# Optional API aplicada

Ellian Aragão Dias

NTconsult Comunidade Java

09 de Maio de 2023

# NullPointer

- The "Billion-Dollar mistake"
- Criado no ALGOL
- Causa do Null Pointer Exception
- Maior causa de erros nas aplicações

# NullPointer - Models

```java
public class Person {
    1 usage
    private Car car;
    1 usage
    public Car getCar() {
        return car;
    }
}
2 usages
public class Car {
    1 usage
    private Insurance insurance;

    1 usage
    public Insurance getInsurance() {
        return insurance;
    }
}
2 usages
public class Insurance {
    1 usage
    private String name;

    public String getName() {
        return name;
    }
}
```

# NullPointer - getter

```java
27 @    public String getCarInsuranceName( @NotNull Person person) {
28          return person.getCar().getInsurance().getName();
29      }
30
        no usages
31      public String getCarInsuraceNameSafety(Person person) {
32          if (Objects.nonNull( obj: person)) {
33              final var car = person.getCar();
34              if (Objects.nonNull( obj: car)) {
35                  final var insurance = car.getInsurance();
36                  if (Objects.nonNull( obj: insurance)) {
37                      return insurance.getName();
38                  }
39              }
40          }
41          return "unknow";
42      }
43
        no usages
44      public String getCarInsuraceNameSafety2(Person person) {
45          if (Objects.isNull( obj: person)) {
46              return "unknow";
47          }
48          final var car = person.getCar();
49          if (Objects.isNull( obj: car)) {
50              return "unknow";
51          }
52          final var insurance = car.getInsurance();
53          if (Objects.isNull( obj: insurance)) {
54              return "unknow";
55          }
56          return insurance.getName();
57      }
```

# NullPointer - O que tem de errado?

- Causa erros (NullPointerException)
- Piora a legibilidade do código
- Uso de ponteiros

# NullPointer - O que fazer neste caso então?

O que viria a ser uma solução para estes problemas?

- NullPointerException
- Legibilidade
- Ponteiros

# Optional - A solução para NullPointers

- Introduzido na versão 8 do Java
- Implementa muito do paradigma funcional
- Estrutura de modo declarativo
- Encapsula os objetos de modo a acessar indiretamente
- Como inicializar:
  - *Optional.of*()
  - *Optional.ofNullable*()
  - *Optional.empty*()

# Optional - Reimplementando o get

```java
45      public String getCarInsuraceNameSafety2(Person person) {
46          if (Objects.isNull( obj: person)) {
47              return "unknow";
48          }
49          final var car = person.getCar();
50          if (Objects.isNull( obj: car)) {
51              return "unknow";
52          }
53          final var insurance = car.getInsurance();
54          if (Objects.isNull( obj: insurance)) {
55              return "unknow";
56          }
57          return insurance.getName();
58      }
59
        no usages
60      public String getCarInsuraceNameOptional(Person person) {
61          return Optional.ofNullable( value: person) Optional<Person>
62                  .map( mapper: Person::getCar) Optional<Car>
63                  .map( mapper: Car::getInsurance) Optional<Insurance>
64                  .map( mapper: Insurance::getName) Optional<String>
65                  .orElse( other: "unknow");
66      }
```

# Optional - filter

```
213  @    @Contract("null->!null")
     public Optional<T> filter( @NotNull Predicate<? super T> predicate) {
214      Objects.requireNonNull( obj: predicate);
215      if (isEmpty()) {
216          return this;
217      } else {
218          return predicate.test( t: value) ? this : empty();
219      }
220  }
```

# Optional - ifPresent e ifPresentOrElse

```
176     public void ifPresent(Consumer<? super T> action) {
177         if (value != null) {
178             action.accept( t: value);
179         }
180     }
181
```

If a value is present, performs the given action with the value, otherwise performs the given empty-based action.

Params: `action` – the action to be performed, if a value is present
`emptyAction` – the empty-based action to be performed, if no value is present

Throws: `NullPointerException` – if a value is present and the given action is `null`, or no value is present and the given empty-based action is `null`.

Since: 9

```
194     public void ifPresentOrElse(Consumer<? super T> action, Runnable emptyAction) {
195         if (value != null) {
196             action.accept( t: value);
197         } else {
198             emptyAction.run();
199         }
200     }
```

# Optional - map e flatMap

```
255  @  public <U> Optional<U> map( @NotNull Function<? super T, ? extends U> mapper) {
256         Objects.requireNonNull( obj: mapper);
257         if (isEmpty()) {
258             return empty();
259         } else {
260             return Optional.ofNullable( value: mapper.apply( t: value));
261         }
262     }
263
```

If a value is present, returns the result of applying the given `Optional`-bearing mapping function to the value, otherwise returns an empty `Optional`.

This method is similar to `map(Function)`, but the mapping function is one whose result is already an `Optional`, and if invoked, `flatMap` does not wrap it within an additional `Optional`.

Params: `mapper` – the mapping function to apply to a value, if present

Returns: the result of applying an `Optional`-bearing mapping function to the value of this `Optional`, if a value is present, otherwise an empty `Optional`

Throws: `NullPointerException` – if the mapping function is `null` or returns a `null` result

```
283  @  public <U> Optional<U> flatMap( @NotNull Function<? super T, ? extends Optional<? extends U>> mapper) {
284         Objects.requireNonNull( obj: mapper);
285         if (isEmpty()) {
286             return empty();
287         } else {
288             /unchecked/
289             Optional<U> r = (Optional<U>) mapper.apply( t: value);
290             return Objects.requireNonNull( obj: r);
291         }
292     }
```

# Optional - orElse e orElseGet

```
349   @   public T orElse( @Nullable @Flow(targetIsContainer = true) T other) {
350           return value ≠ null ? value : other;
351       }
352
```

If a value is present, returns the value, otherwise returns the result produced by the supplying function.

Params: `supplier` – the supplying function that produces a value to be returned

Returns: the value, if present, otherwise the result produced by the supplying function

Throws: `NullPointerException` – if no value is present and the supplying function is `null`

```
363       public T orElseGet(Supplier<? extends T> supplier) {
364           return value ≠ null ? value : supplier.get();
365       }
```

# Optional - orElseThrow

```
375     public T orElseThrow() {
376         if (value == null) {
377             throw new NoSuchElementException("No value present");
378         }
379         return value;
380     }
381
```

If a value is present, returns the value, otherwise throws an exception produced by the exception supplying function.

Params: `exceptionSupplier` – the supplying function that produces an exception to be thrown

Returns: the value, if present

Throws: `X` – if no value is present
`NullPointerException` – if no value is present and the exception supplying function is `null`

API Note: A method reference to the exception constructor with an empty argument list can be used as the supplier. For example, `IllegalStateException::new`

```
    @Contract(pure = true)
399 public <X extends Throwable> T orElseThrow(Supplier<? extends X> exceptionSupplier) throws X {
400     if (value != null) {
401         return value;
402     } else {
403         throw exceptionSupplier.get();
404     }
405 }
```

# Optional - or e stream

```
307  @      public Optional<T> or( @NotNull Supplier<? extends Optional<? extends T>> supplier) {
308             Objects.requireNonNull( obj: supplier);
309             if (isPresent()) {
310                 return this;
311             } else {
312                 /unchecked/
313                 Optional<T> r = (Optional<T>) supplier.get();
314                 return Objects.requireNonNull( obj: r);
315             }
316         }
317
```

If a value is present, returns a sequential `Stream` containing only that value, otherwise returns an empty `Stream`.

Returns: the optional value as a `Stream`

API Note: This method can be used to transform a `Stream` of optional elements to a `Stream` of present value elements:

```
Stream<Optional<T>> os = ..
Stream<T> s = os.flatMap(Optional::stream)
```

Since: 9

```
         @NotNull
333  @      public Stream<T> stream() {
334             if (isEmpty()) {
335                 return Stream.empty();
336             } else {
337                 return Stream.of( t: value);
338             }
339         }
```

# Finalização da apresentação

Muito obrigado pela atenção de todos!