LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

Nebenläufigkeit mit Java

Einheit 01: Einführung in das Java-Threadkonzept

Lorenz Schauer
Lehrstuhl für Mobile und Verteilte Systeme

Heutige Agenda

Organisatorisches

- Ziele, Aufbau und Inhalte
- Zielgruppe, Vergütung, Webseite
- Kontakt

Einführung in Java Threads

- Motivation
  - Theoretische Einführung
- Threads erzeugen
- Threads Zustände & Eigenschaften

Praxis:

- Übungen zu Threads

Lernziele

- Einführung in die parallele Programmierung mit Java
- Thread-Konzept verstehen
- Threads erzeugen und arbeiten lassen

Organisatorisches zum Kurs

- **Ziele**:
  - Grundlagen zur nebenläufigen Programmierung kennenlernen und verstehen
    - Allgemeine Grundlagen in der Java-Programmierung sollten vorhanden sein!
  - Praktischer Einstieg in das Thread-Konzept
    - Kleinere parallele Programme sollen selbstständig entwickelt und ausgeführt werden können
  - Mit Problemen und Lösungen paralleler Programme umgehen können

- **Aufbau**:
  - Mischung aus Vorlesung und praktischen Programmiereinheiten
  - Kleinere Programmieraufgaben müssen während der Veranstaltung selbstständig gelöst werden (ggf. mit Hilfestellung)
    - Bitte bringen Sie daher auch immer Ihr eigenes Gerät (Laptop) mit!

Kursorganisation

- **Zielgruppe:**
  - Studenten der Informatik bzw. mit Nebenfach Informatik, die ihr Wissen in dem angebotenen Thema vertiefen wollen.

- **Vergütung**:
  - Der Kurs stellt ein freiwilliges Zusatzangebot zur Verbesserung der Lehre dar
  - Keine Vergütung!

- **Ort und Zeit:**
  - Insgesamt 4 Einheiten:
    - Vom 11.01 bis einschl. 01.02.2016 immer Montags, von 18.00 -20.00 Uhr c.t.
    - Hauptgebäude (Geschwister-Scholl-Platz 1), Raum: M010

- **Webseite**:
  - http://www.mobile.ifi.lmu.de/lehrveranstaltungen/nebenlaeufigkeit-mit-java/
  - Keine Anmeldung notwendig

Kursinhalte I

- 11.01. (Heute):
  - Organisatorisches und Einführung
  - Java-Thread Konzept
    - Generierung und Starten von Threads
    - Übungsaufgaben

- 18.01.:
  - Weitere Grundlagen von Threads in Java
  - Deadlocks und Synchronisation
    - Atomare Ausdrücke
    - Wechselseitiger Ausschluss
    - Übungsaufgaben

Kursinhalte II

- 25.01.:
  - Weitere Aspekte zur Synchronisation:
    - Semaphoren
    - Monitore
    - Benachrichtigungen
    - Übungsaufgaben

- 01.02.:
  - Zusammenfassende Übungen

Kontakt

- **Veranstalter**:
  - Lorenz Schauer (Wiss. Mitarbeiter)
    - Büro:
      - Lehrstuhl für Mobile und Verteilte Systeme
        Oettingenstraße 67, Raum U160
    - Sprechstunde:
      - Montags, 10 - 12 Uhr
      - Donnerstags, 14.00 - 16.00 Uhr
    - Kontakt:
      - Mail: lorenz.schauer@ifi.lmu.de
      - Tel.: 089-2180-9157
      - Web: http://www.mobile.ifi.lmu.de/team/lorenz-schauer/

Teil 1: Einführung in Java Threads

Mögliche Abläufe:

Sequentielles Programm:
Anweisungen werden Schritt für Schritt hintereinander ausgeführt ("single thread of control")

Paralleles Programm (Nebenläufig):
Anweisungen oder Teile der Anweisungen eines Programms werden nebeneinander ausgeführt ("multi thread of control")

Echt gleichzeitig parallel

Prozesse/Threads werden auf mehreren Kernen echt parallel ausgeführt

Zeitlich verzahnt (quasi-parallel)

Prozesse/Threads teilen sich einen Kern und werden somit durch Scheduling unterbrochen und verzahnt ausgeführt

Vor- und Nachteile paralleler Programmierung

Vorteile:

- Komplexität in parallele Teilbereiche zerlegen
- Höherer Durchsatz
- Performanz
- Ausnutzung bei eingebetteten und verteilten Systemen

Nachteile:

- Erhöhte Komplexität
- Abläufe sind häufig schwer zu durchschauen
- Sehr fehleranfällig
- Schwer zu Debuggen bei Laufzeitfehlern
- Konzepte zur Synchronisation und Thread-Sicherheit erforderlich, um deterministisches Verhalten zu gewährleisten

Ein Prozess:

- Ein Programm in Ausführung
  - Adressraum, Daten, Code, Deskriptor
- Bei modernen BS gehört zu jedem Prozess mindestens ein Thread
  - Der Prozess JVM wird vom BS gestartet
  - JVM erzeugt beim starten des Programms (Aufruf von main()) einen Haupt-Thread (Main-Thread)

Threads:

- Innerhalb eines Prozesses kann es mehrerer Threads geben
- Sind „leichtgewichtige Prozesse“
  - Teilen sich gleichen Adressraum
  - Haben gemeinsamen Zugriff auf Speicher und Ressourcen d. Prozesses
  - Eigener Stack für lokale Variablen
  - Eigener Deskriptor

Bei früheren BS: Nur Simulation der verzahnten Ausführung durch die JVM

- Keine echte Parallelisierung
  - Falls keine direkte Thread-Unterstützung durch das BS möglich

Heute i.d.R.: JVM bildet die Threadverwaltung auf BS ab

- native Threads (bzw. KLT)
- Echte parallele Ausführung auf mehreren Kernen möglich!

Bringt einige Probleme mit sich:

- Nicht deterministisches Verhalten
- Deadlocks
- Sicherheit
- Lebendigkeit (bsp.: Fairness)

Achten auf Thread-Sicherheit und Synchronisation!

Nebenläufigkeit in Java

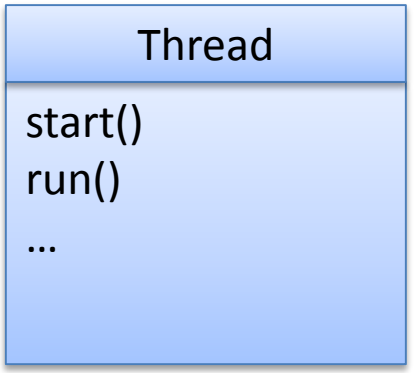
Die Java-Bibliothek besitzt eine Reihe von Klassen, Schnittstellen und Aufzählungen für Nebenläufigkeit:

- Thread
  - Jeder laufende Thread stellt ein Exemplar dieser Klasse dar
- Runnable
  - Programmcode, der parallel ausgeführt werden soll
- Lock
  - Mit Lock können kritische Bereiche markiert werden (nur 1 Thread innerhalb krit. Bereich)
- Condition
  - Threads können auf Benachrichtigungen anderer Threads warten

Threads in Java: Realisierung

Threads in Java stellen Objekte der Klasse `java.lang.Thread` dar

- Die Klasse beinhaltet eine `run()` Methode, die den Code beinhaltet, der parallel ausgeführt werden soll
- Die `run`-Methode muss mittels `threadinstanz.start()` ausgeführt werden, damit der Code parallel zum aufrufenden Thread ausgeführt wird!
- **Achtung**: Ruft man `threadinstanz.run()` auf, wird der Code in der `run-Methode` „ganz normal“ also sequentiell ausgeführt



Threads in Java: Erzeugung

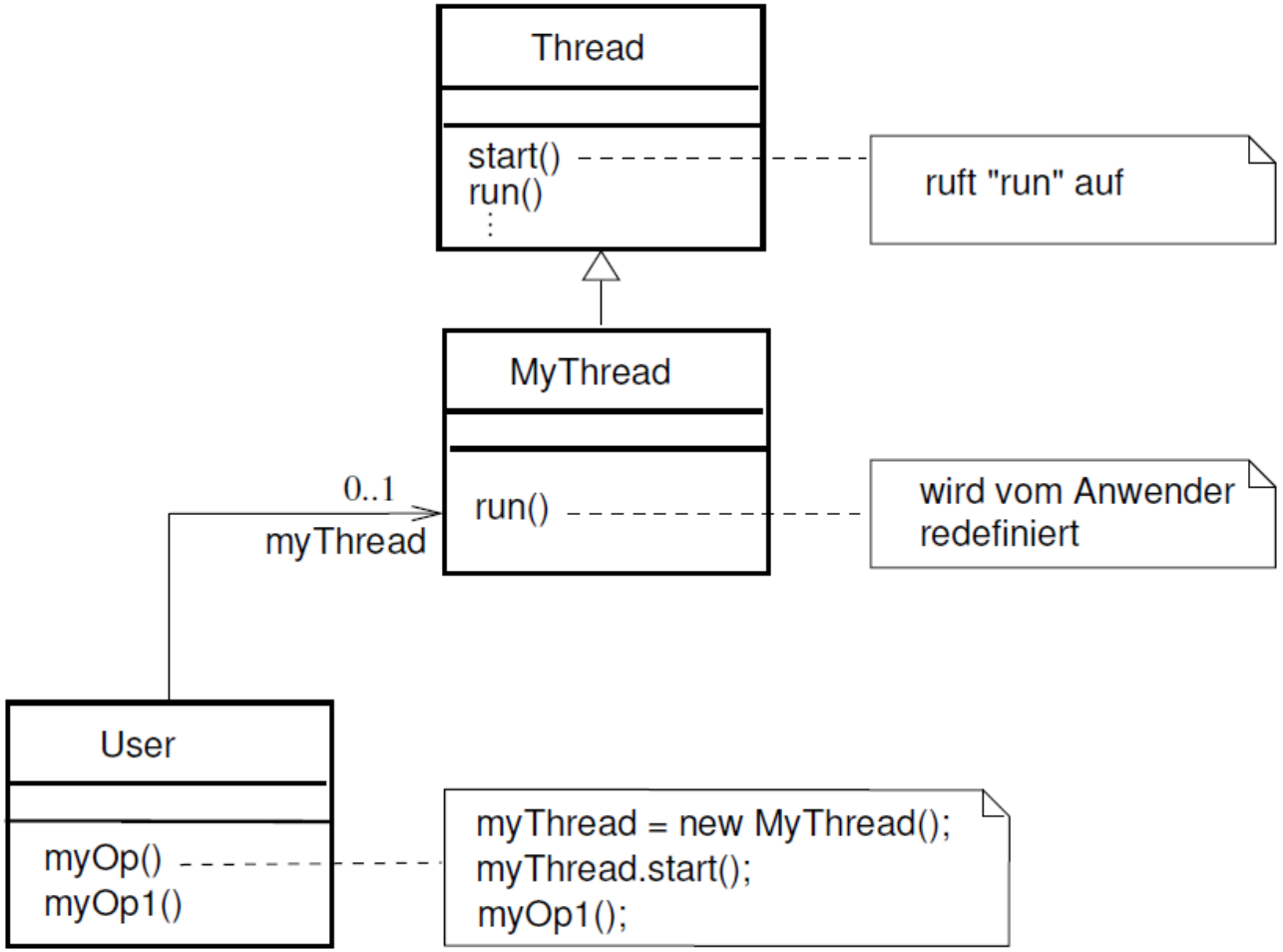
Prinzipiell stehen uns 4 Möglichkeiten zur Verfügung, um Threads in Java programmatisch zu erzeugen:

- Thread als eigene Klasse (Datei):
  - Erben von der Klasse Thread
  - Implementieren des Interfaces `Runnable`
- Threads direkt im Quellcode:
  - Anonyme Klasse
    - Entweder `Runnable` oder im Konstruktor von Thread
  - Lambda Ausdrücke (Seit Java 8)

Die 4 Möglichkeiten basieren auf den beiden Konzepten:

- Threads über Vererbung
  - Nachteil: Das Erben einer weiteren Klasse ist nicht möglich!
- Threads über Interface `Runnable`

Threads mittels Vererbung



Quelle: Skript Parallele Programmierung. R. Hennicker 2011

Threads mittels Vererbung

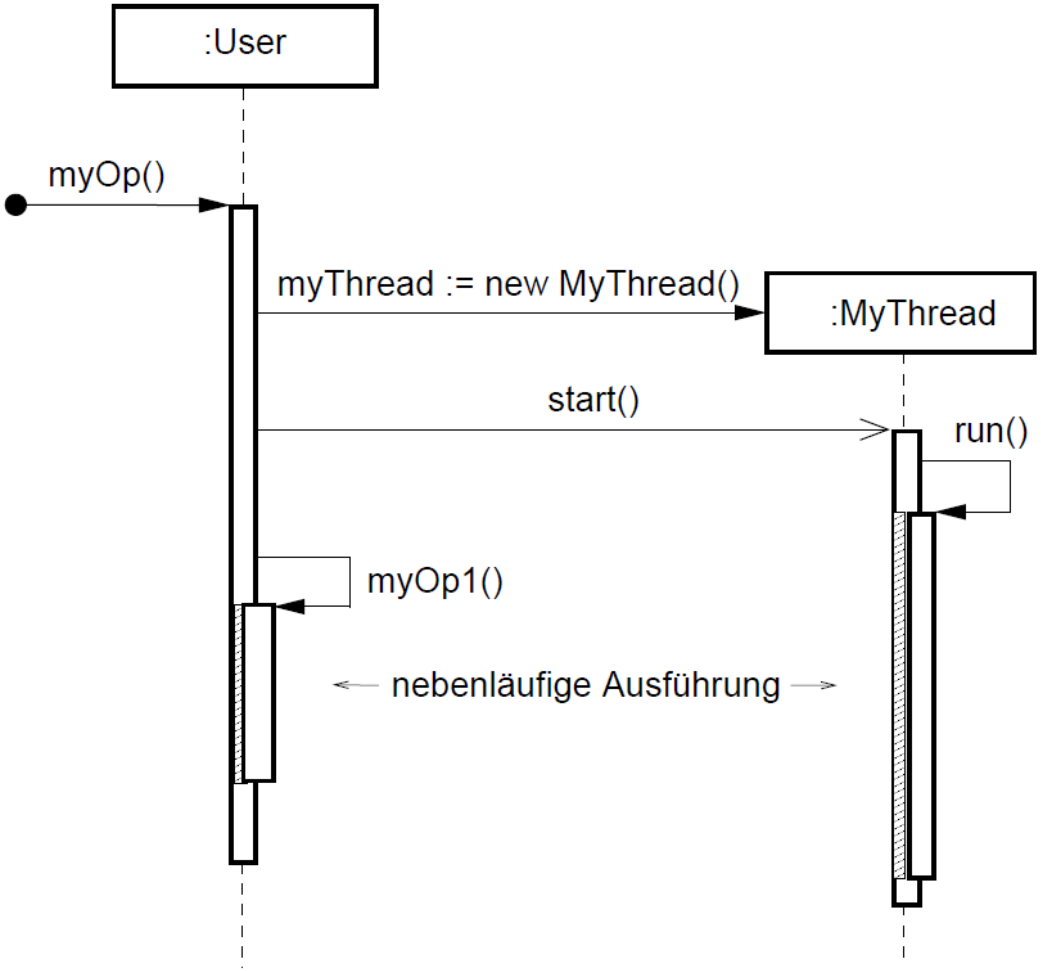
Beispiel-Implementierung mittels Vererbung:

```
//User
public class User {

   public void myOp(){
      MyThread t = new MyThread("Peter");
      t.start();
   }

   public void myOp1(){
      System.out.println("Operation 1.");

   }

   public static void main(String[] args) {
      User u  = new User();
      u.myOp();
      u.myOp1();
   }
}
```

```
// MyThread
public class MyThread extends Thread{


   private String name = "";


   public MyThread(String name){
      this.name = name;
   }

   @Override
   public void run() {
      System.out.println("Hallo ich bin
   "+this.name);
   }
}
```

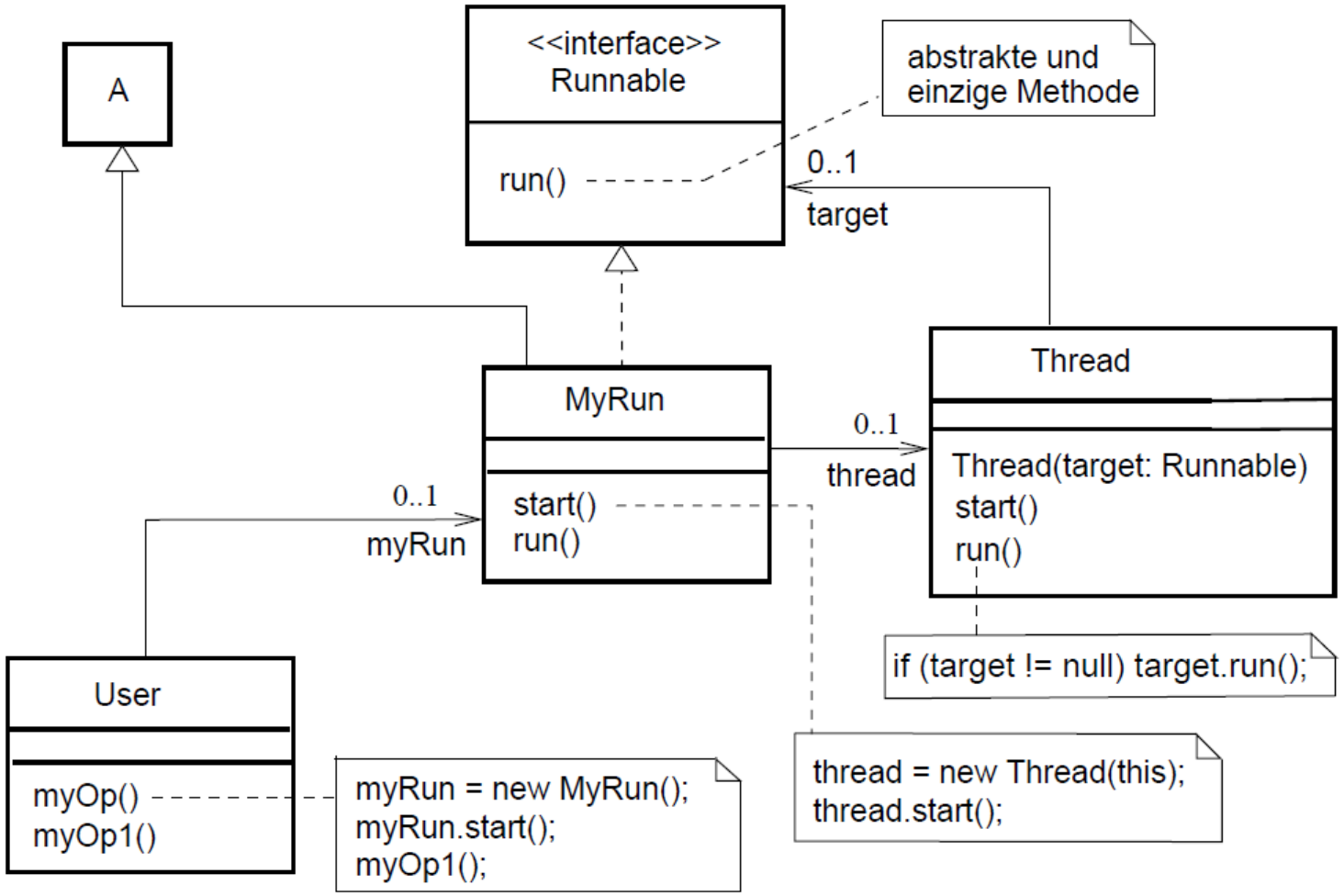
Threads mittels Vererbung

Was passiert: Sequenzdiagramm:



Quelle: Skript Parallele Programmierung. R. Hennicker 2011

Threads über Interface Runnable



Quelle: Skript Parallele Programmierung. R. Hennicker 2011

Threads über Interface Runnable

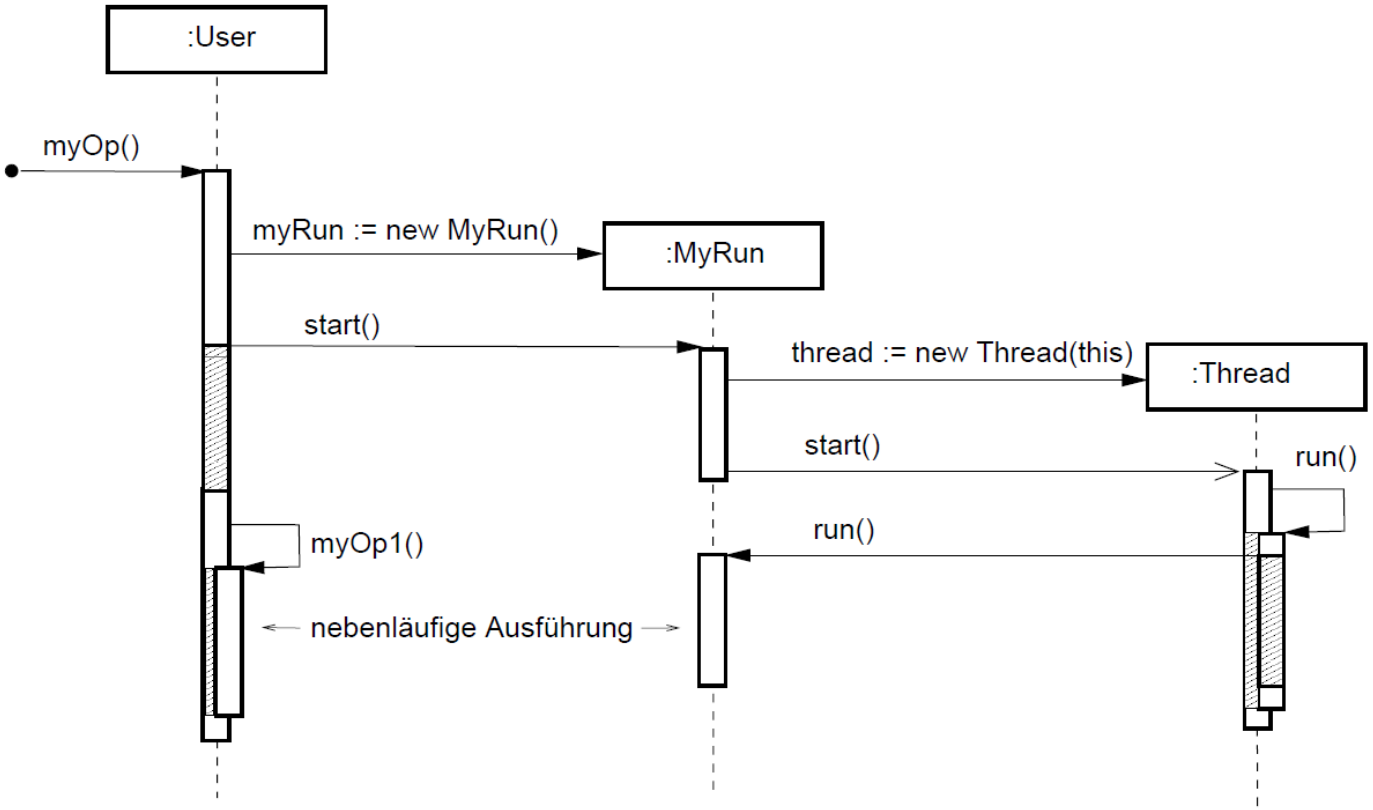
Beispiel-Implementierung mittels Interface Runnable:

```
//User
public class User {

   public void myOp(){
      MyRun myRun = new MyRun("Peter");
      myRun.start();
   }

   public void myOp1(){
      System.out.println("Operation 1.");

   }

   public static void main(String[] args) {
      User u  = new User();
      u.myOp();
      u.myOp1();
   }
}
```

```
// MyRun
public class MyRun implements Runnable{

   private String name = "";

   public MyRun(String name) {
      this.name = name;
   }

   public void start(){
      Thread thread = new Thread(this);
      thread.start();
   }

   @Override
   public void run() {
      System.out.println("Hallo ich bin
      "+this.name);
   }
}
```

Threads über Interface Runnable

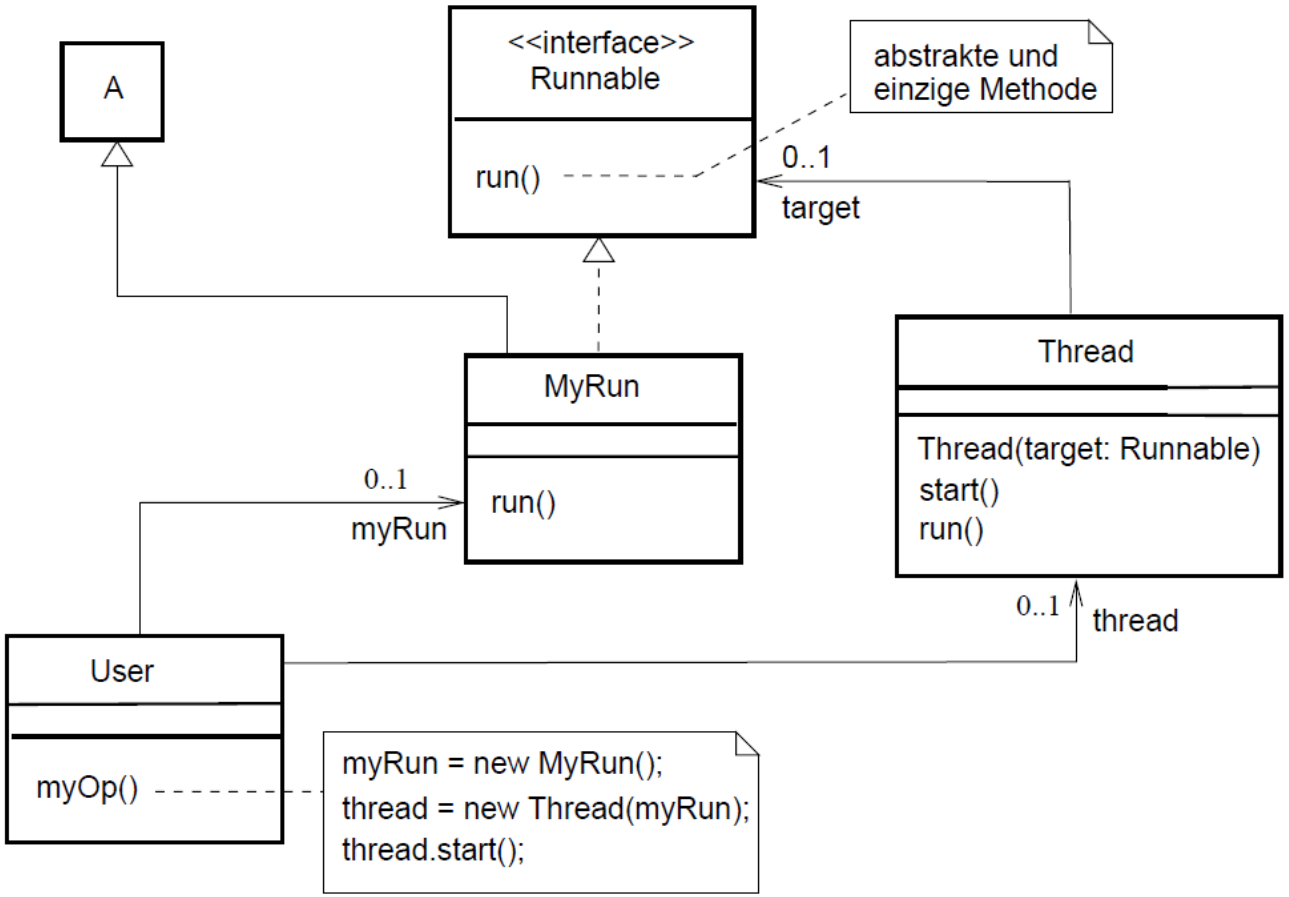
Was passiert: Sequenzdiagramm:



Quelle: Skript Parallele Programmierung. R. Hennicker 2011

Threads über Interface Runnable Variante

Kleine Variante:



Threads über Interface Runnable

Beispiel-Implementierung mittels Interface Runnable (Variante):

```
//User
public class User {

   public void myOp(){
      MyRun myRun = new MyRun("Peter");
      Thread t = new Thread(myRun);
      t.start();
   }

   public void myOp1(){
      System.out.println("Operation 1.");

   }

   public static void main(String[] args) {
      User u  = new User();
      u.myOp();
      u.myOp1();
   }
}
```

```
// MyRun
public class MyRun implements Runnable{

   private String name = "";

   public MyRun(String name) {
      this.name = name;
   }

   @Override
   public void run() {
      System.out.println("Hallo ich bin
   "+this.name);
   }
}
```

Threads direkt im Quellcode: Anonyme Klasse


```
// User
public class User {

   private String name = "";

   public User(String name){
      this.name = name;
   }

   public void myOp(){

      Thread t1 = new Thread(new Runnable() {
         @Override
         public void run() {
            System.out.println("Hallo ich bin "+name);
         }
      });

      t1.start();
   }

   public void myOp1(){
      System.out.println("Operation 1.");
   }

   public static void main(String[] args) {
      User u  = new User("Peter");
      u.myOp();
      u.myOp1();
   }
}
```

Threads direkt im Quellcode: Anonyme Klasse (Alternative)


```
// User
public class User {

   private String name = "";

   public User(String name){
      this.name = name;
   }

   public void myOp(){

      Runnable myRun = new Runnable() {
         @Override
         public void run() {
            System.out.println("Hallo ich bin "+name);
         }
      };
      Thread t = new Thread(myRun);
      t.start();
   }

   public void myOp1(){
      System.out.println("Operation 1.");
   }

   public static void main(String[] args) {
      User u  = new User("Peter");
      u.myOp();
      u.myOp1();
   }
}
```

Einschub: Lambda Ausdrücke

Lambda Ausdrücke wurden in Java 8 eingeführt:

- Ersetzen in den meisten Fällen anonyme Klassen
- Führen zu einfacherem Code
- Hängen direkt mit Funktionsinterfaces zusammen
  - Interface mit genau einer Methode
  - Bsp.: `Runnable`
    - 1 Methode `run();`

Allgemeine Syntax:

- `(Parameter) -> {Body}`
- Passt zu jedem Funktionsinterface, dessen einzige Methode die passende Parameterliste und den passenden Ergebnistyp verlangt.
- Bsp.:
  - `() -> {System.out.println(„Servus Lambda!");}`
  - Ist kompatibel zum Interface Runnable, da kein Parameter und kein Ergebnis verlangt wird

Threads direkt im Quellcode: Lambda Ausdrücke (seit Java 8)

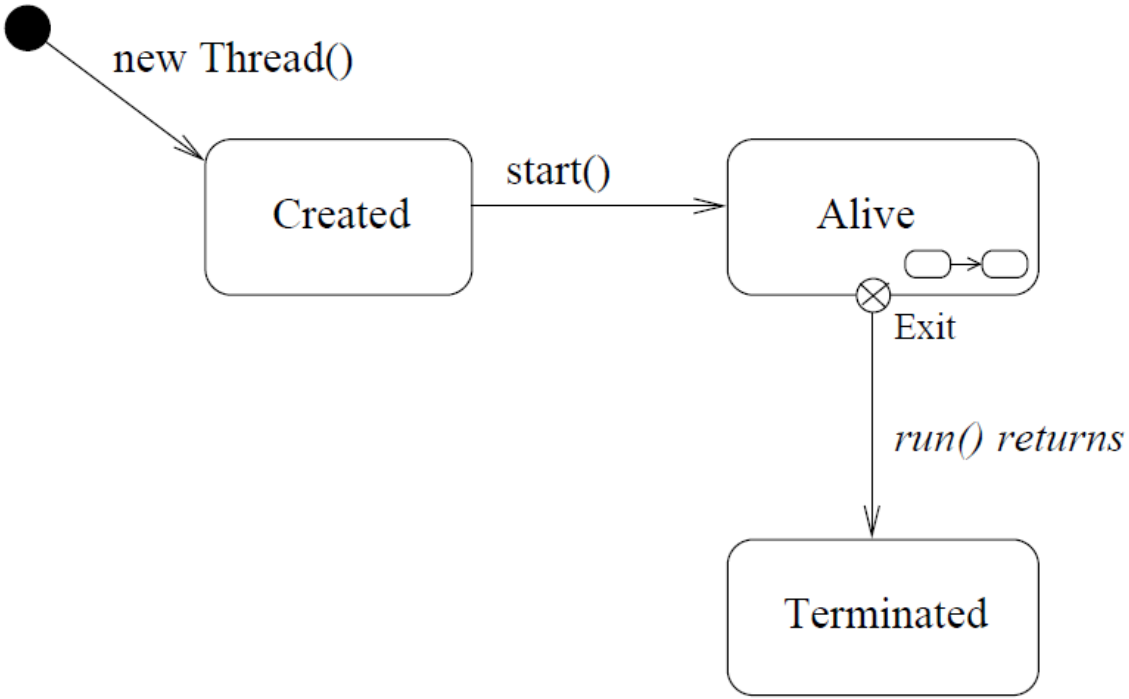

```
public class User {

   private String name = "";

   public User(String name){
      this.name = name;
   }

   public void myOp(){

      Runnable myRun = ()->{
         System.out.println("Hallo ich bin "+this.name);
      };
      Thread t = new Thread(myRun);
      t.start();
   }

   public void myOp1(){
      System.out.println("Operation 1.");

   }

   public static void main(String[] args) {
      User u  = new User("Peter");
      u.myOp();
      u.myOp1();
   }
}
```

Hier kann auf die Instanzvariable des Objekts zugegriffen werden, da keine anonyme Klasse

; nicht vergessen!

Threads in Java: Lebenszyklus

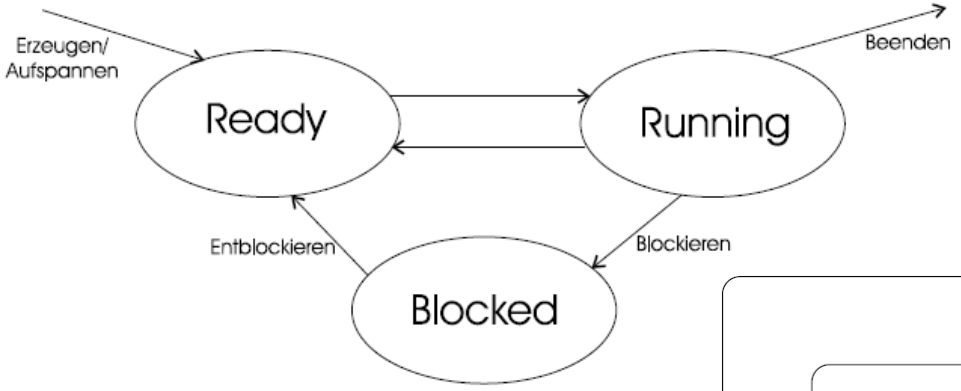
Lebenszyklus eines Java-Threads



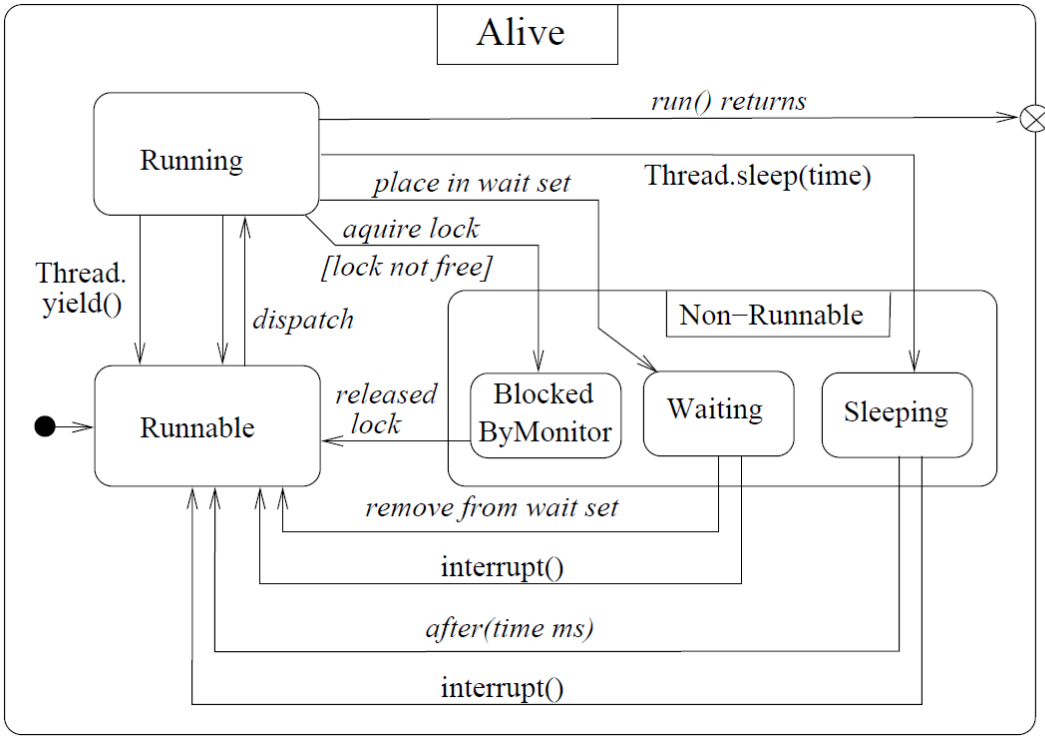
Quelle: Skript Parallele Programmierung. R. Hennicker 2011

Threads in Java: Zustände

Threadzustände (Alive)



Quelle: Skript Betriebssysteme. C. Linnhoff 2015



Quelle: Skript Parallele Programmierung. R. Hennicker 2011

Threads warten lassen

Synchronisation von Threads mittels Monitore

- Nächste Stunde

Threads warten lassen mittels statischer Methode `sleep(long millis);`

- Blockiert den aktuell laufenden Thread für die gegebene Zeitspanne von `long millisekunden`
- Behält aber das Lock des Monitors!
  - Unterschied zu `wait()`
- Wirft `InterruptedException`

```
// Beispiel für Thread.sleep(long millis)

@Override
public void run() {

   for(int i=0;i<100;i++){
      System.out.println("Thread "+this.name+" prints "+i);

      try {
         Thread.sleep(1000);
      } catch (InterruptedException e) {
            e.printStackTrace();
      }

   }

}
```

Übungen mit Threads

Erstellen Sie ein neues Eclipse-Projekt „ThreadsExcercises“ und schreiben Sie nun ein einfaches Programm, um sich mit der Funktionsweise von Java-Threads auseinanderzusetzen:

- Implementieren Sie auf die 4 Arten, die Sie in der Stunde kennengelernt haben eine Run Methode, welche den Namen des ausführenden Threads und eine Zahl von 0 bis 200 hochzählt und ebenfalls ausgibt.
  - Bsp.: „Thread 1 schreibt 35“

- Lassen Sie mindestens 3 Threads für sich arbeiten
- Wie ist der Programmablauf ?
- Wie verhält sich der Programmablauf, wenn Sie statt `thread.start()`, `thread.run()` aufrufen?
- Wie verhält sich der Programmablauf, wenn sie den jeweiligen Thread 100 millisekunden zwischen den Ausgaben warten lassen?