Knowledge and questions for backender candidates

Backend senior greenfield project

Knowledge (bold lines are mandatory)

- **Ability to express complex ideas about previousprojects**
- **Commitment, abilityandaptitude forteamwork**
- **SOLID principles and clean code**
- **Experience with Java8**
- **Strong knowledge Unit testing (JUnit,Mockito)**
- Static code analysis tools (Sonar, PMD, Checkstyle,etc)
- **Design patterns**
- **TDD**
- Knowledge of operating systems(Linux)
- **Git** and shellbash
- **Agile methodologies**
- **Spring, Maven**,Gradle
- Microservice architecture (service discovery, distributed traceability, APIscomposition,
...)

Nice to have

- DDD
- Side-projects
- Jenkins, Ansible,Docker
- Experience with Akka, SpringBoot
- Experience with relational databases and NoSQL

Questions

- **What's new in Java 8? Explain some of them.**

Most important features:

- Use of streams for data management and filtering operations, ordering, etc.
- Default and static methods in interfaces.
- Lambda functions that work as anonymous and can be used in any class or method.

- Functional Interfaces.
- Optional class for handling null values,
- CompletableFuture class for handling asynchronous operations.
- New time API.

· **Given the following list implement a solution in order to get even numbers using**
Java8 Streams

```
List<Integer> list = Arrays.asList(1,2,3,4);
list.stream().filter(number -> number % 2 == 0).forEach(even -> System.out.println(even));
```

· **What do you notice when you do codereview?**
- Clear and legible structure.
- Nomenclature of variables and methods.
- Compliance with quality rules.
- Test coverage.

· **Have you ever worked with Scrum? Tell us what it is, what events do you remember and what roles are involved?**

Yes, in many projects. Scrum is a software project development control methodology based on roles and intermediate tasks that increase functionality to the full functionality of the product.

The roles are:
- Product Owner
- Scrum Master
- Stakeholder
- Development Team

The events are:
- Daily
- Sprint review
- Sprint planning
- Sprint retrospective

· **What access modifiers (or visibility) do you know in Java?**
- Private: This modifier will only allow us to access from the same class.
- Protected: It allows us to access the properties of the parent class, when we are making inheritances. Outside of child classes it is equivalent to private.
- Default: Allows both the class itself and the classes in the same package.
- Public: Allows access from any class or package.

· **Differences between an abstract class and an interface. When would you use one or the other?**

The abstract class has to contain at least one abstract method and can contain the method implementations. The interface only defines the structure of the method without implementations. A new class can extend an abstract class and implement an interface.

Abstract classes are used when we want to define a common functionality for objects of the same type, and an interface when that functionality is common to objects of different types

· **What is Maven and why is it used? What is Maven lifecycle?**

Maven is a dependency manager used for building projects. Lifecycle refers to each of the stages that run in sequence from validation to final deployment.

· **What is Git and what is it used for? List all Git commands that youknow.**

Git es un controlador de versionado de software. Aunque suelo utilizar un frontal de Git como GitKraken, desde consola suelo utilizar también estos comandos:

- git init
- git add
- git status
- git commit
- git clone
- git remote add
- git fetch origin
- git pull origin
- git push origin

· **What is a mock? What would you use it for?**

A mock is an object used so that the tests do not have external dependencies, it is a kind of substitute that is used to create objects, servers, databases, which are needed for the tests, but have no real functionality.

· **How would you explain to someone what Spring is? What can it bring to Their projects?**

Spring is a framework used to support the java applications that are developed. It provides a structure and tools necessary for communication between modules as well as links with external dependencies.

· **What's the difference between Spring and SpringBoot?**

Spring Boot is one of the parts of Spring that facilitates the automatic creation and configuration of applications under Spring, facilitating their configuration and deployment.

- **Do you know what CQRS is? And Event Sourcing?**

CQRS is a system in which the parts corresponding to commands and queries are separated, with the aim of decoupling the responsibility for managing processes and data, and they are synchronized with each other.
Event sourcing refers to an event-based system in which all application changes are stored as a series of events ordered and that can be consumed by a client.

- **Differences between IaaS and PaaS. Do you know any of eachtype?**

IaaS is the infrastructure part and belongs to the Architecture part, while PaaS is the part dedicated to application development and deployment. Example of IAAS are GCP, AWS. PaaS are Heroku, OpenShift.

- **Explain what a Service Mesh is? Do you have an example?**

He is in charge of communication, load balancing, circuit breaker, authentication and securization between microservices in cloud. An example is Istio.

- **Explain what is TDD? What is triangulation?**

TDD means test-oriented design, that is, first implement the tests that the application has to meet and then implement the necessary code to pass those tests.
Triangulation is the process by which cases are implemented from the simplest to the most complex in cases where the algorithm to use is not clear and approximations have to be made.

- **Apply the Factory pattern with lambda expressions**

```
public class CarFactory {
   private CarFactory() {
   }
   private static final Map<String, CarCreator> CAR_FACTORY_MAP =
ImmutableMap.of(
         "Tourism", brand -> new TourismCar(brand),
         "SUV", brand -> new ESuvCar(brand),
         "Sport", brand -> new SportCar(brand),
         "4x4", brand -> new 4x4Car(brand)
   );

   static Car getCar(String type, String brand) {
      if (CAR_FACTORY_MAP.containsKey(type)) {
         return CAR_FACTORY_MAP.get(type).initialize(brand);
      }
      throw new UnsupportedOperationException();
   }

   private interface CarCreator {
      Car initialize(String brand);
   }
}
```

· **Reduce the 3 classes (*OldWayPaymentStrategy*, *CashPaymentStrategy* and *CreditCardStrategy*) into a single class (*PaymentStrategy*). You do not need to créate any more classes or interfaces. Also, tell me how you would use *PaymentStrategy*, i.e. the different payment strategies in the Mainclass**

```
public interface OldWayPaymentStrategy {
double pay(double amount);
}
public class CashPaymentStrategy implements OldWayPaymentStrategy {
@Override
public double pay(double amount) {
double serviceCharge = 5.00;
return amount + serviceCharge;
}
}
public class CreditCardStrategy implements OldWayPaymentStrategy {
@Override
public double pay(double amount) {
double serviceCharge = 5.00;
double creditCardFee = 10.00;
return amount + serviceCharge + creditCardFee;
}
}
public interface PaymentStrategy {
double serviceCharge = 5.00;
double creditCardFee = 10.00;
double cashPaymentStrategy(double amount);
double creditCardStrategy(double amount);
}

public class Main {
public static void main(String[] args) {
PaymentStrategy payment = new PaymentStrategy() {

@Override
public double cashPaymentStrategy(double amount) {
        return amount + serviceCharge;
}
@Override
public double creditCardStrategy(double amount) {
        return amount + serviceCharge + creditCardFee;
}
};
double payCash = payment.cashPaymentStrategy(2.5);
double payCard = payment.creditCardStrategy(2.5);
}
}
```