

Event-Driven, Object-Orientated Programming

Unstrukturiertes Tagebuch

worgtsone @ hush.com

06.03.2005 – 13. Oktober 2011

Inhaltsverzeichnis

1	28.10.2007 : Lösung des Fahrstuhl-Problems in Java	2
1.1	Aufgabe	2
1.2	Ansatz	2
1.3	Quelltext	3
2	03.03.2005 : Zweck	6
3	Definition	6
4	Events	6
5	Kommunikation der Objekte untereinander	7
6	Beispiele	7
6.1	Beispiel Kreise	7
6.2	Beispiel Fahrstuhl	7
6.3	Bildbetrachter und Taschenrechner	7
7	Formale Programmbeschreibung	8
7.1	Kreise	8
7.2	Mehrfädigkeit (Multi-Threading)	8
7.3	Fahrstuhl	9
7.3.1	Erste Liste - obsolet	9
7.3.2	Zweite Liste - besser	10
7.4	Das Puffer-Problem	10
7.5	Das Kommunikations-Problem	10
7.6	Stockwerk-Listen	11
7.7	Pseudocode-Notation	11

1 28.10.2007 : Lösung des Fahrstuhl-Problems in Java

1.1 Aufgabe

Gegeben ist ein Hochhaus mit 10 (oder N) Stockwerken und einem Fahrstuhl.

Die Stockwerke sind von 1 bis 10 numeriert.

In allen Stockwerken außer 1 sind "WillAb"-Knöpfe.

In allen Stockwerken außer 10 sind "WillAuf"-Knöpfe.

In der Kabine sind 10 Knöpfe für die 10 Stockwerke.

Alle Knöpfe leuchten auf, wenn sie gedrückt werden, und gehen wieder aus, wenn ihr Auftrag erfüllt ist.

Der Aufzug fährt nach oben, bis er nicht mehr nach oben fahren muß. Dann fährt er abwärts, bis er nicht mehr muß.

Bauen Sie das in Java.

1.2 Ansatz

```
class Aufzug {                                // one Object only
    public Aufzug () {                        // all that runs is constructor
        initialize();
        direction=up;
        boolean run=true;
        while (run) {
            getNextEtage();
            eventuallyResetButtons();
            goToNextEtage();
            eventuallyResetButtons();
            openDoors();
            userMadeAction();
            closeDoors();
        }
    }
    public void userMadeAction() {
        if (action in a to h ) setUpButton    (n(action));
        if (action in n to w ) setDownButton  (n(action));
        if (action in 1 to 10) setNumberButton(n(action));
    }
}
```

1.3 Quelltext

[Kommentar 28.10.2007 : da ist noch ein Fehler drin. Könnte Sie ihn beschreiben? Lokalisieren?? Oder gar terminieren???)

```

class Aufzug{
// data
int N = 9;
    boolean[] up = new boolean[N + 1];           // up button
    boolean[] dn = new boolean[N + 1];           // down button
    boolean[] cb = new boolean[N + 1];           // cabin button
String direction = "up";
int position = 1;
int sleepy = 100;

public void showAll () {
    System.out.println (N + " etages. pos=" + position + ". dir=" +
        direction);
    System.out.print ("Down : ");
    for (int i = 2; i <= N; i++)
        if (dn[i])
            System.out.print (" " + i);
    System.out.println ();
    System.out.print ("Up   : ");
    for (int i = 1; i < N; i++)
        if (up[i])
            System.out.print (" " + i);
    System.out.println ();
    System.out.print ("Cabin: ");
    for (int i = 1; i <= N; i++)
        if (cb[i])
            System.out.print (" " + i);
    System.out.println ();
}

void initialize () {
    for (int i = 1; i <= N; i++) {
        up[i] = false;
        dn[i] = false;
        cb[i] = false;
    }
    direction = "up";
}

int getNextUpwards () {
    for (int i = position + 1; i <= N; i++)
        if (up[i] || cb[i])
            return i;
    for (int i = position + 1; i <= N; i++)
        if (dn[i])
            return i;
return 0;
}

int getNextDownwards () {
    for (int i = position - 1; i > 0; i--)
        if (dn[i] || cb[i])
            return i;
    for (int i = position - 1; i > 0; i--)
        if (up[i])
            return i;
return 0;
}

int getNextEtage () {
/* fun stuff start
we got to examine
o if we are in 10: direction=down, get next downwards. if there is
  none, do nothing
o else if we are in 1: direction=up, get next upwards. none? do nothing.
o else
- if up: g=getNextUpwards
  if g != 0 return g
  else direction=down; g=getNextDownwards;
  if g!=0 return g
*/
}

```

```

        else return 0
    - else (that is : down)
      g=next downwards
      if g!=0 return g else
        direction=up
        g=next upwards
        return g          ( yes, even in case theres nothing to do,
                          because there is really nothing to do. )
*/
    if (position == N)      {
        direction = "down";
        dn[N] = false;
        return getNextDownwards ();
    }
    if (position == 1)      {
        direction = "up";
        up[1] = false;
        return getNextUpwards ();
    }
// neither 1 nor N
    if (direction == "up")  {
        int g = getNextUpwards ();
        if (g != 0)
            return g;
        else {
            direction = "down";
            return getNextDownwards ();
        }
    } else { //direction==down
        int g = getNextDownwards ();
        if (g != 0)
            return g;
        else {
            direction = "up";
            return getNextUpwards ();
        }
    }
}

public Aufzug (int N)  { // one Object only, run only constructor
    initialize ();
    String direction = "up";
    boolean run = true;
    while (run)        {
        userMadeAction ();
        showAll ();
        int nextEtage = getNextEtage ();
        if (nextEtage > position)
            direction = "up";
        if (nextEtage < position)
            direction = "down";
// set some switches
        if (direction == "up")
            up[position] = false;
        else
            dn[position] = false;

        if (nextEtage == 0)
            System.out.println ("Going nowhere...");
        else {
            System.out.println ("Go from " + position + " to " + nextEtage);
// set some switches
            if (direction == "up")
                up[position] = false;
            else
                dn[position] = false;
            position = nextEtage;
// set some switches
            if (direction == "up")
                up[position] = false;
            else
                dn[position] = false;
                cb[position] = false;
        }
        s (sleepy);
        sleepy = sleepy * 9 / 10;
    }
}

```

```
        if (sleepy < 10)
            sleepy = 10;
    }

    public void userMadeAction () {
// automatic
        int r1 = (int) (Math.random () * 3);
        if (r1 == 0) { // up
            int r2 = (int) (Math.random () * (N - 1) + 1);
            up[r2] = true;
        }
        if (r1 == 1) { // down
            int r2 = (int) (Math.random () * (N - 1) + 2);
            dn[r2] = true;
        }
        if (r1 == 2) { // cabin
            int r2 = (int) (Math.random () * (N) + 1);
            cb[r2] = true;
        }
    }

    public static void main (String[]args) {
        Aufzug a = new Aufzug (10);
    }

    public void s (int msecs) {
        try { Thread.sleep (msecs); } catch (Exception e) { }
    }
}
```

2 03.03.2005 : Zweck

Über die Hälfte aller Software-Projekte gehen in die Hose. Die Gründe sind vielfältig:

- das Pflichtenheft wird nicht erfüllt oder erfüllt nicht das, was sich der Kunde vorgestellt hat.
- Es gibt unterschiedliche Auffassungen über die Bedeutung des Pflichtenheftes zwischen Auftraggeber (Käufer) und Programmierer;
- rätselhafte Datenverluste und gelegentliche Fehler;
- keine User-Akzeptanz;
- ... viele mehr, und Kombinationen.

Die Software-Industrie hat reagiert und stellt EDOO-Plattformen bereit.

In diesem Artikel möchte ich Konzepte untersuchen, Beispiele bewerten und mit etwas Benutzbarem herauskommen.

[Kommentar 28.10.2007 : und es gelingt mit überhaupt nicht.]

3 Definition

EDOO (Event Driven Object-Oriented Programming) ist jedes Programm, mit dem der Benutzer interagiert.

Diese Definition umschließt alle Interaktionsgeräte (Bildschirm, Tastatur, Maus), alle Rechnerplattformen (Workstation, Webserver, Grid, ...) und alle Programmiersprachen.

Ich werde mich jedoch auf banale Sachen wie Schaltflächen, Bilder, Radio- und Checkboxen, Eingabefelder etc. konzentrieren.

4 Events

Events (Ereignisse) sind Sachen, die passieren, zB Mausklicks oder Eingaben.

Ich verwende ein simples Event-Modell:

- Alle Objekte können jeden Event mithören.
- Alle Objekte können Events erzeugen.
- Die Events werden von der darunterliegenden System-Schicht gemanaged, wir müssen uns nicht kümmern.

Das bedeutet unter anderem, daß sie in eine Reihenfolge gebracht werden und vom Programm in dieser Reihenfolge abgearbeitet werden.

Mit ist bekannt, daß es noch andere Kommunikations-Modelle gibt. ZB daß ein Mausklick auf eine Schaltfläche eine Funktion in einem Objekt aufruft.

Ich sehe allerdings keinen Unterschied zwischen beiden. Von einer korrekten Über-Sprache verlange ich, daß sie das transparent handhabt.

5 Kommunikation der Objekte untereinander

Die Objekte kommunizieren untereinander mit get()- und set()-Methoden.

[Kommentar 28.10.2007 : blöde Idee. Objekte, die Botschaften bekommen (ich darf wohl auch Emails sagen) und gelegentlich aufgerufen werden: Ey, arbeite deine Emails ab, sind schlauer.

Leider brauchts dazu `libmessages.so` o.ä.]

6 Beispiele

6.1 Beispiel Kreise

Ohne Interaktion. Öffnet ein Fenster und zeichnet Kreise.

Die Komfort-Version hat einen "Schließen"-Knopf.

Die Luxus-Version läßt die Kreise herumdriften, gegeneinander und gegen den Rand dotzen.

6.2 Beispiel Fahrstuhl

Gegeben ist ein Fahrstuhl in einem Gebäude mit 10 Stockwerken, numeriert von 1 bis 10.

Auf jedem Stockwerk sind 2 Knöpfe: WillAuf und WillAb. Außer im 1. und im 10.

Im Fahrstuhl sind 10 Knöpfe, numeriert von 1 bis 10.

Die Knöpfe werden zweckmäßig beleuchtet.

Der Fahrstuhl steht zu Beginn im Erdgeschoß, Türen zu.

Wenn Leute fahren wollen, fährt der Fahrstuhl zunächst nach oben, hält dabei zweckmäßig an, fährt anschließend nach unten, hält dabei zweckmäßig an, und weiter am Anfang dieses Satzes. Die Knöpfe werden dabei sinnvoll entleuchtet.

6.3 Bildbetrachter und Taschenrechner

Kann Bilder laden, anzeigen, vergrößern, verkleinern, normalisieren, invertieren, speichern.

Taschenrechner halt. Von Kaufhaus-Modell über 1. Luxusstufe=Backspace-Taste bis hin zum wissenschaftlichen Supermodell mit umgekehrter polnischer Notation.

7 Formale Programmbeschreibung

Jedes EDOO-Programm läuft nach folgendem Schema ab:

```
init;  
gib dem User etwas zum Klicken;  
solange (wir leben)  
    verarbeite Events;  
schluß.
```

7.1 Kreise

```
Öffne ein Fenster;  
solange (wir leben)  
    zeichne ein paar Kreise;  
    schlafe ein bißchen;  
    verarbeite Events;  
schluß.
```

Hier sehen wir eins der ersten Probleme: Während das Programm schläft, reagiert es nicht auf Events.

7.2 Mehrfädigkeit (Multi-Threading)

Ein Programm, das nicht auf "Abbrechen" reagiert, wird von den meisten Benutzern doof gefunden.

Das Schlafen und das Events-Verarbeiten sollte es gleichzeitig tun. Das ist besonders sichtbar, wenn es eine Million Werte sortiert und der User gerade "Abbrechen" geklickt hat.

- Das Programm sollte (pseudo-)mehrfädig (multi-threaded) sein.

Mehrfädigkeit ist ein Komfort-Merkmal und für blanke Funktion nicht notwendig. Außer im Pflichtenheft steht etwas wie "muß binnen einer Sekunde auf jede Eingabe reagieren".

Solange wir das in der Über-Sprache beschreiben können, ist es eigentlich egal, wie es umgesetzt wird.

Die SparVersion wird wenigstens die Events in einem Extra-Faden verarbeiten. (Wie wärs mit einer Super-Status-Zeile namens ImmediateResponse, die bei jedem Klick etwas sagt???)

7.3 Fahrstuhl

[Kommentar 28.10.2007 : große Katastrophe. Das Kapitel vom 28.10.2007 zeigt, wies richtig geht.]

Verwenden wir zunächst mal Prosa zur Beschreibung.

7.3.1 Erste Liste - obsolet

Ich schlage folgende Objekte vor:

- Boss. Boss leistet `init`, dh er erschafft die andern Objekte und stößt die Hauptschleife an. Nicht mehr und nicht weniger.
- Knopf.

Wenn er gedrückt wird, geht das Licht an.

Wenn die Kabine auf der passenden Etage in der passenden Richtung anhält, geht es wieder aus.
- Knopf::Raufknöpfe.

Sie können rufen: "Ich wurde gedrückt." Oder besser: "Ein Benutzer im 7. Stockwerk möchte aufwärts fahren." Oder besser: "18, 7". 18 steht dabei für WillAuf, 7 steht fürs Stockwerk.

Daß jedes Event aus 2 Integers besteht, ist natürlich reine Willkür.
- Knopf::Runterknöpfe.

Sie können rufen: "17, 5". 11 steht dabei für WillAb, 5 steht fürs Stockwerk.
- Knopf::Kabinenknöpfe.

Sie rufen: "19, 9", dh: jemand WillZu Stockwerk 9.
- Kabine.

Kann rufen: "BinInStockwerkUndÖffneUndSchließeJetztDieTüren, 3.", "Wohin Jetzt?"
- Kabinentür.

Ruft: "Öffne", "Schließe", "BinOffen", "BinZu."
- RaufListe.

Hört auf die RaufKnöpfe.
- RunterListe.

Hört auf die Runterknöpfe.
- Kabinenliste.

Hört auf die Kabinenknöpfe.
- SchlauesTeil.

Erzeugt Fahrtrichtung und anzustuerndes Stockwerk.

Hört auf Kabine: "Wohin jetzt?"

7.3.2 Zweite Liste - besser

- Boss.
Erzeugt die Objekte und geht anschließend in eine Endlosschleife (while (wir leben ...)).
- Randomizer.
Für den Testbetrieb. Schläft die meiste Zeit, erzeugt manchmal Knopfdrücke.
- Knöpfe.
Brauchen wir gar nicht.
Entweder sie werden in echt gedrückt (dann sind sie Events), oder der Randomizer erzeugt sie durch Zugriff auf die `setButton()`-Methoden von `Scheduler`.
- Scheduler (oder: SchlauesTeil).
Puffert die Knopfdrücke und stellt in einer `get()`-Methode das nächste anzufahrende Stockwerk zur Verfügung.
- Kabine.
Fährt, hält manchmal an, macht Türen auf und zu.

7.4 Das Puffer-Problem

Wir brauchen einen Puffer. Naja – eine Art Puffer. Wenn zB der Fahrstuhl in 3 ist und jemand drückt WillAuf5 und dann WillAuf7, dann müssen beide Knopfdrücke irgendwo zwischengespeichert werden.

Ein Mensch würde das wie folgt handhaben: In der AufListe Stockwerk 5 und 7 als Gewünschte Station eintragen.

7.5 Das Kommunikations-Problem

Unter Menschen funktioniert folgendes ganz einfach:

Kabine: "Ich fahr gerade hoch. Bin In Etage 3. He, AufListe, wo fahre ich als nächstes hin?"

Aufliste: "Fahr nach 5."

(Kabine fährt.)

Kabine: "Ich fahr gerade hoch. Bin In Etage 5. He, AufListe, wo fahre ich als nächstes hin?"

Aufliste: "Fahr nach 7."

(Kabine fährt.)

Kabine: "Ich fahr gerade hoch. Bin In Etage 7. He, AufListe, wo fahre ich als nächstes hin?"

Aufliste: "Habe kein Ziel über 7."

Kabine: "Dann dreh ich meine Fahrtrichtung herum. He, AbListe, wo fahre ich als nächstes hin?"

Abliste: "Bin leer."

Hoppla! Wenn jemand in 3 auf will, wird er dort lange warten.

7.6 Stockwerk-Listen

Die brutal einfache Liste fährt die gewünschten Stockwerke nacheinander an – ohne Optimierung.

Die schlaudere Liste sortiert neu hereinkommende Ereignisse in eine bestehende Schlange ein.

Ich möchte die Liste aber so machen, daß sie auf Anfrage das nächste anzufahrende Stockwerk ausgibt (und dabei die Knöpfe fürs aktuelle Stockwerk löscht).

Nehmen wir an, die Kabine ist in 3 und fährt aufwärts, und es treffen folgende Wünsche ein:

willZu5 , willAuf7 , willAuf2 , Kabine ist fertig in 5 , willZu3 , willAb7 , willAb2 , willZu9.

7.7 Pseudocode-Notation

Das klingt alles ganz einfach. Wie wollen wir es aufschreiben? Und wo verstecken wir die Fahrtrichtung? Ich entscheide mich erstmal für *Scheduler* - wenn das Blödsinn ist, kann mans ja noch ändern.

```
class Boss {
  Boss () {
    rm = new Randomizer();
    erzeuge alle Auf-, Ab- und Kabinenknöpfe;
    sched = new Scheduler();
    kab = new Kabine();
    while (wir leben) {
      schlafe eine Sekunde;
      rm.erzeugeVielleichtEinenKnopfdruck
    }
  }
}
```

```
class Randomizer {
  Randomizer () { }

  erzeugeVielleichtEinenKnopfdruck () {
    if (willst du) {
      sorte = rand (auf, ab, kabine);
      nummer = rand (sorte);
      sched.setSortenknopf (sorte, nummer);
    }
  }
}
```

```
class Scheduler {
  Scheduler () {
    uplist = new list [1..n-1];
    dnlist = new list [2..n];
    kalist = new list [1..n];
    direct = new direction (up);
  }
}
```

```

}

setSortenknopf (sorte, nummer) {
  if (sorte == up) {
    uplist[nummer] = true;
  } ...
}

getNextEtage (currentEtage) {
  while (wir müssen nirgendwo hin)
    schlafe bis wir irgendwo hin müssen;
  else
    if (direction==up)
      if (getNextUpEtage != 0)
        return (getNextUpEtage());
      else
        direction=dn;
        return (getNextDnEtage());
    if (direction==dn)
      if (getNextDnEtage != 0)
        return (getNextDnEtage());
      else
        direction=up;
        return (getNextUpEtage());
}

getNextUpEtage (currentEtage) {
  for (etage=currentEtage+1; etage <= topEtage; etage++) {
    if (wirMüssenZuEtage(etage))
      return (etage);
  }
  return 0;
}

getNextDnEtage (currentEtage) {
  for (etage=currentEtage-1; etage >= topEtage; etage--) {
    if (wirMüssenZuEtage(etage))
      return (etage);
  }
  return 0;
}

wirMüssenZuEtage (etage) {
  if (kalist[etage] || (
    (direction==up) && uplist[etage]
  ) || (
    (direction==dn) && dnlist[etage]
  ) ) {
    return true;
  } else {
    return false;
  }
}

```

```
}  
}
```