Was ist eine Monade und was kann ich damit machen?

Michael Sperber

- Individualsoftware
- branchenunabhängig
- Scala, Clojure, Erlang, Haskell, F#
- Schulungen, Coaching

www.active-group.de

funktionale-programmierung.de

java.util.Optional<T>

```
<U> Optional<U>
flatMap(Function<? super T,
                 Optional<U>> mapper)


static <T> Optional<T>
of(T value)
```

java.util.stream.Stream<T>

```
<R> Stream<R>
flatMap
 (Function<? super T,
           ? extends Stream<? extends R>>
  mapper)

static <T> Stream<T>
of(T t)
```

java.util.Optional<T>

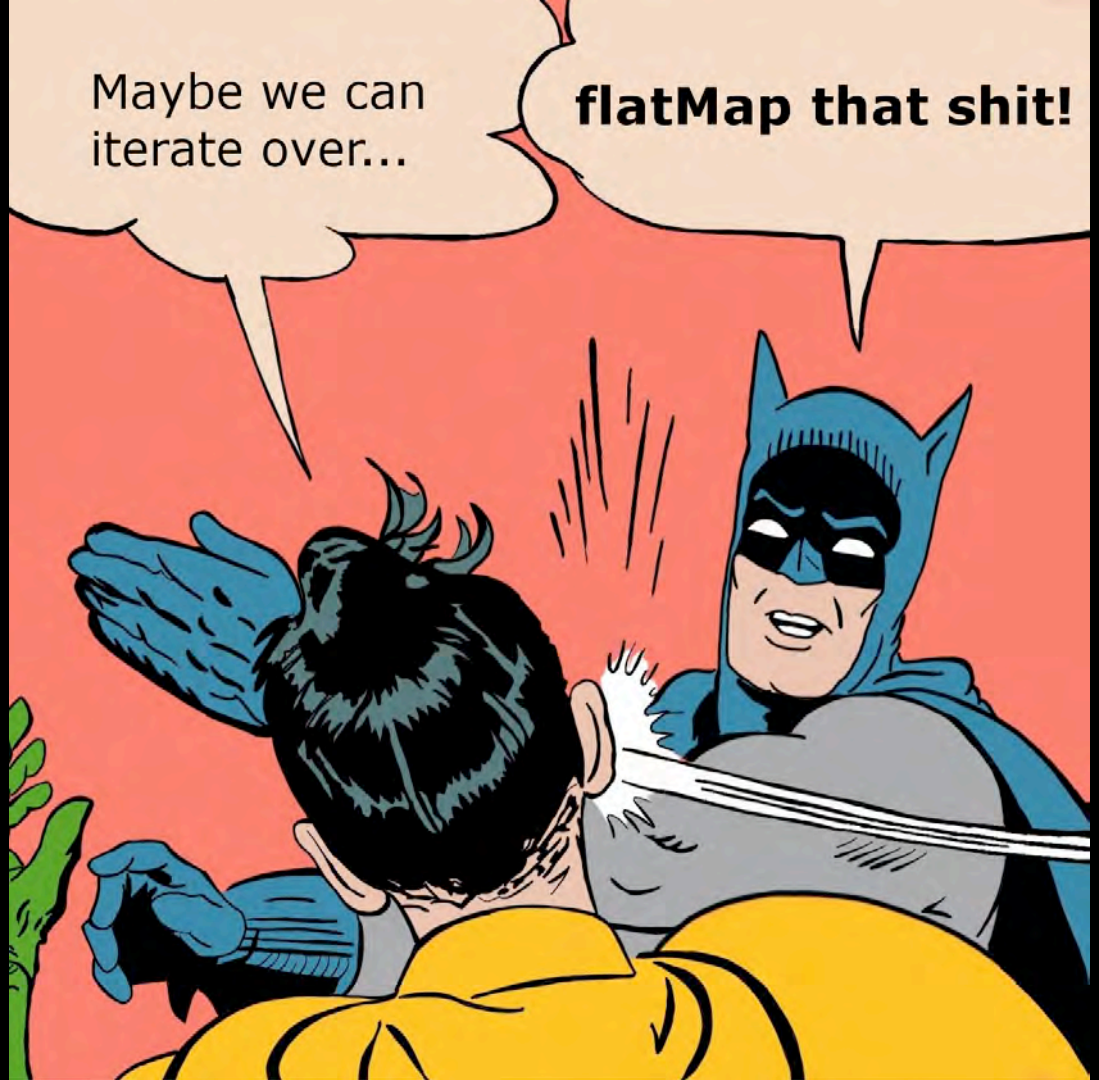
```
<U> Optional<U>
flatMap(Function<T,
                 Optional<U>> mapper)


static <T> Optional<T>
of(T t)
```

java.util.stream.Stream<T>

```
<U> Stream<U>
flatMap(Function<T,
                 Stream<U>> mapper)


static <T> Stream<T>
of(T t)
```



flatmapthatshit.com

```
public class Flight {
  public @Nullable Reservation
      getReservation() { ... }
}
```

```
public class Reservation {
  public @Nullable Seat
      getSeat() { ... }
}
```

```
public class Seat {
  public @Nullable Champagne
      getChampagne() { ... }
}
```

```
public String champagneBrand() {
    if (this.getReservation() == null)
        return "unknown";
    if (this.getReservation().getSeat() == null)
        return "unknown";
    if (this.getReservation().getChampagne() == null)
        return "unknown";
    return
       this.getReservation().getChampagne().getBrand();
}
```

```
public class Flight {
  public Optional<Reservation>
      getReservation() { ... }
}
```

```
public class Reservation {
  public Optional<Seat>
      getSeat() { ... }
}
```

```
public class Seat {
  public Optional<Champagne>
      getChampagne() { ... }
}
```

```
public String champagneBrand() {
 return this.getReservation()
            .flatMap(Reservation::getSeat)
            .flatMap(Seat::getChampagne)
            .map(Champagne::getBrand)
            .orElse("unknown");
}
```

map

```
public class Optional<T> {
  <U> Optional<U>
      map(Function<T, U> mapper);
}

public class Stream<T> {
  <U> Stream<U>
      map(Function<T, U> mapper;
}
```

Teilstrecken eines Flugs

```
public class Flight {
    public List<Leg> getLegs();
}
```

Alle Teilstrecken

```
public static List<Leg>
 allLegs(List<Flight> flights) {
    return flights.stream()
                .flatMap(f ->
                        f.getLegs().stream())
                .collect(Collectors.toList());
}
```

Gemeinsamkeiten

- Typ `M` mit generischem Parameter `T`
- `flatMap: M<T> (T => M<U>) => M<U>`
- `of     : T => M<T>`
- `map    : M<T> (T => U) => M<U>`

Wobei ...

```
public class Stream<T> {
 <U> Stream<U>
  map(Function<T, U> mapper) {
   return
    this.flatMap
        (x -> Stream.of(mapper.apply(x)));
  }
}
```

Monade

- Typ `M` mit generischem Parameter `T`
- `flatMap: M<T> (T => M<U>) => M<U>`
- `of      : T => M<T>`

Monaden in der funktionalen Programmierung

$$m\ \alpha$$

$$\texttt{unit} : \alpha \to m\ \alpha$$

$$\texttt{bind} : m\ \alpha \times (\alpha \to m\ \beta) \to m\ \beta$$

Web-Service

```
public static int balance(String account) {
  CloseableHttpClient cl = HttpClients.custom()
          .setConnectionManager(
                  new PoolingHttpClientConnectionManager())
          .build();

  HttpGet rq = new HttpGet("http://www.account.com/");
  rq.addHeader("account", account);
  HttpResponse rsp = cl.execute(rq);
  int bl =
    Integer.valueOf(EntityUtils.toString(rsp.getEntity()));

  cl.close();
  return bl;
}
```

HttpClient sharen

```
public static int balance(String account) {
  CloseableHttpClient cl = HttpClientManager.getClient();

  HttpGet rq = new HttpGet("http://www.account.com/");
  rq.addHeader("account", account);
  HttpResponse rsp = cl.execute(rq);
  int bl =
      Integer.valueOf(EntityUtils.toString(rsp.getEntity()));

  cl.close();
  return bl;
}
```

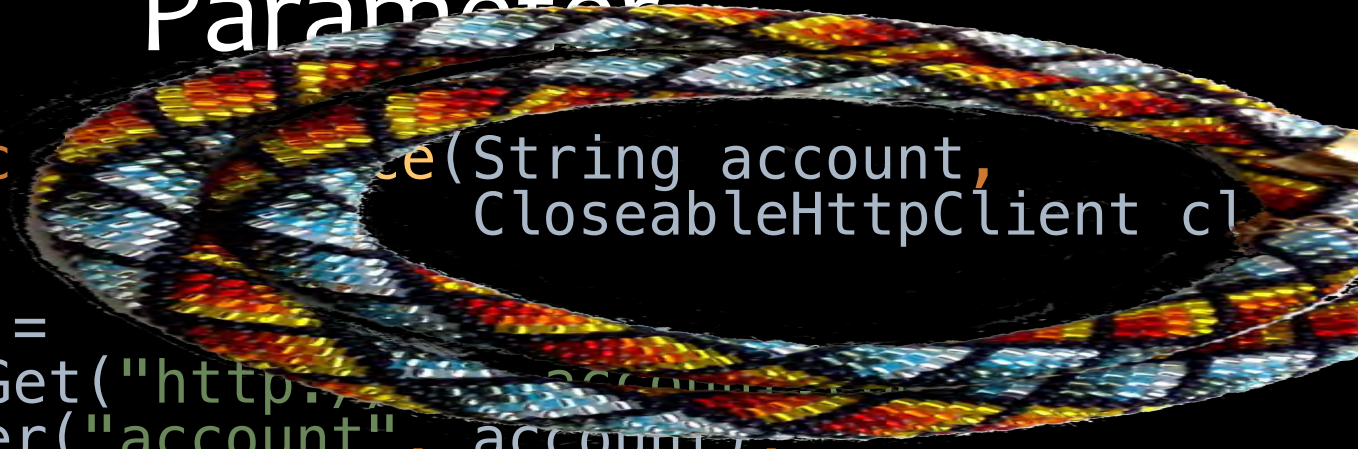
HttpClient sharen

```
public class HttpClientManager {
    private static CloseableHttpClient instance;

    public static void
       setConnectionManager(HttpClientConnectionManager mg) {
        HttpClientManager.instance =
                HttpClients.custom()
                        .setConnectionManager(mg)
                        .build();
    }

    public static CloseableHttpClient getClient() {
       return HttpClientManager.instance;
    }
}
```

Parameter



```
public static                    e(String account,
                                   CloseableHttpClient cl
{
  HttpGet rq =
    new HttpGet("http:/
  rq.addHeader("account", account);
  HttpResponse rsp = cl.execute(rq);
  int bl =
     Integer.valueOf(
       EntityUtils.toString(rsp.getEntity()));
  return bl;
}
```

Überall durchfädeln?

Schönfinkeln

$$\texttt{curry} : (\alpha \times \beta \to \gamma) \to (\alpha \to (\beta \to \gamma))$$

```
public static int balance(String account,
                          CloseableHttpClient cl)
```

Schönfinkeln

```
public interface ConnOp<T> {
    public T run(HttpClient cl);
}
```

Schönfinkeln

```
public static ConnOp<Integer> balance(String account) {
  return cl -> {
    HttpGet rq =
      new HttpGet("http://www.account.com/");
    rq.addHeader("account", account);
    HttpResponse rsp = cl.execute(rq);
    int bl =
      Integer.valueOf(
        EntityUtils.toString(rsp.getEntity()));
    return bl;
  }
}
```

Parameter

```
public static int balance(String account,
                          CloseableHttpClient cl)
{
  HttpGet rq =
    new HttpGet("http://www.account.com/");
  rq.addHeader("account", account);
  HttpResponse rsp = cl.execute(rq);
  int bl =
     Integer.valueOf(
       EntityUtils.toString(rsp.getEntity()));
  return bl;
}
```

Andere Funktionen

```
public static ConnOp<Void> pay(String account,
                               int amount) {
  return cl -> {
    HttpPost rq = new HttpPost("http://www.pay.com/");
    rq.addHeader("account", account);
    rq.addHeader("amount", Integer.toString(amount));
    HttpResponse rsp = cl.execute(rq);
    EntityUtils.consume(rsp.getEntity());
    return null;
  };
}
```

Was ist das?

```
public static <T, U> ConnOp<U>
  flatMap(ConnOp<T> op,
          Function<T, ConnOp<U>> f) {
    return cl ->
             f.apply(op.run(cl)).run(cl);
}

public static <T> ConnOp<T> of(T x) {
    return mg -> x;
}
```

Monadisch programmieren

```
public static ConnOp<Integer>
  rob(String account) {
     return flatMap(balance(account), bl ->
               flatMap(pay(account, bl), _v ->
                          of(bl)));
}
```

Monade

„Inversion of Control“

```
public interface ConnOpRunner {
  public <T> T run(ConnOp<T> op);
}
```

„Dependency Injection“

```
public class PoolingRunner implements ConnOpRunner {
  @Override
  public <T> T run(ConnOp<T> op) {
    CloseableHttpClient cl = HttpClients.custom()
            .setConnectionManager(
              new PoolingHttpClientConnectionManager())
            .build();
    T res = op.run(cl);
    cl.close();
    return res;
  }
}
```

Monaden kapseln Berechnungen

- Zustand
- I/O
- Exceptions
- GUIs
- Nebenläufigkeit
- Parallelität
- Wahrscheinlichkeitsverteilung
- Hoffnung
- DSLs
- …

Zutaten

- Berechnung mit Ergebnis
- „Sequenzialität“
- Schleim

Effekte

Signaturen

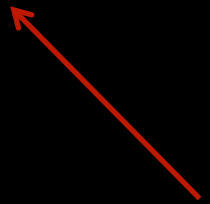
$$m\ \alpha$$

$$\texttt{unit} : \alpha \to m\ \alpha$$

$$\texttt{bind} : m\ \alpha \times (\alpha \to m\ \beta) \to m\ \beta$$

Gesetze

$$
\begin{aligned}
\texttt{bind}(\texttt{unit}(a), f) &\equiv f(a) \\
\texttt{bind}(m, \texttt{unit}) &\equiv m \\
\texttt{bind}(\texttt{bind}(m, f), g) &\equiv \texttt{bind}(m, x \rightarrow \texttt{bind}(f(x), g))
\end{aligned}
$$



Assoziativität!

Monaden-Interface?

```
interface MonadOf<M<_>> {
   ...
}
```

Naaa ....

Scala

```
trait MonadOf[M[_]] {
  def of[T](x: T): M[T]
  def flatMap[T, U](ma: M[T])
                   (f: T => M[U])
    : M[U]
}
```

Mehr Coolness

- Monoiden
- Funktoren
- applikative Funktoren
- Arrows
- …

Monaden

- Wunderwaffe für Effekte & DSLs
- macht mehr Spaß in FP-Sprachen
  Haskell, F#, Racket,
  Scala, Clojure
- mehr Infos:

funktionale-programmierung.de