author: Neil Ernst title: Lecture 16 - Functional Approaches to Design date:

Overview/Learning Objectives

- the nature of functional programming
- several key functional concepts
- leveraging functional idioms for design

Why Functional Approach

Much of FP has focused on programming language idioms e.g. lambdas, purity, typing

But FP is good for parallelism and concurrency, which are increasingly necessary at the design level.

FP is good at reductive reasoning: keeping things simple and side-effect free makes it easier to reason.

Cube Composer

Find someone with a laptop and work through at least levels 0 and 1

Review: Functional Paradigm

(note, a lot is derived from the Fairbanks talk)

Function composition

- Build software by composing small functions (cf 'pipe and filter style')
- functions as arguments `f(g(x))`

Pure Functions

- no side effects -> same arguments yield same results
- Reasoning about the outcome is easier

> curl http://localhost/numberAfter/5 → [always 6]
>
> curl http://localhost/customer/5/v1 → [always v1 of customer]

vs

> curl http://localhost/customer/5 → [may be different]

Statelessness & Immutability

If there's no state

- Easy to reason about

- All instances are equivalent

If things are immutable

- Easy to reason about
- Concurrent access is safe

Idempotence

Idempotence = get the same answer regardless of how many times you do it

> resizeTo100px(image) vs shrinkByHalf(image)

- Often hard to guarantee something is done exactly once
- Eg: Is the task stalled or failed?
- Go ahead and schedule it again without fear that you'll get a duplicate result

---

Declarative (vs imperative)

- Functional programming defines what something *is*
  - ... or how it relates to other things
  - Versus a series of operations that yield desired state
- Definition, not procedure
- Example: how much paint do I need?
  - `while(!done) { fence.paint(); }`
- Vs function parameterized by length and width
  - `paint(l=20,w=3)`

FP and the GoF

- Many of the patterns in the Gang of Four Design Pattern book are idiomatic in FP. That means there is no need for a design pattern, as the language contains built-in features to handle the same design challenge.
- Let's look at Observer from the Mario Fusco presentation.
- That example shows that using a `ConcurrentHashMap` and Java8's `Consumer` interface can radically reduce the amount of code required (check out the GoF example in the same repository).
- This makes the code easier to read (assuming our coder has a basic knowledge of Java8 and functional programming), and easier to maintain.
- Template example. The `Resource` class conducts possibly risky operations (e.g. resulting in resource leaks since they are not disposed of). However, we don't want the client to have to worry about this - the implementer should be responsible for protecting from resource leaks. Therefore the `Template` pattern forces all access to go via a common method, so the client is portected when risky code is run.

FP and Design

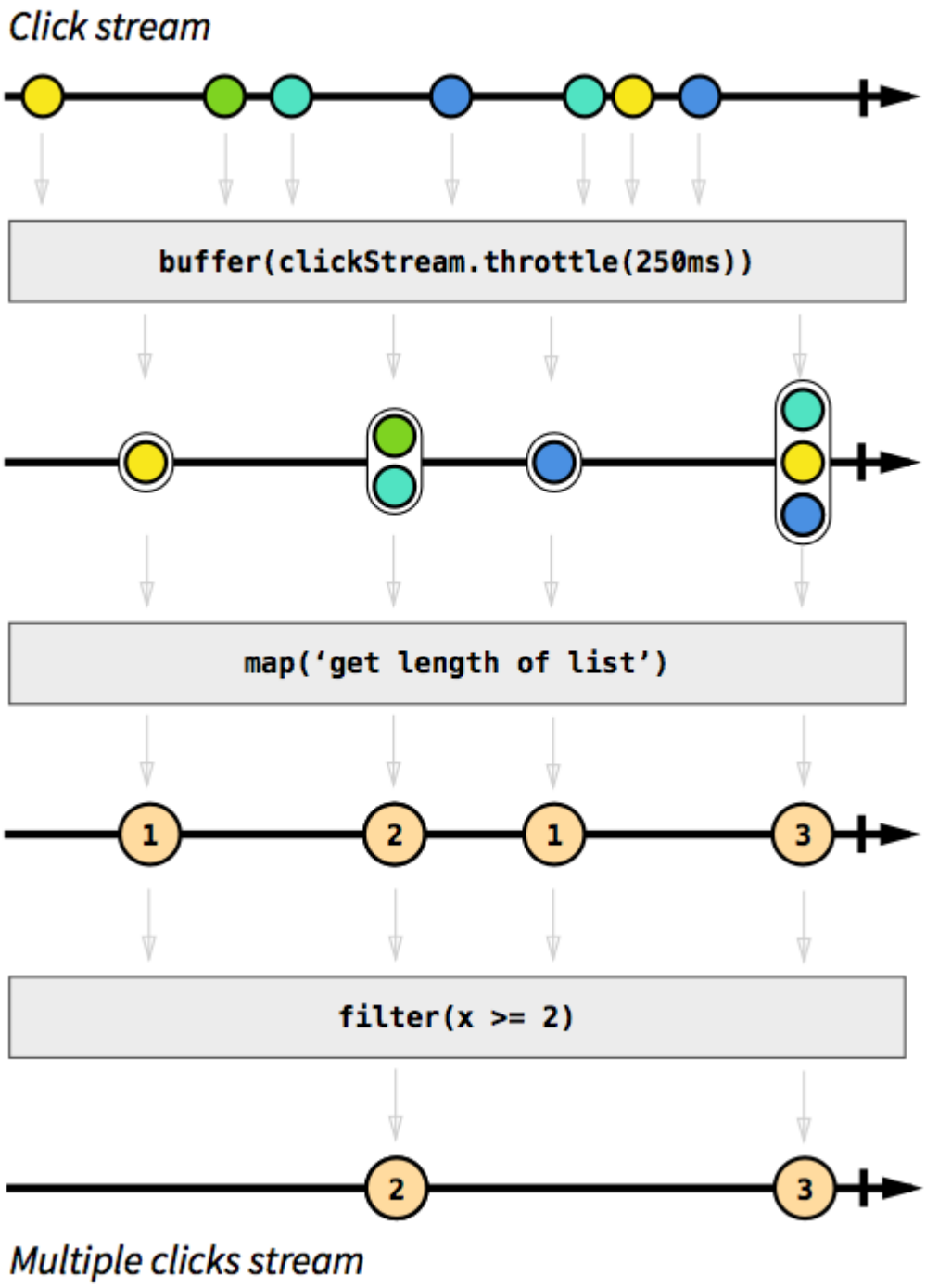
- Streams of events can be processed using functional operators (e.g., `filter`)
- Example: Reactive Programming (not `React` the framework)

(see Andre Staltz intro)

Reactive Programming

It's all streams. We recast our web programming as a series of operations on streams of events and data. For example, to detect double-clicks:

- Rich set of stream operators
  - Transform streams into streams
- Example: Detect double clicks
  - Input: user click stream
  - Transform: group nearby clicks
  - Transform: count group size
  - Transform: drop size-1 groups
  - Output: double click stream



Serverless architectures

- Rather than paying for a machine (IAAS), machine+libraries (PAAS), pay only for resources a single function consumes (FAAS).
    - E.g. Amazon Lambda
- No state, just transformations
- Short-lived functions that start on-demand.
- Think about startup/warmup latency

(more at https://martinfowler.com/articles/serverless.html)

Other examples of FP and design

- Append-Only Data stores
    - Instead of traditional SQL approach, where the DB can be altered, we store every event (stream) into a persistent, immutable storage
    - Reconstruct important events using functional operators (e.g. `select customers = 'jones'`)
- Map-reduce data analysis using Hadoop and Spark
- Machine-learning frameworks (data in, labels out)

Resources

- Mario Fusco talk
-
    - Template Method in Scala/Kotlin https://www.anthony-galea.com/blog/post/the-template-method-pattern/
- https://stackoverflow.com/questions/28417262/java-8-supplier-consumer-explanation-for-the-layperson