Tell me about yourself technically

- Walk me through your latest project

- Tell me specifically about your role in this project

- Explain the data flow

○ Db --> ORM framework --> exposure framework --> consuming framework

- Biggest challenge

○ Always make this into a learning experience/positive

**Four pillars OOP**:

* Inheritance: is a mechanism in which one object acquires all the properties and behaviors of a parent object.
* Abstraction: is a process of hiding the implementation details from the user. Only the functionality will be provided to the user. Abstraction is achieved by using abstract classes and interfaces.
* Encapsulation: is the act of providing a public interface and hiding the implementation details: private instance variables, public getters, and setters.
* Polymorphisms: the ability for an objs to behave in various manners.

Method overloading: same name, different signature (same class).

Method overriding: same name, different implementation (inherited classes).

Can overload a static method, but can not override a static method.

How does java know which overriding methods to call? When you invoke the overriding method with child object, Java will call that overriding method.

Benefit of interface:

* Interface **supports multiple inheritance**, not abstract class.
* You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.

Abstract class:

1. You want to share code among several closely related classes.
2. You want to declare non-static or non-final fields

**String vs stringbuffer**

* Stringbuffer stores data on heap, thread-safe, mutable: append, reverse, substring, toString, chatAt, length, insert, replace.
* String is in string pool, is immutable.

Can you overload a constructor? Yes, but can’t override constructor.

A while loop keeps going until the boolean value is false

The for loop is you set how many loops you want

Describe a switch statement.

Static String alphablet(int num) {

Switch(num)

Case 10: return “a”;

Case 11: return “b””;

Default: return “0”;

}

**Switch data types: byte**, **short**, **char**, and **int** primitive data types. It also works with enumerated types, the **String** class, and wrapper class: **Character**, **Byte**, **Short**, and **Integer.**

**Checked exception** is an exception that is checked by the compiler at compilation-time.

**Unchecked exception** is an exception that occurs at the time of execution. These are also called as **Runtime Exceptions: IOException, IndexOutofBound, InputMishaspexception**

Extend exception or RuntimeException with a new class, add new functionality

Public class MyOwnException extends Exception{}

To convert primitive data into objects, use wrapper class: Double x = 15.5;

Serialization: is a mechanism of converting the state of an object into a byte stream. To make a Java object serializable we implement the **java.io.Serializable** interface.

Static method belongs to class. i.e. Math.pow.

Non-static method belongs an object. Car.run();

Collection

Collection is the root interface of Collections Framework. Other interfaces are List, Set, Queue, and Map.

Collection classes: ArrayList, LinkedList, HashMap, TreeMap, HashSet, TreeSet.

Iterator interface provides methods to iterate over any Collection. We can get iterator instance from a Collection using iterator method.

Iterator<String> iterator = listStrings.iterator();

while (iterator.hasNext()) {

    System.out.println(iterator.next());

}

* Set is a collection that cannot contain duplicate elements.

Set<String> set = new HashSet<>();

Methods: add, contain, remove, size, clear, iterator().

* [List](https://www.journaldev.com/11444/java-list) is an ordered collection and can contain duplicate elements. ArrayList and LinkedList are implementation classes of List interface.

Methods: sort, shuffle, reverse, binarySearch, add.

List<String> str = new ArrayList<>();

* Queue is a collection used to hold multiple elements prior to processing. A Queue provides additional insertion, extraction, and inspection operations.
* Map is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value. Methods: put, get, containsKey, containsValue, size, isEmpty, keyset.

Set<String> keyset = map.keySet();

HashMap:

Hash table based implementation of the Map interface. It permits null values and the null key. HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits null. This class makes no guarantees for the order of the map.

TreeMap: The map is sorted according to the natural ordering of its keys.

Vector is very similar to ArrayList but Vector is synchronized and have some legacy method which collection framework does not have.

Vector v = new Vector();

Methods: add, clear, size, remove, contains, get, indexOf, isEmpty.

== check reference, equals check same value

@Override

Public Boolean equals (Object o) {

//self check

If (this == o) return true;

//null check

If (o == null) return false;

// type check and cast

If (!(o instanceof Person)) return false;

// field comparision

Person person = (Person) o;

Return Objects.equals(firstName, person.firstName)

&& Objects.equals(lastName, person.lastName); }

***hashcode():***returns an integer representation of the object memory address.

**Thread** – a thread is the path followed when executing a program. All Java programs have at least one thread known as the main thread, which is created by the JVM at the program’s start, when the main() method is invoked.

-**Multithreading in java** is a process of executing multiple threads simultaneously.

-Each thread gets its own stack.

-Create a separate thread by extending the Thread class or by implementing the Runnable interface overriding the run() method of the Runnable interface, then calling the start() method of the Thread class.

 If we extend the Thread class, our class cannot extend any other class because Java doesn’t support multiple inheritance. But, if we implement the Runnable interface, our class can still extend other base classes.

How does one thread communicate with other? Wait, notify, notifyAll.

Thread States: New, RUNNABLE, BLOCKED, WAITING, TIMED-WAITING, & TERMINATED.

**Deadlock** describes a situation where two or more threads are blocked forever, waiting for each other ... Use synchronized to prevent deadlock.

Lifecycle of a thread

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

The thread is in running state if the thread scheduler has selected it.

This is the state when the thread is still alive, but is currently not eligible to run. (Blocked = waiting, timed-waiting).

A thread is in terminated when its run() method exits.

Junit

Annotations

@Test – indicates that the following method is a unit test

@Before @ After – run before/after each @Test method

@BeforeClass/@AfterClass - run before/after all @Test methods in the class are executed.

@Assert methods – methods used to define expected behavior & indicate success/failure.

@Runwith, @SuiteCases : Test Suit

@Override

Can junit be used for testing your entire application: yes, use test suit.

* Maven is a project management tool that comes with a dependency manager to import plugins and external dependencies to your project.
* Comes with a cli that can be used to test, build, and package your application.

**Servlet**

Servlet: java classes that can handle HTTP Requests and Responses from web clients.

* Client sends http request
* Server receives request.
* Servlet container consults web.xml – Deployment descriptor – to map the request to appropriate servlet.
* container instantiates the servlet.
* Container calls the init() method of the servlet.
* Container calls the servlet’s service () method.
* HTTP response is returned by servlet to the server then to the client.
* Container calls the servlet’s destroy () method, releasing the instance to save memory. container shut down.

- How do we programmatically create a servlet

In web.xml

<servlet>

<servlet-name>hello</servlet-name>

<servlet-class>com.ex.servlets.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>hello</servlet-name>

<url-pattern>/httpInfo</url-pattern>

</servlet-mapping>

- Servlet config vs servlet context

Servlet context: Data will be visible in every servlet of the application.

<context-param>

<param-name>AppInfo</param-name>

<param-value>Hello there</param-value>

</context-param>

Servlet config: Data only visible for a specific servlet.

<servlet>

<servlet-name>H</servlet-name>

<servlet-class>com.ex.servlets.HelloServlet</servlet-class>

<init-param>

<param-name>secret</param-name>

<param-value>This secret only shared in this servlet</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>…..

- Two important method that help process a request

Public class HelloServlet extends HttpServlet {

Protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {}

Protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {}

}

- Forward vs redirect

req.getRequestDispatcher(“login.html”).forward(req,resp);

resp.sendRedirect(“index.html”);

Forward: the request will be further processed on the server side, URL stays the same, req and resp objects will remain the same objects after forwarding.

Redirect: request is redirected to a different resource, url will change, a new request is created.

In general, a forward should be used if the operation can be safely repeated upon a browser reload of the resulting web page, otherwise, redirect must be used. Typically, if the operation performs an edit on the database, then a redirect is required.

Spring

Spring:

Is an application framework and inversion of control container for java.

Associated with developing websites, very secure, backend, databases

Java framework that makes DI easy for developers.

Create and store dependencies in the container and injects them into registered objects (beans).

Spring modules: Core, Beans, Context, MVC, ORM, Data, Boot, Security, Test.

Spring Bean lifecycle

1. Request bean from ApplicationContext
2. Instantiation
3. Populate properties
4. Set bean name
5. Set bean factory
6. Pre-initiation, bean post processor
7. Initializing bean
8. Custom init method
9. Post-initilization bean post processors
10. Bean is ready to use
11. Container shuts down.

Different ways to configure a class as a spring bean:

1. Autowire, use @Component to register a class a bean. Add @Autowired to a constructor, a field, or a setter method.
2. Configure spring bean in beans.xml or context.xml file.

<bean id=”thisIsABean” class=”com.ex.HelloWorld” scope=”prototype”>

<property name=”message” value=”This is a setter injection”></property>

<constructor-arg name=”message” value=”This is constructor injection”></constructor-arg>

</bean>

To be autowired, must configure annotations so that springs IOC knows where to search for annotations.

<context: annotation-config/>

<context: component-scan base-package=”com.ex.autowiring”>

Spring annotations: Component, Service, Controller, Repository, RestController

* @Component is a generic stereotype for any Spring-managed component, turn java class into beans.
* @Service annotates classes at the service layer
* @Controller annotation marks a class as a Spring Web MVC controller.
* @Repository annotates classes at the persistence layer, which will act as a database repository

Different types of dependency injection you’ve used: setter and constructor injections.

Modes of autowiring: by default no autowiring, byName (setter injection), byType and constructor.

Bean scopes: singleton(default), prototype, request, session, global session.

Singleton: single object instance in the spring IoC container, default scope.

Prototype: many instances of the bean, new instance created whenever bean is requested

**Spring MVC**

Spring MVC architecture

Spring’s web module provides an implementation for the model-view-controller architectural pattern

Implements the front controller design pattern and uses 1 servlet that sends requests to their appropriate @Controller (or @RestController)

Model: Application data (Pojo’s)

View: Renders a view for the client using model data.

Controller: Processes user requests.

ViewResolver: helps render view, useful for server-side HTML rendering, can use any view technology: freemarker, thymeleaf, JSPs, Velocity.

@Controller: turn java classes into beans to handle requests.

**@RestController:** combines @Controller and @ResponseBody. It returns data as json.

Annotations: @RequestMapping, @GetMapping, @PostMapping, @PathVariable, @RequestBody, @ExceptionHandler, @RequestParam

Spring dependencies: Spring-boot-starter-actuator, spring-boot-starter-data-jpa, spring-cloud-starter-feign, spring-cloud-starter-eureka, junit.

**Spring AOP**

Aspect Oriented Programming: the modularization of cross-cutting concerns.

Essentially injecting code that is needed in various layers of the app instead of rewriting it.

Affect many methods without having to change them

Difference between concern and cross cutting concern

The Concern is behavior we want to have in a module of an application. A Concern may be defined as a functionality we want to implement. The cross-cutting concern is a concern which is applicable throughout the application and it affects the entire application. For example, logging, security and data transfer.

Handle exceptions in Spring MVC

* Response Entity: new ResponseEntity<Customer> (HttpStatus.NOT\_FOUND)
* @ResponseStatus: on top of a custom exception.

@ResponseStatus(value=HttpStatus.NOT\_FOUND, reason = “Message”)

* @ExceptionHandler: on top of a method that is going to handle the exception, @ExceptionHandler (CustomException.class). It can also be done globally in a separate class which uses @ControllerAdvice.

Spring boot

The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration, and @ComponentScan

**Web Services**

Webservices: any piece of software that makes itself available over the internet via a standard protocol or messaging system.

* Not tied to any operating system or programming language.
* Discoverable via a simple find mechanism.

Why do we use them?

* Interoperability: make applications platform and technology independent.
* Standardized media types: XML, JSON, HTML.
* Low cost of communication and delivery: HTTP, HTTPS(REST)
* Code reusability: connect other technologies to your existing one.
* Distributed business applications: make your business system decoupled.

Annotations used for SOAP web services

@WebService: it means that we are trying to mark the class or interface as the Web Service.

@SOAPBinding(style=Style.DOCUMENT) : used to specify the SOAP messaging style which can be Document or PRC.

@WebMethod: specifies that the method represents a web service method.

@WebClient.

How to create a SOAP service: 2 ways

Contract first – write the wsdl and generate the service

Contract last – write the service and generate the wsdl.

WSDL structure:

Web Service Description Language is an XML markup language used to described a Web service.

- Binding vs porttype

**portType** (Analogs to Java interface)

* PortType is an abstraction part of WSDL.
* An abstract set of operations supported by one or more endpoints.

**Binding**

* Binding is a concrete part of WSDL.
* Describes how the operation is invoked by specifying concrete protocol and data format specifications for the operations and messages.

Soap message structure:

**Envelope** − Defines the start and the end of the message. It is a mandatory element.

* **Header** − Contains any optional attributes of the message. It is an optional element.
* **Body** − Contains the XML data of the message. It is a mandatory element.
* **Fault** − An optional Fault element that provides information about errors that occur while processing the message.

How can a client consume a soap service: by reading the wsdl.

SOAP: has exception handling built-in, configurable in the body of the message (<fault> tag).

**Restful Web Service**

Annotations: @Produces, @Path, @PathParam, @QueryParam.

-

Handle exceptions: REST doesn’t have built in exception handling, must rely on HTTP status codes

**SQL**

* Primary key (PK) – candidate key which is not null & unique, and will not change.
* Foreign key – reference to another field, usually the PK of another table.

Different kinds of joins:

* **(INNER) JOIN**: Returns records that have matching values in both tables
* **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
* **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
* **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table

- What kind of join allows for null values

Where vs having: WHERE clause is used for filtering rows and it applies on each and every row, while **HAVING** clause is used to filter groups in **SQL**

Delete vs truncate: DROP and **TRUNCATE** are DDL commands, whereas **DELETE** is a DML command. Therefore **DELETE** operations can be rolled back (undone), while DROP and **TRUNCATE** operations cannot be rolled back.

Select \* from employee where salary > 30,000 ;

Insert into employee (id, name) values (123, “John”);

Update employee

Set salary = 50000

Where id=5;

Delete from employee where id =5;

select e1.lastname as EMPLOYEE, e2.lastname as MANAGER

from employee e1

join employee e2

on e1.reportsto = e2.employeeid;

**Functions**:

* Executable block of code which must return a value and may have many parameters.
* DQL statements only

**Stored Procedures**

* Executable block of code without O+|N|OUT parameters & no return value
* Full DML & TCL capabilities.

**Angular**

An NgModule describes how the application parts fit together. Every application has at least one Angular module, the *root*module that you bootstrap to launch the application. By convention, it is usually called AppModule.

[bootstrap](https://angular.io/api/core/NgModule#bootstrap): [AppComponent] // bootstrapped entry component

Declare your components in module.ts:

[declarations](https://angular.io/api/core/NgModule#declarations): [

AppComponent]

The template represents the view. HTML is the language of the Angular template

How do you inject your template into the view: templateUrl

Decorator that marks a class as an Angular component and provides configuration metadata that determines how the component should be processed, instantiated, and used at runtime.

The [HttpClient](https://angular.io/api/common/http/HttpClient) in @angular/common/[http](https://angular.io/api/common/http) offers a simplified client HTTP API for Angular applications that rests on the XMLHttpRequest interface exposed by browsers.

**Microservices**

Pros:

* each microservice is relatively small.
* easier to scale development.
* improved fault isolation.
* each service can be developed and deployed independently.
* eliminates any long-term commitment to a technology stack.

Cons:

TESTING – Spinning up test environments is more involved with microservices due to the increased number of nodes required.

PERFORMANCE – Communication over a network is considerably slower than in memory.

Advantages of using spring boot:

* It reduces lots of development time and increases productivity.
* It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
* It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.

How to load changes to a spring boot app without restarting your server?

Spring-boot-devtools(a dependency), when you save new changes it auto refreshes

Actuators enable production-ready features to a Spring Boot application

Monitoring our app, gathering metrics, understanding traffic or the state of our database becomes trivial with this dependency.

How do you run your spring boot app on a custom port

In application.properties files: server.port = 8081

Application.yml: server:

Port: 8081

How do you enable eureka

@EnableEurekaServer

@SpringBootApplication

@EnableEurekaServer: This microservice will provide the service registry and discovery server.

[@EnableEurekaClient](https://github.com/spring-cloud/spring-cloud-netflix/blob/master/spring-cloud-netflix-eureka-client/src/main/java/org/springframework/cloud/netflix/eureka/EnableEurekaClient.java) this annotation will act like a spring discovery client and will register itself in the eureka server attached to this service.

Application.yml

spring:

application:

name: eureka

eureka:

client:

service-url:

defaultZone: ${EUREKA\_HOME}

register-with-eureka: **false**

fetch-registry: **false**

server:

port: 8761

Zuul

**Zuul** is an edge service that proxies requests to multiple services. It provides a “front door” to your ecosystem.

Zuul filters: pre filters, post filters, route filters, error filters.

Spring cloud config server:

A central config server where all configurable parameters of micro-services are written version controlled. The benefit of a central config server is that if we change a property for a microservice, it can reflect that on the fly without redeploying the microservice.

Docker

**Docker** is a platform for developers to **develop, deploy, and run** applications with containers. Containers can be specified with Docker files and Docker Compose configuration files.

A container is launched by running an image. An **image** is an executable package that includes everything needed to run an application.

Dockerfile to create an image

Use image to create container

Can save images on dockerhub

@EnableCircuitBreaker

@HystrixCommand(fallbackMethod)

**Circuit breaker** monitors the service health and once it gets some issue, the circuit breaker trips and all further calls go to the circuit breaker fall back.

Fallbacks: used when failures occur to keep app running.

Hibernate

Hibernate is an object-relational mapping tool for Java. It provides a framework for mapping an object-oriented domain model to a relational database

ORM tool: is a concept of converting the data from OOP to relational DB and vice versa.

Main configuration is hibernate.cfg.xml, Mapping configuration either in xml or annotations

Hibernate Query Language: is fully object-oriented and understands notions like inheritance, polymorphism, and association

Important interfaces in the Hibernate API

Session represents your session with a database

Methods: save, get, update, delete, beginTransaction, createQuery, createCriteria

Session session = sessionFactory.openSession();

session.close();

SessionFactory used to create session objects, Only need One sessionFactory

Configuration

SessionFactory sessionFactory = new Configuration().configure("hibernate.cfg.xml").buildSessionFactory()

Hibernate object states:

Transient: no session associated with this object

Persistent: object has representation in DB. Hibernate will depict any changes made to an object in this state and will reflect changes at the end of the transaction.

Detached: object was once persisted, but session has been closed, still has representation in DB, but changes to this object will not be reflected in DB, unless it is reattached to a session.

Default fetching strategy in hibernate: Lazy load

Important annotations you’ve used in hibernate mapping: @Entity, @Table, @Id, @Column, @GeneratedValue, @OneToMany, @Cached.

Handle exceptions in hibernate: try catch block

The difference between HQL and SQL

SQL is based on a relational database model whereas HQL is a combination of object-oriented programming with relational database concepts.

SQL manipulates data stored in tables and modifies its rows and columns. HQL is concerned about objects and its properties.

SQL is concerned about the relationship that exists between two tables while HQL considers the relation between two objects.

Criteria vs Query

• A query can be used to perform SELECT, INSERT, UPDATE, DELETE

 SQL injection is possible

• Criteria can only be used to perform SELECT.

List<BankUser> bankUsers = criteria.list();

Transaction: Autocommit is false by default. This means, you must begin and commit a transaction if you modify anything

Transaction tx = session.beginTransaction();

tx.commit();

Hbm2ddl: Update, create, validate: makes no changes, validates data, create-drop.

• ACID

• Atomicity: All operations are done or all operations are not

• Consistency: After a transaction is completed successfully, the data in the database should be consistent

• Isolation: If two transactions are working with the same data, one transaction won't disturb the other

• Durability: After a transaction is completed, the data will persist until another transaction is performed on that data

• Session.load(): Returns a proxy without hitting the database. A proxy is an object given an identifier value, its properties are not yet initialized. If no row is found it throws an ObjectNotFoundException

• Session.get(): Immediately hits the database and returns the actual object representing the row. If no row is found, it throws null

• Session.update() It checks if the object exists in the session cache or not, if the object exists then it throws an exception call NonUniqueObjectException

• Session.merge() Checks if the object exists in the session cache or not, If the object exists then the current changes are copied in to the cache, otherwise it loads the values to the cache

 Update a persistent entity with new field values from a detached entity

• Session.save() It returns an identifier, and if an INSERT is called to get the identifier, it happens immediately regardless if you're inside or outside a transaction

• Session.persist()

Some exceptions associated with hibernate: ConstraintViolationException, DataException, JDBCConnectionException.

Automatic dirty checking: when session is closed, hibernate checks for changes in any persistent objects and puts them into the detached state.

Transactional write behind in the context of a transaction, changes made to an object in the persistent state are not immediately propagated to the DB.

Transitive persistent (cascades): No default cascading of state between related entities.

Available cascade styles: create, merge, save – update, delete, lock, refresh, evict, replicate

Typically cascade one to many, many to one, not one to one or many to many.

Hibernate Caching: improves the performance of the application by pooling objects in the cache. 2 levels:

Level 1: default, bysession

Level 2: must configure with vendor, session factory.

Caching is a mechanism to enhance the performance of a system. It is a buffer memory that lies between the application and the database. Cache memory stores recently used data items in order to reduce the number of database hits as much as possible.

Any third - party cache can be used with Hibernate

Concurrency strategies: transactional, read-write, non-strict read-write, read-only.

Cache provider: EHCache, OSCache, warmCache, JBoss Cache.

**JDBC**

Statement: Use for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters. SQL injection.

PreparedStatement: Use when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.

CallableStatement: Use when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

**Important interfaces in the JDBC API**

* + Connection:
  + Statement:
  + PreparedStatement:
  + CallableStatement:
  + ResultSet:
* **executeUpdate() vs executeQuery()**
* **How, in Java, do i obtain the keys generated (via triggers, etc) in my database?**
  + getAutoGeneratedKeys()

**Advantages of JDBC**

* Clean and easily for small programs
* JDBC provides good performance with large amount of data
* Small JDBC programs can be developed very easily

**Disadvantages of JDBC**

* JDBC is not easily if it is used in large projects.
* Programmer must hardcode the Transactions and concurrency code in the application.
* Handling the JDBC connections and properly closing the connection is also a big issue. Properly closing the connection is must.

**What is ORM?**

**Here are the benefits of ORM technology**

* No need to deal with the SQL Queries to save and retrieve the data
* Simple configuration
* Standardized API to persist the business objects
* Fast development of application
* Concurrency support
* Excellent caching support for better performance of the application
* Injected transaction management
* Configurable logging
* Easy to learn and use

**Disadvantages of ORM**

* Slow performance in case of large batch updates
* Little slower than JDBC