**DECLARATIVE APPROACH IN JAVA**

<https://amigoscode.com/courses/java-functional-programming/lectures/35560249>

<https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html>

**Functional programming in java is also useful because it allows us to use higher-order function, which are functions that respect one of the following conditions:**

1. **They are functions that take one or more functions as parameters**
2. **They return a function as a parameter**

**Also functional programming allows us to have pure functions ( they don’t have inside of them variables declared outside of them, so they don’t have any sideeffects.**

**Let’s say we have a list full of people and we want to filter this list, maybe create a new one, based on the gender of those people.**

**Text

Description automatically generated**

**The declarative approach in java allows us to do this in one swell-swoop without creating more logic, like for loops for iterating through the list and adding the people with the Gender of female into a new list.**

**Predicates**

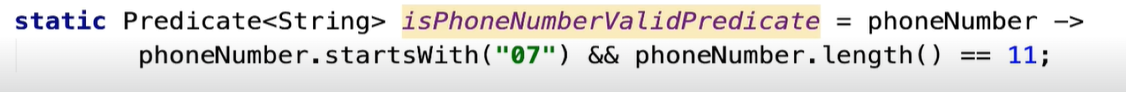
**In the previous example we have used a predicate.**

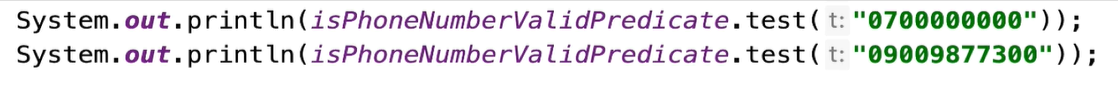
**Text

Description automatically generated**

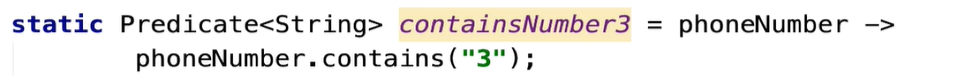
**We can easily declare a predicate before working directly with the stream and after that, using the predicate in our stream.**

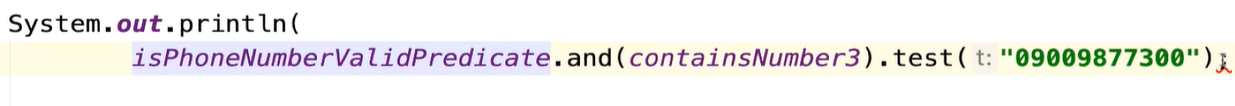
**We can also declare a predicate and use it by itself using the ,,test” method which will return a boolean value.**

****

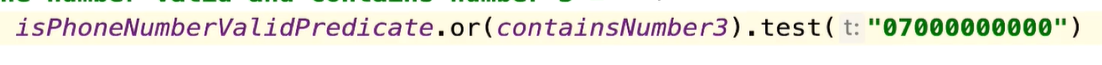
****

**We can also chain predicates using the ,,and” method**

****

****

**We can also use an OR statement using predicates, where only one of the predicates has to hold true.**

****

**FUNCTION<T,R>**

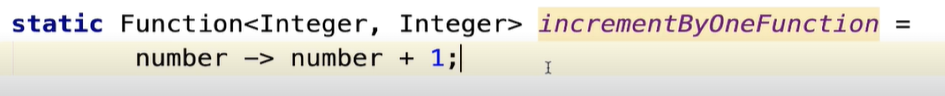
* **Represents a function that accepts one argument and produces a result.**

**Let’s say we have a normal function that simply increments a number.**

**A picture containing chart

Description automatically generated**

**If we want to create this same function but by using the FUNCTION<T,R> template, we would say**

****

**And to call it, we would just call the apply method to it.**

****

**(Chaining functions with andThen)**

**We can easily chain functions by calling the andThen Method**

**Text

Description automatically generated**

**We can also use the andThen method to create a new function, which will represent the combination of the 2 functions we chain.**

**Text

Description automatically generated**

**(BiFunction)**

**BiFunctions behave exaclty like the previous functions but take 2 parameters instead of just 1.**

**Graphical user interface, text, application, email

Description automatically generated**

****

**(Consumer)**

****

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**This second one is a BiConsumer which takes 2 parameters, the second being a flag which chooses whether or not the phoneNumber should be displayed, using the ternary operator.**

**(Supplier)**

## 

**We can use Suppliers to supply us with the data we need**

## 

**We can use a supplier to supply a list also.**

**Text

Description automatically generated**

**(Intro to Streams)**

**Text

Description automatically generated**

**We can easily use streams to transform an element of a list into another one. The result of the previous map, can be passed to another map function and so on.**

**In this example, we take every user and transfrom it into its name, and then take each name and transform it into its length.**

**We can refactor each element from the previous streams operations by clicking**

**CTRL + ALT + V**

**And we can see that there are all variables which we previously learnt.**

**Text

Description automatically generated**

**(Java Optionals)**

**Java Optionals are very usefull to use when we don’t know if the value that will be returned will be null or not. It’s a safe way for dealing with NullPointer Exceptions.**

**For Optional, we have 3 cases:**

**Graphical user interface, text

Description automatically generated**

1. **Optional.of 🡺 means that we for sure know that the value we will pass to it is not null**
2. **Optional.ofNullable 🡺 we use when we don’t know if the value will be null or not**

**Graphical user interface, text, application

Description automatically generated**

**🡺 Graphical user interface, text, application

Description automatically generated**

**We can use methods like „orElse” to do something else in case a certain value isEmpty.**

**Text

Description automatically generated**

**🡺 This will print „hello” cause the string hello is not empty. If it were, we would have printed „world”**

**Text

Description automatically generated**

**In the case above:**

1. **Hello.isPresent returns false**
2. **Hello.isEmpty returns true**
3. **Sout(orElse) will return „world”**

**Now we can use Optionals with streams**

**Graphical user interface

Description automatically generated**

**The first map method gets the optional hello and turns it into an upperString so it will print „HELLO”**

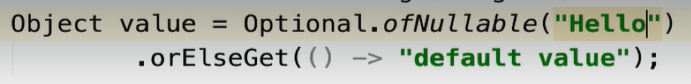
**The orElseGet branch allows us to give to it a supplier. So we do some extra logic there and return a value.**

**Text

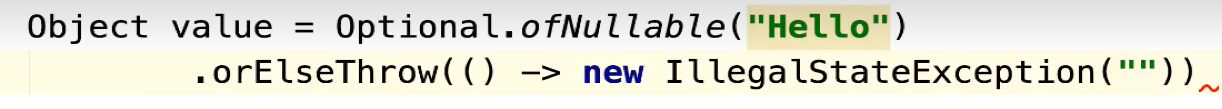
Description automatically generated with medium confidence**

**We can use ifPresentOrElse method**

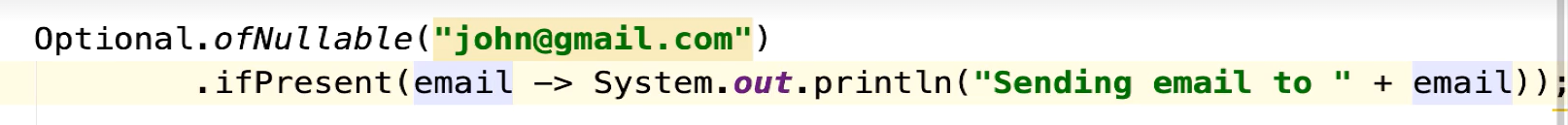
**We can also get the value of the optional by calling the get method or use the .orElseGet method which returns the value of the optional in case in exists and returns a default value in case it doesn’t.**

****

**We can also throw an exception which takes a Supplier**

****

**We can also use the .isPresent method which takes a Consumer**

****

**We can also use the .ifPresent or else which takes a Consumer and a Runnable**

**Graphical user interface, text, application, email

Description automatically generated**

**(More on streams)**

**We can use a stream and see if all the elements match using the .allMatch function which takes a Predicate**

**Graphical user interface, text, application

Description automatically generated**

**We can use the .anyMatch function which takes a Predicate and returns True if any of the elements ( or at least 1) respects the predicate**

**Graphical user interface, text

Description automatically generated**

**We can use the .noneMatch function which takes a Predicate and returns True if none of the elements respect the predicate**

**Graphical user interface, text, application

Description automatically generated**

**(Implementing Combinator Pattern)**

**The combinator patter provides us with an efficient way of dealing with validation.**

**We can get rid of the old method of creating methods for validating each parameter which sometimes poses the problem of not knowing which validation failed.**

**We can start by creating an interface which extends a function. This function will have as 1st parameter the object whose attributes need validation and as a 2nd parameter an enum which holds all possible outcomes of validation.**

**Application

Description automatically generated with medium confidence**

**In the interface file, we start by creating our function for validation. These functions will be static and return an object of the interface’s type.**

**Graphical user interface, text, application, email

Description automatically generated**

**The combinator pattern enables us to chain all these functions together in a new function.**

**Graphical user interface

Description automatically generated with medium confidence**

**And to test it, we simply call it like this:**

**Graphical user interface, text, application

Description automatically generated**

**(Callback like JS)**

**Callback functions are functions which we give into another function. This can be don with functional programming in java using Consumers.**

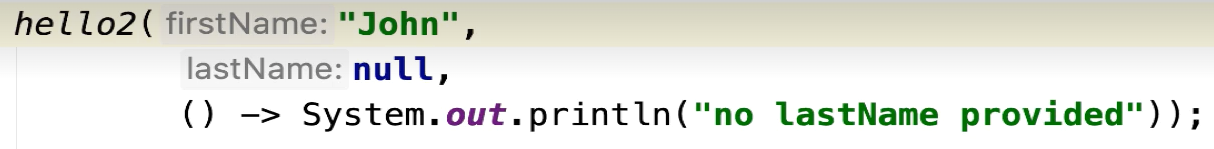
**Graphical user interface, text, application, email

Description automatically generated**

**Or you can use them with Runnables**

**Graphical user interface, text

Description automatically generated**

****

**(Lambdas)**

**Lambdas are what we’ve been using all this time.**

**When we have a lambda expression with only one sentence, we don’t have to use curly brackets, like in the case below**

****

**When we have more statements, we need curly brackets**

**For BiFunctions, or lambda expression with multiple parameters, we have to put the parameters inside paranthesis**

**Graphical user interface, application

Description automatically generated with medium confidence**