Inhalt der Vorlesung

1. Klassen und Objekte
2. Vererbung
3. Enums, Wrapper und Autoboxing
4. Interfaces
5. Generics
6. Exceptions
7. Polymorphismus
8. Grafische Benutzeroberflächen mit JavaFX
9. Streams und Lambda Expressions
10. Leichtgewichtige Prozesse – Threads

Kapitel 10: Leichtgewichtige Prozesse – Threads

Inhalt

Kapitel 10: Leichtgewichtige Prozesse – Threads

Lernziele

[LZ 10.1] Den Thread-Begriff und die Anwendungen von Threads erklären können

[LZ 10.2] Das Thread-Konzept in Java einschl. der Thread-Erzeugung erklären können

[LZ 10.3] Erläutern können, wie Threads terminiert werden

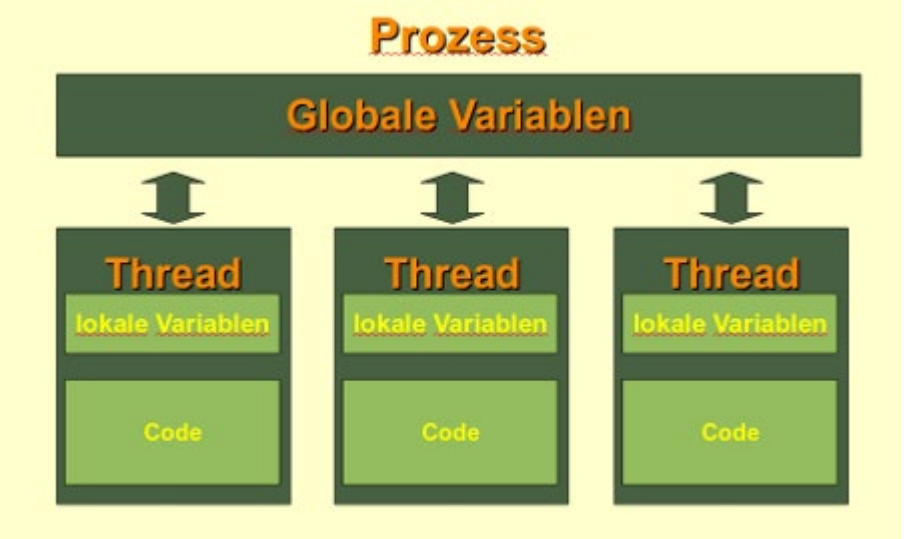
[LZ 10.4] Wichtige Thread-Methoden und ihre Anwendung kennen

10. Threads

10.1 Prinzip

Ein Thread ist ein leichtgewichtiger Prozess, der eingebettet in einem Prozess des Betriebssystems ausgeführt wird. Mehrere Threads teilen sich den (virtuellen) Speicher des Prozesses, wobei jeder Thread einen eigenen Methodenaufruf-Keller besitzt (siehe Bild unten). Jeder Thread führt damit eigene Methodenaufrufe auf Objekten aus. Die auf dem Heap allokierten Java-Objekte können von allen Threads parallel genutzt werden.

Anwendungsbereiche von Threads sind u.a. nicht-blockierende I/O, GUI-Programmierung (parallele UI-Aktualisierung: http://www.sorting-algorithms.com), Textverarbeitung (parallele Rechtschreibprüfung, Speichern im Hintergrund), Beschleunigung der Berechnung von Algorithmen (Nutzung mehrerer CPUs), …



Java bietet in der Laufzeitbibliothek eine Thread-Klasse und ein Interface Runnable:

```
interface Runnable {
    public void run();
}
```

Ein Thread kann prinzipiell auf zwei Arten erzeugt und gestartet werden:

1. Ableitung einer eigenen Klasse von `Thread`: man definiert eine eigene Klasse, welche von Thread erbt. Dabei sollte die `run`-Methode der Thread-Klasse überschrieben werden, sie enthält die Anweisungen, die nach dem Start des Threads mittels der vorgegebenen Methode `start` ausgeführt werden.
2. Erzeugung eines Thread-Objekts, dabei Übergabe eins Objekts, welches das Runnable-Interface implementiert (also eine run-Methode besitzt). Der Thread wird dann wie auch bei 1. durch Aufruf der start-Methode gestartet, was zur Ausführung der run-Methode führt.

Beispiel zu Möglichkeit 1: Ableitung von der Klasse Thread

Die run-Methode wird überschrieben, sie implementiert die Hauptschleife des Threads, welche beim Start des Threads ausgeführt wird.

```
class MyThread extends Thread {
    private int zaehler = 0;
    public void run() {
        System.out.println(zaehler++);   // das, was der Thread tut…
    }
    public static void main(String args[]) {
        Thread t = new MyThread();
        t.start();           // Thread starten, run wird ausgeführt
    }
}
```

Beispiel zu Möglichkeit 2: Thread erzeugen und Runnable-Objekt übergeben
Dies ist die vorzuziehende Möglichkeit, da keine technisch bedingte Vererbung eingeführt wird. Sie ist insbesondere dann geeignet, wenn bereits von einer Klasse geerbt wird, d.h. das Ableiten von Thread nicht mehr möglich ist.
Es wird ein Lambda-Ausdruck „r“ definiert, der die später auszuführende run-Methode enthält. „r“ wird bei Erzeugung des Threads übergeben und kann somit kann bei beim Start des Thread ausgeführt werden.

```
class Starter {
    public static void main(String args[]) throws Exception {
        Runnable r = () -> {
            int zaehler = 0;
            while (true)
                System.out.println(zaehler++);   // endlos den Zähler ausgeben und hochzählen
        };
        Thread t = new Thread(r);
        t.start();           // Thread starten, run wird ausgeführt
    }
}
```

Ohne eine Lambda Expression kann das letzte Beispiel folgendermaßen umgesetzt werden:

```
class Starter {
    public static void main(String args[]) throws Exception {
        Runnable r = new Runnable() {
            public void run() {
                int zaehler = 0;
                while (true)
                    System.out.println(zaehler++);
            }
        };
        Thread t = new Thread(r);
        t.start();          // Thread starten, run wird ausgeführt
    }
}
```

Im obigen Beispiel wird eine anonyme Klasse definiert und instanziiert, die das Runnable-Interface implementiert. Thread-Erzeugung und Starten des Threads sind analog zum Beispiel mit Lambda Expressions auf der vorherigen Seite gestaltet.

Threads terminieren automatisch mit dem Ende der Ausführung ihrer run-Methode. Eine gängige Methode, eine Thread zu beenden, ist folgende Konstruktion (hier mit einer Thread-Ableitung):

```
class MyThread extends Thread {
    private boolean terminiert = false; // der Thread ist aktiv
    public void run() {
        while (!terminiert) {
            // Ausführung der Thread-Aktivität
        }
    }
    public void terminiere() {
        terminiert = true; // der Thread ist inaktiv, s. run()
    }
    public static void main(String args[]) {
        Thread t = new MyThread();
        t.start();           // Thread starten
        Thread.sleep(1000);  // 1000 ms warten
        t.terminiere();      // erzeugten Thread beenden
    }
}
```

Das Attribut „terminiert“ steuert die Ausführung des erzeugten Threads. Solange es den Wert „false“ besitzt, ist der Thread aktiv.

Statische Methoden:

currentThread()	gibt Referenz auf den aktuellen Thread zurück
sleep(long)	pausiert den aufrufenden Thread für die übergebene Anzahl Millisek.

Instanzmethoden:

getName()	liefert den Namen des aktuellen Threads (Thread-0, Thread-1,. . .)
join()	Aufrufer-Thread wartet auf Beendigung des aufgerufenen Threads

Beispiel:

```
Thread t = new Thread(() -> {
    for (int i = 0; i < 1000; ++i)
        System.out.println(i);
};
t.start();
t.join(); // blockieren, bis der Thread terminiert
```