O que faço da vida?

Onde trabalho?

Oieeee!

Eu sou a Catarina Nogueira(Cata)

Estou aqui porque amo passar conhecimento!

DEVS JAVAGIRL

Introduçãozinha Dojo!

O QUE FAREMOS NESSES 40 MINUTINHOS?

Espero que seja bem divertido!

Slides e códigos estão no meu github(link no final)

Programação Funcional

A Lu falou alguns conceitos e sobre a importância do funcional no último encontro, vamos relembrar esse conceitos e aplicá-los ;)

Estorinha...

Temos nossa personagem, a Ada.

Ela tem de ficar equilibrando pratos todos dias, carreira, família, amigos, relacionamento, estudos, contas para pagar etc.

Vamos começar com conceitos simples e no final uns mais avançados...

Hoje vamos ajuda-la usando programação funcional!

01) Filters

- Ada estava sem tempo para os amigos 

```
List<String> life = Arrays.asList("Familia", "Amigos", "Carreira", "Relacionamento",
"Estudo", "Saúde");

List<String> result = life.stream()
        .filter(item -> !"Amigos".equals(item))
        .collect(Collectors.toList());
```

Stream?

Collectors?

O que têm na lista agora?

Vavr(http://www.vavr.io/)

02) Map

- Ada terminou com x crush e vai tirá-lx da lista do seu aniversário

```
List<Person> birthdayInvites = Arrays.asList(
        new Person("crush", 30),
        new Person("amigo1", 26),
        new Person("amiga1", 23),
        new Person("Mami", 69)
);

List<String> newInvites = birthdayInvites.stream()
        .filter(oldInvites -> !"crush".equals(oldInvites.getName()))
       // .map(Person::getName)
        .map(person -> person.getName())
        .collect(Collectors.toList());

 newInvites.forEach(System.out::println);

}
```

O que o map faz?

Person::getName?

O que o forEach printa?

04) FLATMAP

- Ada quer listar os nomes dos convidados dos convidados

```
List<String> newInvites = birthdayInvites.stream()
        .filter(oldInvites -> !"crush".equals(oldInvites.getName()))
        .flatMap(person -> person.getFriends().stream())
        .map(Person::getName)
        .collect(Collectors.toList());
```

```
newInvites.forEach(System.out::println);
```

```
  Person amigoDoAmigo = new Person("amigoDoAmigo", 45);
  Person amigoDoAmigo2 = new Person("amigoDoAmigo2", 32);
  Person amigaDaAmiga = new Person("amigaDaAmiga2", 28);

  List<Person> birthdayInvites = Arrays.asList(
          new Person("crush", 30),
          new Person( "amigo1", 26, Arrays.asList(amigoDoAmigo, amigoDoAmigo2)),
          new Person("amiga1", 23),
          new Person("Mami", 69, Arrays.asList(amigaDaAmiga))
  );
```

03)Optional +funções alta ordem + Interfaces funcionais

- Ada é bem preparada e quer ter um plano B caso o restaurante cancele de ultima hora a reserva dela!

```
final Restaurant op1 = new Restaurant("opcaoA", 678990252, false);
final Restaurant op2 = new Restaurant("opcaoB", 338738737, true);

final Optional<Restaurant> restaurantOpt =
filterRestaurants(Restaurant::isAvailable, op1, op2);

restaurantOpt.ifPresent(System.out::println);
```

```
@FunctionalInterface
public interface Predicate<T>
{
    boolean apply(T t);
}
```

```
private Optional<Restaurant> filterRestaurants(final Predicate<Restaurant> fn, final Restaurant ...restaurants)
{
   for (Restaurant r : restaurants) {
        if (fn.apply(r)) return Optional.ofNullable(r);
    }
    return Optional.empty();
}
```

Como estamos até agora?

04) Futures : Motivação

- Saindo um pouco do nosso exemplo...

Runnable

```
private static void printOrderTest() {
    Runnable runnable = () -> {
        String threadName =
Thread.currentThread().getName();
        System.out.println("Olá " + threadName);
    };

    runnable.run();

    Thread thread = new Thread(runnable);
    thread.start();

    System.out.println("Fim!");
}
```

```
public static void
main(String[] args){
   printOrderTest();
}
```

04) Futures : Motivação

- Saindo um pouco do nosso exemplo...

Runnable

Olá main
Olá Thread-0
Fim!

Olá main
Fim!
Olá Thread-0

Ordem não determinística!

05) Futures : Motivação

- Saindo um pouco do nosso exemplo...

```
private static void printOrderTestWithSleep() {

    Runnable runnable = () -> {
       try {
         String threadName =
Thread.currentThread().getName();
         System.out.println("Olá " + threadName);
         TimeUnit.SECONDS.sleep(2);
         System.out.println("Olá novamente " +
threadName);
       }
       catch (InterruptedException e) {
           e.printStackTrace();
       }
    };
   Thread thread = new Thread(runnable);
   thread.start();
}
```

Runnable

```
public static void
main(String[] args){
   printOrderTestWithSleep();
}
```

Que mão! Como controlar isso!

05) Futures : solução

CompletableFuture

```
final CompletableFuture<String> hello = CompletableFuture.supplyAsync(
    () -> "Hello"
);

final CompletableFuture<String> future = hello.thenApply(
    s -> s + " World"
);

assertEquals("Hello World", future.get());
```

Future provém mecanismos para evaluação

E se Hello falhar?

Tipagem forte

Closures

Lambdas

Pattern Matching

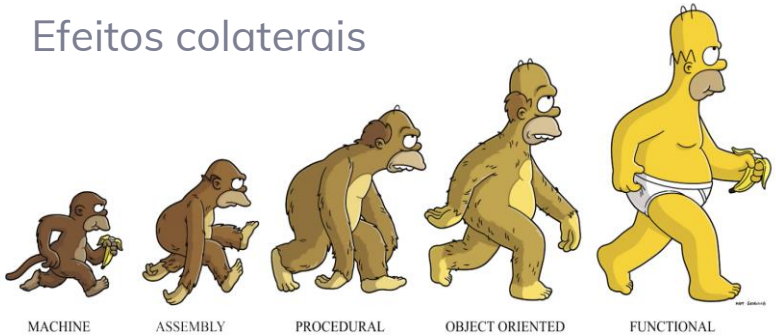
Conjunções de tipos

Falta muita coisa do mundo funcional!

Paradigmas

Recursão

Efeitos colaterais



Concorrência

Funções de primeira ordem

Funções imutáveis

Revisando conceitos

Filter

Remove elementos indesejados com base em alguma(s) condição(ões)..

Map

Itera sobre todos os itens de entrada, devolvendo um novo array transformado conforme a implementação desejada.

Optional

Um objeto container que pode ou não conter valor. Fim ao “null”!

FlatMap

À grosso modo, o FlatMap é uma função de alta ordem que recebe múltiplas listas e devolve uma

Futures

O tipo Future tem seu principal uso em se tratando da recuperação de informação que serão utilizadas no futuro e não estão prontas ainda

Interfaces Funcionais

Existem muitas já prontas, como as BiFunctions<x, y,z>, assim como podem ser criadas contendo um método abstrato, como fizemos no exemplo.

O que é um Dojo?

Atividades do Dojo!

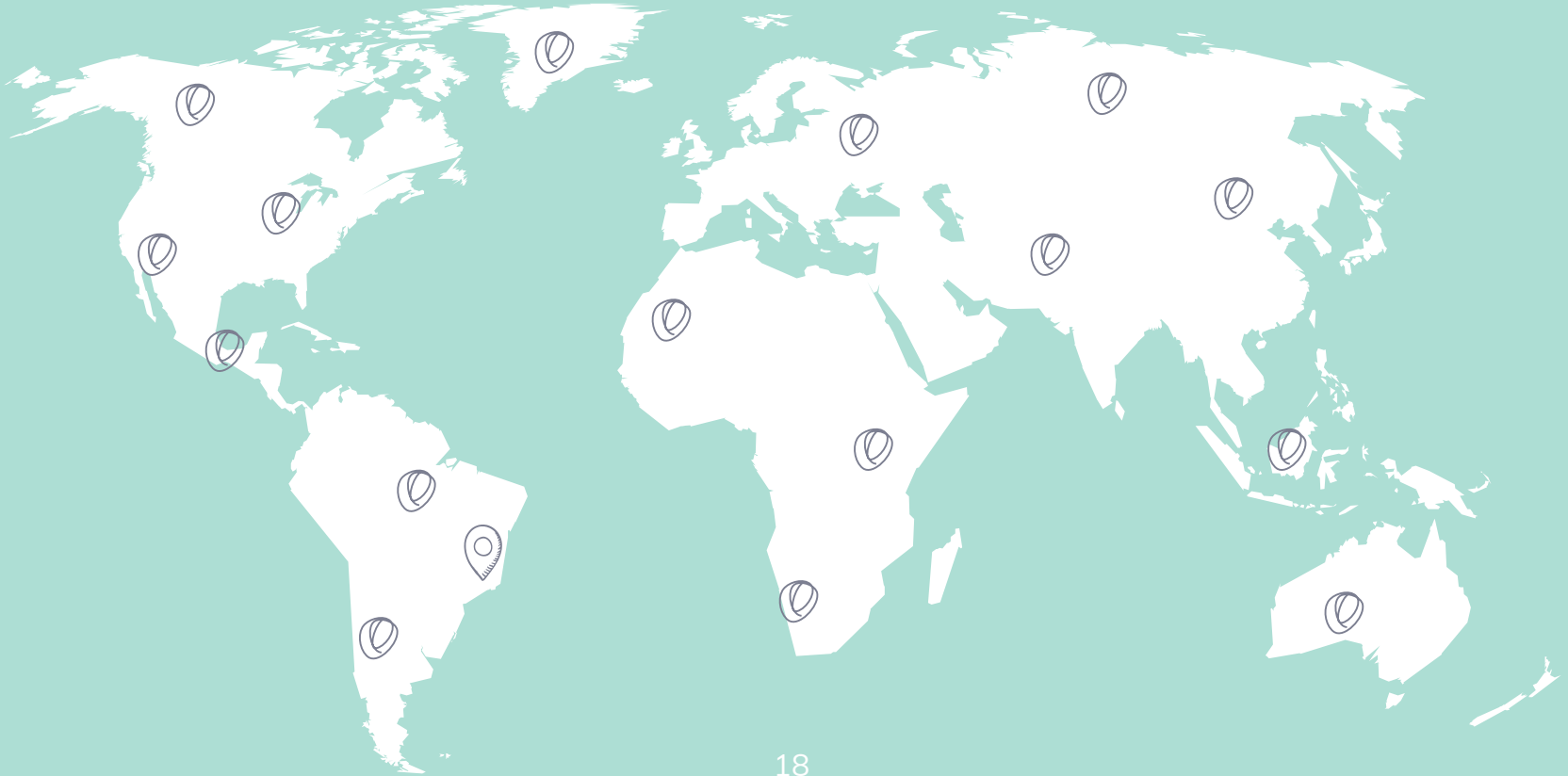
1) **Nós**
Três

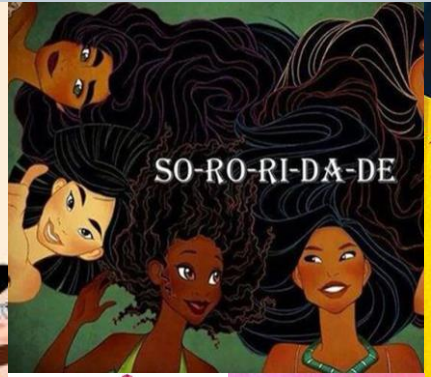
2) **Conseguimos,**
pequenos

3) **Gurias!**
Desafios

Três minutos para cada pessoa!

Let's rule The world, girls!





Obrigada!!

Estou sempre ai para trocar uma ideia!

Email: cvrnogueira@gmail.com

Github: https://github.com/cvrnogueira

Linkedin: https://www.linkedin.com/in/catarina-nogueira/