# Using Function Composition and Chaining to Build Comparators



José Paumard
PHD, JAVA CHAMPION, JAVA ROCK STAR

@JosePaumard https://github.com/JosePaumard



#### Agenda



Applying the principles we saw
On the creation of Comparators
Comparators can be chained
They can be reversed
They can be built with functions



### Implementing a Comparator of Person



```
@FunctionalInterface
public interface Comparator<T> {
   int compare(T t1, T t2);
}
```

```
If the returned integer is positive then t1 > t2

If it is negative then t1 < t2

If it is equal to 0, then t1 = t2
```



```
public class Person {
   private String name;
   private int age;

   // getters and setters
}
```

How can you create a comparator of Person

That would compare users using their name?



#### Demo



Let us write this comparator





A Comparator only depends on

- the type of items it compares
- and a function to extract a key

It can be built from this key extractor

And then can be reversed



### Composing Comparators





#### Given two comparators of Person:

- the first one compares the names
- the second one compares the ages

How can we create a third one

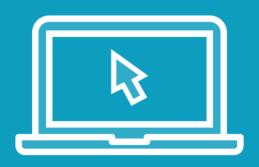
That would compare the ages

In case the names are the same?

And what is the best pattern to create?



#### Demo



Let us do that in the IDE





Using default and static methods

And composition and chaining

Designing a fluent Comparator API becomes very easy!



## Module Wrap Up



You saw how to go one step further
In the design of fluent API

The Comparator interface is a great interface to study this point

