1. **Difference between Git and Github?**

Git is a version control system that lets you manage and keep track of your source code history.

GitHub is a cloud-based hosting service that lets you manage Git repositories. If you have open-source projects that use Git, then GitHub is designed to help you better manage them.

1. **Why Rest API is used?**

The advantages of REST for development

**Separation between the client and the server**: the REST protocol totally separates the user interface from the server and the data storage. it improves the portability of the interface to other types of platforms, it increases the scalability of the projects, and allows the different components of the developments to be evolved independently.

**Visibility, reliability and scalability**. The separation between client and server has one evident advantage, and that is that each development team can scale the product without too much problem. They can migrate to other servers or make all kinds of changes in the database, provided the data from each request is sent correctly. The separation makes it easier to have the front and the back on different servers, and this makes the apps more flexible to work with.

**The REST API is always independent of the type of platform or languages**: the REST API always adapts to the type of syntax or platforms being used, which gives considerable freedom when changing or testing new environments within the development. With a REST API you can have PHP, Java, Python or Node.js servers. The only thing is that it is indispensable that the responses to the requests should always take place in the language used for the information exchange, normally XML or JSON.

### Suppose super class of a new class implement Serializable interface, how can you avoid new class to being serialized?

If Super Class of a Class already implements Serializable interface in Java then its already Serializable in Java, since you cannot unimplemented an interface its not really possible to make it Non Serializable class but yes there is a way to avoid serialization of new class. To avoid Java serialization you need to implement writeObject() and readObject() method in your Class and need to throw NotSerializableException from those method. This is another benefit of customizing java serialization process

1. **Is there any other way instead of using transient in java.**

To avoid Java serialization you need to implement writeObject() and readObject() method in your Class and need to throw NotSerializableException from those method.

1. **Can dictionary be used instead of Hashmap in java?**

In Java the HashMap implements the Map interface while the Dictionary does not. they both do a similar function.

Two terms for the same thing:

*"Map"* is used by Java, C++

*"Dictionary"* is used by .Net, Python

*"Associative array"* is used by PHP

1. **What is query parameter and how to use it with postmapping?**

in the URL, path parameters come before the question mark sign. Secondly, the query parameters are used to sort/filter resources. On the other hand, path parameters are used to identify a specific resource or resources. You can use query parameters to control what data is returned in endpoint responses.

# Challenges of Microservices Architecture

* Bounded Context

the service owns its data and is responsible for its integrity and mutability. It supports the most important feature of microservices, which is independence and decoupling.

* Dynamic Scale up and Scale Down

The loads on the different microservices may be at a different instance of the type. As well as auto-scaling up your microservice should auto-scale down. It reduces the cost of the microservices. We can distribute the load dynamically.

* Monitoring

 The traditional way of monitoring will not align well with microservices because we have multiple services making up the same functionality previously supported by a single application. When an error arises in the application, finding the root cause can be challenging.

* Fault Tolerance

Fault tolerance is the individual service that does not bring down the overall system. The application can operate at a certain degree of satisfaction when the failure occurs. Without fault tolerance, a single failure in the system may cause a total breakdown.

# When should you use Path Variable, and how about Query Parameter?

If you want to identify a resource, you should use Path Variable. But if you want to sort or filter items, then you should use query parameter.

Ex:-

/users # Fetch a list of users  
/users?occupation=programer # Fetch a list of programer user  
/users/123 # Fetch a user who has id 123

You could achieve almost of CRUD process without extra endpoint (Ex. users/crete ) or Query Parameter (Ex. users?action=create ).

@QueryParam is a JAX-RS annotation.

When you want to parse query parameters from a GET request, you can simply define respective arguments to the method that will handle the GET request  and annotate them with @QueryParam annotation. Let’s see how it’s done:

*HelloWorldREST.java:*

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22 | package com.javacodegeeks.enterprise.rest.jersey;    import java.util.List;    import javax.ws.rs.GET;  import javax.ws.rs.Path;  import javax.ws.rs.QueryParam;  import javax.ws.rs.core.Response;    @Path("/helloWorldREST")  public class HelloWorldREST {        @GET      @Path("/parameters")      public Response responseMsg( @QueryParam("parameter1") String parameter1,  @QueryParam("parameter2")  List<String> parameter2) {            String output = "Prameter1: " + parameter1 + "\nParameter2: " +  parameter2.toString();            return Response.status(200).entity(output).build();        }  } |

When you put on your browser:

|  |  |
| --- | --- |
| 1 | <http://localhost:8080/JAXRS-HelloWorld/rest/helloWorldREST/parameters>?  parameter1=JCG&parameter2=Examples&parameter2=QueryParameters |

Outputs:

Prameter1: JCG

Parameter2: [Examples, QueryParameters]

As you can see, one can put multiple values in the same parameter. All of these values will be parsed and become available in a List.

# How do I retrieve query parameters in a Spring Boot controller? I want to accept requests using query parameters:

<http://localhost:8888/user?data=002>

Use **@RequestParam**

@RequestMapping(value="user", method = RequestMethod.GET)

public @ResponseBody Item getItem(@RequestParam("data") String itemid){

Item i = itemDao.findOne(itemid);

String itemName = i.getItemName();

String price = i.getPrice();

return i;

}

1. **when a user fire a request to Rest API, where it will hit first?**

It will hit the server endpoint.

1. **Difference between @Autowired And @Inject?**

@Autowired annotation is used for auto-wiring dependencies in the Spring framework.

**@Autowired** annotation to discover the beans automatically and inject collaborating beans (other associated dependent beans) into our bean. Both the annotation serves for same purpose but @Inject is a standard annotation defined by JSR-330 and @Autowired is spring specific.

# Will a finally block execute after a return statement in a method in Java?

Yes, the finally block will be executed even after a return statement in a method.

The **finally block** will always execute even an exception occurred or not in Java. If we call the **System.exit()** method explicitly in the**finally block** then only it will not be executed. There are few situations where the finally will not be executed like **JVM crash**, **power failure**, **software crash** and etc. Other than these conditions, the **finally block** will be always executed.

1. **Can abstract classes have a** **constructor**?

**Yes**.. The main purpose of the constructor is to initialize the newly created object. In abstract class, we have an instance variable, abstract methods, and non-abstract methods. We need to initialize the non-abstract methods and instance variables, therefore abstract classes have a constructor.

even if we don’t provide any constructor the compiler will add default constructor in an abstract class.

The constructor inside the abstract class can only be called during constructor chaining i.e. when we create an instance of sub-classes. This is also one of the reasons abstract class can have a constructor.

1. **How to secure microservices?**

API Gateway is used to secure micro services. The API Gateway is the entry point to all the micro services of our application. It’s responsible for service discovery (from the client side), routing the requests coming from external callers to the right microservices, and fanning out to different microservices

OAuth is used along with, API Gateways to handle the authentication and authorization from the external callers to the microservice level.

1. **How microservices communicate with each other?**

Feign provides an abstraction over REST-based calls via annotation, by which microservices can use to communicate with each other without writing detailed REST client code.

1. **What to do to maintain the flow of microservices uninterrupted in case of any microservices goes down?**

Circuit breaker is used maintain the flow of microservices uninterrupted in case of any microservices failure. Netflix's Hystrix library provides an implementation of the circuit breaker pattern. When you apply a circuit breaker to a method, Hystrix watches for failing calls to that method, and if failures reached a threshold (limit), Hystrix opens the circuit and redirects calls to the fallback method.

The circuit breaker has three distinct states: Closed, Open, and Half-Open:

* **Closed** – When everything is normal, the circuit breaker remains in the closed state and all calls pass through to the services. When the number of failures exceeds a predetermined threshold the breaker trips, and it goes into the Open state.
* **Open** – The circuit breaker returns an error for calls without executing the function.
* **Half-Open** – After a timeout period, the circuit switches to a half-open state to test if the underlying problem still exists. If a single call fails in this half-open state, the breaker is once again tripped. If it succeeds, the circuit breaker resets back to the normal, closed state.

1. **Is Spring application synchronized**?

If scope of bean is singleton, then it will be synchronized i.e. the same instance will be reused by multiple threads.

If scope of bean is defined prototype, then it will be thread safe because a new bean instance will be created per request.

## Shallow Copy Vs Deep Copy In Java :

Below is the list of differences between shallow copy and deep copy in Java.

|  |  |
| --- | --- |
| **Shallow Copy** | **Deep Copy** |
| Cloned Object and original object are not 100% disjoint. | Cloned Object and original object are 100% disjoint. |
| Any changes made to cloned object will be reflected in original object or vice versa. | Any changes made to cloned object will not be reflected in original object or vice versa. |
| Default version of clone method creates the shallow copy of an object. | To create the deep copy of an object, you have to override clone method. |
| Shallow copy is preferred if an object has only primitive fields. | Deep copy is preferred if an object has references to other objects as fields. |
| Shallow copy is fast and also less expensive. | Deep copy is slow and very expensive. |

## Difference Between == Operator And equals Method In Java :

Below is the list of differences between == operator and equals method in java.

|  |  |
| --- | --- |
| **“==” Operator** | **equals() Method** |
| It is a binary operator in java. | It is a public method of java.lang.Object class. |
| It compares the two objects based on their location in the memory. | The default version of equals method also does the comparison of two objects based on their location in the memory. But, you can override the equals method so that it performs the comparison of two objects on some condition. |
| It can be used on both primitive types as well as on derived types. | It can be used only on derived types. |
| It is best suitable for primitive types. | It is best suitable for derived types. |
| You can’t override the “==” operator. It behaves same for all objects. | You can override the equals method according to your business requirements. |

1. **Fail Fast Vs Fail Safe Iterators In Java :**

|  |  |
| --- | --- |
| **Fail-Fast Iterators** | **Fail-Safe Iterators** |
| Fail-Fast iterators doesn’t allow modifications of a collection while iterating over it. | Fail-Safe iterators allow modifications of a collection while iterating over it. |
| These iterators throw ConcurrentModificationException if a collection is modified while iterating over it. | These iterators don’t throw any exceptions if a collection is modified while iterating over it. |
| They use original collection to traverse over the elements of the collection. | They use copy of the original collection to traverse over the elements of the collection. |
| These iterators don’t require extra memory. | These iterators require extra memory to clone the collection. |
| Ex : Iterators returned by *ArrayList*, *Vector*, *HashMap*. | Ex : Iterator returned by *ConcurrentHashMap.* |

|  |  |
| --- | --- |
| **Spring** | **Spring Boot** |
| **Spring Framework** is a widely used Java EE framework for building applications. | **Spring Boot Framework** is widely used to develop **REST APIs**. |
| It aims to simplify Java EE development that makes developers more productive. | It aims to shorten the code length and provide the easiest way to develop **Web Applications**. |
| The primary feature of the Spring Framework is **dependency injection**. | The primary feature of Spring Boot is **Autoconfiguration**. It automatically configures the classes based on the requirement. |
| It helps to make things simpler by allowing us to develop **loosely coupled** applications. | It helps to create a **stand-alone** application with less configuration. |
| The developer writes a lot of code (**boilerplate code**) to do the minimal task. | It **reduces** boilerplate code. |
| To test the Spring project, we need to set up the sever explicitly. | Spring Boot offers **embedded server** such as **Jetty** and **Tomcat**, etc. |
| It does not provide support for an in-memory database. | It offers several plugins for working with an embedded and **in-memory** database such as **H2**. |
| Developers manually define dependencies for the Spring project in **pom.xml**. | Spring Boot comes with the concept of **starter** in pom.xml file that internally takes care of downloading the dependencies **JARs** based on Spring Boot Requirement. |

**What are idempotent operations?**

An idempotent HTTP method is an HTTP method that can be called many times without different outcomes. It would not matter if the method is called only once, or ten times over. The result should be the same.

1. POST is NOT idempotent.
2. GET, PUT, DELETE, HEAD, OPTIONS and TRACE are idempotent.

**How do you check whether an ExecutionService task executed successfully?**

We can use a Future to check the return value. Below example shows how it can be done. Future get

method would return null if the task finished successfully.

Future future = executorService1.submit(new Runnable() {

public void run() {

System.out .println("From executorService1");

}

});

future.get(); // returns null if the task has finished correctly.

**What is Callable? How do you execute a Callable from ExecutionService?**

Runnable interface's run method has a return type void. So, it cannot return any result from executing a

task. However, a Callable interface's call method has a return type. If you have multiple return values

possible from a task, we can use the Callable interface. Example shows how to create a Callable

interface and execute it using an executor service. The return value is printed to the output.

Future futureFromCallable = executorService1.submit(new Callable() {

public String call() throws Exception {

return "RESULT";

}

});

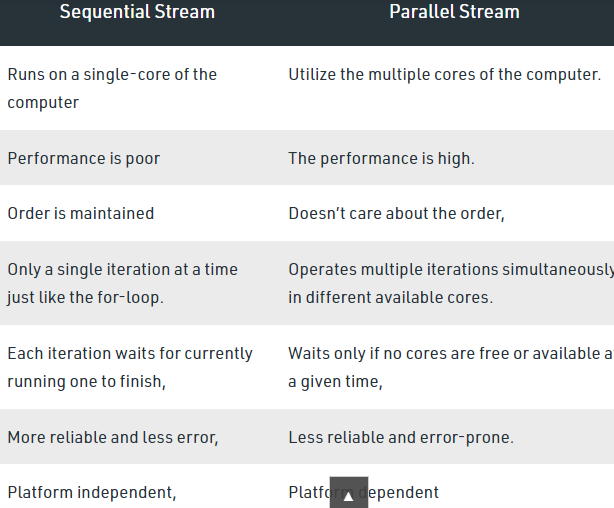
System.out .println("futureFromCallable.get() = "

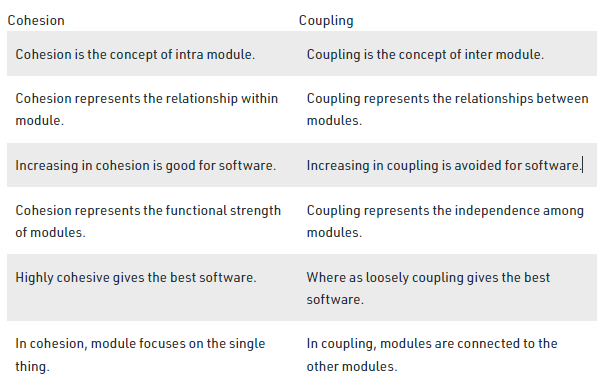
+ futureFromCallable.get());

**Where singleton design pattern is used?**

It is used **where only a single instance of a class is required to control the action throughout the execution**. A singleton class shouldn't have multiple instances in any case and at any cost. Singleton classes are used for logging, driver objects, caching and thread pool, database connections

**Difference between putmapping and postmapping?**

**@configuration, @enable autoconfiguration**

****