Knowledge and questions for backender candidates

Backend senior greenfield project

Questions

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

- Streams: flows of data to make easy the processing and operation over collections using declarative way.
- Lambdas: Anonymous functions to help make code concise and more readable, less verbosity using declarative sintaxis
- Functional Interface: this type of interface can have methods with implementations, they are called “default methods”. This interface has one abstract method only
- static methods in interfaces: they are associated to the interfaces, is not necessary instances of implementations class to use them.
- Optional: helps to avoid NullPointerException and control null values choosing alternative value or actions.
- LocalDate: class represents dates without time
- LocalTime: class represents time without date
- LocalDateTime: class combine LocalDate and LocalTime

  LocalDate, LocalTime and LocalDateTime belong to API that offers methods and utilites to operate over dates and time objects easily

- 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 == 0)
 .collect(Collectors.toList())
```

- What do you notice when you do code review?
  Code review verifies the code accomplishes the development guidelines of the company and best practices, to ensure code quality.

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

  Yes, I have worked with Scrum. Scrum is a framework of agile methodologies that allows deliver a product in incremental way, i.e, in shorter and iterative periods deliver some functionalities until finish the final product. Help to team collaboration and best adaptation to changes in software development.

  Events:
  -Refinement
  -Planning
  -Daily
  -Review
  -Retrospective

  Roles:
  -Scrum Master
  -Developer team
  -Product Owner

- What access modifiers (or visibility) do you know in Java?
  public
  private
  protected

- Differences between an abstract class and an interface. When would you use one or the other?
- In an interface all variables are static; in an abstract class not.
- In a interface all methods and variables are public; an abstract class can have all access modifiers (public, protected, private)
- In an interface all methods are abstract. An abstract class can have abstract and concrete methods. However, since Java 8, with FunctionalInterface we have default and static methods in interfaces.
- An interface doesn't have a constructor method, an abstract class does.

  I would use an abstract class when I have classes in a hierarchy and I want the subclasses inherit some behaviours of abstract class. I would use interface when want standardize the methods or different behaviours in classes not related; the classes can implement methods of distinct interfaces.

- What is Maven and why is it used? What is Maven life cycle?
  Maven is a tool to build projects automatically. It's used to manage dependencies, compile, create distributable files, validate, test the code, deploy.

  Maven lifecycle is:

- validate: verify the project is ok
- compile: compile the source code
- test: run tests the code using unit test classes
- package: convert compiled code in a distributable file (jar, war, EAR)
- verify: validate code packaged is ok
- install: copy the code packaged to local repository to be used as dependency in others projects locally.
- deploy: send the file to a remote repository to share with others distributed projects.

- What is Git and what is it used for? List all Git commands that you know.
  Git is a system to control the source code version. It's used to log the changes over files, synchronize these changes done over shared files.

  Some git commands:
  git clone
  git checkout
  git branch
  git pull
  git fetch
  git merge
  git status
  git add
  git commit
  git push

- What is a mock? What would you use it for?
  Mock is a dummy/fake object to simulate data or behaviour of the real methods or objects of applications. It used for unit tests to test objects and simulate its behaviour expected.

- How would you explain to someone what Spring is? What can it bring to their projects?
  Spring is a dependencies injection framework, that creates a context to load beans.

- What's the difference between Spring and Spring Boot?
  Spring is just a framework and Spring Boot is a utility to make easy running applications because create embedded server as standalone

- Do you know what CQRS is? And Event Sourcing?
  CQRS is a pattern to separate command operations of query operations. It means have methods only to get application status and others for updating application status.

  I don't know what is Event Sourcing.

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

  IaaS : offers services of computing, storage, servers, networking and others resources in

internet and the companies pay for them on demand. Example: AWS, Microsoft Azure, Google Cloud.

PaaS: offers tools and environment to create and manage applications on premise. Examples: OpenShift, DC/OS

- Explain what a Service Mesh is? Do you have an example?
  I haven't worked with Service Mesh tools but I know is a set of services to help manage communication between microservices, like, discovery, routing, logging, traceability, security.

- Explain what is TDD? What is triangulation?
  TDD is a software development technique where first write the test, and after write the functionality code.

  Triangulation is a TDD technique where we must follow next steps:
- Write test to the simpler case of the solution
- Write functional code with that simpler example
- Test fails and then refactor the functional code
- Repeat previous steps adding cases more complex until you finish the solution.

- Apply the Factory pattern with lambda expressions

  Example:
  Create objects of classes Dog and Cat that implement the Animal interface

  Factory method with the lambda expressions is:

```
public Animal createAnimal(String animalType){
 Map<String, Supplier<Animal>> map = new HashMap<>();
 map.put("dog", Dog::new);
 map.put("cat", Cat::new);
 return map.get(animalType).get();
}
```

- 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 {
  //write here your solution
  double pay(double amount);

  static PaymentStrategy cashPayment(){
   double serviceCharge = 5.00;
   return amount -> (amount + serviceCharge);
  }

  static PaymentStrategy creditCardPayment(){
   double serviceCharge = 5.00;
   double creditCardFee = 10.00;
   return amount -> (amount + serviceCharge + creditCardFee);
  }
}


public class Main {

    public static void main(String[] args) {
      double amount=100;
      PaymentStrategy.cashPayment().pay(amount);
      PaymentStrategy.creditCardPayment().pay(amount);
    }
}
```

I converted the interface to a Functional Interface with just one abstract method (pay) and static methods that implement the payment calculations with lambda expressions.

In Main Class I invoke the static method for each payment type (cash, creditcard) to get calculation result for payment