Questions

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

Lambdas – Easy creation of anonymous functions.
Stream API – Mainly, it allow some fluent API for collections , sometime in a functional way.
Default and static methods in interface – Some kind of multiple inheritance.
New Java classes for date and time – Improvements over java.util.Date
Optional – In some cases we can manage null values.

• 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);
```

```
import java.util.Arrays;
import java.util.List;

import static java.util.stream.Collectors.toList;

public class EvenNumbers {
    public static void main(String[] args) {
        List<Integer> list = Arrays.asList(1, 2, 3, 4);

        List<Integer> result = list.stream()
                .filter(i -> i % 2 == 0)
                .collect(toList());

        System.out.println("result = " + result);
    }
}
```

• What do you notice when you do code review?

Gain from improving code quality, maintainability and readability.

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

Yes, we follow this metodology since 2016. It is a way working and team selforganization for deliver value to the business.
Common events are: Sprint, daily meetings, plannings, retrospective, refinament.
I think there are 3 main roles:
 Product owner – They define which tasks to do and its priority.
Scrum master - It is a facilitator, it solves problems that are not within the sprint tasks
Engineers – They try to complete the sprint tasks and make an estimation of task duration.

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

Default, I mean, no specific modifier (package private) – Only classes of the same package can access to this member.
private – Only member that are inside the class can access to this member.
protected – Only subclasses of the class can access to this member.
public – Other classes can access to this member.

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

An abstract class is a class where some methods are abstract, i.e., they aren't defined.
An interface is a class where all its methods are abstract, static of default.
Depending on the problem, we can use abstract classes of interface.
If we want to force that some classes implement some methods and anything more we should use interfaces.
On the other hand, if we want that some classes share behaviour and implement partial methods we should use abstract classes.

• What is Maven and why is it used? What is Maven life cycle?

Maven is a build tool to compile, run, test, package, install, etc. programs in the JVM.
Each Maven life cycle is a group of phases that run together. For example: test life cycle implies compile test, copy classes to target directory, run test, etc.

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

Git is a VCS (Version Control System), it allows to handle diferent versions of source code files and integrate changes those versions.

Commands list:
git clone
git pull
git status
git add
git commit
git push
git log

• What is a mock? What would you use it for?

A mock is a type of test double, it used to replace a dependency in an unit test.

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

Spring is a framework for applications development. It facilitates the development and organization of these applications.
Main feature of this framework is depencency injection.

• What's the difference between Spring and Spring Boot?

Spring Boot gives you Spring plus a series of additonal technologies that make eaiser applications development.
Most dependencies are imported inside de different starters that Spring Boot has.

You don’t need deploy a application in a application/container server (Tomcat for example), it is embeded as a dependency.

• Do you know what CQRS is? And Event Sourcing?

Yes I know a little about these architectural design patterns, but I have never implemented a system with those designs.

• Differences between IaaS and PaaS. Do you know any of each type?

IaaS (Infrastructure as a Service), you get a virtual machine with basic configuration.
For example, AWS EC2
PaaS (Platform as a Service), you get a virtual machine ready to install or create your application.
For example, Google App Engine

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

I think that is a set of independent components that help in microservice development.
For example Consul discovery service.
I know very little about this.

• Explain what is TDD? What is triangulation?

A development technique, first you create the test and then the code that makes the test pass and iteratively,
 you create tests and refactor so that all the tests pass.
I don’t know what triangulation is, but I would like learn it.

• Apply the Factory pattern with lambda expressions

I am not sure how to do that. Maybe something like this:

```
Function<Integer, Function<Integer, Integer>> add = x -> y -> x + y;

Function<Integer, Integer> add2 = add.apply(2);

System.out.println(add2.apply(4));
System.out.println(add2.apply(8));

Function<Integer, Integer> add3 = add.apply(3);

System.out.println(add3.apply(1));
System.out.println(add3.apply(5));
```

• 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

```
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 {
    static double payByCash(double amount) {
        double serviceCharge = 5.00;
        return amount + serviceCharge;
    }

    static double payByCreditCard(double amount) {
        double serviceCharge = 5.00;
        double creditCardFee = 10.00;
        return amount + serviceCharge + creditCardFee;
    }
}

public class Main {
    public static void main(String[] args) {
        double paymentByCash = PaymentStrategy.payByCash(18);
        System.out.println("paymentByCash = " + paymentByCash);

        double payByCreditCard = PaymentStrategy.payByCreditCard(5);
        System.out.println("payByCreditCard = " + payByCreditCard);
    }
}
```