Back Dev. Case

Jorge Stiven Perdomo Serrano

1. Design pattern: Choose one design pattern (Factory, Strategy, etc.) and explain how you've used it in a real-world Java project.

R/

At this point I choose the builder pattern because it is one of the best known and one of the most used, this pattern basically in the java world helps us with the construction of complex objects using a step by step construction and supporting us with the build method that either you create it or you support it with the annotation in Lombok of @builder.

2. Java Streams: How do Java Streams work? Can you give an example of transforming a list of objects into a summary map?

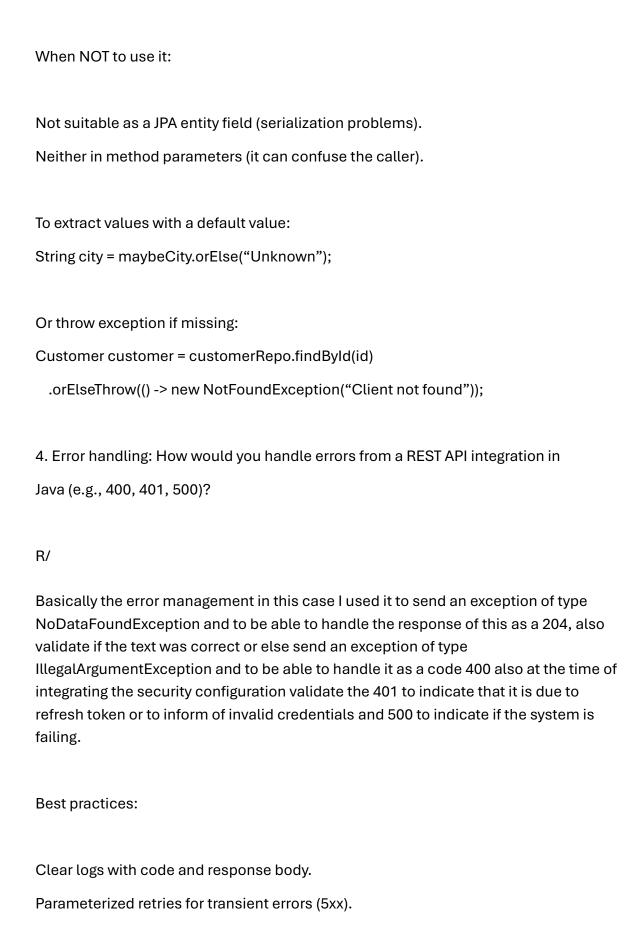
R/

The Streams in java I understand them as internal functions of the code with which the lists can be worked, as for example some cases when using the filter, the map, the foreach and many other things that these functions offer us.

An example is to transform an Order list in a map that groups the total invoiced by client.

List<Order> orders = getOrders();

```
Map<String, BigDecimal> totalByCustomer = orders.stream()
  .collect(Collectors.groupingBy(
   Order::getCustomer,
   Collectors.mapping(Order::getAmount, Collectors.reducing(BigDecimal.ZERO,
BigDecimal::add))
 ));
groupingBy groups by customer name.
mapping extracts the amount of each order.
reducing sums the amounts per customer.
3. Handling nulls and Optional: How do you use Optional effectively? When is
it appropriate?
R/
Effective use of Optional:
Serves to wrap a value that may or may not be present, avoiding NullPointerException.
Appropriate situations:
In the signature of methods that search in database: Optional<User> findByld(Long
id).
When chaining safe transformations:
Optional<String> maybeCity = optionalClient
 .map(Customer::getAddress)
  .map(Address::getCity);
```



Circuit breakers (e.g. Resilience4j) to avoid overloading downed systems.

Timeouts and fallback if the API takes too long.