

(Modern) Java for Clojure Programmers



James Henderson (@jarohen)

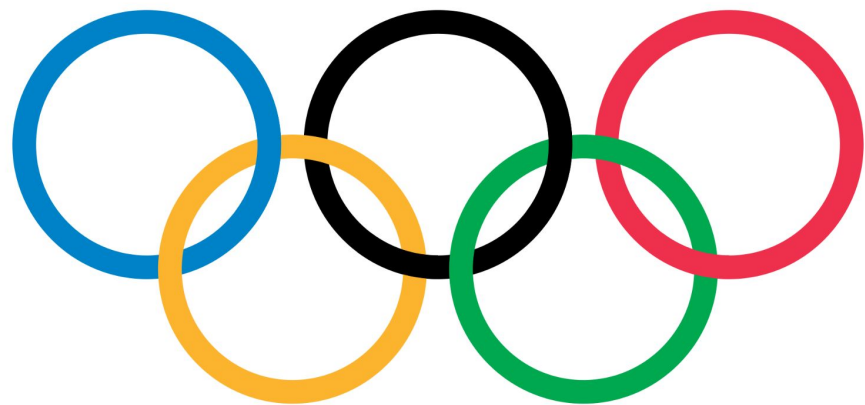
2023-07-12

Why?



2012:

2012:



2012:



2012:



2012:



2012:



YouTube ^{GB} Search

Clojure

A Dynamic Programming Language for the JVM

An Introduction for Java Programmers

Rich Hickey

0:42 / 1:48:12

Clojure for Java Programmers Part 1 - Rich Hickey

 ClojureTV
27.6K subscribers

Subscribe

Like | Comment | Share | Save | ...

122K views · 10 years ago


Part 1 of a presentation by Rich Hickey to the NYC Java Study Group. A gentle introduction to Clojure, part 1 focuses on reader syntax, core data structures, code-as-data, evaluation, special operators, functions, macros and sequences. No prior exposure to Lisp is presumed. [Show more](#)

Simple Made Easy

👍 LIKE

📄 25



View Presentation   

Speed: 1X 1.25X 1.5X 2X



Download MP3

01:01:26

Summary

Rich Hickey emphasizes simplicity's virtues over easiness', showing that while many choose easiness they may end up with complexity, and the better way is to choose easiness along the simplicity path.

Simple Made Easy

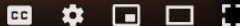
Rich Hickey



or, Hammock-driven Development



0:36 / 39:48 • Intro >



Hammock Driven Development - Rich Hickey



ClojureTV
27.6K subscribers

Subscribe

Like



Share

Save



269K views 10 years ago Clojure Conj 2010

Rich Hickey's second, "philosophical" talk at the first Clojure Conj, in Durham, North Carolina on October 23rd, 2010.

Show more



2012

The Language of the System

Rich Hickey



1:37 / 1:02:49 • Intro >



The Language of the System - Rich Hickey

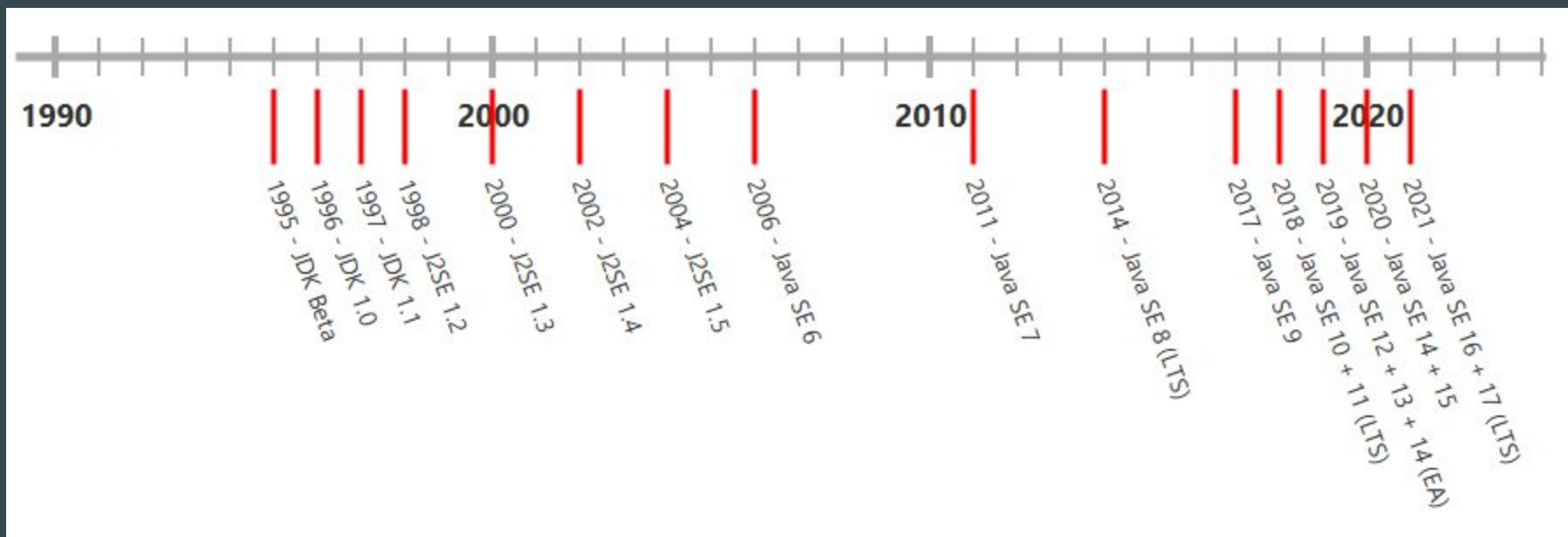


ClojureTV
27.6K subscribers

Subscribe



156K views 10 years ago Clojure/conj 2012



Java releases ([foojay.io](https://www.foojay.io))

Aside: Kotlin

`var` - local variable type inference (Java 10, JEP 286)

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
```

```
// =>
```

```
var baos = new ByteArrayOutputStream();
```

```
var list = new ArrayList<String>();
```

```
var stream = list.stream();
```

```
var path = Paths.get(fileName);
```

```
var bytes = Files.readAllBytes(path);
```

`switch` expressions (Java 12-14, JEP 361)

```
int numLetters;

switch (day) {
    case MONDAY:
    case FRIDAY:
    case SUNDAY:
        numLetters = 6; break;
    case TUESDAY:
        numLetters = 7; break;
    case THURSDAY:
    case SATURDAY:
        numLetters = 8; break;
    case WEDNESDAY:
        numLetters = 9; break;
    default:
        throw new ...
}
```

```
int numLetters = switch (day) {
    case MONDAY, FRIDAY, SUNDAY -> 6;
    case TUESDAY                 -> 7;
    case THURSDAY, SATURDAY      -> 8;
    case WEDNESDAY               -> 9;
};
```

Pattern matching for `instanceof` (Java 14-16, JEP 394)

```
if (obj instanceof String) {  
    // grr...  
    String s = (String) obj;  
    ...  
}
```

```
if (obj instanceof String s) {  
    // wahey!  
    ...  
}
```


Pattern matching in `switch` (Java 17-21, JEP 441)

```
static String formatter(Object obj) {
    String fmt = "unknown";
    if (obj instanceof Integer i) {
        fmt = String.format("int %d", i);
    } else if (obj instanceof Long l) {
        fmt = String.format("long %d", l);
    } else if (obj instanceof Double d) {
        fmt = String.format("double %f", d);
    } else if (obj instanceof String s) {
        fmt = String.format("String %s", s);
    }

    return fmt;
}
```

```
static String formatter(Object obj) {
    return switch (obj) {
        case Integer i
            -> String.format("int %d", i);
        case Long l
            -> String.format("long %d", l);
        case Double d
            -> String.format("double %f", d);
        case String s
            -> String.format("String %s", s);
        default
            -> obj.toString();
    };
}
```

`record`s (Java 14-16, JEP 395)

```
class Point {
    private final int x;
    private final int y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    int x() { return x; }
    int y() { return y; }

    public boolean equals(Object o) {
        if (!(o instanceof Point)) return false;
        Point other = (Point) o;
        return other.x == x && other.y == y;
    }

    public int hashCode() {
        return Objects.hash(x, y);
    }

    public String toString() {
        return String.format("Point[x=%d, y=%d]", x, y);
    }
}
```

```
record Point(int x, int y) {
}
```

`sealed` classes and interfaces (Java 15-17, JEP 409)

```
sealed interface Expr {...}
```

```
record ConstantExpr(int i) implements Expr {...}
```

```
record AddExpr(Expr a, Expr b) implements Expr {...}
```

```
record MulExpr(Expr a, Expr b) implements Expr {...}
```

```
record NegExpr(Expr e) implements Expr {...}
```

Record patterns (Java 19-21, JEP 440)

```
public int eval(Expr expr) {
    return switch (expr) {
        case ConstantExpr(var i) -> i;
        case AddExpr(var a, var b) -> eval(a) + eval(b);
        case MulExpr(var a, var b) -> eval(a) * eval(b);
        case NegExpr(var e) -> - eval(e);

        // `Expr` is sealed, so no `default` required!
    }
}
```

Virtual threads (aka 'Project Loom' - Java 19-21, JEP 444)

- Problems:
 - 'platform' (OS) threads are heavyweight and expensive
 - asynchronous style introduces complexity and observability tradeoffs
- Loom:
 - Targets applications written in 'thread-per-request' style
 - Increases **throughput** in a concurrent application, not speed of a single request
 - Decouples 'virtual' threads from 'platform' threads
- Quite different to Kotlin's approach [1]

[1] "Coroutines and Loom behind the scenes", Roman Elizarov (Kotlin Project Lead at JetBrains). <https://youtu.be/zluKcazqkV4>

Virtual threads (aka 'Project Loom' - Java 19-21, JEP 444)

- Blocking (often I/O) requests transparently 'park' virtual threads to allow platform threads to work on other tasks
- Virtual threads scheduled with a JVM `ForkJoinPool` - no time-sharing.
- (Deliberately) re-uses existing `java.lang.Thread` API: start with `Thread.ofVirtual().start(Runnable)`
- Can adapt existing thread pools with `Executors.newVirtualThreadPerTaskExecutor()`

Virtual threads: Don'ts

- Don't pool virtual threads - no need
- Don't limit virtual thread numbers to constrain concurrency - consider other concurrency primitives, like Semaphore
- Don't use ThreadLocals to cache expensive resources any more - you'll get one per virtual thread.

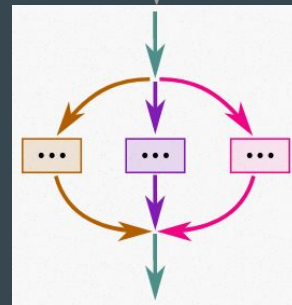
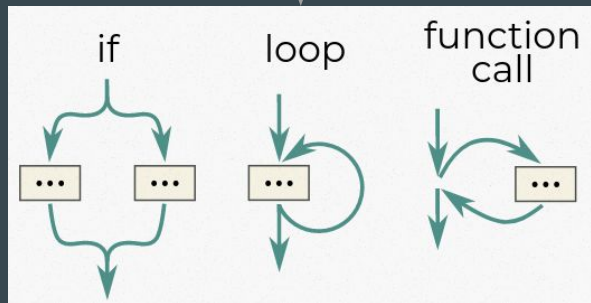
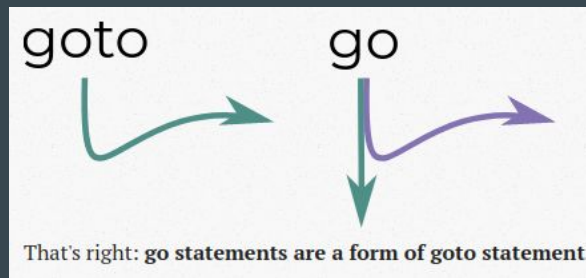
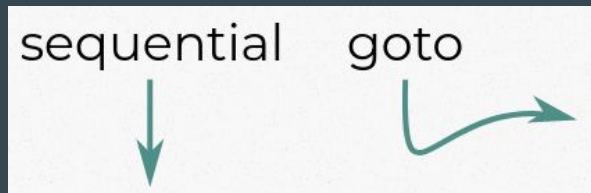
Structured Concurrency

- First coined by Martin Sústrik [1]
- Popularised by Nathaniel J. Smith in ‘*Notes on structured concurrency, or: Go statement considered harmful*’ [2]
 - Paraphrased: “go is to concurrent programs what goto was to serial programs”
 - Cancellation (incl. timeouts) and error-handling in concurrent code is complex and error-prone.
 - Introduced ‘nurseries’ - blocks which waited for all of the threads that spawned within it to finish.
 - Code that is less powerful can be better analysed/optimised. (cf. `while`, `if` vs `goto`)

[1] <https://250bpm.com/blog/71/>

[2] <https://vorpus.org/blog/notes-on-structured-concurrency-or-go-statement-considered-harmful/>

Structured Concurrency



Structured Concurrency (Java 19-21+, JEP 453)

```
import java.util.concurrent.StructuredTaskScope;

Response handle() throws ExecutionException, InterruptedException {
    try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
        Supplier<String> user = scope.fork(() -> findUser());
        Supplier<Integer> order = scope.fork(() -> fetchOrder());

        scope.join()           // Join both subtasks
            .throwIfFailed(); // ... and propagate errors

        // Here, both subtasks have succeeded, so compose their results
        return new Response(user.get(), order.get());
    }
}
```

questions (n., pl.)

from Latin: 'quaestio', derived from 'quaerere': 'to ask', 'to seek'.