

Entwurf und Implementierung paralleler Programme

Prof. Dr. Rolf Hennicker

23.04.2009

Kapitel 2

Prozesse und Java-Threads

2.1 Prozessbegriff

Prozess:

Programm in Ausführung

Prozesszustand (zu einem Zeitpunkt):

Wird charakterisiert durch die Werte von

- expliziten Variablen (vom Programmierer deklariert)
- impliziten Variablen (Befehlszähler, organisatorische Daten)

Zustandsübergang (eines Prozesses):

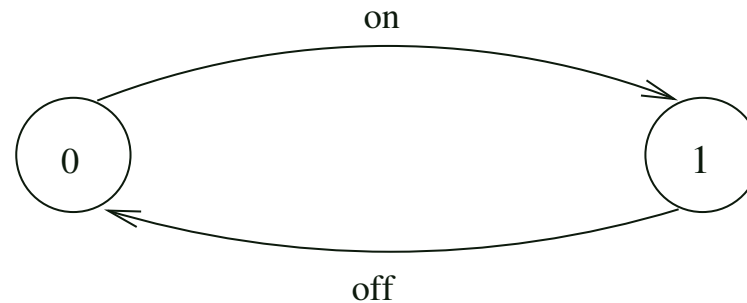
Wird von einer Aktion bewirkt. Aktionen sind elementar, d.h. nicht unterbrechbar.

Bemerkung:

- Im Gegensatz zu UML wird hier nicht zwischen Ereignis und Aktion unterschieden.
- Im Folgenden abstrahieren wir von den konkreten Zustandsdarstellungen.

2.2 Modellierung durch endliche Zustandsmaschinen

Beispiel: Lichtschalter als Zustandsmaschine



Aktionsfolge (Ablauf, “trace“): $\text{on} \rightarrow \text{off} \rightarrow \text{on} \rightarrow \text{off} \rightarrow \dots$

Konvention:

Zustände werden in der graphischen Darstellung von 0 beginnend durchnummeriert. 0 ist der Anfangszustand.

Beachte:

- Wir betrachten nur Prozesse mit endlich vielen Zuständen und endlicher Menge von Aktionen.
- Das Verhalten eines Prozesses kann aber unendlich sein (nicht terminierend).

Die hier betrachteten Zustandsmaschinen sind formal *endliche markierte Transitionssysteme* (“labelled transition systems“), abgekürzt LTS.

Definition:

Sei $States$ eine universelle Menge von Zuständen und ACT eine universelle Menge von Aktionen. Ein endliches LTS ist ein Quadrupel

$$\langle S, A, \Delta, q \rangle,$$

wobei

- $S \subseteq States$ eine endliche Menge von Zuständen ist
- $A \subseteq ACT$ eine endliche Menge von Aktionen ist
- $\Delta \subseteq S \times A \times S$ eine Übergangsrelation ist
- $q \in S$ ein Anfangszustand ist

Beispiel: Lichtschalter

2.3 Prozessausdrücke

Prozesse werden kompakt beschrieben durch Ausdrücke der Sprache FSP (“finite state processes”) [Magee, Kramer].

FSP orientiert sich

- syntaktisch an CSP [Hoare]
- semantisch an CCS [Milner]

Die *Semantik* eines Prozessausdrucks E wird durch Übersetzung in ein LTS gegeben.

Im Folgenden werden Prozessausdrücke induktiv definiert.

Jedem Prozessausdruck E wird eine Menge von freien Variablen $FV(E)$ zugeordnet.

Konstante Prozessausdrücke und Prozessidentifikatoren

Sei PID eine universelle Menge von Prozessidentifikatoren (Bezeichnern).

Definition:

1. STOP ist ein (konstanter) Prozessausdruck mit $FV(\text{STOP}) = \emptyset$.
2. Jeder Prozessidentifikator $P \in \text{PID}$ ist ein Prozessausdruck mit $FV(P) = \{P\}$.

Wirkung:

Aktionspräfix

Definition:

Ist $a \in \text{ACT}$ eine Aktion und E ein Prozessausdruck, dann ist das *Aktionspräfix* $(a \rightarrow E)$ ebenfalls ein Prozessausdruck mit $\text{FV}((a \rightarrow E)) = \text{FV}(E)$.

Statt von Prozessausdrücken sprechen wir häufig kurz von „Prozessen“.

Wirkung:

Der Prozess $(a \rightarrow E)$ engagiert sich zunächst in die Aktion a und verhält sich dann wie E .

Beispiele:

Auswahl

Definition:

Sind a_1, \dots, a_n Aktionen und E_1, \dots, E_n Prozessausdrücke, dann ist $(a_1 \rightarrow E_1 \mid \dots \mid a_n \rightarrow E_n)$ ein Prozessausdruck mit $FV((a_1 \rightarrow E_1 \mid \dots \mid a_n \rightarrow E_n)) = FV(E_1) \cup \dots \cup FV(E_n)$.

Wirkung:

Der Prozess engagiert sich entweder

- in a_1 und verhält sich danach wie E_1 oder
- in a_2 und verhält sich danach wie E_2 oder
- \vdots
- in a_n und verhält sich danach wie E_n .

Beispiel:

Prozessausdrücke mit Rekursion

Werden nur als Hilfskonstrukt zur Definition der Semantik von Prozessidentifikatoren P im Kontext einer rekursiven Prozessdeklaration " $P = E$." benötigt.

Definition:

Sei P ein Prozessidentifikator und E ein Prozessausdruck, so dass $P \in FV(E)$.

Dann ist $\text{rec}(P = E)$ ein Prozessausdruck mit $FV(\text{rec}(P = E)) = FV(E) \setminus \{P\}$.

(Rekursive) Prozessdeklarationen

z.B. SWITCH = (on \rightarrow off \rightarrow SWITCH).

Definition:

Ist P ein Prozessidentifikator und E ein Prozessausdruck, dann ist

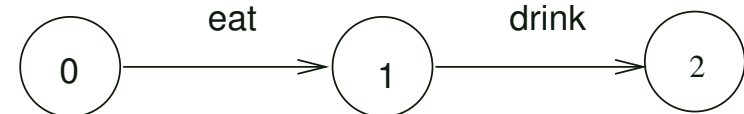
$$P = E.$$

eine *Prozessdeklaration*. Die Deklaration ist *rekursiv*, wenn P in dem Ausdruck E frei vorkommt, d.h. $P \in FV(E)$.

Beispiele:

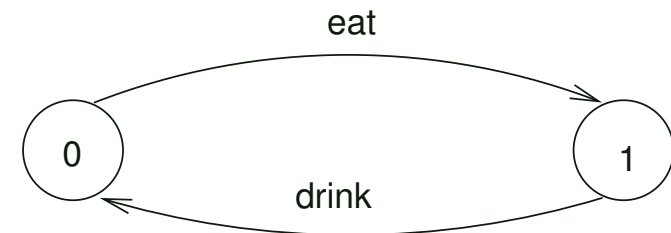
1. PERS = (eat \rightarrow drink \rightarrow STOP).

Dann ist das LTS von PERS gegeben durch:



2. PERSON = (eat \rightarrow drink \rightarrow PERSON).

Dann ist das LTS von PERSON gegeben durch:



Äquivalente Prozessbeschreibung mit lokalen Prozessdeklarationen:

PERSON = EATING,
EATING = (eat → DRINKING),
DRINKING = (drink → PERSON).

Beispiel (DRINKS):

DRINKS = (red → coffee → DRINKS | blue → tea → DRINKS).

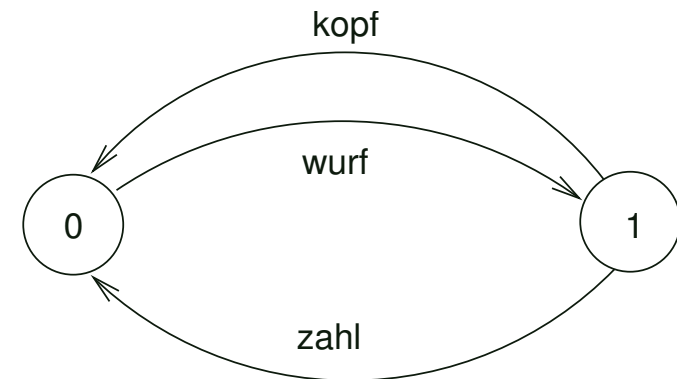
Zugehöriges LTS:

Bemerkungen:

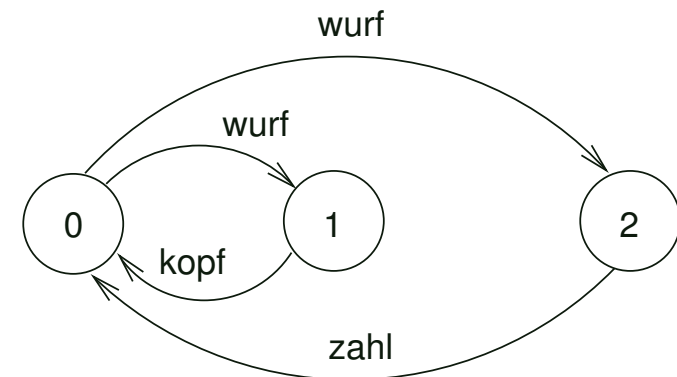
- blue, red $\hat{=}$ Input-Aktionen (der Umgebung)
- coffee, tea $\hat{=}$ Output-Aktionen
- Häufig beginnen die Alternativen einer Auswahl mit Input-Aktionen.
- Im Beispiel DRINKS gibt es unendlich viele mögliche Abläufe (“traces“):
 - red \rightarrow coffee \rightarrow red \rightarrow coffee \rightarrow ...
 - red \rightarrow coffee \rightarrow blue \rightarrow tea \rightarrow blue \rightarrow
...
 - blue \rightarrow tea \rightarrow red \rightarrow coffee \rightarrow ...
 - ...
 - ...

Beispiel (Münzwurf):

MÜNZE1 = (wurf \rightarrow (kopf \rightarrow MÜNZE1
| zahl \rightarrow MÜNZE1)).



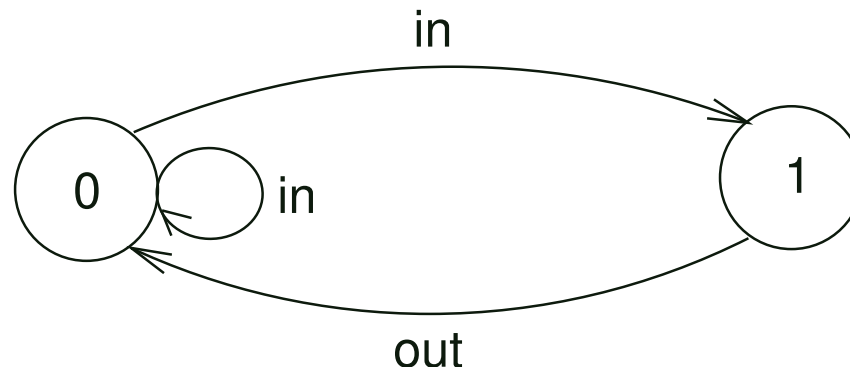
MÜNZE2 = (wurf \rightarrow kopf \rightarrow MÜNZE2
| wurf \rightarrow zahl \rightarrow MÜNZE2).



Beachte:

Beide Prozesse haben dieselben Ablauffolgen, jedoch verschiedene (nicht äquivalente) LTSs.

Beispiel (Fehlerhafter Übertragungskanal):

$$F_CHAN = (in \rightarrow out \rightarrow F_CHAN \\ | in \rightarrow F_CHAN).$$


Der Prozess ist nichtdeterministisch!

Im Folgenden betrachten wir notationelle Vereinfachungen.

Indizierte Aktionen und Prozesse

Indizierte Aktionen:

Können zur Modellierung von Daten (als Parameter von Aktionen) verwendet werden.

Beispiel (Korrektur Übertragungskanal für Daten):

$$\text{CHAN} = (\text{in}[i:0..2] \rightarrow \text{out}[i] \rightarrow \text{CHAN}).$$

ist Kurzschreibweise für:

$$\begin{aligned} \text{CHAN} = & (\text{in}[0] \rightarrow \text{out}[0] \rightarrow \text{CHAN} \\ & | \text{in}[1] \rightarrow \text{out}[1] \rightarrow \text{CHAN} \\ & | \text{in}[2] \rightarrow \text{out}[2] \rightarrow \text{CHAN}). \end{aligned}$$

Beachte:

Der Indexbereich muss beschränkt sein.

Indizierte Prozesse:

Dienen zur Vereinfachung von Prozessdeklarationen (mit lokalen Prozessen).

Beispiel (Kanal):

$$\text{CHAN} = (\text{in}[i:0..2] \rightarrow \text{TRANSMIT}[i]),$$
$$\text{TRANSMIT}[i:0..2] = (\text{out}[i] \rightarrow \text{CHAN}).$$

steht für:

Mehrfache Indizes, arithmetische Ausdrücke und Deklarationen von Konstanten und Bereichen:

Beispiel (SUM):

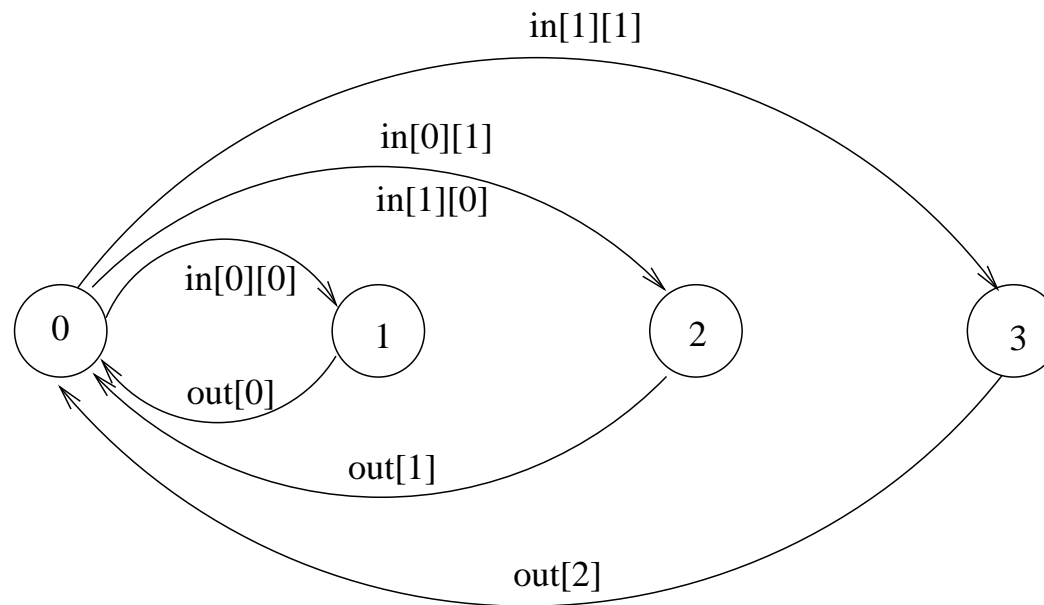
const N = 1

range T = 0..N

range R = 0..2*N

SUM = (in[a:T][b:T] → TOTAL[a+b]),

TOTAL[s:R] = (out[s] → SUM).



Bewachte Aktionen

(when B a \rightarrow E | b \rightarrow F).

Die Aktion a kann nur dann gewählt werden, wenn die Bedingung B erfüllt ist.

Bemerkung:

- Bewachte Aktionen können bei der Deklaration indizierter, lokaler Prozesse verwendet werden:

$$P[i:T][j:R] = (\text{when } B \text{ a } \rightarrow E \mid \dots)$$

- Die Bedingung B darf an Variablen höchstens die Indizes der Prozessdeklaration und formale Parameter (von parametrisierten Prozessen) enthalten.
- Prozessdeklarationen mit bewachten Aktionen sind Kurzschreibweisen für Prozessdeklarationen ohne bewachte Aktionen.

Beispiel (Countdown):

$$\begin{aligned} \text{COUNTDOWN} &= (\text{start} \rightarrow \text{CD}[2]), \\ \text{CD}[i:0..2] &= (\text{when } (i > 0) \text{ tick} \rightarrow \text{CD}[i-1] \\ &\quad | \text{when } (i == 0) \text{ beep} \rightarrow \text{STOP} \\ &\quad | \text{stop} \rightarrow \text{STOP}). \end{aligned}$$

ist Kurzschreibweise für:

Parametrisierte Prozesse

- Parametrisierte Prozesse erlauben eine generische Definition von Prozessen.
- Der Prozessparameter muss bei der Deklaration einen "Defaultwert" erhalten (sonst kein endliches LTS).
- Der Prozess kann jedoch für einen beliebigen aktuellen Parameter in einer anderen Prozessdeklaration aufgerufen werden.

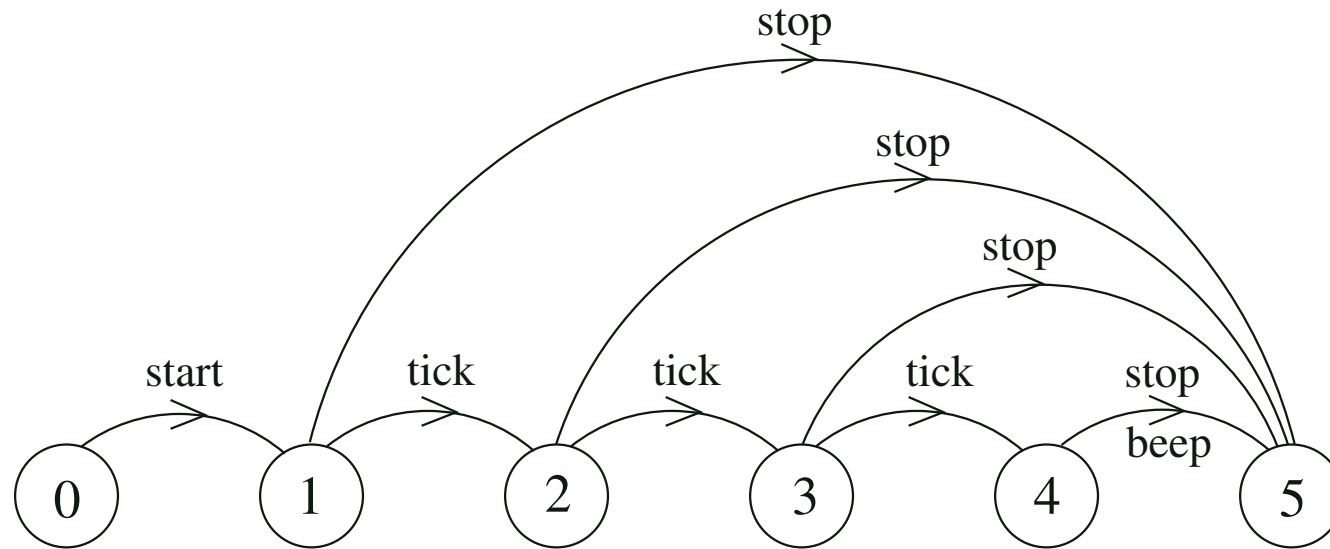
Beispiel (COUNTDOWN(N)):

$$\begin{aligned} \text{COUNTDOWN}(N=2) &= (\text{start} \rightarrow \text{CD}[N]), \\ \text{CD}[i:0..N] &= (\text{when } (i > 0) \text{ tick} \rightarrow \text{CD}[i-1] \\ &\quad | \text{when } (i == 0) \text{ beep} \rightarrow \text{STOP} \\ &\quad | \text{stop} \rightarrow \text{STOP}). \end{aligned}$$

Beachte:

Parameter werden in globalen Prozessdeklarationen verwendet, Indizes in lokalen Prozessdeklarationen.

Anwendung: MY_COUNTDOWN = COUNTDOWN(3).



2.4 Semantik von Prozessausdrücken

Starke Äquivalenz von LTSen

Es bezeichne \mathcal{T} die Menge aller (endlichen) LTSen über States und ACT.

Definition (Starke Bisimulation):

Seien $T, T' \in \mathcal{T}$, $T = (S, A, \Delta, q_0)$, $T' = (S', A', \Delta', q_0')$ mit $A = A'$.

Eine *starke Bisimulation* zwischen T und T' ist eine Relation $R \subseteq S \times S'$, so dass für alle $(q, q') \in R$ gilt:

(1) $(q \xrightarrow{a} p) \in \Delta \implies \exists p' \in S'$ mit $(q' \xrightarrow{a} p') \in \Delta'$ und $(p, p') \in R$.

(2) $(q' \xrightarrow{a} p') \in \Delta' \implies \exists p \in S$ mit $(q \xrightarrow{a} p) \in \Delta$ und $(p, p') \in R$.

Bemerkung:

Sei $T = (S, A, \Delta, q_0) \in \mathcal{T}$.

Die Identität $= \subseteq S \times S$ ist eine starke Bisimulation zwischen T und T .

Definition (Starke Äquivalenz von LTSen):

Seien $T, T' \in \mathcal{T}$, $T = (S, A, \Delta, q_0)$, $T' = (S', A', \Delta', q_0')$.

T und T' sind *stark äquivalent*, geschrieben $T \sim T'$, wenn gilt:

- (a) T und T' haben dieselben Aktionen, d.h. $A = A'$.
- (b) Es gibt eine starke Bisimulation $R \subseteq S \times S'$ zwischen T und T' , so dass $(q_0, q_0') \in R$.

Lemma:

\sim ist eine Äquivalenzrelation auf \mathcal{T} .

Bemerkung:

Stark äquivalente LTSen haben dieselben Abläufe.

Die Umkehrung gilt jedoch nicht (vgl. Beispiel Münzwurf von oben).

Beispiele:

Definition der Semantik von Prozessausdrücken

Es bezeichne \mathcal{E} die Menge aller Prozessausdrücke.

Die Semantik von Prozessausdrücken ist gegeben durch eine Funktion

$$\text{Its}: \mathcal{E} \longrightarrow \mathcal{T}$$

die gemäß der Struktur von Prozessausdrücken folgendermaßen induktiv definiert ist (vgl. Vorlesungsmitschrift):

Definition (Starke Äquivalenz von Prozessen):

Zwei Prozesse $E, F \in \mathcal{E}$ sind *stark äquivalent* (stark bisimilar), geschrieben $E \sim F$, wenn gilt: $\text{Its}(E) \sim \text{Its}(F)$.

Beispiele:

2.5 Implementierung von Prozessen

Betriebssystem-Prozesse und Threads

Ein *BS-Prozess* besitzt einen eigenen Adressraum und wird repräsentiert durch

- Daten (globale und lokale Variable);
die lokalen Variablen sind in einem Keller organisiert
- Code (Befehle)
- Deskriptor (organisatorische Daten und Werte der Maschinenregister)

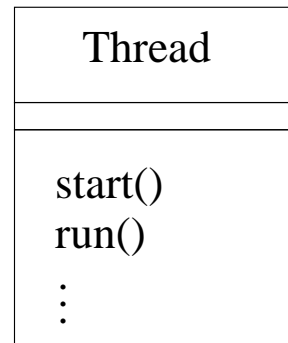
Ein BS-Prozess ist ein “schwergewichtiger Prozess“ (z.B. Ausführung eines Anwendungsprogramms).

Ein *Thread* ist ein “leichtgewichtiger Prozess“, der innerhalb eines BS-Prozesses (evt. parallel zu anderen Threads) abläuft.

- Jeder Thread besitzt einen eigenen Stack für seine lokalen Variablen und einen eigenen Deskriptor.
- Der Thread-Code ist im Code-Segment des BS-Prozesses enthalten.
- Jeder Thread hat Zugriff auf die globalen Variablen des BS-Prozesses.

Realisierung von Threads in Java

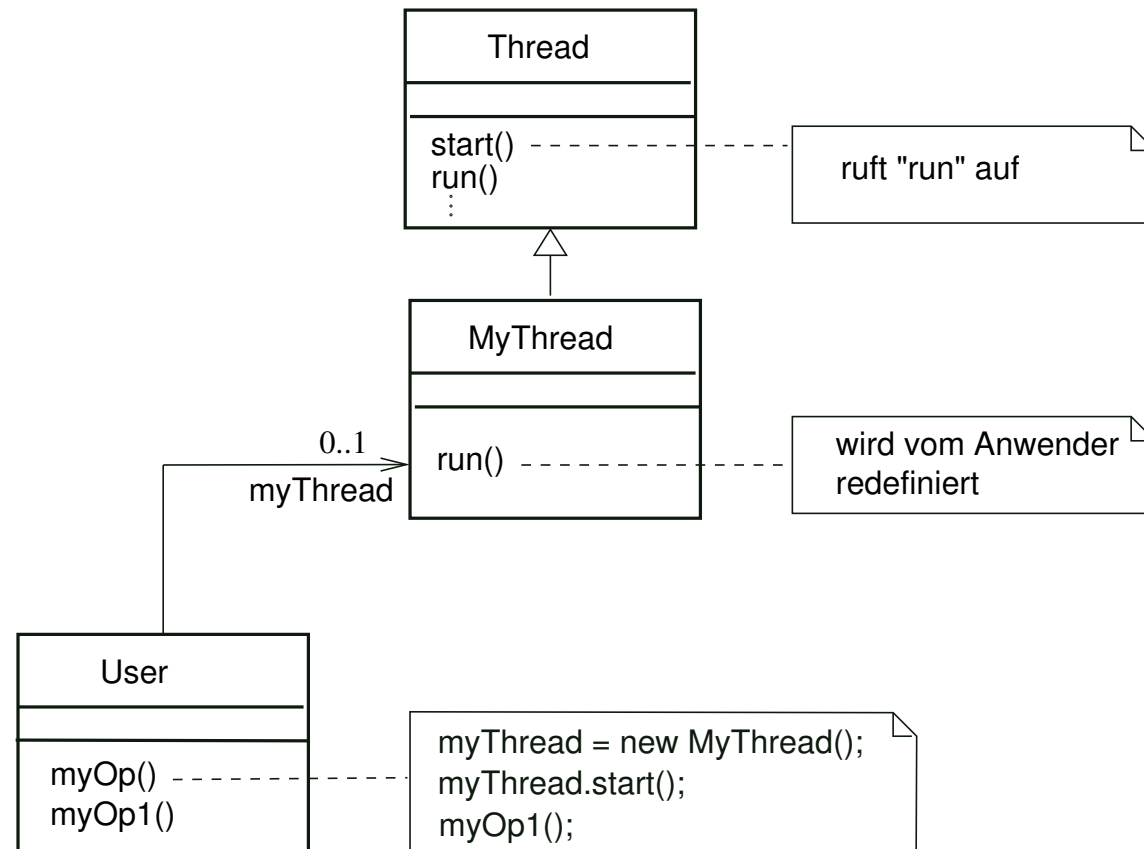
Threads werden in Java durch Objekte der Klasse "Thread" realisiert.



Es bezeichne t ein Objekt der Klasse Thread oder einer Subklasse von Thread.

- Der Methodenaufruf $t.start()$; bewirkt, dass das Thread-Objekt t aktiviert wird und seine `run`-Methode aufgerufen wird.
- Der aufrufende Thread setzt dann seine Tätigkeit parallel zur Ausführung der `run`-Methode des Threads t fort.

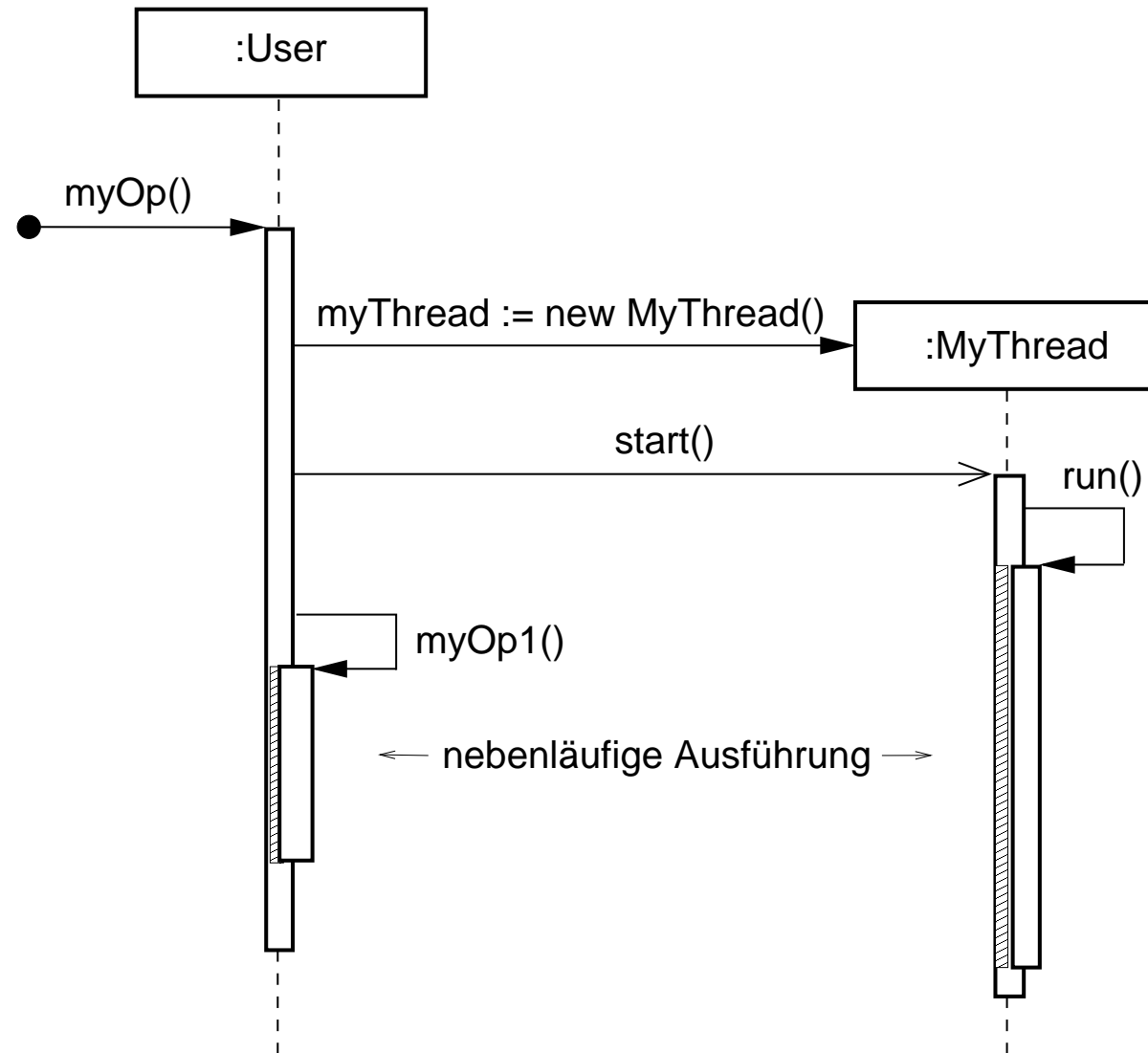
1. Realisierung von Threads mittels Vererbung



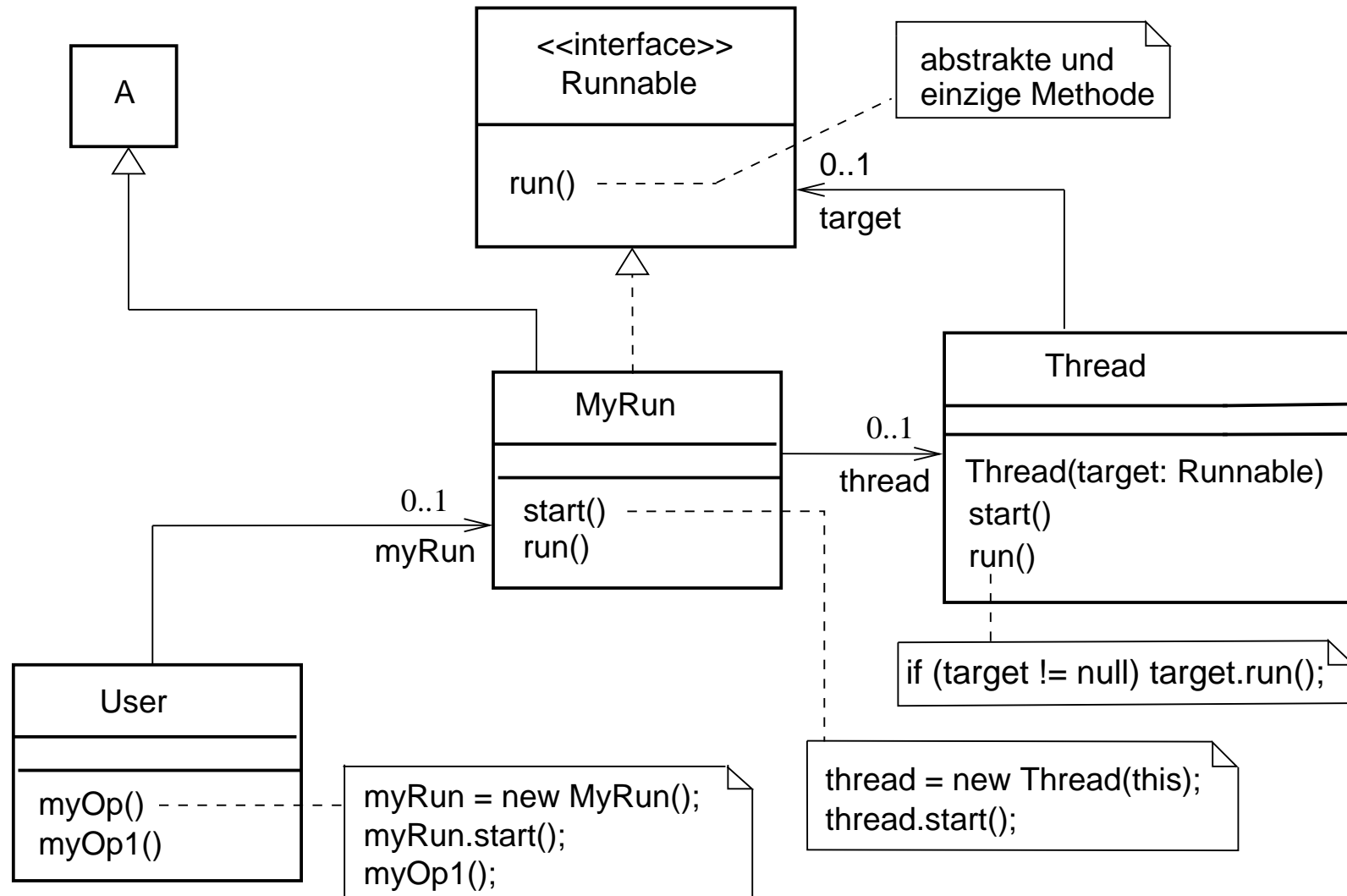
Beachte:

“MyThread“ kann nicht Erbe einer weiteren Klasse sein, da in Java Mehrfachvererbung nicht möglich ist!

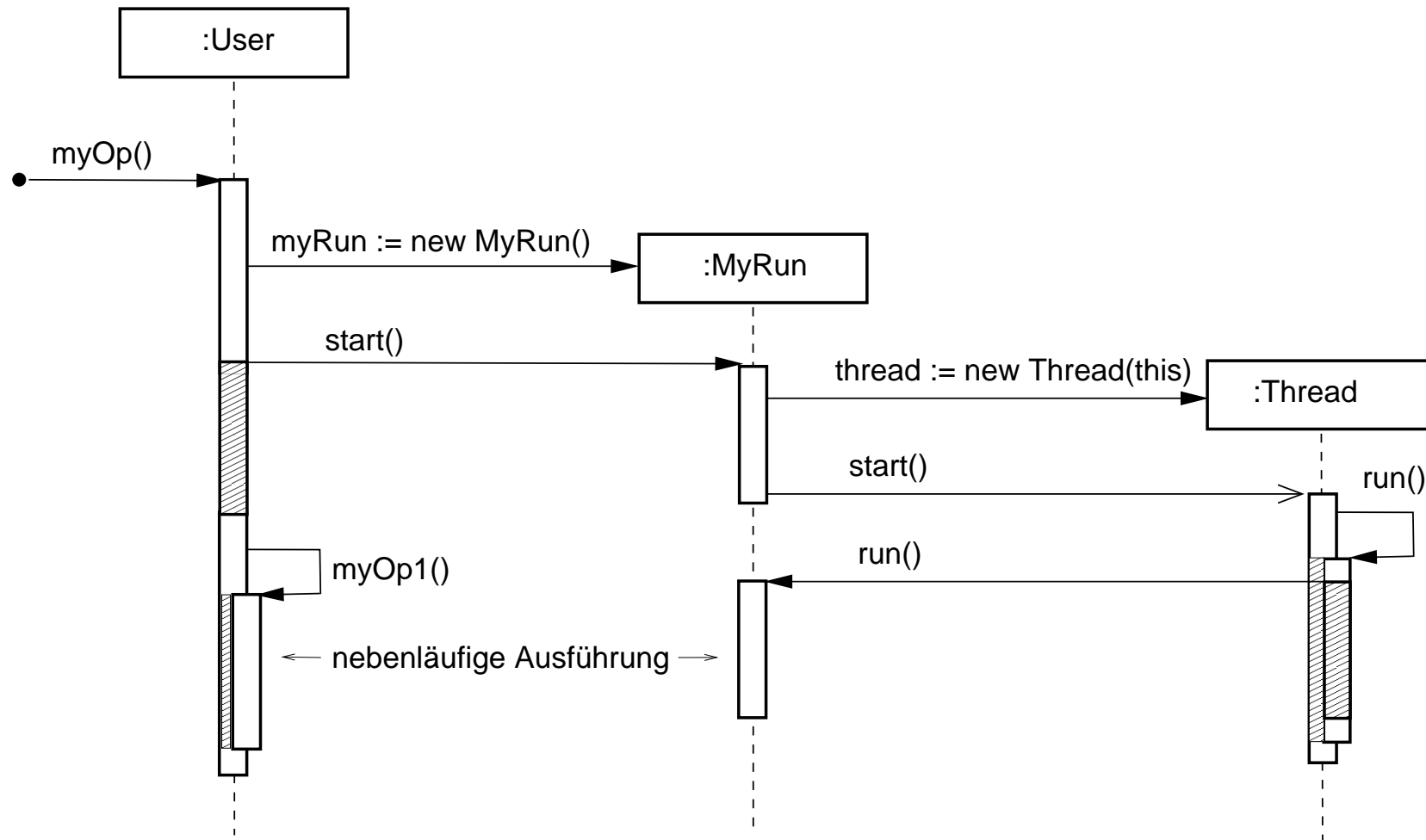
Sequenzdiagramm mit Objekt der Klasse MyThread



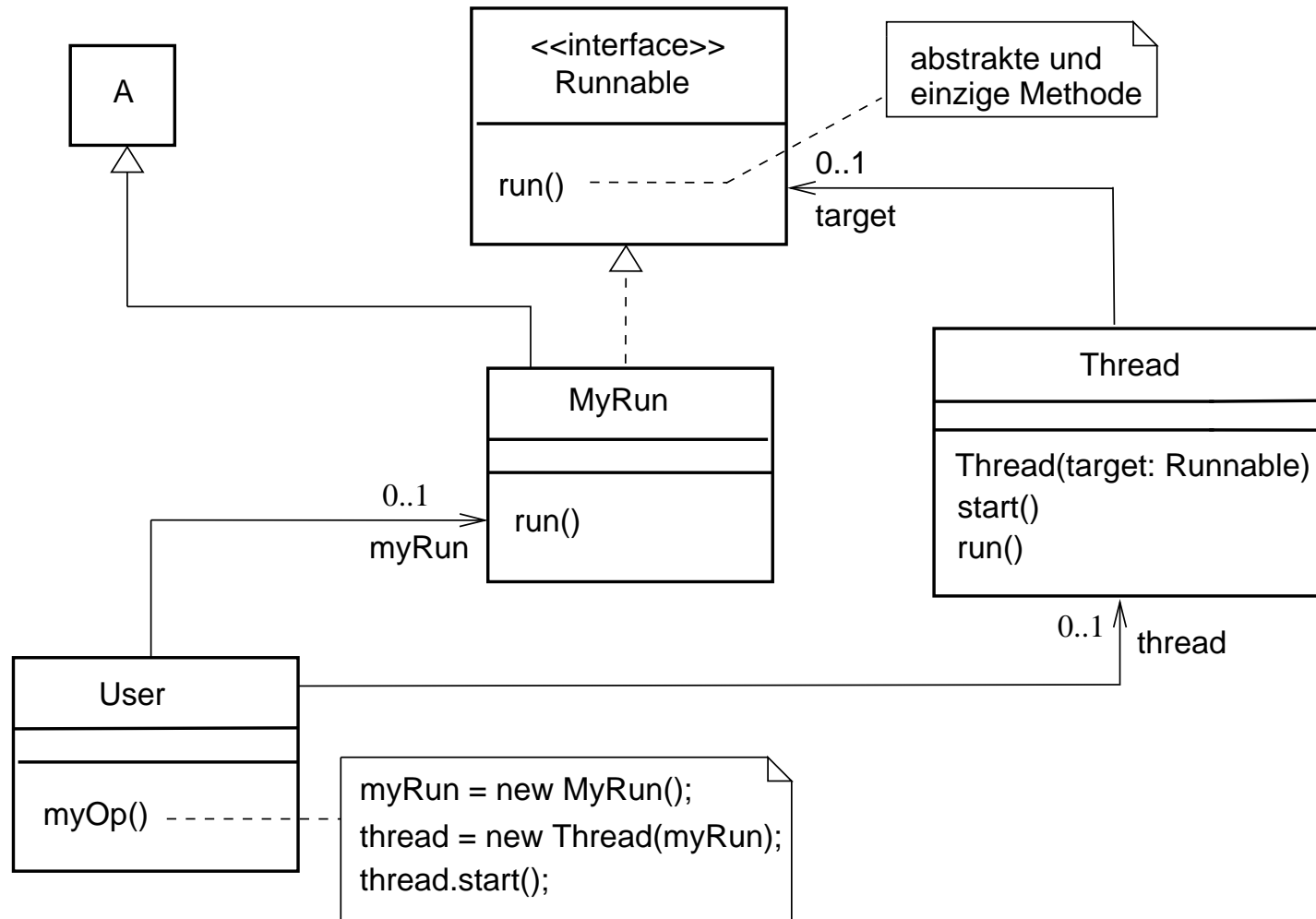
2. Realisierung von Threads durch Verwendung des Interfaces "Runnable"



Sequenzdiagramm mit Objekt der Klasse MyRun



Klassendiagramm mit Interface Runnable (Variante)



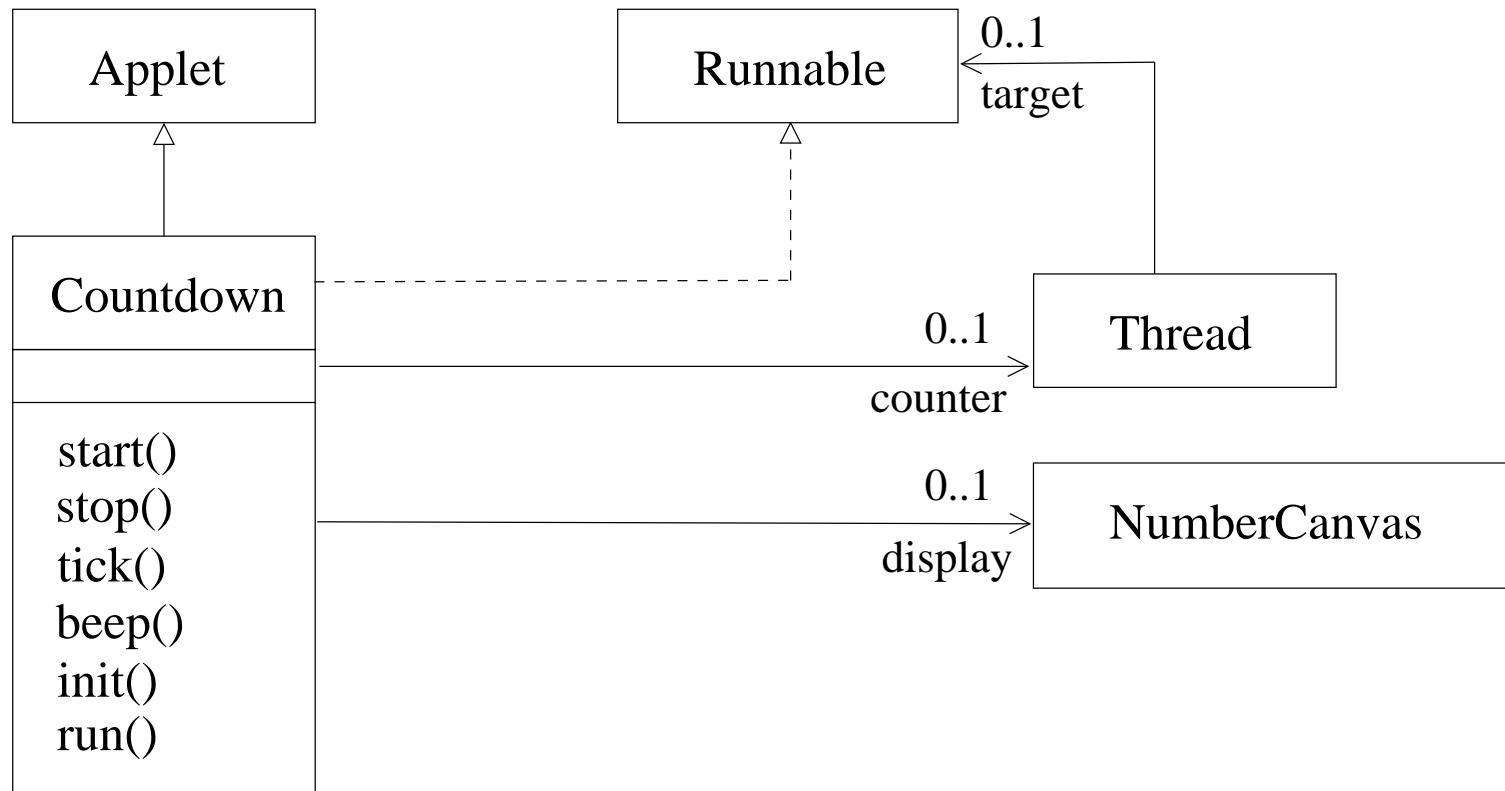
Beispiel (Implementierung des Countdown-Prozesses):

$$\begin{aligned} \text{COUNTDOWN}(N=10) &= (\text{start} \rightarrow \text{CD}[N]), \\ \text{CD}[i:0..N] &= (\text{when } (i > 0) \text{ tick} \rightarrow \text{CD}[i-1] \\ &\quad | \text{when } (i == 0) \text{ beep} \rightarrow \text{STOP} \\ &\quad | \text{stop} \rightarrow \text{STOP}). \end{aligned}$$

Aktionen:

- externe: start, stop
- interne: tick, beep

Klassendiagramm der Implementierung



Java-Implementierung:

```
public class Countdown extends Applet implements Runnable {
    final static int N = 10;
    int i;
    Thread counter;
    AudioClip beepsound, ticksound;
    NumberCanvas display;

    public void start() {
        i = N;
        counter = new Thread(this);
        counter.start();
    }

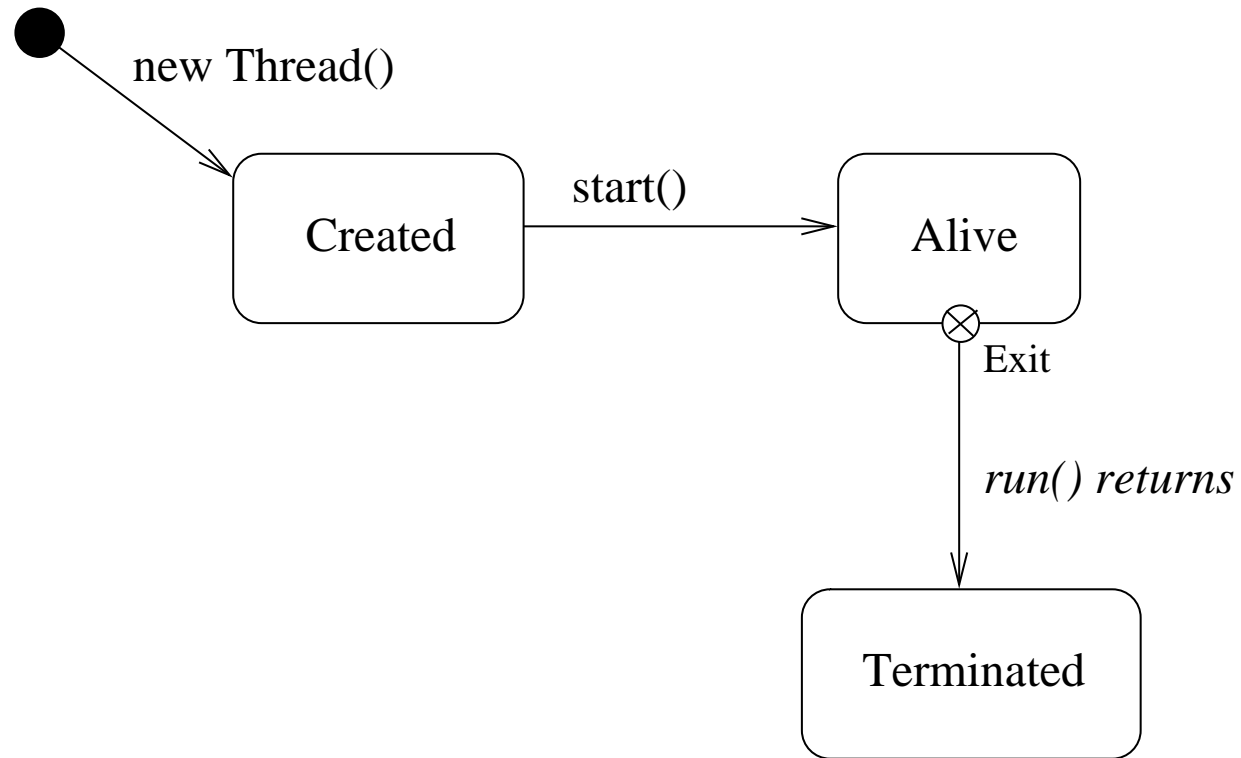
    public void stop() {
        counter = null;
    }
}
```

```
private void tick() {...}
private void beep() {...}

public void init() {...}

public void run() {
    while (true) {
        if (counter == null) return;
        if (i > 0) {tick(); i = i-1;}
        if (i == 0) {beep(); return;}
    }
}
}
```

Lebenszyklus eines Java-Threads



Untorzustände des Alive-Zustands

