Optional no Java 8

Entre as novidades do Java 8 talvez `Optional` seja uma das menos comentadas. A principal proposta deste recurso é encapsular o retorno de métodos e informar se um valor do tipo `<T>` está presente ou ausente.

Com isso é possível:

- Evitar erros `NullPointerException`.
- Parar de fazer verificações de valor nulo do tipo `if (cliente != null)`.
- Escrever código mais limpo e elegante.

Primeiro vou apresentar alguns métodos e em seguida alguns cenários de uso.

empty - Retorna uma instância de `Optional` vazia.

```
Optional<Cliente> retorno = Optional.empty();
```

of - Retorna um `Optional` com o valor fornecido, mas o valor não pode ser nulo. Se não tiver certeza de que o valor não é nulo use `Optional.ofNullable`.

```
Optional<Cliente> retorno = Optional.of(buscaCliente(cpf));
```

ofNullable - Se um valor estiver presente, retorna um `Optional` com o valor , caso contrário, retorna um `Optional` vazio. Este é um dos métodos mais indicados para criar um `Optional`.

```
Optional<Cliente> retorno =
Optional.ofNullable(buscaCliente(cpf));
```

filter - Se um valor estiver presente e o valor corresponder ao predicado retorna um `Optional` com o valor, se não, retorna um `Optional` vazio.

```
Optional<Cliente> retorno = buscaCliente(cpf)
   .filter(cliente -> !cliente.getNome().isEmpty());
```

isPresent - Se um valor estiver presente retorna `true`, se não, retorna `false`.

```
Optional<Cliente> retorno =
Optional.ofNullable(buscaCliente(cpf));
if (retorno.isPresent()) {
   System.out.println("Cliente encontrado.");
} else {
   System.out.println("Cliente não encontrado.");
}
```

O código acima não é o cenário ideal do uso de `Optional`.

get - Se um valor estiver presente retorna o valor, caso contrário, lança `NoSuchElementException`. Então para usar `get` é preciso ter certeza de que o `Optional` não está vazio.

```
Optional<Cliente> retorno =
Optional.ofNullable(buscaCliente(cpf));
if (retorno.isPresent()) {
   Cliente cliente = retorno.get();
}
```

ifPresent - Se um valor estiver presente, executa a ação no valor, caso contrário, não faz nada.

```
public void login(String cpf, String senha) {
   dao.buscaPorCPF(cpf).
      ifPresent(cliente -> cliente.realizaLogin(senha));
}
```

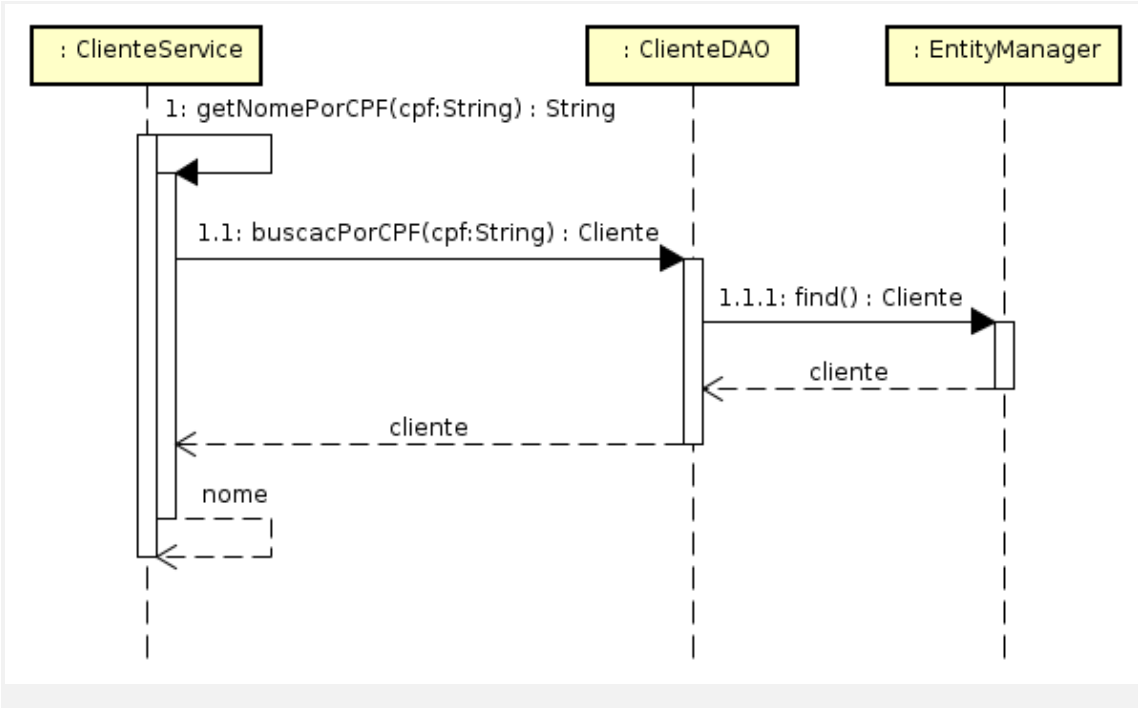
map - Se um valor estiver presente retorna um `Optinal` com o resultado da aplicação da função de mapeamento no valor, caso contrário, retorna um `Optional` vazio.

```
if (optCliente.isPresent()) {
   String nome = optCliente.map(Cliente::getNome);
}
```

flatMap - Semelhante a `map`, se um valor estiver presente, retorna o resultado da aplicação da função de mapeamento no valor, caso contrário retorna um `Optional` vazio. A diferença é que `flatMap` pode se aplicado a um mapeamento que também retorna um `Optional`.

```
Endereco endereco =
   buscaCliente(cpf).flatMap(Cliente::getEndereco).get();
```

Vamos ver alguns cenários de uso.



Classes utilizadas nos exemplos

O método na classe `ClienteDAO` retorna um `Cliente`. O retorno pode ser nulo caso não seja encontrado um cliente com o CPF informado.

```
//Método na classe ClienteDAO
public Cliente buscaPorCPF(String cpf) {
   return busca(cpf);
}
```

Na classe `ClienteService` o método `getNomePorCPF` deve retornar apenas o nome do cliente, mas ele também precisa tratar o cenário onde um cliente não é encontrado
e `ClienteDAO` retorna nulo:

```
//Método na classe ClienteService
public String getNomePorCPF(String cpf) {
   Cliente cliente = dao.buscacPorCPF(cpf);
   if (cliente != null) {
      return cliente.getNome();
   } else {
      return "DESCONHECIDO";
   }
}
```

Com Java 8 o método na classe `ClienteDAO` pode ser alterado para retornar um `Optional<Cliente>`. Como o cliente pode ser nulo vamos usar `Optional.ofNullable` como método de criação.

```
//Método na classe ClienteDAO alterado
public Optional<Cliente> buscaPorCPF(String cpf) {
   return Optional.ofNullable(busca(cpf));
}
```

Agora que `ClienteDAO.buscaPorCPF` retorna um `Optional<Cliente>` a classe `ClienteService` pode utilizar os recursos de `Optional` para tratar o cenário onde um cliente não é encontrado de várias maneiras interessantes.

orElse - Se um valor estiver presente, retorna o valor, caso contrário, retorna o valor definido no parâmetro.

```
//Método na classe ClienteService alterado
public String getNomePorCPF(String cpf) {
   return dao.buscaPorCPF(cpf).map(Cliente::getNome)
      .orElse("DESCONHECIDO");
}
```

orElseGet - Se um valor estiver presente, retorna o valor, caso contrário, retorna o resultado produzido pelo parâmetro.

```
//Método na classe ClienteService alterado
public String getNomePorCPF(String cpf) {
   return dao.buscaPorCPF(cpf).map(Cliente::getNome)
      .orElseGet(Constantes::getNomePadrao);
}
```

orElseThrow - Se um valor estiver presente, retorna o valor, caso contrário, lança a exceção informada no parâmetro.

```
//Método na classe ClienteService alterado
public String getNomePorCPF(String cpf) {
   return dao.buscaPorCPF(cpf).map(Cliente::getNome)
```

```
      .orElseThrow(IllegalArgumentException::new);
}
```

Novidades para Optional no Java 9

ifPresentOrElse - Se um valor estiver presente, executa uma determinada ação, caso contrário, executa uma outra ação.

Veja o código abaixo:

```
//Método na classe ClienteService
public void login(String cpf, String senha) {
   Cliente cliente = dao.buscaPorCPF(cpf);
   if (cliente != null) {
      cliente.realizaLogin(senha);
   } else {
      registraFalhaLogin();
   }
}
```

Com `ifPresentOrElse` o código pode ser escrito assim:

```
//Método na classe ClienteService alterado
public void login(String cpf, String senha) {
   dao.buscaPorCPF(cpf).ifPresentOrElse(cliente ->
      cliente.realizaLogin(senha), () ->
registraFalhaLogin());
}
```

or - Se um valor estiver presente, retorna o valor, caso contrário, executa uma ação. Por exemplo, é possível tentar obter um cliente do cache e no caso do retorno estar vazio tentar obter o cliente do banco de dados.

```
Optional<Cliente> cliente = cache.getCliente(cpf).or(() ->
dao.buscaPorCPF(cpf));
```

stream - Se um valor estiver presente, retorna uma `Stream` contendo apenas esse valor, caso contrário, retorna uma `Stream` vazia.

```
dao.buscaPorCPF(cpf).stream();
```

Para ajudar a entender onde usar `Optional` e onde não usar, siga as sete regras de Stuart Marks:

1. Nunca, nunca, use `null` para uma variável `Optional`ou valor de retorno.
2. Nunca use `Optional.get()` a menos que você possa provar que o `Optional` está presente.
3. Prefira métodos alternativos do que `Optional.isPresent()` e `Optional.get()`.
4. Geralmente é uma má ideia criar um `Optional` para o propósito específico dos métodos de encadeamento e a partir dele obter um valor.
5. Usar um `Optional` com um outro `Optional` aninhado, ou que tenha um resultado intermediário de `Optional`, provavelmente é uma solução muito complexa.
6. Evite usar `Optional` em campos, parâmetros de método e coleções.
7. Não use `Optional` para envolver qualquer tipo de coleção (`List, Set, Map`). Em vez disso, use uma coleção vazia para representar a ausência de valores.