**Laura Alonso**

**Questions**

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

What comes to my mind is:

* Lambda expressions for functional interfaces (interfaces with only one method) and method references (way to use a lambda expression in a more easy to read way)
* Streams: allow you to work with Collections making bulk operations among other things.
* Optional class: class that simplifies the code as you dont need to check for references that are null.
* Static and default methods in interfaces: you can now add implemented methods to an interface that allow you to add new functionality to interfaces and ensure compatibility with older implementations of that interface. While default methods can be overriden, static methods can´t.
* Improvements in concurrency
* New date-time classes.

• 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> listEven = list.stream().filter(x -> ((Integer)(x%2)).equals(0))).collect(Collectors.toList());

• What do you notice when you do code review?

I learn a lot, sometimes what to do, and other times what not to do.

I notice how important is to comment and document code.

I see how using self explaining names in your classes, methods, variables… facilitates the reading.

I spot when the description of a method or corner case doesn´t match with the actual code.

I have also realized that with code review silly mistakes are spoted easily, and though it might seem like you are loosing time, you are indeed not, as it saves you lots of time in the future in case of easy to spot bugs.

• 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 Scrum, which is an agile methodology that facilitates developing software without having everything especified from the very beginning. I´d say it is a ‘customer friendly’ methodology as it facilitates sharing progress with the client and the inclusion of changes and its also ‘developer friendly´as it involves the developer in the planning.

I like to work with scrum because it involves the whole team as everyone agrees-on and works together towards the same goal.

What we used to have while applying scrum was:

* **spring planning**: at the beginning of the sprint to plan what to be done, for how long…
* **daily standups**: daily short meetings to share progress and support each other in case of locks.
* **demo/sprint review with customer:** at the end of the sprint to share the product and listen to feedback.
* **sprint reviews with the team/sprint retrospective**: at the end of the sprint to review last sprint, see what to be improved and what to keep.

The roles involved… I would say the team, a scrum master within the team, stakeholders and customer or product owner.

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

* **Private:** only to be seen in the same class
* **Public:** you can access it from anywhere
* **Protected:** only to be seen in the same class, its subclass or package.
* **Default**: you can access from any class in the package

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

No constructors in an interface (you can´t instantiate it).

All fields in an interface are public, static and final.

An interface can only be public whereas you can use access modifiers in an abstract class.

While a class can extend only one abstract class, it can implement several interfaces.

In general, I prefer interfaces as they work well in Spring with dependency injection.

I consider abstract classes when I need to restrict access and when I need non-final or non-static fields.

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

Maven is a software that helps you manage a project.

It is used to make the life of a developer easier and ensures everyone in the team has the same environment. I especially like how it simplifies the build-up process by automatically downloading all the libraries and dependencies needed and I also like the project folder structure that it creates.

I would say the Maven life cycle is the different steps or phases that occur while building or deploying a Maven-project. Some of the steps that I do know are validation, get resources, compilation, package, test and deployment. You can also clean the project and get documentation.

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

Git is an open source version control software that is used for software development. It is ideal for distributed teams as each member can have a copy of the master repo and their own repo, updating their local repo and uploading their code to different branches.

It is used for managing the different versions of your code so a Good practice is that the developer saves the code each time he has a new working functionality, so you can always go back to that point in case something doesn´t work in the next steps. It is also used to keep track of the changes as the labels help you identify what was made in relation to that label. It also facilitates the integration of the code developed by different people, checking for incompatibilities.

Git commands that I know are: git… init, clone, commit, pull, push, merge, add, status, diff. Though I usually use Git through IDE or Sourcetree and not with command line.

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

A mock is a simulation of something.

I use it for testing purposes.

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

Spring is a framework that makes the life of the developer easier as it removes a lot of boiler plate code and provides the dependency injection, eliminating complexity from the developer so that he can focus on the business logic. It also facilitates integration with other tools.

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

Spring Boot is an addition to Spring and focusses on building the Project structure and the dependencies needed based on a selection made by the developer/architect. You can get to the same point using Spring and SpringBoot but Spring boot acelerates the creation and configuration of the Project.

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

Even though I don’t know what CQRS, I will be happy to learn and apply it, as I have done in previous projects where I came across new software that needed to be used.

I am familiar with what Event Sourcing is as I have worked in a project where we where building an ERP and had a need for auditing. We made some research and came across Event Sourcing. Even though, I have never used it because COVID19 made the customer to freeze the project.

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

In PaaS the service provided is ‘wider’, so the customer can manage less things by themselves. It allows the customer to focus only on the development of applications.

As IaaS provides ´less´ services the customer needs to do more work, but has more flexibility.

An example of IaaS is AWS though, it could also be seen as PaaS. I don’t know any examples of PaaS.

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

I don’t know what a Service Mesh is and do not have an example. Even though, as with CQRS, I will be happy to learn it and add it to my box of tools.

• Explain what is TDD? What is triangulation?

TDD is test driven development which means that you start writing your tests first and then your code. In this way you first have to think on what an outcome of your code must be, and write a test that validates that outcome. Once you have your test, you code and check if the code passes the test, if so, you continue to add tests and functionality, always adjusting (refactoring) your code and making sure it passes your tests.

I think that triangulation is a technique in TDD but I have never used it.

• Apply the Factory pattern with lambda expressions

• 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 pay (String paymentType, double amount) {

switch (paymentType) {

case "Cash": return PaymentStrategy.CashPayment(amount);

case "Card": return PaymentStrategy.CreditCardPayment(amount);

default: throw new IllegalArgumentException ("Payment method not supported");

}

}

static double CashPayment(double amount) {

double serviceCharge = 5.00;

return amount + serviceCharge;

}

static double CreditCardPayment(double amount) {

double serviceCharge = 5.00;

double creditCardFee = 10.00;

return amount + serviceCharge + creditCardFee;

}

}

public class Main {

public static void main(String[] args) {

PaymentStrategy.pay("Cash", 1000));

PaymentStrategy.pay("Card", 1000));

}

}