Java 8 (2014)

Lambdas

```
public static List<Persion> filterPerson(List<Person> persons, PersonPredicate predicate)

filterPerson(person, (Person p) -> p.getCountry().equals"Germany"));
filterPerson(person, (Person p) -> p.getAge() > 10));
```

Funktionales Interfaces

```
void tueEtwas(Runnable rn) {
   rn.run();
}

tueEtwas() -> {System.out.println("Hello World") }
```

Abstrakte Interfaces --> Default-Keyword

```
public interface List<E> extends Collection
{
   default void replace()
   {
      //....
   }
}
```

Factory Interfaces

```
public interface Stream<T>
{
   public static<T> Stream<T>of(T...values)
   {
      return Arra<.stream(values);
   }
}
```

Komposition

Compose-Funktion (inkl. UnaryOperator-Interface)

Lazyness

```
StringHelper.of("TeXeT)
   .transform(String::toLowerCase)
   .transform(String::trim)
   .toString();
```

Stream API

Idee ; Verarbeitung von Collection-Daten

Beispiel: Maximal eine Person die aus Deutschand kommt

```
List<Person> personList = personList
                    .stream()
                    .limit(1)
                    .filter(p -> p.getCountry().equals("Germany"))
                    .forEach(p -> { System.out.println(p) } )
```

```
IntStream.iterate(i, i -> i+1)
   .limit(10)
   .forEach(System.out::print);

Stream.builder
   .add("AAA")
   .add("BBB")
   .add("CCC")
   .add("DDD")
   .build
   .forEachOrdred(System.out::print);

Stream.of(1, 2, 3, 4, 5).forEach(System.out::print);

Stream.of(PersonUtil.getListe1(), PersonUtil.getListe2())
   .flatMap(pList -> plist.stream() )
   .forEach(System.out::println)

Stream.of(PersonUtil.getListe1(), PersonUtil.getListe2())
   .flatMap(pList -> plist.stream().map(p -> p.getFirstname() + " " + p.getName() ) )
   .forEach(System.out::println)
```

Komma seperierte Liste

```
liste.stream()
   .map(Personen::getFilename())
   .collect(Collectors.joining(","));
```

Personen mit dem Anfangsbuchstaben "M"

```
liste.stream()
   .filter(p -> p.getFirstname().startWith("M"))
   .forEach(System.out::print);
```

Aggregationen

Sortierung

```
liste.stream()
   .sorted()
   .forEach(System.out::println)
```

Anzahl Elemente

```
liste.stream().count()
```

```
liste.stream()
   .collect(Collectors.groupBy(Person::getCountry(), Collectors.counting()))
   .forEach( (county, count) --> System.out.println(String.format("%s (%s)", country,
count)) );
```

Transformationen

- Mapping = Objekte mit Eingabetype in Objekte eines Ausgabetyps umwandeln
- FlatMap = Collection-Elemente in Stream-Elemente begradigen (Array[][] --> Array[])

// Combining / Shortening

- Liste verkleinern = Elemente aus dem Ergebnis werfen
- Filter = Bestimmte ELemente nach Kriterien aussortieren
- Reduce = ELemnete miteinander verbinden zur Reduktion auf ein Element
- Concat/Joining = Elemente miteinander verbinden (z.B. Strinfs)
- Limit = Anzahl der Ergebnisse begrenzen

Aggregation

- Sorting = Sortieren der ELemente im Stream mit Comparator
- Min/Max = Maximal- und Minimal Werte einer Collection bestimmen
- Grouping = Gruppieren gleicher Elemente
- Discinct = Einzigartige ELemente bestimmen Equals-Methode überschreiben

Ergebnisse generieren

- Lazyness = Operationen generieren im regelfalL KEIN ERGEBNIS
- forEach = Iteration über die resultierende Elemente
- Collectors = Aufbauen neuer Collections aus xdem Ergebnissen der Operationen
- toArray = Aufbauen eines Arrays aus den Ergebnissen der Operation

Parallelisieren

- Anstatt "stream" --> "parallelStream" auf Collection
- parallel() auf Stream

- Primitive Streamns --> IntStream....,

// Zeiten messsen
Instant start = Instant.now();
Instant end = Instant.now();
System.out.println(Duration.between(start, end));

DateTime API

LocalTime
LocalDate
LocalDateTime
ZonedDateTime
Instant
TemporalAdjuster
Duration
DateTimeFormater

*** Zeit bestimmen

LocalDate.now() = Aktuelles Datum
LocalTime.now() = Aktuelle Zeit
LocalDateTime.now() = Aktuelle Zeit + Datum
Instant.now() = Aktuelle Maschinenzeit
LocalDate.of(2014, Month.MARCH, 18) = Bestimmte Zeitpunkte aufbauen

*** Zeitoperationen

LocalTime.now().plusHours(1)
LocalTime.now().getHour()
LocalTime.of(23,59).isBefore(LocalTime.of(0.0))

*** Datumsoperationen

LocalDate.now().plusDays(1)
LocalDate.now().getDaysOfWeek()

*** Zeitzone

ZonedDateTime newYork = ZonedDateTime.of(
LocalDate.now(),
LocalTime.new(),
ZoneId.of("America/New_York)
);

*** TemporalAdjuster

.next(DayOfWeek.MONDAY)

.previos(DayOfWeek.MONDAY)
.lastInMonth(DayOfWeek.MONDAY)

*** DateTimeFormater

DateTimeFormatter.ofLocalizeDate(FormatStyle.SHORT)
DateTimeFormatter.ofPattern("dd.MM.yyyy")

siehe
https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html

JavaFX 8

Nashorn (Java-Script-Engine)

Optional-Konzepi

Problem : Umgehen der Null-Pointer-Exception
Lösung : Optional

Methode:

```
Optional<String>MachWas
{
   try {
      String text = .....
   }
   catch (IOException) {
      return Optional.empty();)
   }
   return Optional.of(text(text);
}
```

Aufruf:

```
Optional<String> result = Machwas(;
if (result.isPOresent) result.get();

String result = MachWas().orElse("War leer");

MachWas().ifPresent(System.put::println);
```

Doppelte Annotationen

```
@Target(ElementType.TYPE_USE)
public @interface Unique {}

void method()
```

```
{
   @Unique String name = "";
}
```

Speicherverwaltung

-XX:MaxPermSite/PermSize ist überflüssig