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

- Stream collections. They consist on a flow of data, a series of operations and a final operation to produce a result: Sum, count, list, etc.
  They are a new way of going through a list of elements and apply some operations to it: filter, map, calculate average, min, max, etc.
- Lambda expressions.
  The lambda expressions allow to start programming in java like we would in a functional language. They are functions with no name associated to it and that can be passed as arguments to other functions.
  In my opinion, they are the most disruptive change introduced in java 8.
- Reference methods.
  By using Reference methods, we can use already implemented methods as lambda expressions.
  We can do it with static methods, constructors or even methods of an instance.
- Functional Interfaces.
  They are the perfect complement to be used with lambda expressions. They are interfaces with only one abstract method.
  By combining functional interfaces with lambda expressions, we can reduce code, and make it more readable. There are already defined functional interfaces on java like Runnable, Comparator, etc.
- Default methods on interfaces
  It is an approach to multiple inheritance in java. It allows to implement some methods on interfaces.
- Optional objects
  By defining Optional objects we can avoid Nullpointers exceptions. They allow to know if an object is available or not.
- Type-based annotations
- Changes in the Date API

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.stream().filter(n->n%2==1).collect(Collectors.toList()
```

What do you notice when you do code review?

- Security issues. I try to detect possible errors on the code. Possible exceptions not handled, methods with design flaws, etc.
- Good error handling. I do not like the code that assumes there is no errors.
- Duplicate code. I hate copy&paste of several lines of code with only little changes. Refactor!
- Unreachable code. There is legacy code with hundreds of lines that are not used anymore.

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

I have worked with agile methodologies, adapted to our particular business.
Events: Planning, Daily meeting, Sprint, Demo Spring meeting
Roles: Product owner, Scrum manager and developers

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

- private. Only class access.
- default (no modifier specified). Class and package access
- protected. Class, package and inheritance access.
- public. Everyone can access.

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

Both of them are the key concept for polymorphism, but with some differences.

- **Interfaces**. An interface is used to specify some methods all implementing classes must implement. Originally, interfaces were nor supposed to contain any code implementation, but it changed with default methods on Java 8.
- **Abstract classes**. An abstract class define and codify a default behaviour for all implementing classes. An abstract class can also define abstract methods forcing all implementing classes to implement them. Those abstract methods can be used in common code to achieve different forms to do the same task.

With Java 8 and default methods on interfaces, interfaces and abstract classes are more alike than before, but a class can implement several interfaces and extend only one abstract class. This could be considered as an approach to multiple inheritance.

I would use abstract classes to define default behaviors with a common implementation having in mind that superclasses should redefined some of this behaviors.

I would use an interface to define a set of behaviors that can be implemented in different ways. An API is a clear example.

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

Maven is a software tool to automatize constructions of software projects.
Maven also allow us to handle project dependencies.

Maven life cycle are al the phases defined to be executed in order to build an artifact.
Basically: clean, compile, test, package, install and deploy

When we define the build of an artifact on the pom file, we can intervene in that phases by referencing some goals of that phases and specifying what to do with plugins.

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

Git is a software for version control. It is used to have control of all the historic changes the files have been through.

I haven't used git, I have used SVN and always through GUI, but the concept is similar except for the push/pull concepts.

Operations I recall from SVN that I suppose are similar in git: add, delete, diff, commit, branch, tag, checkout, merge, list, etc.

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

A Mock is an object that is used to supply certain functionality that we want to be certain of.

Mocks are used in testing to assure the desired behavior of some code (results, method invocation, etc.).

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

Spring is a development framework with a lot of features and modules to make developers lives easier.

The main concepts behind Spring are Inversion of Control (IoC) and Dependency injection (DI).

- With IoC, the control of instantiated objects is transferred to a container (Context). This allows to decouple the code, make it easy to modularize a project and allows to switch between different implementations.
- Dependency injection is the pattern that allow to implement IoC. The objects that are handled by the context can be injected on other beans.

But there is a lot more to that. There are a lot of implemented modules ready to be used and adapted to particular needs: security, data, transactions, mvc, etc.

What's the difference between Spring and Spring Boot?

Spring is the framework designed to helps the developers build applications. It gives us a million tools and a lot of flexibility.

Spring Boot allows developers to create applications using Spring very fast and with lesser code lines.

Do you know what CQRS is? And Event Sourcing?

I know CQRS and event sourcing are arquitectural patterns, but I do not know much about them.

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

IaaS stands for Infrastructure as a service. A third party, for example: Amazon Web Services, Microsoft Azure, offers infrastructures easily automatized and scalable as a service to business to deploy their applications, services. It can be summarized as a renting of infrastructures (machines, storage, computation, etc.) where the client only pays for what they use.

Paas stands for Platform as a Service. In this case, the providers (AWS and Azure again, but some providing services) provides services and applications to developers to build their own applications without having to be aware of infrastructures (memory, storage, etc.)

The main difference between them consist on who is in charge of security and maintenance. In Iaas, the business that rents the infrastructure is the one in charge of secure and maintain the platform, while in PaaS it is something the service provider does.

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

I do not know the concept.

Explain what is TDD? What is triangulation?

TDD stands for Test Driven Development. It is one way to develop software where the test are written at the beginning thinking on the requirements. Requirements must be broken into very specific test cases. Then, we write the code to pass the test and then the test must be improved and refactor. This two things are repeated in a loop until we are satisfied by the final result.

I am not vey familiar with the term triangulation.

Apply the Factory pattern with lambda expressions

```
public class FactoryPatternExample {
	public static void main(String[] args) {
		ComputerLambdaFactory factory = new ComputerLambdaFactory();
		System.out.println(factory.getComputer(ComputerType.DESKTOP));
		System.out.println(factory.getComputer(ComputerType.LAPTOP));
	}
}

public class ComputerLambdaFactory {

final static Map<ComputerType, Supplier<Computer>> map = new HashMap<>();
	static {
		map.put(ComputerType.LAPTOP, ()->new Laptop());
		map.put(ComputerType.DESKTOP, Desktop::new);
	}
	public Computer getComputer(ComputerType computerType) {
		Supplier<Computer> computer = map.get(computerType);
		if (computer != null) {
			return computer.get();
		}else {
			return null;
		}
	}
}
```

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 {
	public double pay(double amount);
}
public class Main {
	public static void main(String[] args) {
	   double amount = 15.0;
	   double serviceFee = 4.00;
	   double creditCardFee = 10.00;
	   PaymentStrategy cashPaymentStrategy = (a) -> a + serviceFee;
	   System.out.println("CPS: " + cashPaymentStrategy.pay(amount));
	   PaymentStrategy ccPaymentStrategy = (a) -> (a + serviceFee + creditCardFee);
	   System.out.println("CCPS: " + ccPaymentStrategy.pay(amount));
	}
}
```