CONHECENDO E APLICANDO UM OUTRO PARADIGMA SEM SAIR DE SUA LINGUAGEM FAVORITA!

JAVA é claro! \o/

Rafael Nunes Vieira

7 anos de experiência com Desenv.
Pós em Arquitetura de Sistemas Distribuídos
Consultor Desenvolvedor na **Sensedia**

Me interesso muito por:

#java #javascript #clojure #cleancode #apis #containers #functionalprogramming #architetura

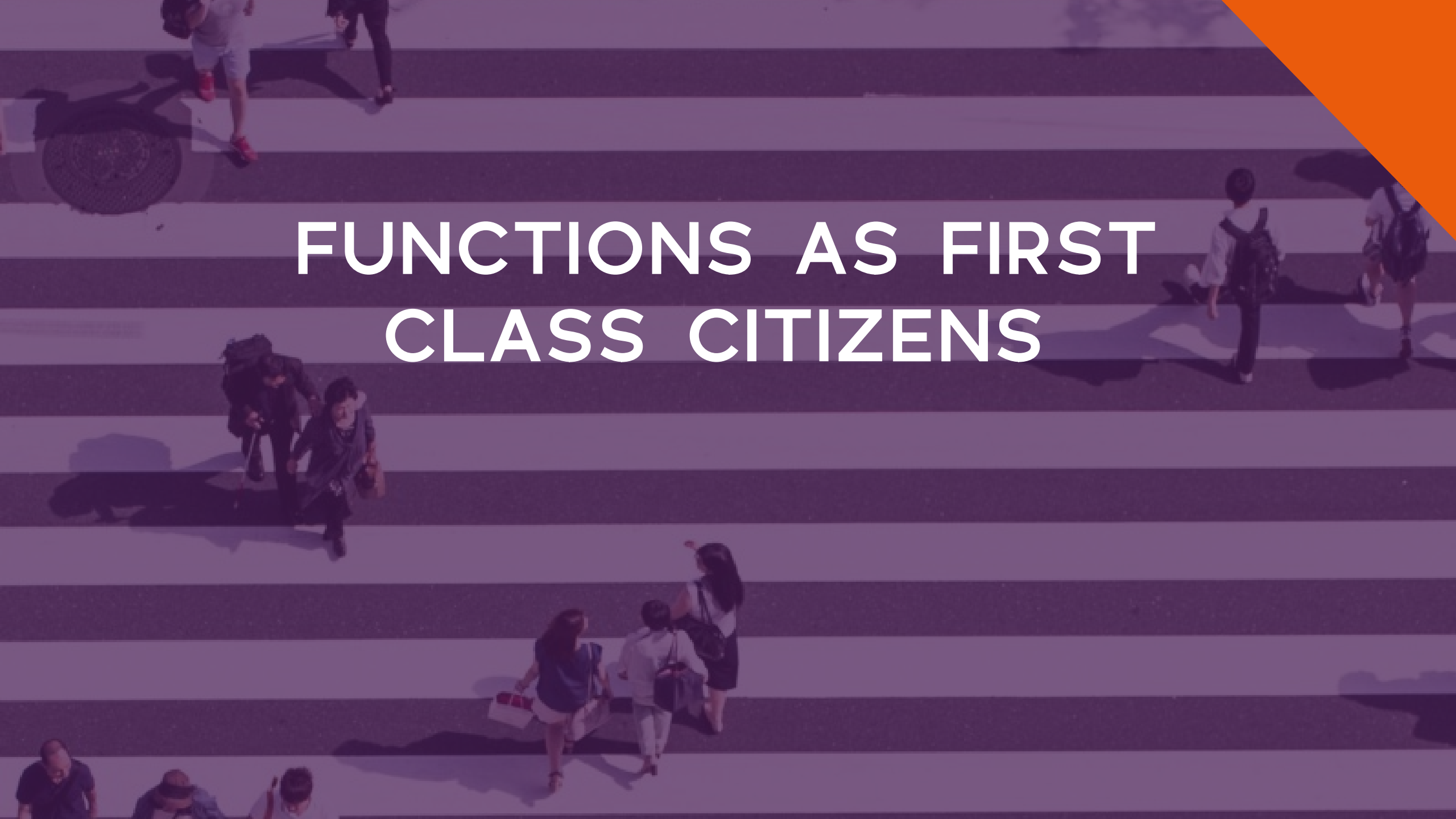

sensedia


elo

PARADIGMA FUNCIONAL NO JAVA

This is a new way in Java, one that will make our code more expressive, easier to write, less error prone, and easier to parallelize than has been the case with Java until now

Venkat Subramanian, Functional Programming in Java



FUNCTIONS AS FIRST CLASS CITIZENS

Implements Runnable

```
public class MyRunnable implements Runnable {
    public void run(){
        System.out.println("MyRunnable running");
    }
}

Thread thread = new Thread(new MyRunnable());
thread.start();
```

Anonymous Implementation of Runnable

```
Runnable runnable = new Runnable(){
    public void run(){
        System.out.println("MyRunnable running");
    }
};

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

```
public class MyRunnable implements Runnable {
    public void run(){
        System.out.println("MyRunnable running");
    }
}
```

```
Runnable runnable = new Runnable(){
    public void run(){
        System.out.println("MyRunnable running");
    }
};
```

Lambda Implementation

```
Runnable myRunnable = () -> System.out.println("MyRunnable running");

Thread thread = new Thread(myRunnable);
thread.start();
```

FunctionalInterface

```
interface Function<T, R>
interface Supplier<T>
interface Consumer<T>
interface Predicate<T>
...
```

FunctionalInterface

```
@FunctionalInterface
interface Callback {
    void call(String response);
}
public static void search(String term, Callback callback) {
    // make async call when it returns run callback
    // function with response
    // {...}
    //

    callback.call(term + " found");
}
```

High Order Functions

```
users.stream().filter((User user) -> user.gender == Gender.MALE)
              .map((User user) -> user.cardLimit)
              .reduce((v1, v2) -> v1 + v2)
              .get()
```


PURE FUNCTIONS

Pure Function

```
public static int sum(int a, int b) {
    return a + b;
}
```

Impure Function

```
public static int subtract(int a, int b) {
    return new Random().nextInt() - a - b;
}

public static int multiply(int a, int b) {
    int result = a * b;

    OtherClass.persist(result);

    return result;
} // With side effects
```

Functions should do one thing. They should do it well. They should do it only

Robert C. Martin Clean Code

```java
@Override
public UserCardResponse getCardById(UUID userId, UUID cardId, String expand) {
  // verifica se o usuario existe e nao esta com o status igual a DELETADO.
  UserDomain userDomain = userService.getById(userId);
  if (userDomain == null) {
    throw new NotFoundException(this.messageError.create(Messages.USER_NOT_FOUND));
  }
  // verifica se o card existe
  Integer positionCard = this.positionCard(userDomain.getCard(), cardId);
  if (positionCard < 0) {
    throw new NotFoundException(this.messageError.create(Messages.CARD_NOT_FOUND));
  }
  CardDomain cardDomain = userDomain.getCard().get(positionCard);
  if (ObjectUtils.isEmpty(cardDomain)) {
    throw new NotFoundException(this.messageError.create(Messages.CARD_NOT_FOUND));
  }
  ...
```

```
  if (!ObjectUtils.isEmpty(cardDomain.getNumberBin())) {
    Optional<BinDomain> binDomain = this.binRepository.findById(cardDomain.getNumberBin());
    binDomain.ifPresent(cardDomain::setBin);
  }

  UserCardResponse userCardResponse = this.userCardConverter.toCard(cardDomain, expand);
  try {
    CardExpiry cardExpiry = this.getDecryptedExpiry(cardDomain.getDateExpiry());
    userCardResponse.setExpiry(cardExpiry);
  } catch (NullPointerException | NoSuchPaddingException | NoSuchAlgorithmException |
    InvalidKeyException | IllegalBlockSizeException | BadPaddingException | IllegalArgumentException e) {
    log.warn("UserCardController.getCard: " + e.getMessage(), e);
    throw new UnprocessableEntityException(this.messageError.create(Messages.DECIPHER_ERROR),
      "Decipher error", e);
  }
  return userCardResponse;
}
```

Refactoring

```
@Override
public UserCardResponse getCardById(UUID userId, UUID cardId, String expand) {
  UserDomain user = getActiveUser(userId);
  Integer positionCard = validateCardExists(cardId, user);
  CardDomain card = getCardFromUser(user, positionCard);
  defineCardBinData(card);
  UserCardResponse userCardResponse = this.userCardConverter.toCard(card, expand);
  defineCardExpiry(card, userCardResponse);

  return userCardResponse;
}
```

IMMUTABILITY

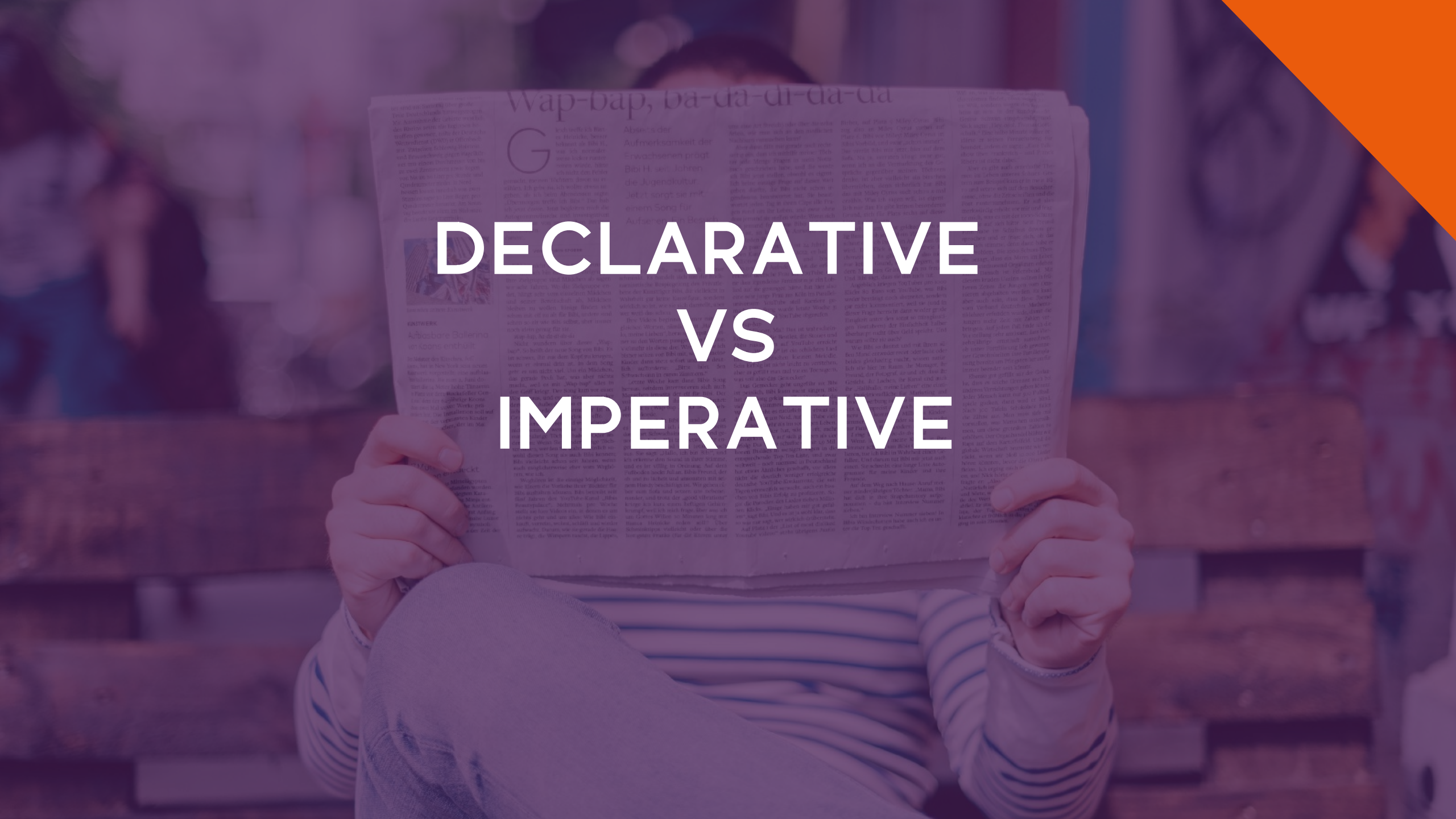
```java
public class User {
    private final String name;
    private final String email;

    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }
    ...
}
```

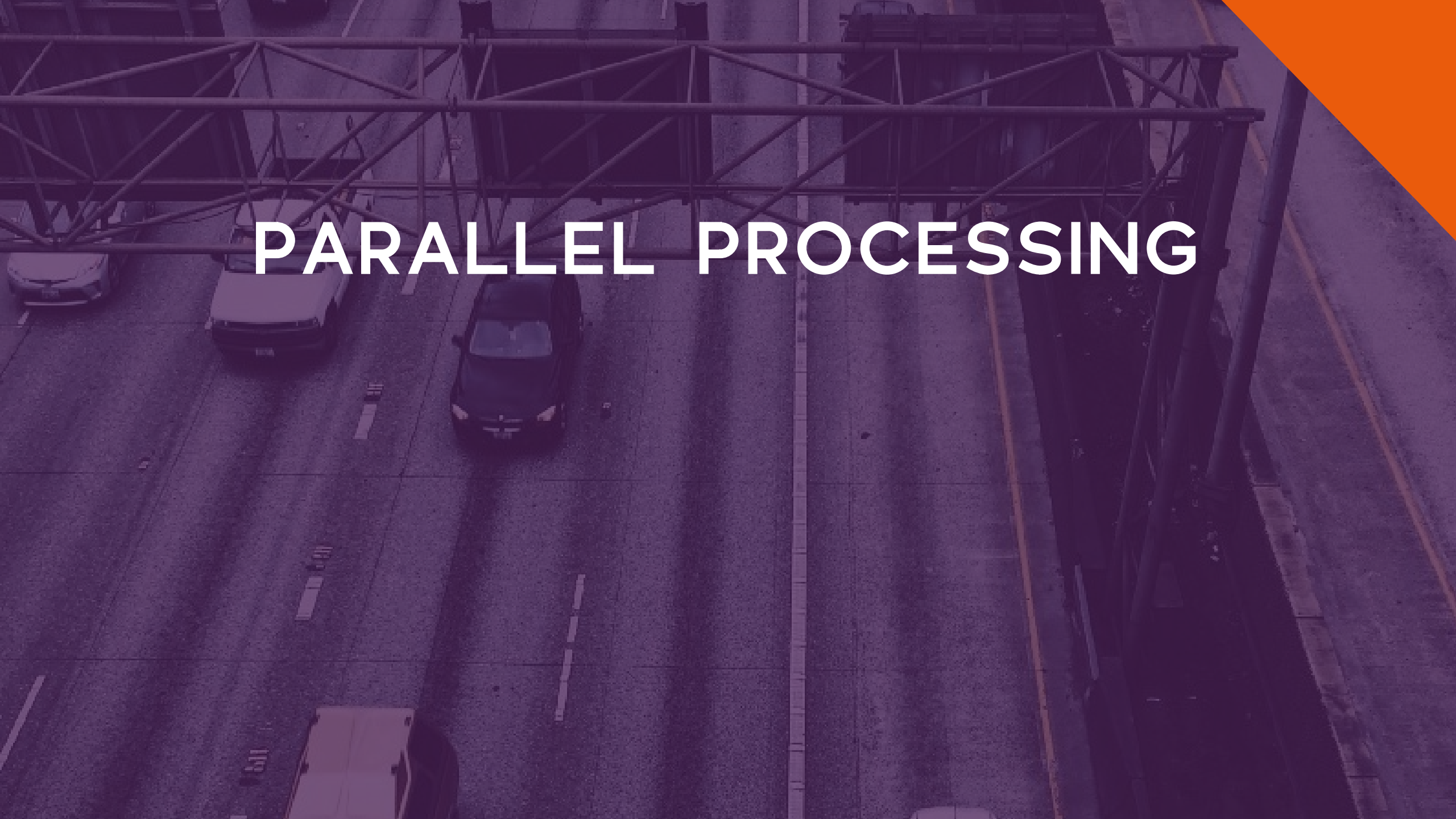
```java
final User joao = new User("João", "joao@gmail.com");
```

Tools:

- immutables/immutables
- rzwitserloot/lombok

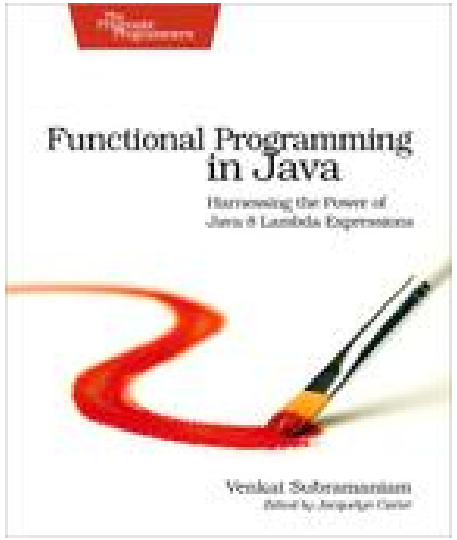


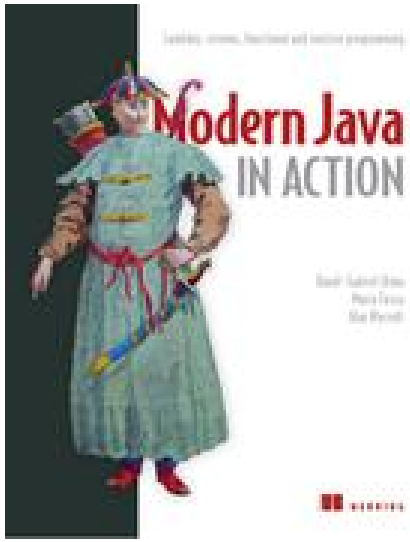
DECLARATIVE VS IMPERATIVE

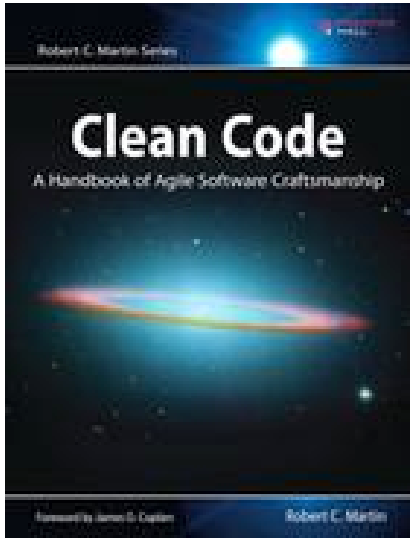


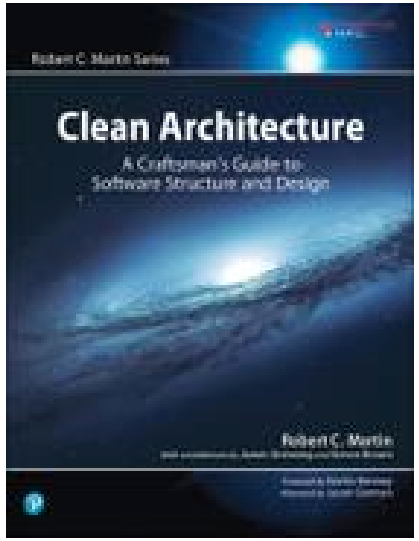
PARALLEL PROCESSING

Bibliografia









DÚVIDAS?


OBRIGADO!
@nvieirarafael
in/nvieirarafael