Questions

Cristhian Xavier Urcuyo Rivera

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

- Lambda Expressions

  Give the ability to pass a functionality as a method argument. One application would be to create anonymous classes of functional interfaces.

- Streams and Pipelines

  Stream is used to propagate elements from a source through a pipeline. It is a sequence of elements. Additional to that, Streams provide the utility to perform parallel Operations.

- Default Methods

  Before Java 8, Interfaces could only contain abstract methods. But in Java 8 Default methods were introduced to give interfaces the ability to contains non-abstract methods.

- Functional Interfaces

  An interface with exactly one abstract method Is called a Functional Interface, it usually contains a @FunctionalInterface annotation.

- Date and Time API

  It contains a best list of utility methods used for regular operations.

* **Given the following list implement a solution in order to get even numbers using Java 8.**

See code on **https://github.com/cxur/PaymentStrategyProject/tree/master/src/main/java/paymentStrategy/question2**

* **What do you notice when you do code review?**

- variable names
- method names
- number of lines in a class
- indentation
- That methods and classes perform related task only

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

Scrum is a process framework used to manage product development and other knowledge work.

One of the SCRUM process I've been part of is the **Daily Scrum**

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

- Public
- Protected
- Private
- Static
- Final
- Abstract
- Synchronized
- Volatile

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

Abstract Class	Interface
Can contain Abstract methods and concrete methods (methods with implementation)	Can contain Abstract methods and Default Methods
A child class can only extend ONE abstract class	A child class can implement multiple interfaces
An abstract class can inherit from only ONE abstract class or concrete class	An interface can implement multiple interfaces
Abstract Class can have an access modifier.	The interface does not have access modifiers. Everything defined inside the interface is assumed public modifier.
The class can have data fields.	The interface cannot contain data fields.
An abstract class can inherit a class and multiple interfaces.	An interface can inherit multiple interfaces but cannot inherit a class.

When it comes to deciding which one to use, I think it depends on the case … for example, when you want a class to inherit from multiple sources then an interface should be use. When you have classes repeating the same data fields and method implementations then an abstract class should be created.

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

Maven is a tool that is used for building and managing any Java-based project. And it has three lifecycles default, clean and site, each lifecycle contains different Phases.

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

Git is an open source distributed version control system designed to track and control changes in source code during software development.

- git clone
- git config
- git pull
- git fetch
- git push
- git commit
- git diff
- git reset
- git status
- git init
- git rm
- git merge
- git stash
- git rebase
- git checkout
- git branch

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

A mock to me is a copy of something, in JAVA mocks are mostly use for testing purposes, you can mock classes, database objects, implementations to test you code without using the actual elements.

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

Spring is a framework that contains a number of projects that help creating robust applications, from server level applications to web applications. Within the Spring Framework you have Spring MVC which is use to build web applications , Spring Data use to make interaction with the database easier, Spring Security that provides authorization and authentication tools to protect your applications, among others.

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

More than a difference I think they complement each other, since Spring Boot makes configuration for Spring Applications a lot easier since In Spring Boot, everything is auto configured; no manual configurations are needed. But is also very flexible since it lets you customize and create your own configurations

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

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

Infrastructure as a service (IaaS) is an instant computing infrastructure, provisioned and managed over the internet. For example **Website hosting. Test and development. Big data analysis.**

Platform as a service (PaaS) is a complete development and deployment environment in the cloud, it provides everything IaaS provides with the addition of **Development tools, business intelligence (BI) services, database management systems.**

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

Service mesh is an infrastructure layer use to control how different parts of an application share data with one another. For Example and one of the most common **Traffic management.**

* **Explain what is TDD? What is triangulation?**

(TDD) is a software development process which includes test-first development. Which technically means that the developer has to write the test cases first before writing the production code.

*** Apply the Factory pattern with lambda expressions**

See code on

https://github.com/cxur/PaymentStrategyProject/tree/master/src/main/java/paymentStrategy/question16

*** Reduce the 3 classes (OldWayPaymentStrategy, CashPaymentStrategyandCreditCardStrategy) 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**

See code on **https://github.com/cxur/PaymentStrategyProject/tree/master/src/main/java/paymentStrategy/question17**

```
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;
    }
```

}