How Java9+ helps you REACT- Reactive Programming?

Jayashree S Kumar, IBM
Agenda

• Why? What? and When?
• Reactive - System, Principles, Programming, Streams
• Go reactive in Java: Popular Tools/Libs
• Java9’s Flow API and the differentiation.
• TakeAways
About Me

• IBM Java- Classes Library developer

• Worked Extensively on JDK’s Testing

• IDT Evaluator: IBM Patents Portfolio

• Runtimes team @ IBM Software Labs
PARADIGMS?
Programming Paradigms

Procedural

Imperative

Object-Oriented

Declarative

Functional Programming

Logic

Prolog

Data-Flow

Reactive Programming

C

Fortran

C++

Java

Scala

Erlang

Haskell

Ocaml

Elm

Clojure

What?

How?
\begin{align*}
a & = 10; \\
b & = a + 1; \\
a & = 11; \\
(\text{Imperative } b & = 11) \quad \ldots \quad (\text{Reactive } b & = 12) \\
b & = a + 1; \\
\end{align*}
Streams

Synchronous

Asynchronous
WWW
REACTIVE
Why Reactive?

Evolving changes/demands in the Computing Ecosystem

- Improved H/w level
- Virtualization & Cloud based Infrastructure
- Applications S/w Requirement
- Huge Volumes of Data
- Multi-userness
- Impatient Us!!
What is Reactive?

Reactive Manifesto
https://www.reactivemanifesto.org

Declaration of ideals for Modern software design

* More flexible systems
  * Highly responsive
* More tolerant of failures
  * Handling of failures
Reactive Principles
Reactive Streams

Blueprint for how to implement reactive programming.

- JVM Standard for asynchronous stream processing with non-blocking back pressure

Working group: Netflix, Pivotal, LightBend – later joined by Oracle, Twitter, Red Hat, spray.io
Generalized Stream processing architecture

Streaming pipeline with backpressure

Borrowed from: https://blog.softwaremill.com/how-not-to-use-reactive-streams-in-java-9-7a39ea9c2cb3
Reactive Streams

—> 3 Key Factors:

- Data processed asynchronously,
- Backpressure is non-blocking,
- Downstream’s slowness represented

—> Specification consists of:

- * API specifics [Java API & Textual]
- * Technology Compatibility Kit (TCK)
- Implementation Example
Reactive Programming
a, b, c, d are emitted values
X is an error
| is the 'completed' signal
→> is the timeline

Subscribing Observers Observable
Benefits of RP to Users?

- Increased performance (not speed)
- Increased Resource Utilization
- Improved UX
- Simplified modification & Updates
When/Where RP?

High-load or multi-user applications:

• Social networks, chats
• Games
• Audio and video apps
• Server code that serves highly interactive UI elements
• Proxy servers, load balancers
• Artificial intelligence, machine learning
• Real-time data streaming
Reactive Manifesto

Reactive Systems

Reactive Principles

Message-Driven

For

Reactive Programming

Reactive Streams

Reactive Streams Specification

Event-Driven

(Reactive Extension or ReactiveX)

Must Adhere

Follows
Reactive in Java
Java’s Impact on Modern World

MOBILE TECHNOLOGIES
Android devices are all driven by Java Libraries. Java is the official language for developing Android apps.

HOSPITALS
All management of hospital functions and operating on Java.

MILITARY
Java is used in military mobile apps transforming away from Ada for more secure applications.

CITY INFRASTRUCTURE
Java is used by cities to monitor and control water and traffic systems across desktops and mobile devices.

SPACE
NASA uses Java to simulate physics in space.

BANKS
Java is a good alternative to COBOL as it is a modern language and can be documented more easily.

GAMING
Java has moved beyond the Swing interface in creating virtual and augmented reality games.

CONSUMER electronics & IoT
100% of Blu-Ray players ship with Java.

Automotive
Java is used in automotive electronics.

AUTOMOTIVE
Responsible for operating the accelerometer, thermal sensors for tires and brakes, in-car heart rate monitors, and touchscreen controls.

Borrowed from : Guide to Modern Java by DZone
Java 8

- Most feature Packed Release
- Introduced Functional programming

Java 9..11 n onwards

- New release train model
- The module system - Project Jigsaw
- Introduced Reactive Streams

Java EE 8 - “Modern Web" & “Ease of” Development
Reactive Stream Interfaces (earlier separate libs) became part of JDK in `java.util.concurrent.Flow` class.

4 basic interfaces:

- Flow.Publisher
- Flow.Subscriber
- Flow.Subscription
- Processor

*SubmissionPublisher
Goals of Reactive Stream in JDK

- Provide SPI layer
- Future Java Development
- Improve Interoperability
  Among existing stream abstractions:
  - `java.io.InputStream/OutputStream`
  - `java.util.Iterator`
  - `java.nio.channels.*`
  - `javax.servlet.ReadListener/WriteListener`
  - `java.sql.ResultSet`
  - `java.util.Stream`
  - `java.util.concurrent.Flow.*`
import java.util.List;
import java.util.concurrent.SubmissionPublisher;
import java.util.concurrent.Flow;

public static void main(String[] args) {
    List<String> items = List.of("1", "2", "3", "4", "5", "6", "7", "8", "9");
    SubmissionPublisher<String> publisher = new SubmissionPublisher<>();
    publisher.subscribe(new MySubscriber<>());

    items.forEach(s -> {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        publisher.submit(s);
    });

    publisher.close();
}
public static class MySubscriber<T> implements Flow.Subscriber<T> {

    private Flow.Subscription subscription;

    public void onSubscribe(Flow.Subscription subscription) {
        this.subscription = subscription;
        this.subscription.request(1);
    }

    public void onNext(T item) {
        System.out.println("item = [" + item + "]");
        subscription.request(1);
    }

    public void onError(Throwable throwable) {
        throwable.printStackTrace();
    }

    public void onComplete() {
        System.out.println("DONE");
    }
}
OTHERs
Java Reactive Programming

• RxJava

• Spring Reactor

Reactive Systems in Java

• Akka

• Eclipse Vert.x
Last Example

Mary finishes work -> Orders the pizza -> waits till done -> Picks up her friend. Finally (Friend and pizza) makes it home -> gets down to the movie : **Sync approach**

Mary orders pizza  
-> Calls Friend asks to come -> Head home  
-> Has Pizza Delivered  
-> Starts watching movie (eating Pizza) -- Without Friend : **Async Approach**

Mary orders pizza  
-> Calls Friend asks to come -> Head home  
-> Has Pizza Delivered  
-> Waits for Friend to come -- Only then watches movie or eats Pizza : **Reactive Approach**.

*Wait till all async actions (changes) are completed and then proceed with further actions.*
to each his own
Recap + Takeaways

• Not for the "faint of heart" but for the determined

• **Reactive programming != Reactive systems** (or FRP)

• Reactive is "way of thinking".

• Flow SPI is inter operation specification; not an user API

• Reactive Programming $\uparrow$ Scalability & Stability not Speed.

• Introduce Reactive-components in Bite-size.
EVALUATE THE SESSIONS

Sign in and vote using the conference app or eclipsecon.org

-1 0 +1
THANK U!
Benefits of RP to Programmers?

• avoid “callback hell”
• a lot simpler to do async work
• very simple to compose streams of data
• complex threading becomes very easy
• you end up with a more cleaner, readable code base
• easy to implement back-pressure
import io.reactivex.Observable;
import io.reactivex.functions.Consumer;

// defining the source
Observable<Integer> source = Observable.range(1, 5);

// defining the consumer
Subscriber<Integer> consumer = new Subscriber<Integer>() {
    public void onNext(Integer number) { System.out.println(number); }
    public void onError(Throwable e) { System.out.println("error"); }
    public void onCompleted() { System.out.println("completed"); }
};

// connecting the consumer to source
source.subscribe(consumer);
 Observable.range(1, 5).subscribe(
    number -> System.out.println(number),
    error -> System.out.println("error"),
    () -> System.out.println("completed")
);
At a glance...

public interface Publisher<T> {
    public void subscribe(Subscriber<? super T> s);
}

public interface Subscriber<T> {
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}

public interface Subscription {
    public void request(long n);
    public void cancel();
}

public interface Processor<T, R> extends Subscriber<T>, Publisher<R> {