Questions

- **What's new in Java8? Explain some of then.**

La inclusión de las expresiones lambda, la inclusión de los Streams, la posibilidad de implementar default methods para las interfaces y las interfaces funcionales.

Las interfaces funcionales son interfaces que incluyen un único abstract method (a parte del resto de métodos que puedan tener). Se han definido algunas en java.util como Predicate o Function, que tienen sus métodos por defecto definidos.

Las expresiones lambda son funciones que implementan el abstract method de una interfaz funcional. Pueden implementarse a partir de cualquier interfaz funcional pero resultan muy útiles para algunos nuevos métodos de Collections o de los Streams que esperan como parámetro una de las interfaces funcionales definidas en java.util.

- **Given the following list implement a solution in order to get even numbers using Java 8 Streams**
  List<Integer> list = Arrays.asList(1,2,3,4);

```
List<Integer> solutionList = list.stream().filter(l -> l%2==0).collect(Collectors.toList());
```

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

Nunca he trabajado con Scrum 100% aplicado pero hace unos meses obtuve la certificación PSMI y estoy deseando poder usar los conocimientos obtenidos.

Scrum es un framework de trabajo en equipo que presenta herramientas para la mejora continua del equipo y el trabajo obtenido.

Los eventos son Sprint Planning, donde participan Product Owner, Scrum Master y Development Team; Daily Meeting, para el Development Team; Sprint Review, donde participan obligatoriamente Product Owner, Scrum Master, Development Team y existe la posibilidad de invitar stakeholders; y Sprint Retrospective para Scrum Master y Development Team.

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

Public, private y static.

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

Mientras que una clase puede implementar varias interfaces diferentes, solo puede extender de una única clase abstracta. En mi opinión, las clases abstractas son más bien bases para la construcción de otras clases y las interfaces son más sitios de los que tomar herramientas (o métodos) según los necesitemos.

- **What is Maven and why is it used? What is Maven life cycle?**

Maven es una herramienta que ayuda con la compilación y despliegue de proyectos java. Permite configurar la compilación para diferentes entornos y administrar con facilidad los plugins.

El life cycle es compilación, test, validación y despliegue.

- **What is Git and what is it used for? List all Git commands that you know.**

Git es un sistema de control de versiones sobre repositorios.

Casi siempre lo he usado mediante interfaces gráficas como SmartGit o GitKraken, pero en consola recuerdo los comandos add, commit (con la opción -m para incluir mensajes), push, mesh, merge y pull.

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

No he escuchado hablar de ello hasta ahora, pero no tengo ningún problema en aprender cosas nuevas.

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

Spring es un framework que facilita enormemente la conectividad de un servicio web con su lógica de negocio y sus bases de datos. Está orientado a un modelo vista-controlador con conectividad REST y es muy útil en el desarrollo de microservicios.

Con Spring se ahorra muchísimo tiempo en el desarrollo de las comunicaciones, cambiando mucha implementación y depuración por unas pocas anotaciones.

- **What's the difference between Spring and Spring Boot?**

Mientras que Spring no deja de ser solo un framework, Spring Boot te levanta un servidor tomcat propio que facilita las pruebas sobre tu aplicación.

- **Do you know what CQRS is? And Event Sourcing?**
- **Differences between IaaS and PaaS. Do you know any of each type?**
- **Explain what a Service Mesh is? Do you have an example?**
- **Explain what is TDD? What is triangulation?**

Desconozco todos estos conceptos, pero les echaré un vistazo a todos.

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

```
import java.util.function.Function;
```

```
public class PaymentStrategy {
   private static double serviceCharge = 5.00;
   private static double creditCardFee = 10.00;

   private static Function<Double, Double> cashPaymentStrategy = (amount) -> amount + serviceCharge;
   private static Function<Double, Double> creditCardStrategy = (amount) -> amount + serviceCharge +
creditCardFee;

   public static double calculatePayment(Double amount, Function<Double, Double> strategy){
      return strategy.apply(amount);
   }


   public static void main(String[] args) {

      double payment = 5.00;
      double cashPayment = calculatePayment(payment, cashPaymentStrategy);
      double cardPayment = calculatePayment(payment, creditCardStrategy);

   }


}
```