

# Entwurf und Implementierung paralleler Programme

---

Prof. Dr. Rolf Hennicker

16.10.2006

# **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):

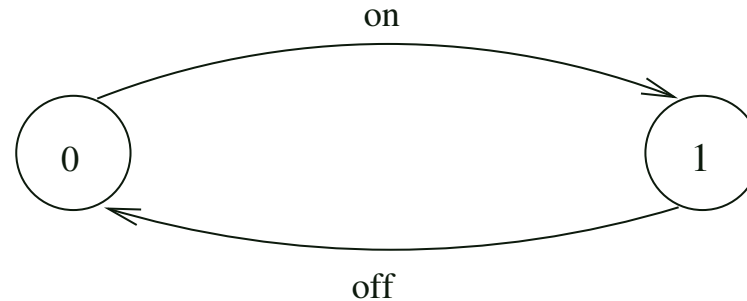
Wird von einer Aktion bewirkt. Aktionen sind elementar, d.h. nicht unterbrechbar.

### **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



Einzig mögliche Aktionsfolge (“trace“):

on → off → on → off → ... (unendliche Folge)

#### **Konvention:**

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

#### **Beachte:**

- Wir betrachten nur Prozesse mit endlich vielen Zuständen.
- Das Verhalten eines Prozesses kann aber unendlich sein.

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 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.

## 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.  $P \in FV(E)$ .

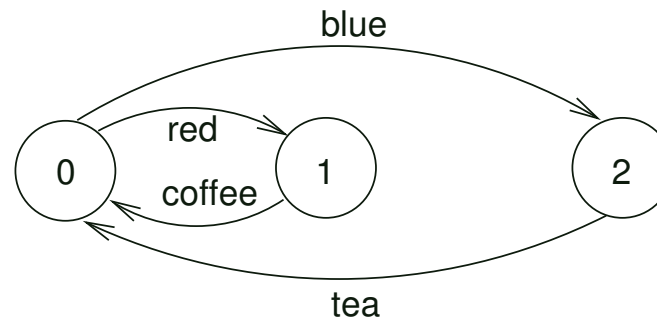
### Beispiele:

Äquivalente Prozessbeschreibung mit lokalen Prozessdeklarationen:

**Beispiel (DRINKS):**

$\text{DRINKS} = (\text{red} \rightarrow \text{coffee} \rightarrow \text{DRINKS} \mid \text{blue} \rightarrow \text{tea} \rightarrow \text{DRINKS}).$

Zugehöriges LTS:

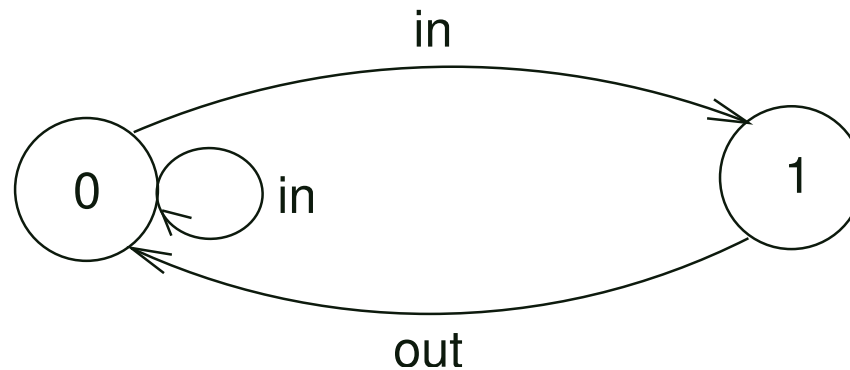


## Bemerkungen:

- blue, red  $\hat{=}$  Input-Aktionen (der Umgebung)
- coffee, tea  $\hat{=}$  Output-Aktionen
- Häufig besteht die Auswahl aus 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  $\rightarrow$  ...
  - ...
  - ...
- In CSP wird die “trace“-Semantik für Prozesse verwendet.

## Beispiel (Münzwurf):

## 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:

## Indizierte Prozesse:

Dienen zur Vereinfachung von Prozessdeklarationen.

## Beispiel (Kanal):

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

steht für:

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

### Beispiel (SUM):

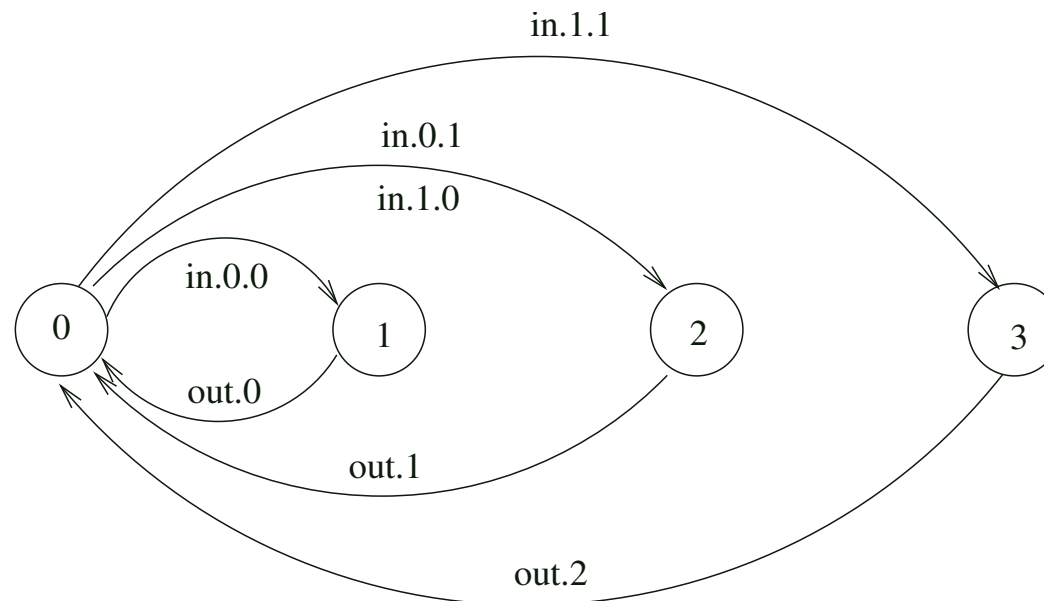
const  $N = 1$

range  $T = 0..N$

range  $R = 0..2*N$

SUM = (in[a:T][b:T]  $\rightarrow$  TOTAL[a+b]),

TOTAL[s:R] = (out[s]  $\rightarrow$  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 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):

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

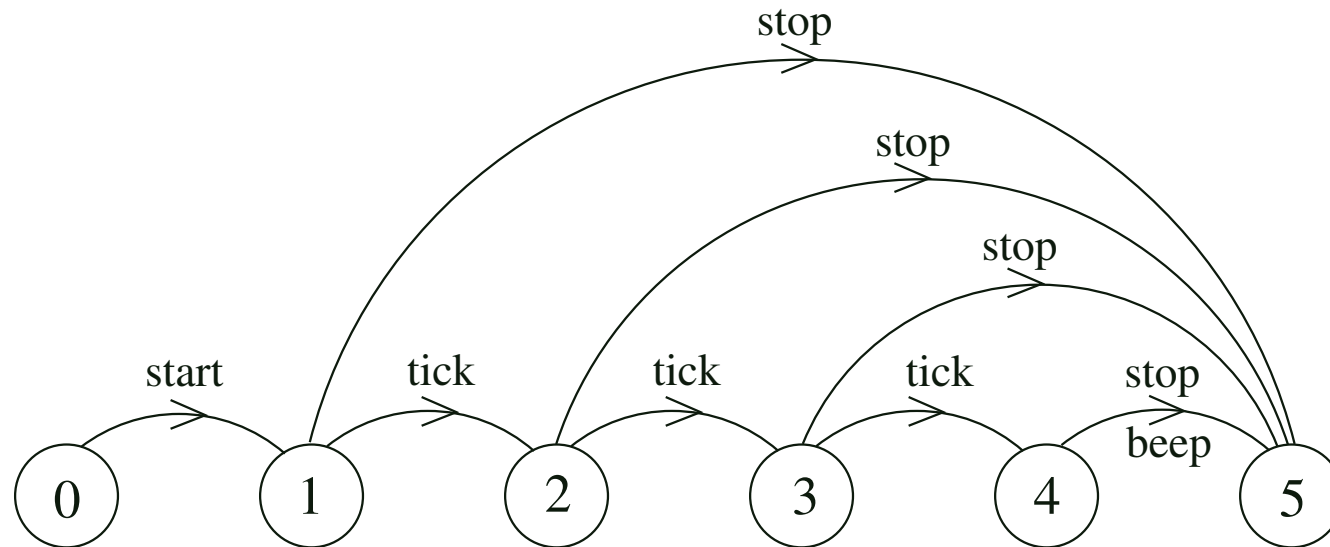
ist Kurzschreibweise für:

## 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)):**

**Anwendung:** MY\_COUNTDOWN = COUNTDOWN(3).



## 2.4 Semantik von Prozessausdrücken

### Grundlegende Konzepte

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

#### Definition:

Seien  $T, T' \in \mathcal{T}$  mit  $T = (S, A, \Delta, q)$ ,  $T' = (S', A', \Delta', q')$   
und sei  $a \in A$ .

$T$  geht mit  $a$  über in  $T'$  (geschrieben  $T \xrightarrow{a} T'$ ), falls gilt:

$$(1) (q, a, q') \in \Delta$$

$$(2) S' = \begin{cases} S & \text{falls } q \text{ von } q' \text{ aus mit } \Delta \text{ erreichbar ist} \\ S \setminus \{q\} & \text{sonst} \end{cases}$$

$$(3) A' = A$$

$$(4) \Delta' = \{ (q_1, b, q_2) \in \Delta \mid q_1, q_2 \in S' \}$$

## Beispiele:

**Definition (Starke Äquivalenz von LTSen):**

Die *starke Äquivalenz* (starke Bisimulation) ist die größte Relation  $\sim \subseteq \mathcal{T} \times \mathcal{T}$ , so dass für alle  $T, T' \in \mathcal{T}$  mit  $T \sim T'$  gilt:

(0)  $T$  und  $T'$  haben dieselben Mengen von Aktionen.

(1)  $T \xrightarrow{a} R$  ( $a \in \text{ACT}, R \in \mathcal{T}$ )  $\implies$   
 $\exists R' \in \mathcal{T}$  mit  $T' \xrightarrow{a} R'$  und  $R \sim R'$ .

(2)  $T' \xrightarrow{a} R'$  ( $a \in \text{ACT}, R' \in \mathcal{T}$ )  $\implies$   
 $\exists R \in \mathcal{T}$  mit  $T \xrightarrow{a} R$  und  $R \sim R'$ .

**Lemma1:**

$\sim$  ist die Vereinigung aller Relationen  $Rel \subseteq \mathcal{T} \times \mathcal{T}$ , die (0), (1) und (2) erfüllen.

**Bemerkung:**

Die Identität  $= \subseteq \mathcal{T} \times \mathcal{T}$  erfüllt (0), (1) und (2).

Folglich gilt nach Lemma 1:  $T = T'$  impliziert  $T \sim T'$ .

**Lemma2:**

$\sim$  ist eine Äquivalenzrelation.

**Bemerkung:**

Stark äquivalente LTSe 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:

## 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)

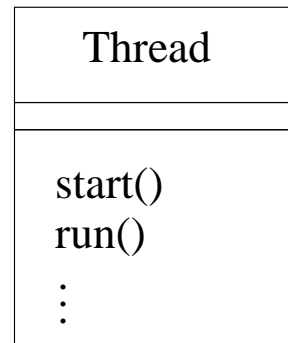
Ein BS-Prozess ist ein “schwergewichtiger Prozess“ (z.B. Ausführung eines Anwendungsprogramms).

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.

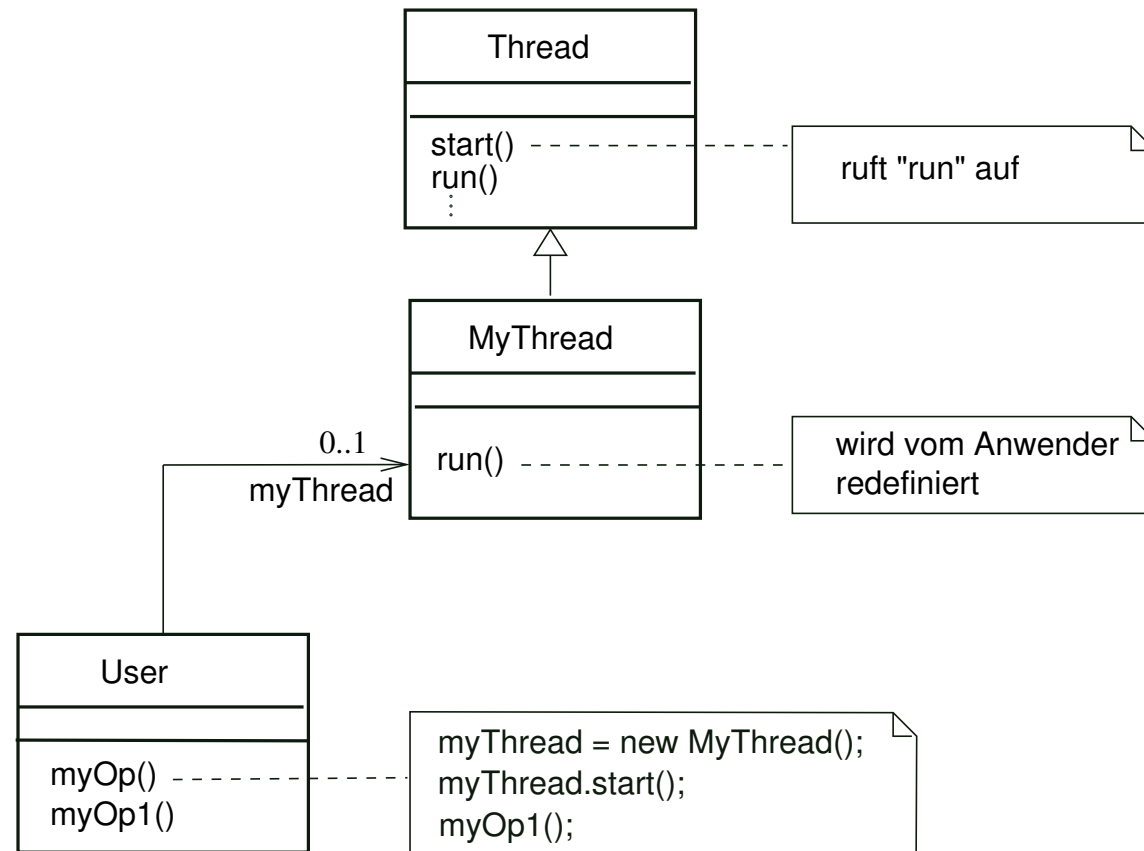


Es bezeichne  $t$  ein Objekt der Klasse Thread oder einer Subklasse von Thread.

Der Methodenaufruf  $t.start()$  ;

- aktiviert das Thread-Objekt  $t$  und ruft seine run-Methode auf;
- erfolgt durch asynchrone Nachrichtenübertragung, d.h. das aufrufende Objekt setzt nach dem Senden von start an  $t$  seine Tätigkeit parallel zum Thread-Objekt  $t$  fort.

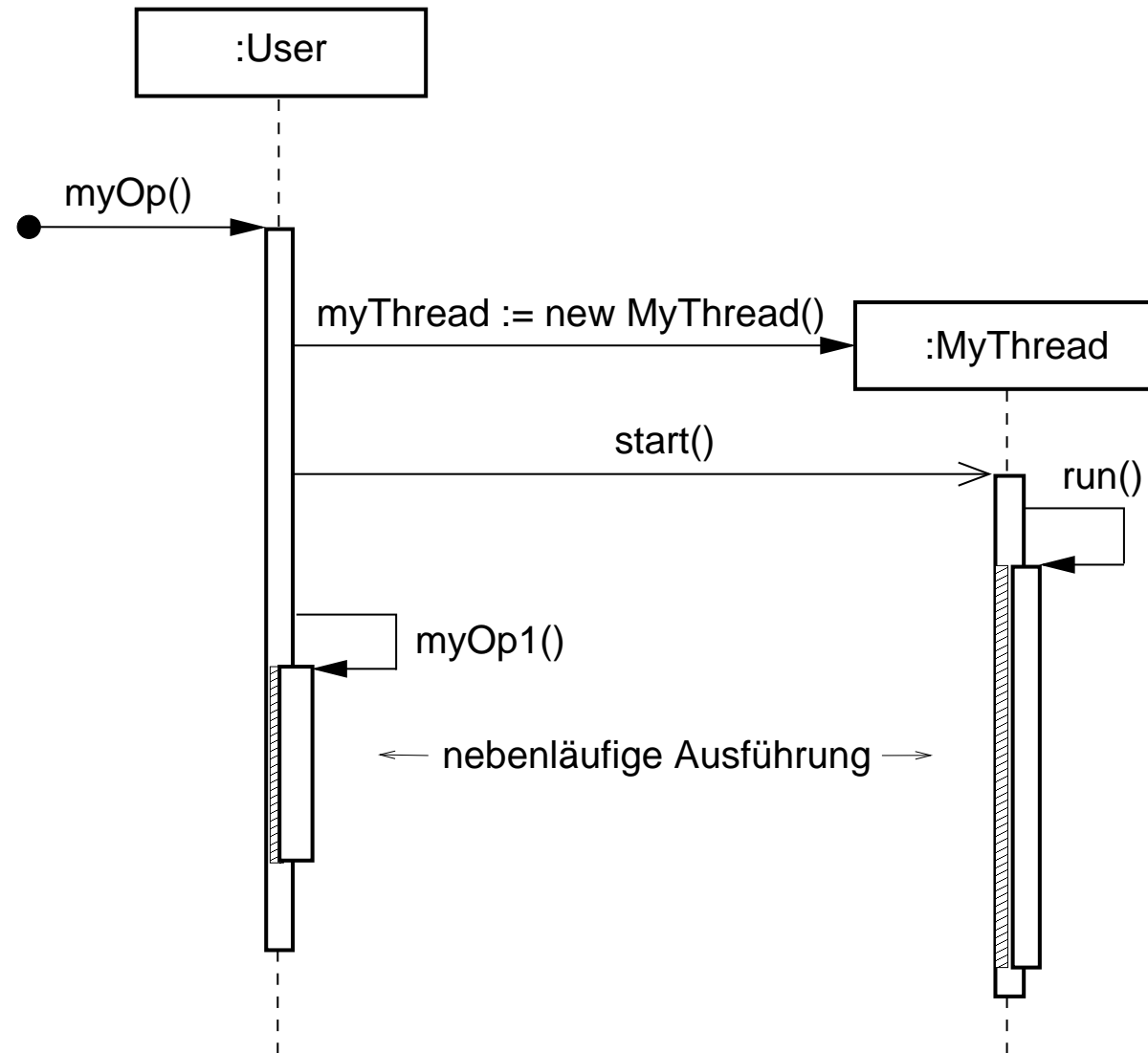
# 1. Realisierung von Threads mittels Vererbung



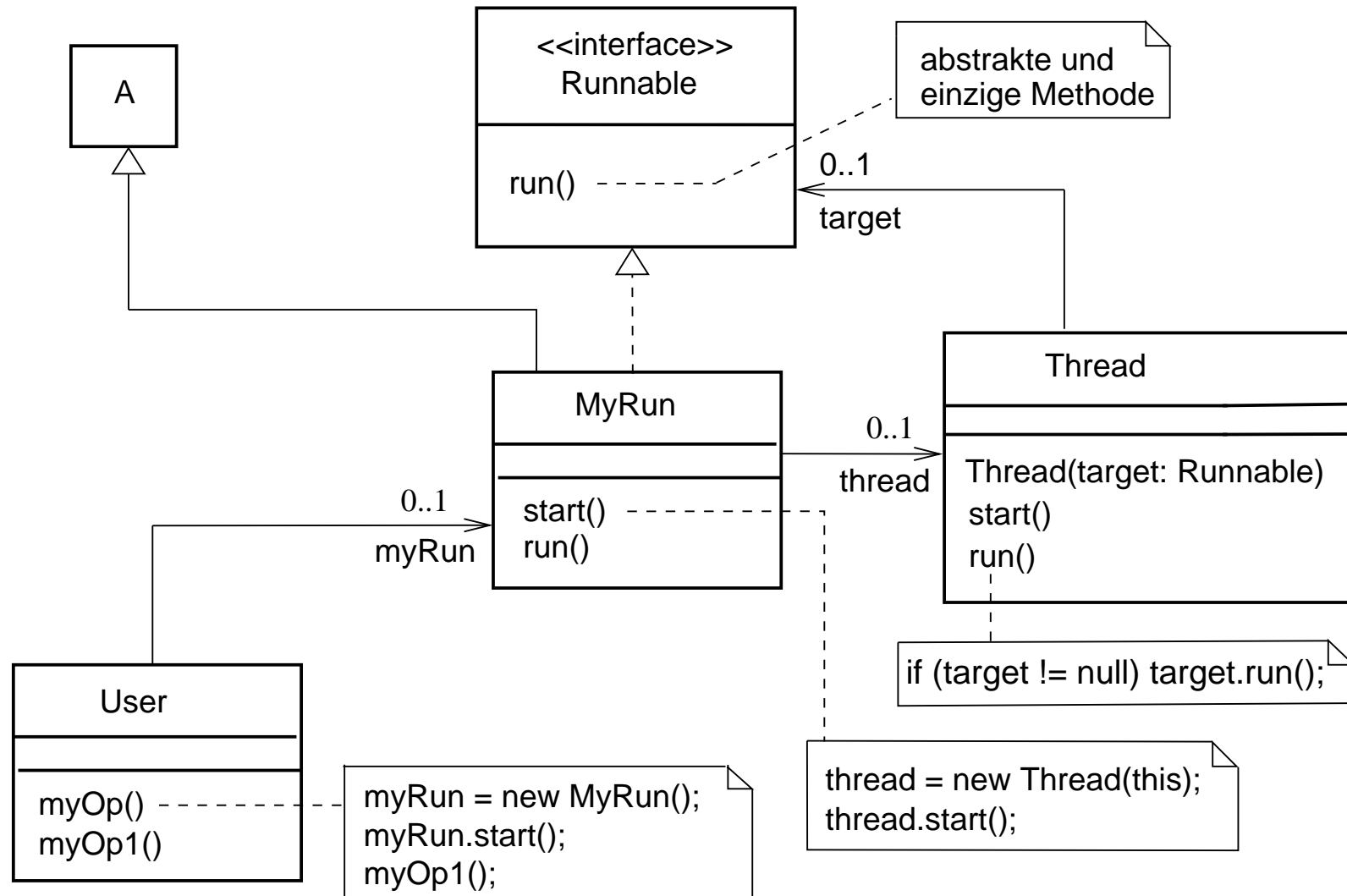
## Beachte:

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

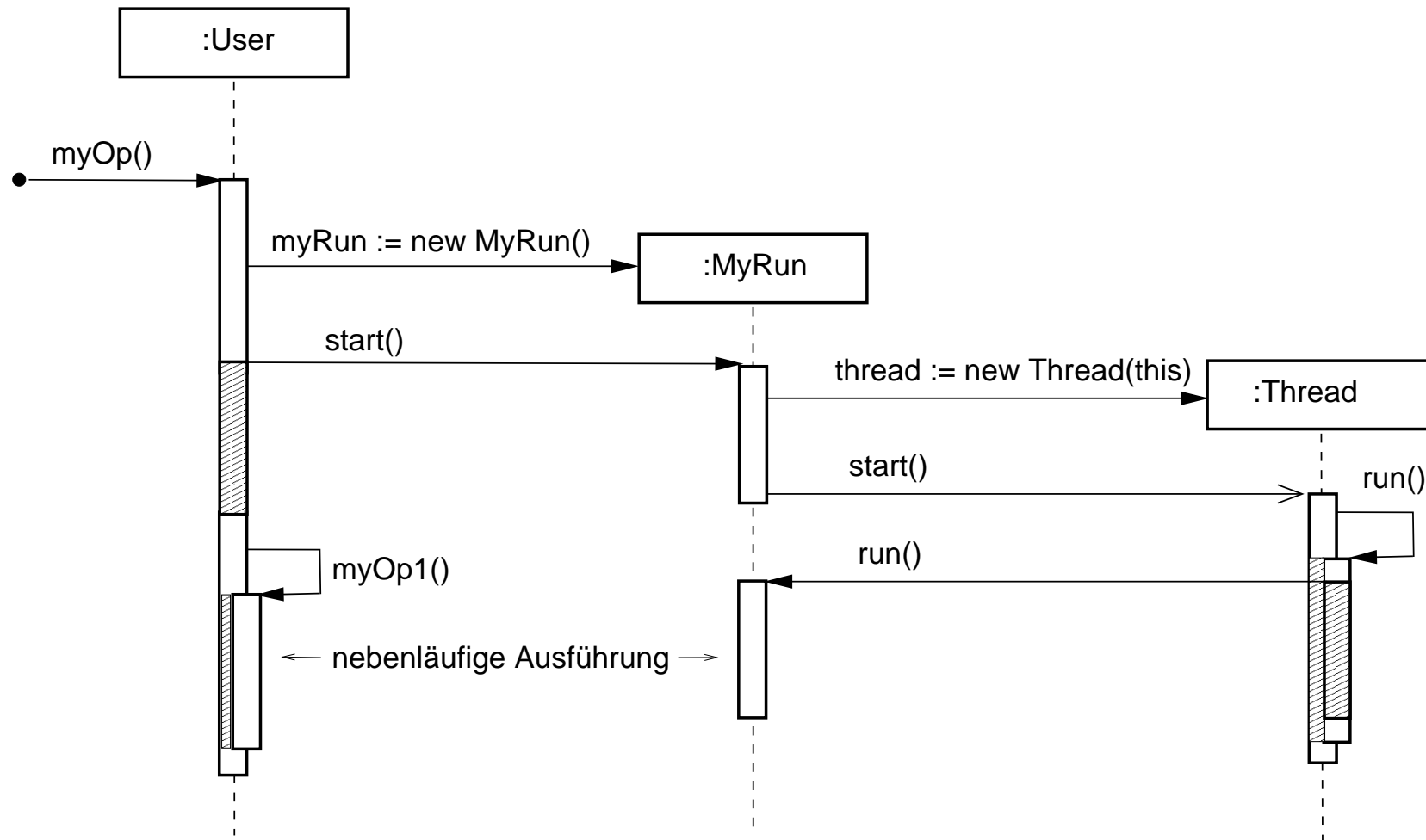
## Sequenzdiagramm mit Objekt der Klasse MyThread



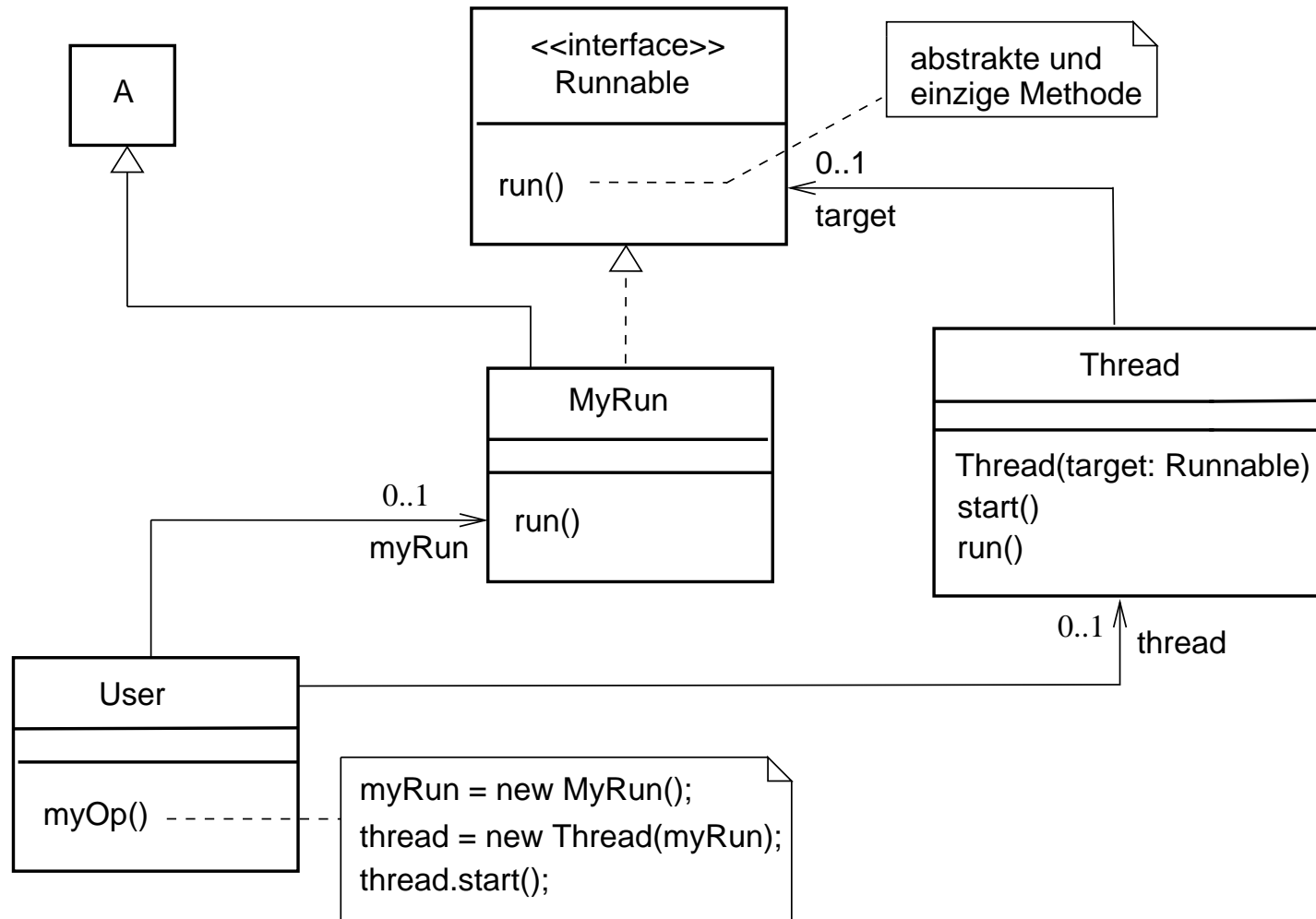
## 2. Realisierung von Threads durch Verwendung des Interfaces "Runnable"



## Sequenzdiagramm mit Objekt der Klasse MyRun



## Klassendiagramm mit Interface Runnable (Variante)



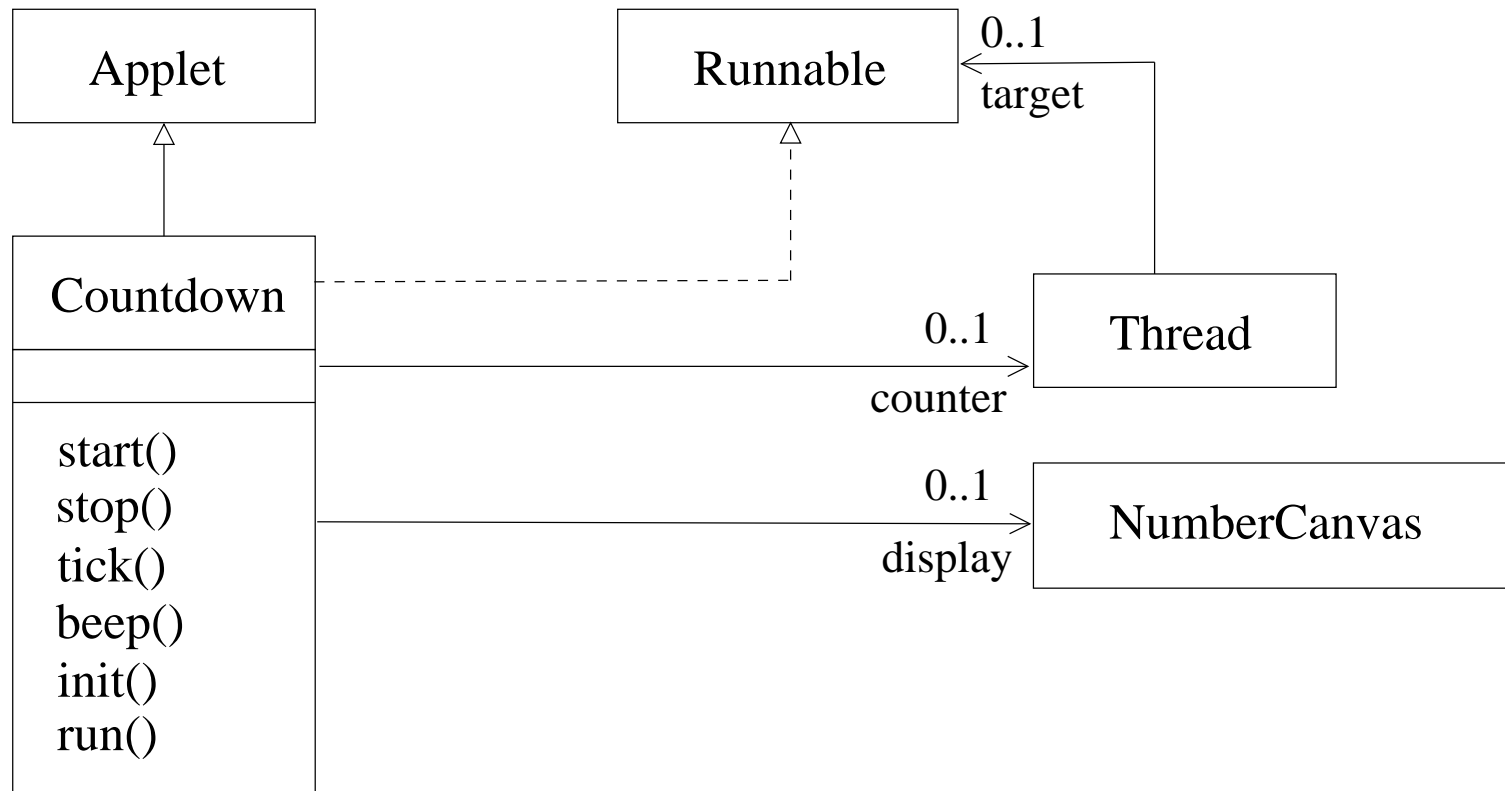
## Beispiel (Implementierung des Countdown-Prozesses):

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

### 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();
    }

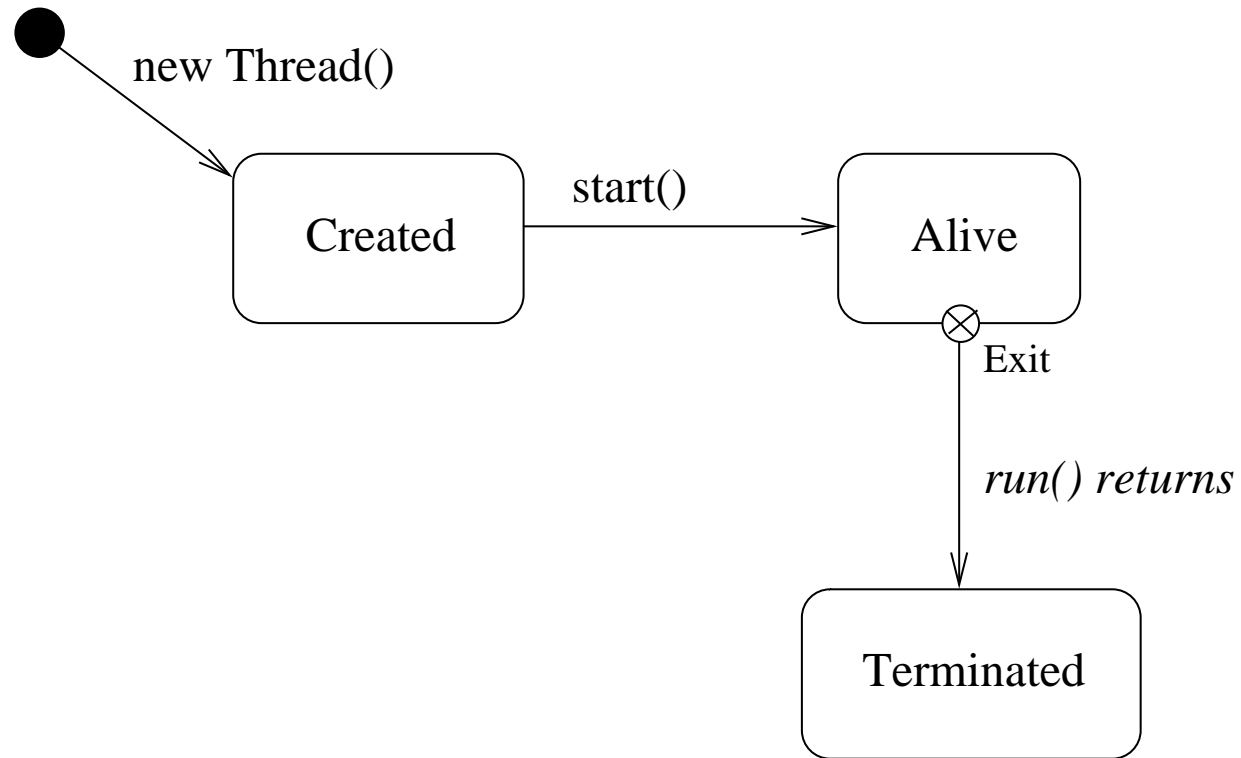
    public void stop() {
        counter = null;
    }
}
```

```
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

