Threads

- Umgangssprachlich Faden, Diskussionsfaden, Gewinde, Faser, Garn, roter Faden
- Threads ermöglichen Nebenläufigkeit (parallele Ausführung von Anwendungsteilen).
- Typisch für Threads ist, dass sie zu einem Prozess gehören und dessen Ressourcen (Variablen) gemeinsam nutzen.
- Java Threads können auf unterschiedliche Weise erzeugt werden. Bei den verschiedenen Möglichkeiten unterscheidet sich der jeweilige Kontext des Threads.

Threads

Eine typische Anwendung ist die Prozessüberwachung mit einem Timeout. Der Prozess wartet in einem Thread auf eine Verbindung, ein Download …, in einem anderen Thread wird eine Zeit lang gewartet. Kommt einer der beiden Threads zum Ende, beendet sich auch der andere Thread. Entweder war die Operation erfogreich oder es wurde ein Timeout erzeugt.

Eine andere Anwendung könnte eine Fortschittsanzeige sein.

Bausteine für Threads

- Ein Objekt der Klasse Thread oder einer von Thread abgeleiteten Klasse
- Eine Methode **`public void run()`**, als überschriebene Methode der Klassse Thread oder als implementierte Methode des Interface Runnable.
- Aufruf der Methode start() der Klasse Thread, um damit die zuständige Methode run, dioe den Thread beschreibt, zu starten.

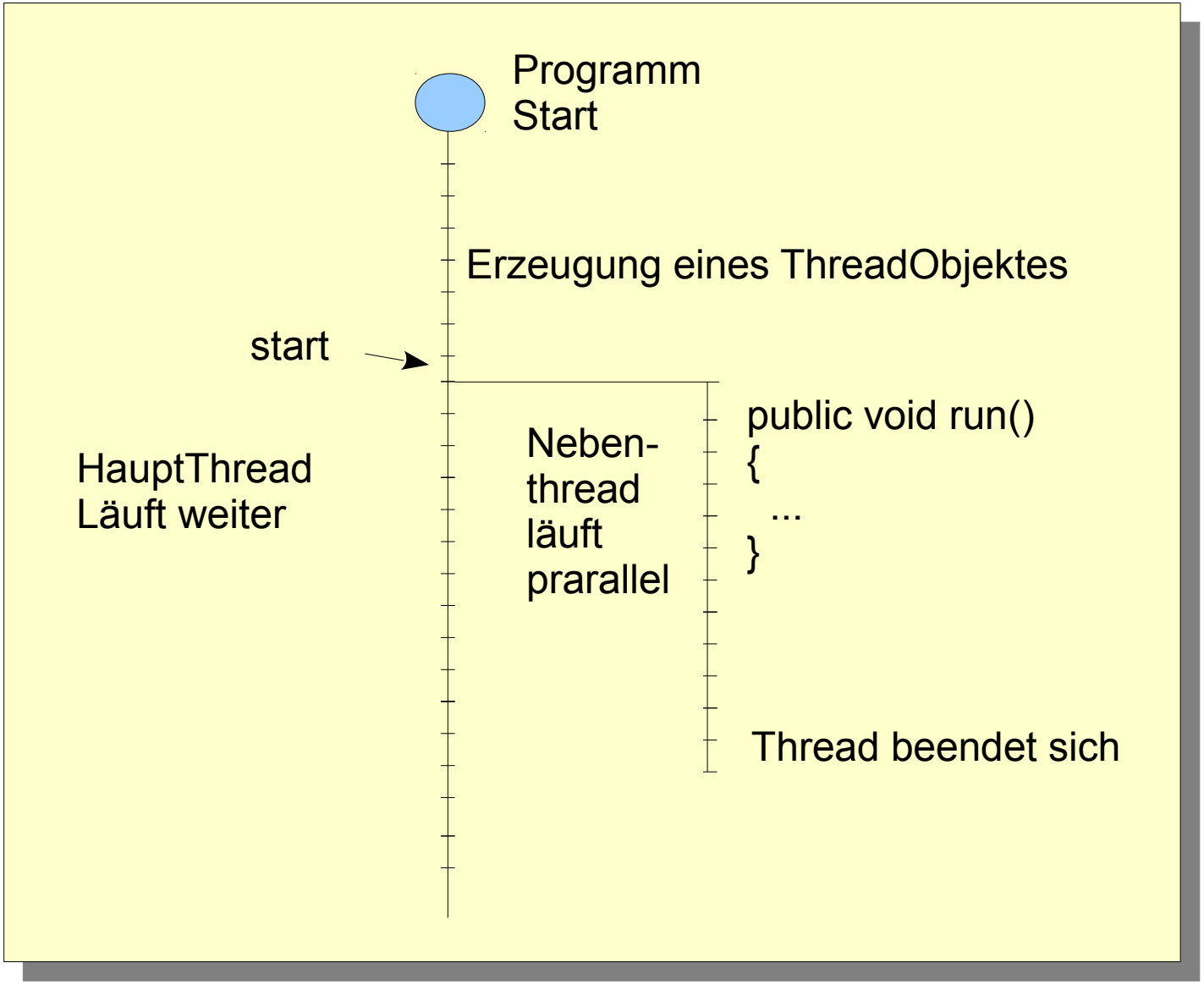
Lebenszyklus von Threads

- Erzeugung eines Thread durch Instanzierung
  - einer Klasse, die von Thread abgeleitet ist oder
  - der Klasse Thread selbst, wobei ein Objekt, dass das Interface Runnable implementiert, als Instanzierungsparameter anzugeben ist.
- Starten des Thread durch Aufruf der Methode start() der Klasse Thread. Dies resultiert in die Ausführung der Methode **`public void run().`**
- Beenden des Thread durch return der run Methode.
- Thread.sleep(long ms) gestattet zeitabhängiges Warten.
- Jeder Thread kann genau einmal gestartet werden.

Lebenszyklus von Threads

- Soll ein Thread nach seiner Beendigung erneut gestartet werden, so ist ein neues Threadobjekt zu erzeugen.
- Über start() des neuen Threadobjektes wird der Thread dann, mit der selben methode run() getstartet.

Ablauf eines Thread



Synchronisation

- Synchronisation von Zugriffen auf gemeinsame Objekte mittels synchronized
- Synchronisation von Ereignissen mittels wait und notify

Beispiel Thread1

- Die wesentlichen Bestandteile finden sich im Constructor:
  - Es wird ein Objekt der Klasse Thread erzeugt, als Parameter wird this übergeben. Dieses Objekt enthält die Methode run, die die auzuführenden Aktionen des nebenläufigen Threads beschreibt.
  - Via T.start() wird der nebenläufige Thread gestartet.
  - Da er sich selbst nicht beendet, läuft er wenigstens bis zum Ende des Hauptthread.
  - Der HauptThread läuft ebenfalls in einer Endlosschleife. Er erwartet eine Eingabe und gibt den eingegebenen String aus.
  - Das Programm wird mit Ctrl-C beendet.

```
import java.io.*;
class Thread1 implements Runnable
{
  public void run() // run beschreibt den auszufuehrenden Thread
  {  // Dieser Code wird nebenlaeufig ausgefuehrt
     while (true)
     {
      System.out.println("Run Thread: ich laufe");
      try{Thread.sleep(1000);}catch (Exception e1){} // 1s warten
     }
  }
  Thread T=new Thread(this);
  public Thread1()
  {
    String s="";
    BufferedReader br
                  =new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Start");
    T.start(); // Hier wird der nebenlaeufige Thread gestartet
    while(true)
    {
      System.out.println("HauptThread: ich laufe "+s);
      try{s=br.readLine();}catch (Exception e1){}
    }
  }
  public static void main(String args[])
  {    new Thread1();  }
}
```

Beispiel Thread2

- Der selbe Ablauf, wie in Beispiel1
  - Es wird die Klasse Thread abgeleitet und die Methode run darin überschrieben.
  - In der Zeile `(new myThread()).start();` Wird das Objekt der abgeleiteten Klasse erzeugt und der Thread via start in Gang gesetzt.
  - Da er sich selbst nicht beendet, läuft er weigstens bis zum Ende des Hauptthread.
  - Der HauptThread läuft ebenfalls in einer Endlosschleife. Er erwartet eine Eingabe und gibt den eingegebenen String aus.
  - Das Programm wird mit Ctrl-C beendet.

```
import java.io.*;
class Thread2
{
  class myThread extends Thread
  {
    public void run()
    {
       while (true)
       {
        System.out.println("Run Thread: ich laufe");
        try{Thread.sleep(1000);}catch (Exception e1){}
       }
    }
  }
  public Thread2()
  {
    String s="";
    BufferedReader br =new BufferedReader(new InputSt...r(System.in));
    System.out.println("Start");
    (new myThread()).start();
    while(true)
    {
      System.out.println("HauptThread: ich laufe "+s);
      try{s=br.readLine();}catch (Exception e1){}
    }
  }
  public static void main(String args[])
  {    new Thread2();  }
}
```

Beispiel Thread3

- Der selbe Ablauf, wie in Beispiel1
  - Hier wird die eigene Klasse von Thread abgeleitet und die Methode run darin überschrieben.
  - Da das eigene Objekt ja bereits existiert, wird der nebenläufige Thread hier mit `start();` oder `this.start();` gestartet.
  - Da er sich selbst nicht beendet, läuft er weigstens bis zum Ende des Hauptthread.
  - Der HauptThread läuft ebenfalls in einer Endlosschleife. Er erwartet eine Eingabe und gibt den eingegebenen String aus.
  - Das Programm wird mit Ctrl-C beendet.

```
import java.io.*;

class Thread3 extends Thread
{
  public void run()
  {
    while (true)
    {
     System.out.println("Run Thread: ich laufe");
     try{Thread.sleep(1000);}catch (Exception e1){}
    }
  }
  public Thread3()
  {
    String s="";
    BufferedReader br=new BufferedReader(new InputStr...er(System.in));
    System.out.println("Start");
    Start();   // oder this.start();
    while(true)
    {
      System.out.println("HauptThread: ich laufe "+s);
      try{s=br.readLine();}catch (Exception e1){}
    }
  }
  public static void main(String args[])
  {    new Thread3();  }
}
```

Beispiel Thread2StartStopp

- Der selbe Grundaufbau, wie in Thread2
  - Die Eingabe im HauptThread wird für die Threadsteuerung genutzt.
  - Die Eingabe wird in der Instanzvariable s gepeichert. Enthält sie “stop“, so beendet sich der nebenläufige Thread, in dem er die run Methode verlässt.
  - Bei Eingabe von start wird, wenn es noch keinen laufenden Nebenthread gibt ein neues Threadobjekt erzeugt und der Thread über “start“ gestartet.
  - “quit“ beendet den HauptThread.
  - Wird das Programm bei “quit“ beendet, wenn ein Nebenthread läuft? Finden Sie es heraus!

wait/notify

- erweitert den Synchronisationsmechanismus
- object.wait() hält einen Thread an, bis die Methode object.notify() aufgerufen wird.
- Beide Aufrufe müssen in einem syncronized Block zu dem selben Objekt aufgerufen werden, also mit dem selben lock operieren.
- Somit wird eine Ereignissynchronisation möglich.

wait/notify

- Die Ausführung des wartenden Threads kann frühestens nach dem Verlassen der kritischen Region des freigebenden Prozesses erfolgen (nach notify, nach Verlassen des synchronized block)
- Gibt es mehrere durch wait() wartende Threads, wird nur ein Thread durch notify() freigegeben.
- Über einen Parameter bei wait kann eine timeout Zeit angegeben werden.
- Beispiele folgen in dem Kapitel Images und drawing.

daemon/user Thread

- Jedes Java-Programm läuft zunächst als ein user Thread, der mit dem Programm beendet wird.
- User Threads laufen ansonsten weiter, auch wenn der erzeugende Thread beendet ist.
- Daemon Threads werden dagegen beendet, wenn der erzeugende Thread beendet wird.
- Über setDaemon(boolean) kann eingestellt werden, um welche Art Thread es sich handeln soll. Dies muss vor dem start des Threads erfolgen.