**Reactive programming**

October, Sergey Morenets, 21st 2018

2018

Sergey Morenets, 2018

DEVELOPER

TRAINER

WRITER

14 YEARS

6 YEARS

4 BOOKS

Sergey Morenets, 2018

FOUNDER

SPEAKER

Sergey Morenets, 2018

Sergey Morenets, 2018

Sergey Morenets, 2018

**Software communication**

Sergey Morenets, 2018

Component 1 Component 2

**Software communication rules**

Sergey Morenets, 2018

**Task #1. Installation & configuration**

1. Install and configure your IDE 2. Import training project as Maven/Gradle projects. 3. Review project and its structure/content. 4. What are project components? How do they interact with

each other? 5. What are advantages/disadvantages of the current

component communication?

Sergey Morenets, 2018

**Java. Asynchronous programming**

Sergey Morenets, 2018

Thread, Introduced in 1.0 with

Runnable and Callable types Was refined in Java 5 using Executors library and additional

types like Queue, Semaphore, CountDownLatch and CyclicBarrier Thread

-safe collections like CopyOnWriteArrayList/ ConcurrentHashMap Uses

synchronized/volatile keywords or Lock-based types to provide synchronization/locking Added

ForkJoinPool in Java 7 Introduced

CompletableFuture in Java 8

Future is read-only reference to the expression that may be calculated or failed to calculate

**Promise**

Promise is write-once reference to the expression that will be provided

**Asynchronous helpers**

Sergey Morenets, 2018

**Future**

**Java 5. Futures**

Sergey Morenets, 2018

**Java 5. Futures workflow**

Cancels current task with interrupting if needed

Sergey Morenets, 2018

**Java 8. Parallel streams**

Sergey Morenets, 2018

**Task #2. Asynchronous programming**

1. Review training project. How would you rewrite it using

asynchronous (blocking) programming paradigm? 2. Change WaiterService so that it operates asynchronously. Try

to run Starter class and confirm that application works properly. 3. How would you change Starter/WaiterService to cancel

current order on timeout?

Sergey Morenets, 2018

**CompletableFuture**

Sergey Morenets, 2018

Introduced in Java 8

CompletionStage Implements Future and

interfaces Allows to combine the results of different asynchronous

executions or computations in the chain of steps Error handling

**CompletableFuture**

Name Description runAsync Asynchronously complete this Runnable supplyAsync Asynchronously complete this task anyOf/allOf Executes several tasks acceptEither Executes current or given task thenAccept Allows to execute action using completion result thenApply Allows to execute function using completion

result cancel Cancel current task runAfterEither Allows to execute action after current or given

task completes

Sergey Morenets, 2018

**CompletableFuture**

Name Description thenCombine Allows to execute in parallel two independent

CompletableFuture tasks and combine result thenCompose Allows to combine execution of two

CompletableFuture tasks thenRun Run the action when current task completes handle Allows to handle exception/task result in one

method orTimeout Allows specify timeout for the current task join Returns execution result of throws an exception completeExce ptionally

Throws given exception if not completed

Sergey Morenets, 2018

**Java 8.**

Sergey Morenets, 2018

**CompletableFuture**

**Java 8.**

Sergey Morenets, 2018

**CompletableFuture**

**Java 8.**

Sergey Morenets, 2018

**CompletableFuture**

**Java 8.**

Sergey Morenets, 2018

**CompletableFuture**

**CompletableFuture Task #3.**

1. Review methods from CompletableFuture class. 2. Change WaiterService so that it returns CompletableFuture

now for long-running operations. How would you change Starter class? 3. Try to run Starter class and confirm that application works

properly. 4. What is disadvantages of the current solution?

Sergey Morenets, 2018

**Reactive manifesto**

Sergey Morenets, 2018

*The system is resilient in case of failures(software, hardware, timeout, human errors)*

**Elastic**

*-driven The system rely on asynchronous messaging to enable communication between components*

**Reactive manifesto**

Sergey Morenets, 2018

**Responsive**

*The system responses in timely manner Resilient*

*The system stays responsive under varying workload Message*

**Reactive manifesto. Patterns**

Sergey Morenets, 2018

**Responsive**

*Circuit breaker, offline mode Resilient*

*Fault tolerance, sharding, replication Elastic*

*Vertical and horizontal scalability Message*

*-driven Message queues, location transparency*

**Reactive programming history**

Sergey Morenets, 2018

created Erlang

in 1987 by Joe Armstrong and brought Actors for distributed calculations 3 seconds downtime per 100 years

-involved Re

immutability and functional programming Reactive programming is software paradigm about data flows

and propagation of changes Data stream is sequence of ordered events

**Reactive programming**

Sergey Morenets, 2018

**Leads to asynchronous programming**

Flow of the application is easier to understand

Provides universal error handling

Source of information is entity that emits data

Stream data types are value type, signal of error and signal of

completion Cold source of information publishes data even if there are no

subscribers Hot source of information publishes data after first

subscription

**Developer kit**

Sergey Morenets, 2018

Akka

Disruptor

Apache Storm

ReactiveX

**Reactive streams**

Sergey Morenets, 2018

**Reactive streams**

Sergey Morenets, 2018

Sergey Morenets, 2018

**Reactive streams**

Sergey Morenets, 2018

Version 1.0 released

Core types are Subscriber/Publisher/Subscription/Processor

Akka, Implementations include

MongoDB, Reactor, RxJava and Slick Incorporated in

Java 9 as Flow API

**ReactiveX**

Sergey Morenets, 2018

**Observer/Iterator Best ideas from**

pattern and functional programming Developed by

Netflix Implemented in Java as

RxJava Supports JavaScript, Scala,

Clojure, Swift and .NET

Sergey Morenets, 2018

**RxJava**

Sergey Morenets, 2018

JDK6+ Designed for

and Andriod 2.3+ Support for Java 8

lambdas Developed since 2014

18 Stable 1.x branch and brand new 2.x branch(

dev months!) Scala/

Kotlin/Clojure support Synchronous and asynchronous execution

Reactive and functional programming

rd-party No 3

libraries Supports back

-pressure in 2.x By default single

-threaded

**RxJava**

Sergey Morenets, 2018

**. Maven dependencies**

**Java SE. Iterator & Observer**

Sergey Morenets, 2018

**Observable. Creation methods**

Method Description empty() Creates Observable with no items but completion event error() Creates Observable that invokes onError callback

fromArray() Converts array into Observable fromCallable() Creates Observable based on Callable instance fromFuture() Converts Future into Observable generate() Creates synchronous stateless generator of values just() Creates Obserable that generates given events never() Creates Observable with no items range() Creates Observable with given range of integers create() Creates Observable that wraps non-reactive behavior

Sergey Morenets, 2018

**RxJava**

Sergey Morenets, 2018

**API**

Single

– publisher that emits either single event or error with no completion event Maybe

– publisher that emits single event, no event or error with completion event support Completable

– computation that can generate only error or completion event

**RxJava**

Sergey Morenets, 2018

**. Marbles**

**RxJava**

Intermediate operators

Sergey Morenets, 2018

**. Terms**

Downstream operators

Emission Item Signal Event Message Data

Upstream operators

**Observable generation**

Sergey Morenets, 2018

**Observable generation**

Sergey Morenets, 2018

**RxJava**

Sergey Morenets, 2018

**2. Callbacks**

Calls when onComplete/onError signal received

**RxJava**

Sergey Morenets, 2018

**2. Emitter**

**Observable transformation**

Method Description interval() Returns Observable that pushes items with specified

interval delay() Adds delay before items generation

count() Calculates total number of generated items distinct() Returns Observable with distinct values doOnEach() Executes specified consumer for each generated item elementAt() Returns item at the specified index or completes if

there are less items than specified index retry() Retries items generation in case of error replay() Replays the generated items for new subscribers

Sergey Morenets, 2018

**Observable transformation**

Sergey Morenets, 2018

**Observable flow control**

Method Description first() Returns only the first item firstOrError() Returns the first item of signals error if Observable is

empty last() Returns the last item or the default value take(N) Returns only first N items takeLast(N) Returns only last N items takeWhile() Returns items while condition matches and then

completes takeUntil() Returns items while condition doesn’t match and then

completes skipWhile() Skip items while certain condition matches defaultIfEmpty() Produces item with default value if stream is empty

Sergey Morenets, 2018

**Observable transformation**

Sergey Morenets, 2018

**Observable transformation**

Method Description map() Performs one-to-one mapping flatMap() Performs one-to-many mapping

scan() Performs scanning of all the items with possible result

accumulation groupBy() Allows to group by items into groups by some

condition

Sergey Morenets, 2018

**Observable transformation**

Sergey Morenets, 2018

**Observable transformation**

Sergey Morenets, 2018

**Observable joining**

Method Description join() Combine the overlapping items of two Observables combineLatest() Combine the latest items of the multiple Observables

using given function merge() Combine multiple Observables into one stream mergeWith() Combine second Observable with first Observable

only when the latter complete zip() Combines two or more Observables together using

given function

Sergey Morenets, 2018

**Observable joining**

Prints 1,2,3,4

Sergey Morenets, 2018

Prints 1\_a,2\_b

**RxJava**

Sergey Morenets, 2018

**types.**

**Conversion**

**Disposable**

Method Description dispose() Cancels current task isDisposed() Returns true if current task/resource is disposed

Cancel subscription

Sergey Morenets, 2018

Subscription result

Enables to cancel

subsription

**2 Task #4. Introduction into**

1. Add RxJava 2 dependency to your project: 2. Create new class RxJavaStarter with main() method and try

to create Observable object in different ways and provide various intermediate operators. 3. Try to cancel subscription 4. Try joining multiple Observables. 5. Try to create Observable that emits latin lower-case

characters.

Sergey Morenets, 2018

**RxJava**

**. Task #**

**Using Observable**

1. Change MealRepository so that its getMeal method returns

Observable. 2. Update CookService/WaiterService so that it uses Observable

as well. 3. Modify Starter class and run it.

Sergey Morenets, 2018

**5**

**RxJava**

PublishSubject AsyncSubject ReplaySubject

Sergey Morenets, 2018

**2. Bridges**

Subject

Observable

Observer

**2. RxJava**

**Bridges**

Class Description PublishSubject Emit items to current subscribers and terminal events

to current (and late) subscribers AsyncSubject Emits the last item and terminal events to subscribers

ReplaySubject Replay items to current (and late) subscribers BehaviorSubject Emits most recent item and all later items to

subscribers

Sergey Morenets, 2018

**RxJava**

Sergey Morenets, 2018

**2. Bridges**

**RxJava**

Sergey Morenets, 2018

**2. Bridges**

**Asynchronous programming**

Sergey Morenets, 2018

**Schedulers**

Method Description newThread() Each unit of work will be executed in the new separate thread without further thread reusage computation() Each unit of work will be executed in the separate

thread pool where number of threads are equal to number of cores. Not recommended for blocking operations io() Each unit of work will be executed in the separate

reusable thread pool with infinite number of threads. Recommended for slow I/O blocking operations single() Each unit of work will be executed sequentially in the

same background thread shutdown() Shutdown standard schedulers

Sergey Morenets, 2018

**Schedulers. Subscriptions**

Sergey Morenets, 2018

Run in main thread

Change thread for downstream operators

Change thread when Observable is subscribed

**Schedulers. Quiz**

Sergey Morenets, 2018

**Schedulers API**

Sergey Morenets, 2018

Stops worker

**Parallel execution**

New type in RxJava 2.x

Executed in parallel

Executed sequentially

Sergey Morenets, 2018

**Task #6. Schedulers& Subject**

1. Try to create different types of Subject’s in RxJavaStarter

class and observe their behaviors. 2. Try to use different types of Scheduler’s when emitting

items. 3. Try to use parallel/sequential processing in Flowable. 4. How would you change CookService/WaiterService so that

items were processed asynchronously?

Sergey Morenets, 2018

**Observable. Error handling**

Method Description onErrorReturnItem(T) Instructs Observable to return given item in

case of error and don’t invoke onError onErrorReturn(Function) Instructs Observable to return result of the

function execution in case of error and don’t invoke onError

onErrorResumeNext Invokes another Observable in case of error onExceptionResumeNext Invokes another Observable in case of

Exception

doOnError(Consumer) Invokes an action if Observable emits onError

signal

Sergey Morenets, 2018

**Observable. Error handling**

Prints 1

Sergey Morenets, 2018

**Observable. Error handling**

Sergey Morenets, 2018

Prints 1

**Observable. Error handling**

io.reactivex.exceptions.OnErrorNotImplementedException: The exception was not handled due to missing onError handler in the subscribe() method call. Further reading: https://github.com/ReactiveX/RxJava/wiki/Error-Handling | 2 exceptions occurred.

at io.reactivex.internal.functions.Functions$OnErrorMissingConsumer.accept(Functio ns.java:704)

Sergey Morenets, 2018

**Observable. Error handling**

Sergey Morenets, 2018

Default error handler

Retry twice if error

**Task #7. Error handling**

1. What are the possible errors in the current workflow? 2. Try to implement error handling in the repositories/services

of the project.

Sergey Morenets, 2018

**Overloading and back**

Sergey Morenets, 2018

**-**

**pressure**

**Overloading and back**

Sergey Morenets, 2018

Cold Observable

**-**

**pressure**

Hot Observable

**Overloading.**

Exception: java.lang.OutOfMemoryError thrown from the UncaughtExceptionHandler in thread "RxSchedulerPurge-1"

Exception: java.lang.OutOfMemoryError thrown from the UncaughtExceptionHandler in thread "RxCachedWorkerPoolEvictor-1"

Sergey Morenets, 2018

**RxJava**

**1.x**

**Overloading.**

**2.x**

Caused by: io.reactivex.exceptions.MissingBackpressureException: Can't deliver value 128 due to lack of requests

at io.reactivex.internal.operators.flowable.FlowableInterval$IntervalSubscribe r.run(FlowableInterval.java:96)

Sergey Morenets, 2018

**RxJava**

Buffer items if Observable emits Items faster than observer can process

**Back**

Discard items if Observable emits Items faster than observer can process

Sergey Morenets, 2018

**-**

**pressure.**

**RxJava**

**2.x**

**Back**

Sergey Morenets, 2018

**-**

**2.x pressure.**

Buffer size

Also DROP\_LATEST or DROP\_OLDEST

Action on buffer overflow

**RxJava**

Group items in batch

**pressure Task #8. Back**

1. What is available places of overloading in the project? What

are suitable back-pressure strategies? 2. Try to apply back-pressure in the project workflow.

Sergey Morenets, 2018

**-**

**Projector Reactor**

Sergey Morenets, 2018

**Project Reactor**

Sergey Morenets, 2018

framework Foundational

for asynchronous applications Developed by Pivotal,

Inc since 2013 Inspired by

Reactor pattern Main contributor is Stephane

Maldini High

-performance platform with Netty/Kafka support

**Types. Reactor vs**

Sergey Morenets, 2018

**RxJava**

Reactor RxJava Flux Observable

Mono Single

**Types. Reactor**

Publisher

Sergey Morenets, 2018

Flux Mono

Reactive Streams

**Reactor. Maven dependencies**

Sergey Morenets, 2018

**Mono. Generation**

Method Description create(Consumer) Returns Mono based on Consumer empty() Returns Mono that emits no items

error(Throwable) Returns Mono that produces an error from(Publisher) Returns Mono based on given Publisher just(T) Returns Mono that emits specified item justOrEmpty(T) Returns Mono that emits specified item if it’s not null never() Returns Mono that emits no signals fromFuture Returns Mono based on CompletableFuture fromRunnable Returns Mono that completes after Runnable has

finished zip Returns Mono that aggregates Mono(s) into tuple

Sergey Morenets, 2018

**Mono transformation**

Method Description delay(Duration) Delays onNext by given duration block() Blocks this thread until Mono emits item or completes

cache() Cache the last emitted item for the next subscribers doFinally(Consu mer)

Triggers Consumer when Mono terminates for any reason doOnCancel Invokes Runnable when Mono is cancelled doOnNext Triggers Consumer when next item is emitted doOnError Triggers Consumer then Mono completes with error flux() Converts this Mono into Flux stream repeat Repeatedly subscribes to the source retry(long) Re-subscribes to the source if it produces an error

Sergey Morenets, 2018

**Mono transformation**

Method Description retryBackoff Returns Mono that makes a retry based on backoff

strategy only if error occurres subscribe() Subscribes to this source and returns Disposable

take(Duration) Completes this Mono if no item was emitted during

timeout timeout(Duration) Raises an error if no item was emitted during timeout toFuture() Converts this Mono into CompletableFuture map(Function) Converts emitted item synchronously using Function filter(Predicate) Emits item only if it Predicates returns true onErrorReturn(T) Emits given item in case of error onErrorResume Subscribes to fallback in case of error

Sergey Morenets, 2018

**Disposable**

Method Description dispose() Cancels current task isDisposed() Returns true if current task/resource is disposed

Sergey Morenets, 2018

Subscription result

Enables to cancel

subsription

**Projector Reactor. Samples**

Sergey Morenets, 2018

**Projector Reactor. Bridges**

Sergey Morenets, 2018

**Task #9. Project Reactor**

1. Add new dependency to your project: 2. Create new class ReactorStarter and try to create new

instances of types Mono/Flux/DirectProcessor. Try to apply intermediate operators. 3. Change RxJava 2.x types in services/repositories in the

project with Reactor types.

Sergey Morenets, 2018

**Spring 5**

Sergey Morenets, 2018

Jigsaw Java 9 and

project support Java EE 7 support

HTTP/2 Servlet 4.0 and

support(Tomcat/Jetty/Undertow) Junit 5 support

**1.0 Kotlin**

support Dropped support for

**Hibernate 3/4, Velocity and Guava Java 8**

usage in core functionality Reactive

architecture (Spring WebFlux) Current version 5.1

**Reactive streams**

Sergey Morenets, 2018

asynchronous Reactive programming provides

data processing in non-blocking mode Affects

infrastructure(web servers, database drivers, web frameworks) Reactive

datastore (Postgres, MongoDB, Couchbase, Redis, Cassandra) Similar to

**CompletableFuture in Java 8**

Sergey Morenets, 2018

Sergey Morenets, 2018

**Blocking I/O**

Sergey Morenets, 2018

**Non**

Sergey Morenets, 2018

**-**

**blocking I/O**

1. Return back to pull model 2. Increase input buffer(queue) 3. Ignore elements

**Back**

Sergey Morenets, 2018

Fast publisher shouldn’t overload slow consumer

Solutions can be

**-**

**pressure**

**Spring 5**

Sergey Morenets, 2018

**spring-web-flux New**

module adapting Reactive Streams specification Based on

Project Reactor Reusing Spring MVC programming model but in non

-blocking mode Based on non

-blocking Servlet API or native SPI connectors(Netty, Undertow) Supported by Tomcat/Jetty/

Netty/Undertow Reactive client

WebClient Support unit

**WebFlux**

**-testing with WebTestClient**

**Spring Web Flux**

Sergey Morenets, 2018

Sergey Morenets, 2018

**Spring Web Flux. Maven**

Sergey Morenets, 2018

**REST controller. Mono**

Sergey Morenets, 2018

**REST controller. Flux**

Sergey Morenets, 2018

**REST controller. Flux**

Sergey Morenets, 2018

**Web client**

Sergey Morenets, 2018

**Spring Task #10.**

**Web Flux**

1. Add Sping WebFlux dependency to pom.xml: 2. Create REST controller that returns stream of lower-case

Latin characters c as Flux type. Run Spring Boot application and verify in the browser that prime numbers are displayed on the page 3. Create REST client that uses WebClient class and prints

received prime numbers to the console.

Sergey Morenets, 2018

Sergey Morenets, 2018

Sergey

Morenets, sergey.morenets@gmail.com