Closing Remarks

José Paumard
PHD, Java Champion, JavaOne RockStar

@JosePaumard https://github.com/JosePaumard

Agenda

Understanding how the API is organized

- what model for your tasks

- tasks chaining 1 – 1 and P – 1

- exception handling

Performances

Creating a Task

Creating a CompletableFuture

Task	Example	Pattern	Executor
Supplier	`() -> getWeather()`	`CompletableFuture<Weather> cf = CompletaleFuture.supplyAsync(supplier)`	yes
Runnable	`() -> logger.info("Done")`	`CompletableFuture<Void> cf = CompletableFuture.runAsync(runnable)`	yes

Creating a CompletableFuture

Task	Pattern	Executor
Completes when all completes	`CompletableFuture<Void> cf = CompletableFuture.allOf(...)`	no
Completes on one of the fastest	`CompletableFuture<Object> cf = CompletableFuture.anyOf(...)`	no

Modeling Tasks

CompletableFuture Supported Tasks

Task	Example	Method	CF method
Runnable	`() -> logger.info("System OK")`	`void run()`	`thenRun()`
Consumer	`n -> logger.info(n + " users read")`	`void accept(Long)`	`thenAccept()`
Function	`id -> readUserFromDB(id)`	`User apply(Long)`	`thenApply()`

Chaining tasks

1 – 1 CompletableFuture Chaining

	Parameter type	Async	Executor
thenRun()	`Runnable`	**yes**	**yes**
thenAccept()	`Consumer<T>`	**yes**	**yes**
thenApply()	`Function<T, U>`	**yes**	**yes**
thenCompose()	`Function<T, CompletionStage<U>>`	**yes**	**yes**

2 – 1 CompletableFuture Chaining

	Parameter type	Async	Executor
runAfterBoth()	`Runnable`	**yes**	**yes**
thenAcceptBoth()	`BiConsumer<T, U>`	**yes**	**yes**
thenCombine()	`BiFunction<T, U, V>`	**yes**	**yes**

2 – 1 CompletableFuture Chaining

	Parameter type	Async	Executor
runAfterEither()	`Runnable`	**yes**	**yes**
acceptEither()	`Consumer<T>`	**yes**	**yes**
applyToEither()	`Function<T, V>`	**yes**	**yes**

Exception Handling

Exception Handling

	Parameter type	Can recover	Async
exceptionally()	`Function<Throwable, T>`	**yes**	**yes**
handle()	`BiFunction<T, Throwable, U>`	**yes**	**yes**
whenComplete()	`BiConsumer<T, Throwable>`	**no**	**yes**

Performances

Identify in-memory vs. I/O computation

Use asynchronous calls for long running tasks

Use executors with the right number of threads

Avoid moving data from one thread to the other

https://jdk.java.net/loom

Course Wrap Up

What did you learn?

The CompletionStage API

Create efficient asynchronous pipelines

How to model a task, how to chain them

How to recover from exceptions

Up Next: Design Efficient Applications!