

2.– 5. September 2013
in Nürnberg



Herbstcampus

Wissenstransfer
par excellence

Unter Beobachtung

Reactive Programming auf der JVM

Joachim Hofer

@johofer – imbus AG

„Callback Hell“

„Callbacks are the modern goto“ (Elm)

```
fs.readdir(source, function(err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function(filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function(err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function(width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(destination + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }).bind(this)
        }
      })
    })
  }
})
```

Fahrplan

- Was bedeutet Reactive?
- Was ist Rx?
- Was ist ein Observable?
- Komposition von Observables
- JVM-Frameworks
- Warum RxJava?
- Beispiel mit RxJava

Kurzvorstellung

- > 10 Jahre Teamleiter Softwareentwicklung
- > 10 Jahre *zu viel* Enterprise Java
- Praktizierender™ Scrum Master
- **Scala-Enthusiast**
- Committer bei **RxJava**
- Autor diverser **sbt-Plugins**
 jacoco4sbt, findbugs4sbt, cpd4sbt, ant4sbt etc
- sporadischer **Blogger** (und **Zwitscherer**)

Reactive Manifesto (1)

<http://www.reactivemanifesto.org/>

„**reactive**“ = „readily responsive to a stimulus“

Eigenschaften:

- **interactive**: „react to users“
- **event-driven**: „react to events“
- **scalable**: „react to load“
- **resilient**: „react to failure“

Reactive Manifesto (2)

<http://www.reactivemanifesto.org/>

Warum Reactive?

„Reactive applications represent a balanced approach to addressing a wide range of contemporary challenges in software development.“

Rx (Reactive Extensions)

*„a library to compose **asynchronous** and **event-based** programs using **observable** collections and LINQ-style query operators.“* (siehe **MSDN**)

- **.NET**-basiert
- auch **RxJS** für **JavaScript**
- seit November '12 **Open Source**

„Champion“: **Erik Meijer**

Für die JVM? – bisher Fehlanzeige...

Iterable

Iterable:

```
trait Iterable[+A] {  
  def iterator: Iterator[A]  
}
```

Iterator:

```
trait Iterator[+A] {  
  def next: A  
}
```


Observable

Observable:

```
trait Observable[+A] {  
  def subscribe(observer: Observer[A]): Subscription  
}
```

Observer:

```
trait Observer[-A] {  
  def onCompleted: Unit  
  def onError(error: Throwable): Unit  
  def onNext(value: A): Unit  
}
```

Iterable: Pull

```
val nums = Seq(1, 2, 3, 4)
```

Pull-Prinzip:

```
val iter = nums.iterator
```

```
iter.next // 1
```

```
iter.next // 2
```

Wir **holen** uns die Werte, wenn wir sie brauchen.

Observable: Push

```
val nums = Seq(1, 2, 3, 4)
```

Push-Prinzip:

```
val obsv = new Observer[Int] {  
    def onNext(value: Int) = println(value)  
    ...  
}
```

```
val sub = Observable from nums subscribe obsv
```

Wir **bekommen irgendwann** Werte, die wir behandeln müssen.

Push vs Pull

Pull: **Consumer** bestimmt

- meist synchron

Push: **Producer** bestimmt

- Event-Handling
- Asynchronizität

Dualität

Iterable

Observable

pull

push

synchron

asynchron

next: T

onNext(T)

wirft Exception

onError(Exception)

kehrt zurück

onCompleted

Dualität: Beispiel

Iterable Observable

```
// Iterable[String]
getDataFromLocalMemory
  skip 10
  take 5
  map (_ + "-trans")
  foreach println
```

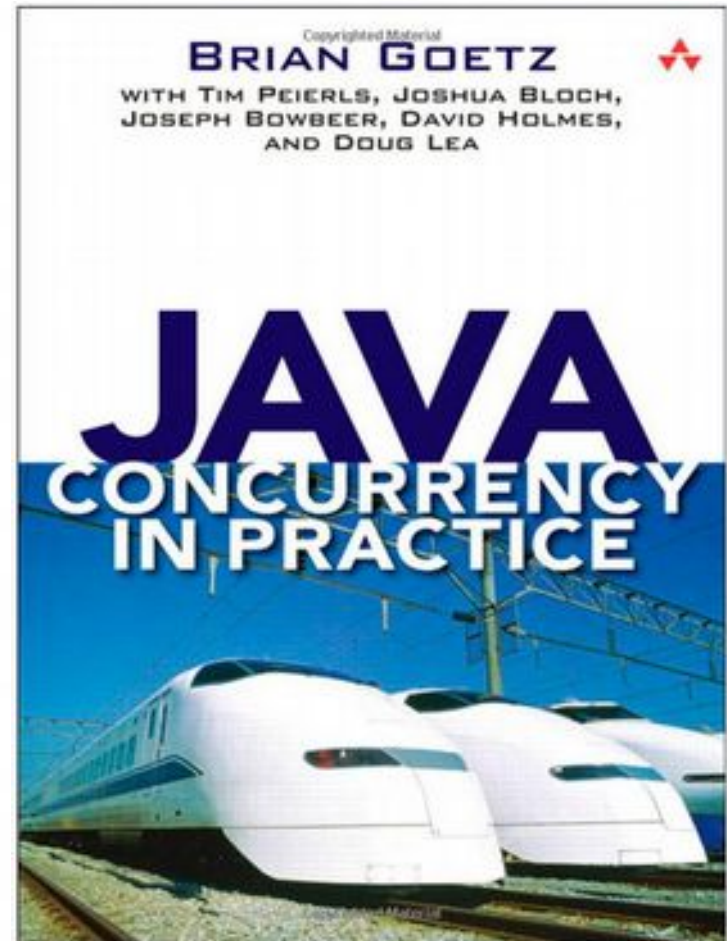
```
// Observable[String]
getDataFromNetwork
  skip 10
  take 5
  map (_ + "-trans")
  subscribe println
```

Asynchron: Concurrency

Wir brauchen
High-Level-Abstraktionen
für Concurrency!

Sonst muss **jeder** Entwickler
dieses Buch lesen!

(gutes Buch – unbedingt lesen!)



Einordnung

	einzelnen	mehrere
sync	T	Iterable[T]
async	Future[T]	Observable[T]

Synchron – Einzelwert

	einzelnen	mehrere
sync	T	Iterable[T]
async	Future[T]	Observable[T]

```
val v: T = getData(args)
```

```
if (v == x) // do something  
else       // do something else
```

Synchron – mehrere Werte

	einzelnen	mehrere
sync	T	Iterable[T]
async	Future[T]	Observable[T]

```
val vs: Iterable[T] = getData(args)
```

```
for (v ← vs) {  
  if (v == x) // do something  
  else      // do something else  
}
```

Asynchron – Einzelwert

	einzelnen	mehrere
sync	T	Iterable[T]
async	Future[T]	Observable[T]

```
val f: Future[T] = getData(args)
```

```
if (f.get == x) // do something  
else           // do something else
```

Asynchron – Einzelwert (Guava)

	einzeln	mehrere
sync	T	Iterable[T]
async	Future[T]	Observable[T]

```
val f: Future[T] = getData(args)
Futures.addCallback(f, new FutureCallback[String] {
  def onSuccess(v: T): Unit = {
    if (v == x) // do something
    else      // do something else
  }
  def onFailure(t: Throwable): Unit = { /*...*/ }
}, executor)
```

Asynchron – Einzelwert (akka)

	einzelnen	mehrere
sync	T	Iterable[T]
async	Future[T]	Observable[T]

```
val f: Future[T] = getData(args)
f map { v =>
  if (v == x) // do something
  else       // do something else
}
```

Asynchron – mehrere Werte

	einzeln	mehrere
sync	T	Iterable[T]
async	Future[T]	Observable[T]

```
val vs: Observable[T] = getData(args)
vs map { v =>
  if (v == x) // do something
  else       // do something else
}
```

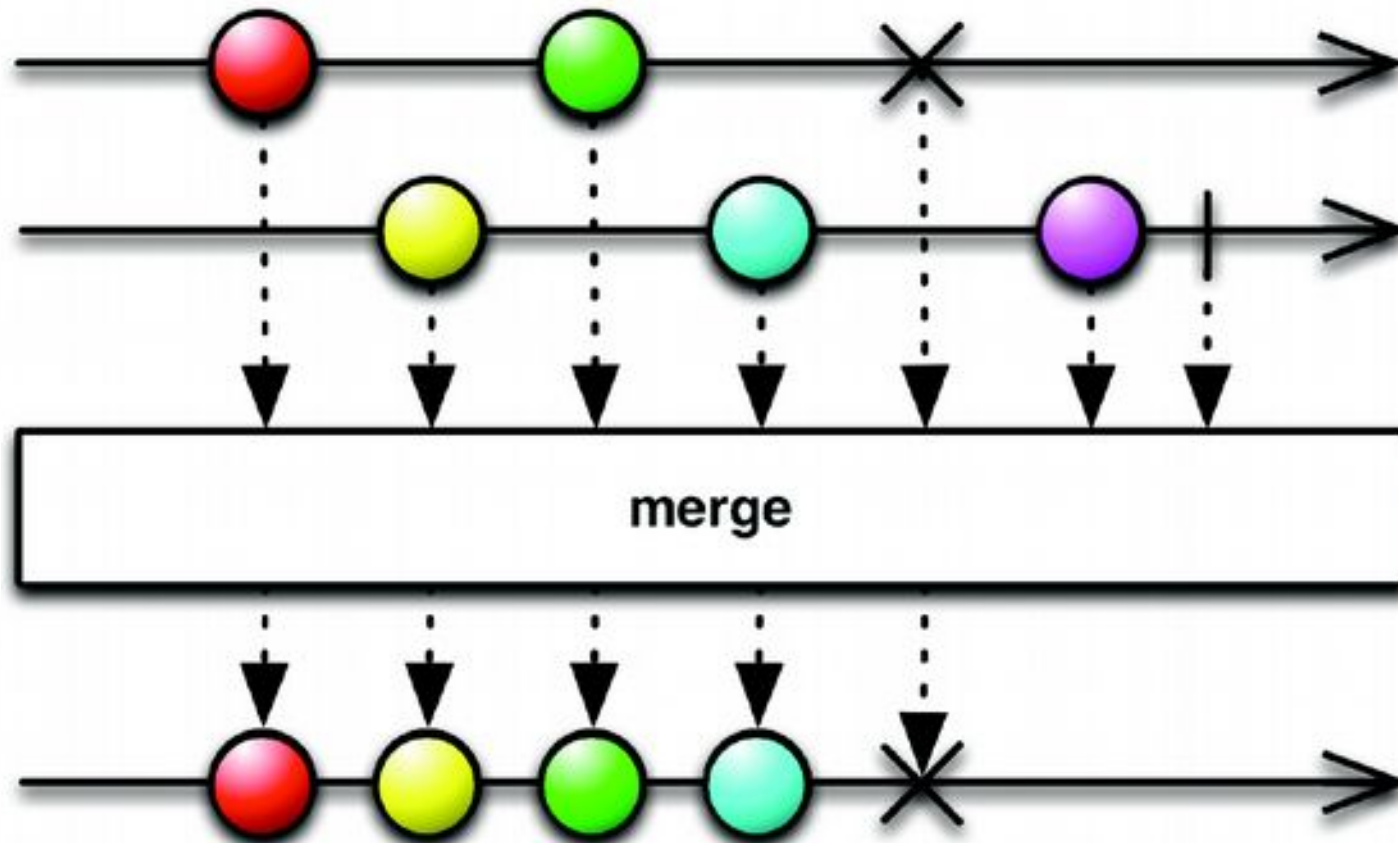
Vorteile ggü klassischem Event Handling

- Verknüpfen mehrerer Event-Quellen
- Filtern von Events
- Manipulieren von Events
- **Komposition** (Observable ist monadisch)
- stark reduzierte Thread-Blockade-Gefahr

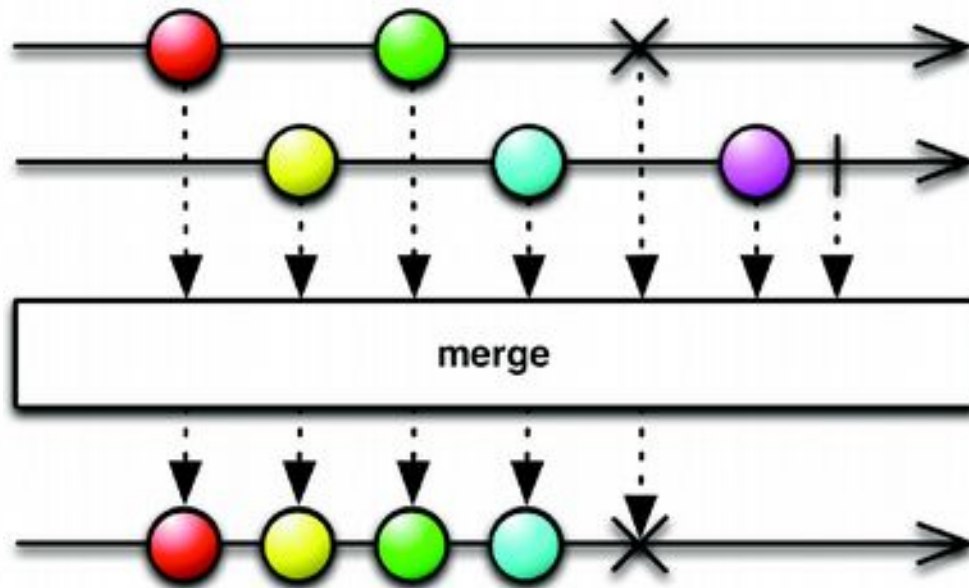
Kompositionsmöglichkeiten

- **Umwandeln:**
map, flatMap, reduce, scan...
- **Filtern:**
take, skip, sample, takeWhile, filter...
- **Kombinieren:**
concat, merge, zip, combineLatest,
multicast, publish, cache, refCount...
- auch so: **Concurrency & Fehlerbehandlung**

Beispiel: merge



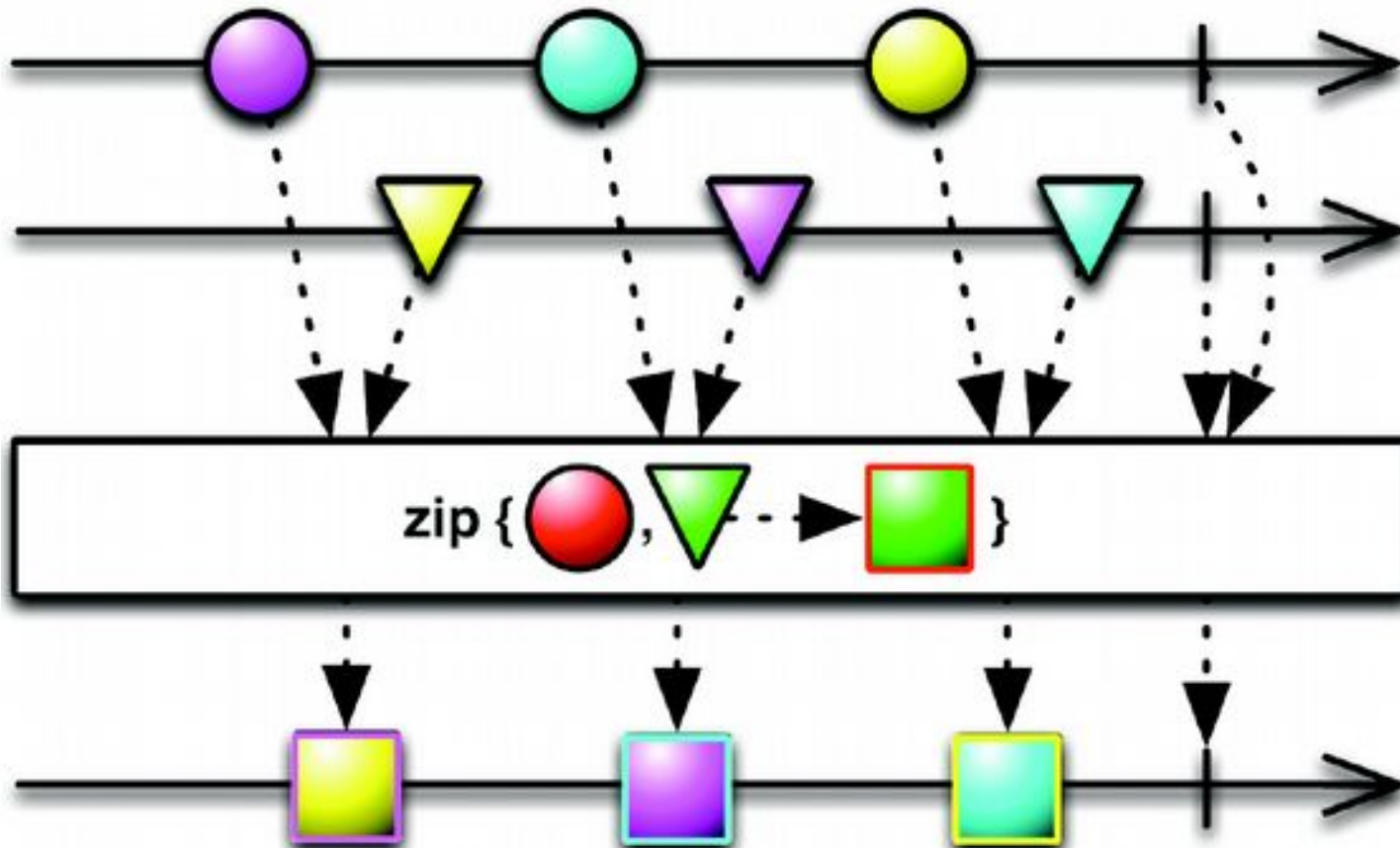
Beispiel: merge



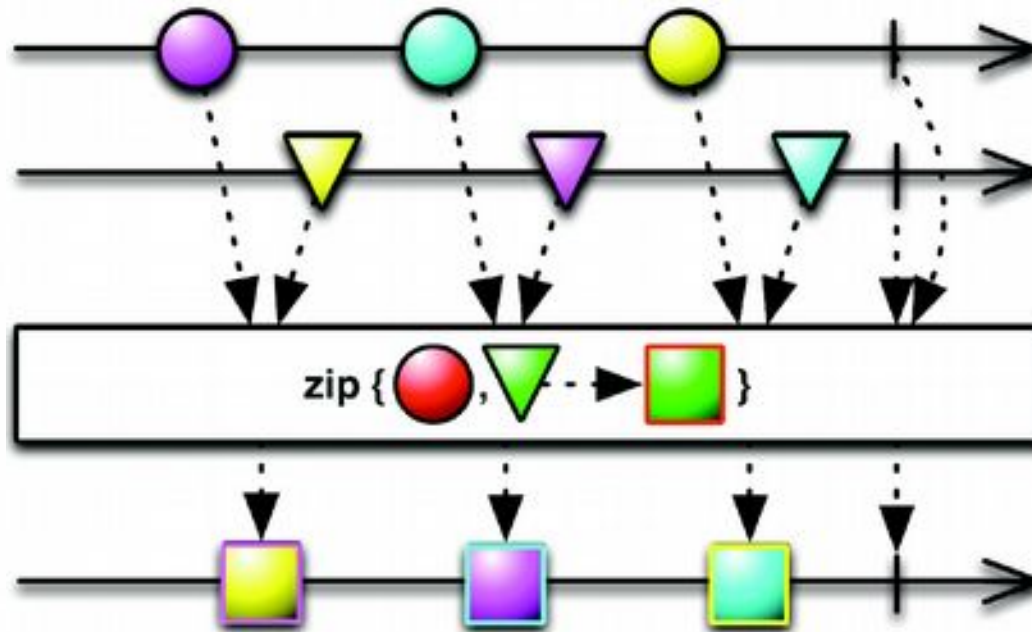
```
val as: Observable[T] = getDataA  
val bs: Observable[T] = getDataB
```

```
Observable merge (as, bs) subscribe println
```

Beispiel: zip



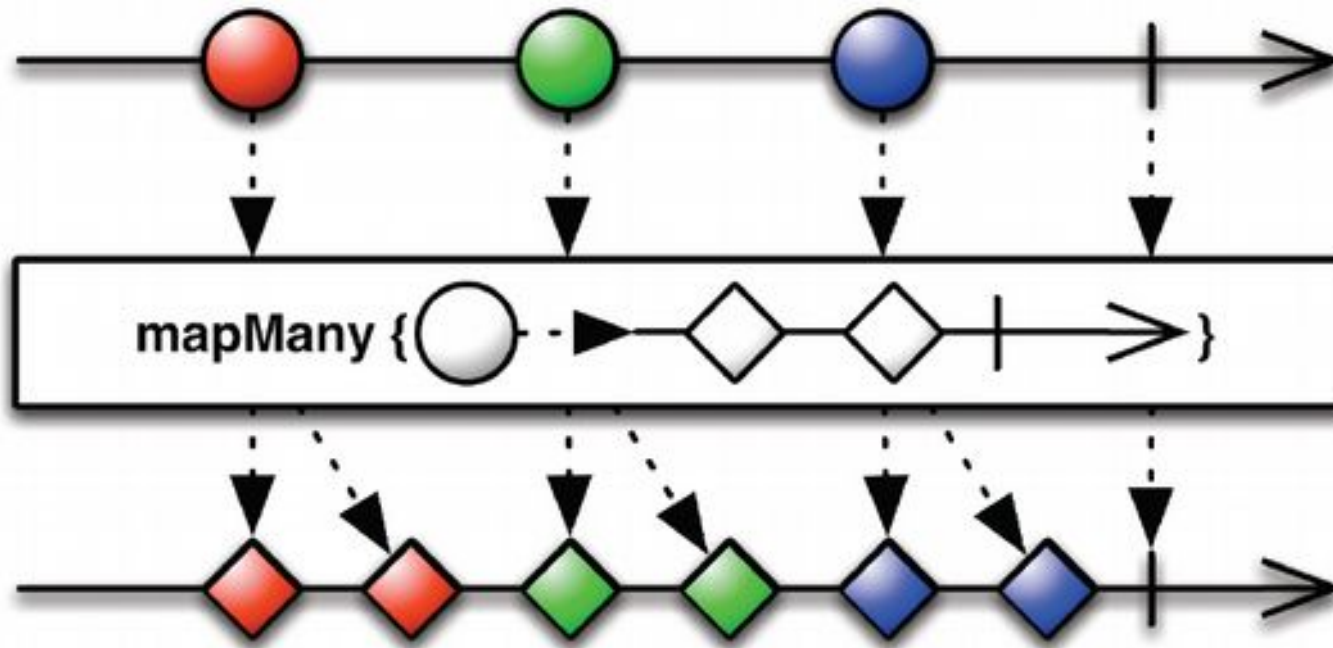
Beispiel: **zip**



```
val as: Observable[A] = getDataA
val bs: Observable[B] = getDataB
val f: (A, B) => String = (a, b) => s"$a, $b"
```

```
Observable zip (as, bs, f) subscribe println
```

Beispiel: `flatMap` / `flatMapMany`



```
val as: Observable[A] = getDataA
val bs: Observable[B] = as flatMap getDataB
// where getDataA: () => Observable[A]
// and    getDataB: A  => Observable[B]
```

Wozu **flatMap**?

- Synchroner API:

```
trait VideoService {  
  def getMovies(u: UserId): List[Video]  
  def getRating(v: Video): Rating  
  ...  
}
```

- Asynchroner API:

```
trait VideoService {  
  def getMovies(u: UserId): Observable[Video]  
  def getRating(v: Video): Observable[Rating]  
  ...  
}
```

Wozu **flatMap**?

- Synchroner API:

```
val ratings = getMovies(user) map getRating
```

- Asynchroner API:

```
val ratings = getMovies(user) flatMap getRating
```

```
trait VideoService {  
  def getMovies(u: UserId): Observable[Video]  
  def getRating(v: Video): Observable[Rating]  
}
```


Rx auf der JVM

- viele kleine (oft inaktive) Projekte (Auswahl)
 - **reactive4java** (Java),
 - **reactive-clojure** (Clojure)
 - **scala-reactive**, **nafg/reactive**,
dylemma/scala.frp, **dire** (Scala)
 - **Scala.React**
 - auf Basis von „**Deprecating the Observer Pattern**“
 - Forschungsprojekt, in Open Source inaktiv
- **RxJava**

RxJava

- von **Netflix** initiiert und nach Open Source gestellt
- bei Netflix für die interne API **im Einsatz**
 - mehr als 36 Mio „Subscriber“ in > 50 Ländern
 - > 2 Mrd API-Requests pro Tag
 - → **praxiserprobt**

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	36.8%	Netflix	33.0%	Netflix	28.8%
2	HTTP	9.83%	YouTube	14.8%	YouTube	13.1%
3	Skype	4.76%	HTTP	12.0%	HTTP	11.7%
4	Netflix	4.51%	BitTorrent	5.89%	BitTorrent	10.3%
5	SSL	3.73%	iTunes	3.92%	iTunes	3.43%
6	YouTube	2.70%	MPEG	2.22%	SSL	2.23%
7	PPStream	1.65%	Flash Video	2.21%	MPEG	2.05%
8	Facebook	1.62%	SSL	1.97%	Flash Video	2.01%
9	Apple PhotoStream	1.46%	Amazon Video	1.75%	Facebook	1.50%
10	Dropbox	1.17%	Facebook	1.48%	RTMP	1.41%
	Top 10	68.24%	Top 10	79.01%	Top 10	76.54%

 sandvine

RxJava

- aktives Open-Source-Projekt
 - 27 Committer
 - gut dokumentiert
 - Erik Meijer beobachtet / berät
- eng angelehnt an Rx.NET
- aber: noch nicht „fertig“

RxJava

- **multilingual**
 - derzeit: Java, Scala, Groovy, Clojure
 - in Arbeit/Planung: JRuby, Kotlin
- Unterstützung für Swing und **Android**
 - ansatzweise, in Arbeit
- austauschbare **Scheduler**
 - derzeit: synchron, ExecutorService, Swing, Android
 - genauso möglich: zB **Akka**

Fazit

asynchrone Entwicklung zunehmend wichtig

Reactive ist der aktuelle Hype

- **Observables** sind hierfür ein wichtiger Baustein
- **RxJava** ist dafür eine gute Umsetzung

Siehe auch:

Coursera: **Principles of Reactive Programming**
(Odersky, Meijer, Kuhn)

Live Coding: Pong

siehe **rxjava-samples** auf **Github**

2.– 5. September 2013
in Nürnberg



Herbstcampus

Wissenstransfer
par excellence

Vielen Dank!

Joachim Hofer

@johofer – imbus AG

imbus AG

Spezialisierte Lösungsanbieter für
Software-Qualitätssicherung und Software-Test
Innovativ seit 1992

Erfahrung und Know-how aus über 4.000
erfolgreichen Projekten

> 200 Mitarbeiter an fünf Standorten in Deutschland

Beratung, Test-Services, Training, Tools,
Datenqualität

Für den gesamten Software-Lebenszyklus

www.imbus.de

