Java 8 auf einen Blick (Screenshot zum Buch!)

(Bereits März 2014 erschienen)

1. Funktionale Programmierung (Lambda Ausdrücke)
   * Das Kernmerkmal von FP ist also das Fehlen von Zustand und Seiteneffekten; Alle Funktionen sind "pur", d. h. sie bekommen etwas übergeben und liefern etwas zurück, und das ist alles was sie tun. Also keine Seiteneffekte, wie zB etwas zu verändern, dass sich ausserhalb der Funktion befindet.
2. Massenoperationen auf Collections (Filterung, Transformation, Sortierung)
3. Neue Date und Time API
4. Diverses
5. (Java FX 8)

Lambda ist vom Verständnis her ähnlich eine einfache Methode, die aber keinen namen hat, keinen expliziten rückgabetyp und keine informationen über ausgelöste Exceptions.

{Parameterliste} -> {Ausdruck oder Anweisung}

//Ref 1

// does not compile: incompatible types: Object is not a functional interface

Object hello = () -> (System.out.println("Hello");)

Object temp = (int x, int y) -> {return x + y;};

Wie rufen wir diesen Lambda Ausdruck auf? Bis Java 8 konnte jede Referenz auf Object abgebildet werden. Lambdas kann man nur „Functional Interfaces“ zuweisen.

Im java.util.function package gibt es vordefinierte Funktionale interfaces

@FunctionalInterface

interface MathOperation {

int operation(int x, int y);

}

Funktionales Interface == Interface mit einer abstrakten Methode == auch SAM Typ (Single Abstract Method) genannt

Type Inference 🡪 ähnlich zum Diamond Operator kann man explizite Typ Angaben weglassen. Compiler ermittelt aus dem Einsatzkontext was wir wollen.

List<E> wurde auch erweitert zb durch .sort()

Companies.forEach(…) 🡪 siehe Java Doc 🡪 Interfaces default implementierung!

Fazit:

Gut für eine kompakte Schreibweise, aber nur für kurze logik, debuggen schwierig

Massenoperation auf Collections

Externen Iteration 🡪 Traversierung im Applikationscoede

Interne Iteration 🡪 Framework übernimmt die Iteration

Fazit: Iterationsalgorithmen werden in Java 8 obsolet.

Wir wissen wie man eine interne iteration vornimmt, es wäre sinnvoll die ganzen if abfragen wo attribut X auf bedingung Y geprüft wird irgendwie zu vereinfachen, dies führte zu 🡪

Interface Predicate <T>

* Sind boolsche Ausdrücke

Filterung möglich, ja , aber man fühlt sich vom framework immer noch eingeschränkt, zb nur ein Prädikat möglich auf removeIf 🡪 Abhilfe:

Streams:

* Folge von Verarbeitungsschritten auf Daten (Analogie: Fließband)
* keine Speicherung der Daten (!)
* können nur einmal durchlaufen (traversiert) werden
* bestehen 3 Operationstypen:
  + Create (Erzeugung)
  + Intermediate(Berechnung)
  + Terminal(Ergebnisermittlung)

parallelStream <> stream warum nicht immer parallelstream? Sorgt für einen großen Overhead um die Threads zu koordinieren 🡪 sequentielle Verarbeitung ist bei millionen von integer werten schneller (!) als parallelStream, erst so ab 1 Milliarde beträgt der faktor 1:4 für (!) parallelstream

Streams 🡪 mapToInt zb, .sum(), .max(), .min(), .average()

stream(). 🡪 zb allMatch(), anyMatch(), noneMatch(), findFirst(), findAny()

reduce();

Source 🡪 Stream 🡪 Op1 + Op2 + Op3 … 🡪 Result

Todo .stream(). Methoden erklären können

person -> "" + person.getAge()) 🡪 wandelt int zu String um