzeugte Ereignisse

- Für die GUI wird mindestens ein separater Thread gestartet, der auf die Ereignisse wartet und sie bearbeitet.
- Methoden zur Beobachtung bestimmter Ereignisse; explizite Anmeldung der Beobachter

Ereignisse (events)



Widget

- Ein Interaktionselement in einer GUI
- Fenster
- Menü
- Schaltfläche (Button)
- Bildlaufleiste (Scrollbar)
- ...

AWT und Swing

- zwei weit verbreitete Bibliotheken zur Realisierung von GUIs in Java
- AWT (Abstract Window Toolkit):

  Java Hülle um die größte *gemeinsame* Menge an Widgets der Betriebssysteme
- Swing:

  Umfangreichere Bibliothek an GUI Elementen; Was auf einem BS nicht vorhanden ist, wird basierend auf AWT Widgets emuliert

Fenster

```
import java.awt.*;
class Fenster extends Frame {
 Fenster (String titel) {
  super(titel);
  setSize(160,120);
  setVisible(true); }
}
public class FensterDemo {
 public static void main(String[] args) {
 Fenster zumHof = new Fenster("zum Hof");
 Fenster zurTür = new Fenster("zur Tür");
 }
}
```

Fensterbeobachter

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

class FensterBeobachter extends WindowAdapter{
  public void windowClosing(WindowEvent e) {
   System.exit(0);
  }
}
class Fenster extends Frame {
 Fenster (String titel) {
...
  FensterBeobachter fb = new FensterBeobachter();
  addWindowListener(fb);
...
```

Knöpfe und Knopfbeobachter

```
class KnopfBeobachter implements ActionListener{
 public void actionPerformed(ActionEvent e) {
   System.out.println("Knopf gedrückt!"); }
}

class Fenster extends Frame {
 Fenster (String titel) {
...
  Button knopf = new Button("Knopf!");
  knopf.addActionListener(new KnopfBeobachter());
  add(knopf);
...
```

aktive Komponenten

Zur Verwendung aktiver Komponenten (Knöpfe, Schalter, ...) sind immer folgende Schritte nötig

- Erzeugen – `Button b = new Button ()`
- Registrieren – `add(b)`
- Verbinden zur Ereignisbehandlung –

  `add....Listener(this)`
- Methode zur Ereignisbehandlung schreiben
  - `actionPerformed, itemStateChanged, adjustmentValueChanged,...`

anonyme Beobachter

```
class Fenster extends Frame {
 Fenster (String titel) {
...
  Button knopf2 = new Button("Knopf2");
  knopf2.addActionListener (new ActionListener (){
   public void actionPerformed(ActionEvent e) {
   System.out.println("Knopf2 gedrückt!"); }});
  add(knopf2);
...
```

Layout-Fragen

```
class Fenster extends Frame {
 Fenster (String titel) {
...
  setLayout(new FlowLayout());
  Button knopf2 = ...
```

mehrere verschiedene Layouts verfügbar!

- BorderLayout
- FlowLayout
- GridLayout
- CardLayout
- GridBagLayout

Schachtelung von Layouts möglich

Beispiel: eine Stoppuhr

- Zeitanzeige in Minuten:Sekunden:Hunderstel

```
class UhrzeitThread extends Thread{
   public void run(){
    while(running){
      elapsedTime = (System.currentTimeMillis() - startTime);
      csec = (int) (elapsedTime % 1000 / 10);
      sec = (int) (elapsedTime / 1000) % 60;
      min = (int) (elapsedTime / 60000);
      zeitAnzeigeCsec.setText(((csec < 10)? "0" : "") + csec);
      zeitAnzeigeSec.setText(" : " + ((sec < 10)? "0" : "") + sec + " : ");
      zeitAnzeigeMin.setText(((min < 10)? "0" : "") + min);
    }
  }
}
```

Knöpfe für Start/Stop und Reset

- class ButtonListenerStartStop implements ActionListener{
  public void actionPerformed(ActionEvent e){
  if (! running){
  running = true;
  startstop.setLabel("Stop");
  startTime = System.currentTimeMillis();
  UhrzeitThread uhrAnzeige = new UhrzeitThread();
  uhrAnzeige.start();
  } else {
  running = false;
  startstop.setLabel("Start");
  }
  }
  }
- class ButtonListenerReset implements ActionListener{
  public void actionPerformed(ActionEvent e){
  running = false;
  elapsedTime = 0;
  zeitAnzeigeMin.setText("00");
  zeitAnzeigeSec.setText(" : 00 : ");
  zeitAnzeigeCsec.setText("00");
  startstop.setLabel("Start");
  }
  }

Rahmenprogramm

```
class Zeitmessung extends Frame {
public Zeitmessung(){
    zeitAnzeigeMin = new Label();
    zeitAnzeigeMin.setBounds(50, 100, 120, 200);
    add(zeitAnzeigeMin);
   // ... und genauso für zeitAnzeigeSec, zeitAnzeigeCsec

    startstop = new Button("Start");
    startstop.setBounds(50,650,100,50);
    startstop.addActionListener(new ButtonListenerStartStop());
    add(startstop);

    reset = new Button("Reset");
    reset.setBounds(290, 650, 100, 50);
    reset.addActionListener(new ButtonListenerReset());
    add(reset);


public static void main(String[] args) {
    Zeitmessung uhr = new Zeitmessung();
    uhr.setBounds(0, 0, 400, 400);
    uhr.setVisible(true);
  }
}
```

8.2 Graphikprogrammierung

- Mausevents werden nach dem selben Schema behandelt

```
addMouseListener(new MouseAdapter() {
  public void mousePressed(MouseEvent e) {
   x1 = e.getX(); y1 = e.getY(); } ...
```

- Ausgabe graphischer Elemente (Linien, Kreise, Vielecke, Polygone...) mit Objekt der Klasse Graphics
- Methoden: drawLine, fillPolygon, drawOval, ...
- Konstruktor getGraphics()

ein Malprogramm

```
...
addMouseListener(new MouseAdapter() {
   public void mousePressed(MouseEvent e) {
    x1 = e.getX(); y1 = e.getY();   }
   public void mouseReleased(MouseEvent e) {
    x2 = e.getX(); y2 = e.getY();
    Graphics g = getGraphics();
    g.setColor(Color.blue);
    int w=Math.abs(x1-x2), h=Math.abs(y1-y2);
    switch (modus) {
    case 1: g.fillRect(x1,y1,w,h); break;
    case 2: g.fillOval(x1,y1,w,h); break;
    } } });
...
```

Animationen

- Bei Veränderung des Fensters (z.B. Vergrößern) geht die Zeichnung verloren
  - ➔`paint (Graphics g)` zur Ausgabe der Zeichnung, wird bei Fensterveränderungen automatisch aufgerufen
  - ➔paint muss so programmiert werden, dass es alle Zeichnungsobjekte ausgibt
  - ➔Eintrag von Zeichnungsobjekten in Collection (z.B. Set), iterative Ausgabe aller Elemente
- Die selbe Methode kann für Animationen benutzt werden
  - Aufruf von paint() in eigenem Thread, alle 40 ms