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
- **Agilemethodologies**
- **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 ofthem.**
  Se incluyeron:
  - La implementación de JavaFX.
  - Modificación en interfaces para poder hacer métodos estáticos y asi tener métodos como los que tendríamos en una clase normal y no tener que sobrescribir. También podemos hacer métodos default que funcionan como los estáticos pero si lo deseamos donde implementemos la interface los podremos sobrescribir.
  -Implementación de Stream para facilitar el realizar una misma acción a un conjunto de datos bajo un mismo flujo como puede ser: modificar todos los valores de una lista incrementando en 1, o realizar algún filtro según alguna condición.
  -Implementación de Opctional<T>: de esta conozco su existencia pero nunca lo he llegado a usar realmente. Simplemente se que uno de sus usos es para evitar NullPointerException.
  -Implementación de expresiones lambdas para hacer uso de funciones anónimas.

-Incorporación de nuevas API para manejas fechas y tiempo.

- **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);
Stream<Integer> par = list.stream().filter(number -> number % 2 == 0);
par.forEach(n -> System.out.println(n));
```

- **What do you notice when you do codereview?**

  Cuando reviso un código a lo que atiendo es a como estar estructurado y que puedo aprender que a mi no se me había ocurrido. Hago por entenderlo todo y analizar si esta de la forma mas legible posible, o si tiene una estructura que no conocía aprendo de ella y veo que es una forma mas optima de trabajo.
  También atiendo a la ortografía para pillar errores que al programador que lo escribio se le hayan podido pasar y así que sea más robusto.

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

  Si, actualmente trabajo con bajo el flujo que define Scrum. Es un conjunto de buenas practicas para que haya un mejor flujo de trabajo entre equipos técnicos y equipos funcionales y conseguir un producto final mas robusto.
  Algunas terminologías de Scrum pueden ser:
  -Sprints: que son periodos de tiempo de entre 2 o 3 semanas en los que se planifica que tareas se acometerán en ese tiempo. Esto se defini previamente a comenzar dicho periodo bajo un Sprint Backlog.
  -Daily: Son reuniones diarias muy cortas al principio de la jornada para hacer saber a todo el equipo en que va a trabajar esa persona durante el día y conocer el estado de las tareas todos,
  -Review: Es una reunión al final de sprint para ver, sobre lo estimado, que tareas se han terminado y en que estado han quedado.
  -Historia de usuario: Es la mínima funcionalidad que se le entrega a un cliente y debe funcionar por si misma de forma independiente y completa.
  Con respecto a los roles los mas importantes serian:
  -Product Owner: Que se encarga de que los equipos trabajan de forma adecuada.
  -Scrum Master: se encarga de llevar los equipos técnicos y funcionales bajo un ritmo de trabajo conociendo bien ambos campos y facilitando la comunicación entre dichos equipos.

- **What access modifiers (or visibility) do you know in Java?**

  public: que hace accesible a la propiedad desde donde se desee instanciar.
  protected: que la propiedad es accesible desde la misma clase mismo paquete y las subclases.
  private: cuya variable es accesible unicamente desde la misma clase.
  default: este valor se pone por defecto si no se especifica modificador de acceso. Se tiene acceso a la propiedad desde la misma clase y desde el mismo paquete.

- **Differences between an abstract class and an interface. When would you use one ortheother?**

  -Las clases abstractas se heredan y las interfaces se implementan.

-Las clases abstractas pueden implementar interfaces o heredad de otras clases sean abstractas o no, y las interfaces solo pueden implementar otras interfaces.
-Las clases abstractas pueden contener métodos abstractos o métodos normales y las interfaces no.
-En las interfaces puede definir parte del cuerpo del método que se sobrescribiera con la palabra default en el método, en las clases abstractas no
-En las clases abstractas, como son herencias, se puede usar el modificador protected.

- **What is Maven and why is it used? What is Maven lifecycle?**
  Maven es una herramienta de java para, mediante un archivo xml llamado pom poder gestionar el proyecto, sus dependencias y la construcción del mismo. Es muy útil para implementación de librerías externas de una forma muy agil.
- **What is Git and what is it used for? List all Git commands that youknow.**
  Git es un controlador de versiones de proyectos . Se usa para gestionar las diferentes versiones de nuestro proyecto y dividir las tareas a acometer en diferentes ramas para trabajar sobre nuevas funcionalidades de un proyecto con la posibildad de volver al estado previo de empezar para no romper nada en caso de no desempeñar bien la tarea.
  Los comandos que conozco son: git commit (se puede agregar -m para un comentario), git push, git pull, git merge, git clone, git add(seguido del archivo a incluir), git rm(seguido del archivo a borrar), git branch(si se escribe un nombrep ara la rama se crea la rama, si no te dara al lista de las ramas de tu repositorio local y si se especifica -d la borraras), git checkout(seguido del ronmbre de la rama, si se especifica -b te creara la rama y te llevara a ella).
  A git rm se le puede agregar --cache para que no te borre el archivo de tu repositorio local.
- **What is a mock? What would you use itfor?**
  Un mock es una simulación de un objeto para burlar algunas partes de la función que deseamos testear. Con esto nos ahorraremos instanciar objetos reales y depender de otras librerías cada vez que pasemos nuestros test unitarios, ya que un test unitario se debe probar exclusivamente a si mismo sin depender de nada externo. Con estos objetos simulados tambien podremos definir su comportamiento para que nos devuelva otros objetos que definamos o incluso saber cuantas veces se ha llamado a ese objeto (o a una funcion deese objeto) en la función que testeamos.
  He trabajado con mocks mediante Mockito para Java y unittest.mock para Python3
- **How would you explain to someone what Spring is? What can it bring to theirprojects?**
  Spring es un framework de Java para la realización de proyecto de aplicaciones web de una forma ágil.
  Aconsejo usar Spring para la creación de aplicaciones web ya que te permite la creación y gestión del proyecto de una forma rápida y mas sencilla pudiéndote centrar mas en lo que es el desarrollo de la aplicación. Spring ademas hace mas sencillo e intuitivo el desarrollo, sobre todo el partado dela creación de su API Rest. Ademas incluye herramientas para dar soporte también en persistencia de datos con spring data, seguridad con spring security
- **What's the difference between Spring and SpringBoot?**
  Spring es un framework que abarca todos los campos necesarios para el desarrollo de aplicaciones web, y Spring Boot es una de las partes de spring que esta mas encargada de la creación de la API Rest y de facilitarte la creación y gestión del proyecto y la subida al servidor para ponerlo en producción, haciendo que te centres principalmente en el desarrollo de la aplicación en si.
  Ademas Spring Boot ya tiene un servidor envevido para probar tu aplicación de forma local que es Apache Tomcat y un sitio web para la inicialización de tu proyecto con

todo configurado y las dependencias que asignes ya incluidas el cual luego te importar a tu IDE.

- **Do you know what CQRS is? And EventSourcing?**
  No conozco demasiado de estos patrones. Simplemente se que están enfocados a modularizar una aplicación dividiendola en partes para hacer mas sencillo modificaciones posteriores y su mantenimineto.
- **Differences between IaaS and PaaS. Do you know any of eachtype?**
  No conozco demasiado de esto. Diria que la diferencias es que Iaas se centra mas en la infrestructura y Paas en el desarrollo de la aplicación.

- **Explain what a Service Mesh is? Do you have anexample?**
  Esta pregunta no sabria responderla.

- **Explain what is TDD? What istriangulation?**
  Es una forma de trabajo en la que antes de desarrollar una funcionalidad, se realizan los test de la misma para conseguir como objetivo una vez los test definidos, que la funcionalidad pase los test y así asegurarnos que se ha cumplido como deseábamos y nos da robustez en el desarrollo. Una vez hecho esto se pasa a refactorizar el código creado.

- **Apply the Factory pattern with lambdaexpressions**
  Teniendo un ejemplo con un interface Instrument y 3 clases: Cuerda, Percusion, y Viento que implementan Instrument, la clase de la Factoria quedaria asi:

```
class FactoryInstrument{

    enum TypeInstrument{
          cuerda,
          percusion,
          viento
    }

    public Instrument getInstrument(TypeInstrument instrument) {
          if(instrument == TypeInstrument.cuerda) {
                return Cuerda::new;
          }else if(instrument == TypeInstrument.percusion) {
                return Percusion::new;
          }
          else if(instrument == TypeInstrument.viento) {
                return Viento::new;
          }else {
                return null;
          }
    }

}
```

- **Reduce the 3 classes (*OldWayPaymentStrategy*,*CashPaymentStrategy*and*CreditCardStrategy*) into a single class (*PaymentStrategy*). You do not need to createany 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 {
  //write here your solution

 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);

    }
}
```