**Master java**

**Environment**

1. Java
2. Maven
3. Git

**Java basics**

1. OOP

* **Inheritance**

Class can be extended and the subclass can use variable/methods from super class by specifying “super”

* **Polymophism**

These are 2 or more classes related through inheritance.

* **Encapsulation**

This makes sure that sensitive data is hidden from the user by declaring variable as private and and writes its setter and getter methods

* **Abstraction**

This is a process of hiding certain details and showing only essential information. You can achieve abstraction by Abstract classes and interface.

|  |  |
| --- | --- |
| Abstract | Interface |
| Can have both abstract (without implementation) and non-abstract (with implementation) methods | Interface does not have implementation |
| Subclass of abstract class can write implementation of the abstract method in the super class by Overriding the method. | A different class must implement the interface and write implementations of the methods in interface by Overriding them. |
| You can access the methods/ variable of the abstract class through its subclass. | You can access through the interface |

1. Design Patterns/Princiles

Solid is an acronym for the following five principle

* Single Responsibility
* Open-Closed Principle

The Open-Closed Principle (OCP) states that classes should be open for extension but closed for modification. “Open to extension” means that you should design your classes so that new functionality can be added as new requirements are generated. “Closed for modification” means that once you have developed a class you should never modify it, except to correct bugs.

* Liskov Substitution
* Interface Segregation
* Dependency Inversion

1. Constructors

Constructor in java is used **to create the instance of the class** and can also be used to set variables while creating the object. Constructors are almost similar to methods except for two things - its name is the same as the class name and it has no return type. Sometimes constructors are also referred to as special methods to initialize an object.

1. Override vs Overload

|  |  |
| --- | --- |
| Override | Overload |
| Overriding occurs when the method signature is the same in the superclass and the child class. | Overloading occurs when two or more methods in the same class have the same name but different parameters. |
|  | It is used **so that the programmer does not have to remember various function names**. |
|  |  |
|  |  |

1. Collections

Collections are implementation of Data Structures.

|  |  |  |  |
| --- | --- | --- | --- |
| Arrays | Lists | Maps | Sets |
| Fixed size | Not Fixed in size | A value is assigned to a key. |  |
| Specify size or initialize on creation | Implementation: ArrayLists, Stacks  Vector | Keys can not be duplicated and can assign one value in a key. |  |
|  |  | Get value using the key |  |
|  |  |  |  |
|  |  |  |  |

1. Treads

Threads allows a program to operate more efficiently by doing multiple things at the same time. Threads can be used to perform complicated tasks in the background without interrupting the main program.

There are two ways to create a thread.

* Extending the Tread class and overriding the run class.

public class Main extends Thread {

public static void main(String[] args) {

Main thread = new Main();

thread.start();

System.out.println("This code is outside of the thread");

}

public void run() {

System.out.println("This code is running in a thread");

}

}

* Implement the Runnable interface and override the run method

public class Main implements Runnable {

public static void main(String[] args) {

Main obj = new Main();

Thread thread = new Thread(obj);

thread.start();

System.out.println("This code is outside of the thread");

}

public void run() {

System.out.println("This code is running in a thread");

}

}

**Callenge**: Because threads run at the same time as other parts of the program, there is no way to know in which order the code will run. When the threads and main program are reading and writing the same variables, the values are unpredictable. The problems that result from this are called concurrency problems.

**Solution**:  If attributes need to be shared, one possible solution is to use the isAlive() method of the thread to check whether the thread has finished running before using any attributes that the thread can change.

public class Main extends Thread {

public static int amount = 0;

public static void main(String[] args) {

Main thread = new Main();

thread.start();

// Wait for the thread to finish

while(thread.isAlive()) {

System.out.println("Waiting...");

}

// Update amount and print its value

System.out.println("Main: " + amount);

amount++;

System.out.println("Main: " + amount);

}

public void run() {

amount++;

}

}

gdg

1. Difference between JVM, JRE, JDK
2. Garbage collection
3. Best practices

**Services**

1. Soap (Simple Object Access Protocol) service

**SOAP** is a protocol which was designed before REST to ensure that programs built on different platforms and programming languages could exchange data in an easy manner. SOAP stands for Simple Object Access Protocol.

1. Rest (Representational State Transfer)

**REST** was designed specifically for working with components such as media components, files, or even objects on a particular hardware device. Any web service that is defined on the principles of REST can be called a RestFul web service. A Restful service would use the normal HTTP verbs of GET, POST, PUT and DELETE for working with the required components. REST stands for Representational State Transfer.

1. Soap vs Rest

|  |  |
| --- | --- |
| Soap | Rest |
| SOAP can only work with XML format. As seen from SOAP messages, all data passed is in XML format. | REST permits different data format such as Plain text, HTML, XML, JSON, etc. But the most preferred format for transferring data is JSON. |
| Performance: slower that Rest  SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the amount of data transfer using SOAP is generally a lot. | Performance: Quick  REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. |
| API class cannot be cached | API calls can be ceched |
| Stateless by default but its possible to make a SOAP API stateful | stateless |
| WS-security with SSL support. Its also has a built in asset compliance. | Supports HTTPS and SSL |
| Transfer protocols: HTTP, SMTP, UDP and others. | Transfer protocol: HTTP |
|  |  |

1. Recommendations to use SOAP and REST?

Soap is recommended for high security applications like payment/financial applications and other application with sensitive data.

Rest is recommended to be used is public API’s for web services and applications that does not contain sensitive data.

1. Challenges of SOAP service

* WSDL file: REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. Making changes to the WSDL file will cause clients to make changes.
* Size of SOAP message: The large size of soap messages transferred from client to server can cause performance problems.

1. Challenges of REST service

* Lack of security: Rest does not impose any sort of security like SOAP.
* Lack of state: Most web applications require a stateful mechanism. For example, if you had a purchasing site which had the mechanism of having a shopping cart, it is required to know the number of items in the shopping cart before the actual purchase is made. Unfortunately, the burden of maintaining this state lies with the client, which just makes the client application heavier and difficult to maintain.

1. Steps to create a Rest service
2. Soap
3. Steps to create a Soap service

* Service Definition
* XML – Define XML request and XML response structure.

Structure (Request and Response): Envelope – Header (optional) – Body (contain request/response object

* XSD – Data being passed In the body
* WSDL
* XML Java Binding
* JAXB – Configure XML binding. Map java objects to XML and XML to Java objects. Configure a plugin in the pom.xml
* Framework Configuration
* Endpoint – Write method and specify namespace
* @Endpoint  
  public class Notes {  
    
   @Payload(namespace = "http://lwazi.com/gae", localPart = "RequestPayload")  
   @ResponsePayload //convert object response to XML  
   public DetailsResponse getDetails(@RequestPayload DetailsRequest request){  
   DetailsResponse response = new DetailsResponse();  
   return response;  
   }  
  }
* WSDL-generation

1. Swagger documentation
2. Exceptions
3. Logging
4. Auditing

**Testing**

1. Integration testing
2. Unit testing
3. Repository testing

**Java Frameworks**

1. Spring
2. Play
3. Quarks

**Spring framework**

1. Why we need spring framework?

With spring, we get **Dependency Injection** (identify the beans, identify their dependencies and wire them together). With this, we don’t have to focus on creating objects and managing their dependencies. Spring has a number of modules/projects like Spring Data JPA, Spring boot, Spring MVC, spring starter test (Mockito, Junit).

1. Types of dependency injections

* **Constructor-based**: Dependencies are set by creating a constructor on your bean and setting the variable bean in the constructor. You can have “@Autowire” in you bean variable declaration or on the constructor.
* **Setter-based**: Dependencies are set on a setter method on your bean class. You can have “@Autowire” in you bean variable declaration or on the setter method.
* **Field**: No setter or constructor. Dependencies are injected using reflection.

1. Recommended type of dependency injection.

Constructor-based because all dependencies are automatically set when an object is created.

1. How spring works

* **@Component** to allow spring to create a bean of the class
* Component scan will scan the package to look for all classes with annotation @Component.
* **Application context** creates the beans.
* **@Autowire** – Identifying the spring dependencies and wiring them.

1. Spring jars

We use maven to download spring jars to support our spring implementation.

**Spring Projects**

1. Spring MVC

A Spring MVC is a Java framework which is used to build web applications and REST API. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

1. Spring boot

* World before spring boot.

Setting up an project was not easy and took time.

* **Dependency Management**: Need add, manage frameworks/Dependencies and their versions in the pom file., e.g to build REST API (dependencies: Spring framework, Spring MVC, JSON binding framework). Unit tests (Spring test, Mockito, Junit)
* **Web.xml**: Configure dispatcher sevlet for spring MVC
* **Spring Configuration**: Define component scan, view resolver(jsp path)
* **None-functional requirements**: configure Logging, Error handling, Monitoring
* Spring boot helps to build **Production-Ready** apps **Quickly**.
* Build **Quickly**
* **Spring initializer**

You can create a spring project by specifying the name, build tool and dependencies in the spring initializer website.

* **Spring boot starter project**

Spring boot starter projects are dependencies that contain a list of pre-defined dependencies inside them for a specific feature.

**REST API and Web Applications** – Spring Boot Starter Web (spring-webmvc, spring-web, spring-boot-starter-tomcat, spring-boot-starter-json)

**Unit Test** – Spring Boot Starter Test (Spring test, Mockito, Junit)

**Database connection** – Spring Boot Data JPA

**Security** – Spring Boot Starter Security

* **Spring boot Auto Configuration**

By adding starter projects in the pom file, maven will download the dependencies for that starter project and place in the class path. Spring boot auto configures based on the frameworks that exist in the class path.

Auto Configuration decide bases on:

* Which frameworks are in the class path
* What is the existing configuration

Example: Spring Boot Starter Web will configure:

* Dispatcher servlet (DispatcherServletAutoConfiguration)
* Tomcat
* Default error page ( when you hit incorrect url endpoint)
* Bean to and from JSON (JacksonHttpMessageConvertersConfiguration)

All logic for spring boot auto configuration sits in a jar called SpringBootAutoConfiguration.jar

* **Spring boot DevTool**

Spring Boot devtools increate developer productivity. You don’t have to manually restart the services after writing a new code change. After the change, spring boot will auto restart the application.

* Build **Production-Ready**
* Logging

You can specify the logging level you want on your application property file. Level log:

Trace, debug, info, warning, error, off

* Different Configuration for different Environment (Profiles, ConfigurationProperties)

Provides easy configuration of profiles for different environments. Create separate application.properties for DEV, QA and PROD with their own configurations ( eg different databases, log level). You can set the application property file you want to active on your main property file.

* Monitoring (Spring Boot Actuator)

Spring boot starter actuator provides list of endpoints to monitor your application:

* Beans – Complete list of Spring beans in the app
* Health – Application health information
* Metrics – Application metrics
* Mappings – Details around Request mappings.

By default when you launching the actuator url, it gives the health url. For more features, you need to enable them in application.properties. For all endpoints:

Management.endpoints.web.exposure.include=\*;

1. Spring security
2. Spring data
3. Spring cloud

**Data bases**

1. What are databases and why we need them?

Used to store information

1. SQL vs NoSQL

|  |  |  |
| --- | --- | --- |
| **Category** | **SQL** | **NoSQL** |
| **Structure** | Relational (Tables that can have relationships) | Implementation dependent (documents, Graphs, tables) |
| **Storage** | Concentrated (One node contain the entire copy of the data. It is not partitioned) | Hashing input. Data is partitioned into multiple nodes. |
| **Scale** | Vertical (increase RAM, CPU in your server) | Horizontal (add more servers to handle traffic) |
| **Access** | SQL (Structured query language), JPQL, Direct connection | Rest API |
|  |  |  |

1. When to use SQL and NoSQL

|  |  |
| --- | --- |
| **SQL** | **NoSQL** |
| When your access patterns are not defined | When your access pattern is defined |
| When you want to perform flexible queries | When your primary key is known |
| When you want to perform relational queries | When your data model fits (graph) |
| When you want to enforce field constraints | When you need high performance and low latency |
| When you want to use well documented access language (SQL) |  |

1. SQL and NoSQL database engine examples

|  |  |
| --- | --- |
| **SQL** | **NoSQL** |
| MySQL | mangoDB |
| SQLServer | DynamoDB |
| PostgreSQL | Elasticsearch |
| Amazon Aurora | Amazon ElastiCeche |

1. How to connect java code with a database?

Create a JPA repository and specify your entity class.

1. JPA and Hibernate

|  |  |
| --- | --- |
| **JPA** | **Hibernate** |
| JPA is described in **javax.persistence** package. | Hibernate is described in **org.hibernate** package. |
| It describes the handling of relational data in Java applications. | Hibernate is an Object-Relational Mapping (ORM) tool that is used to save the Java objects in the relational database system. |
| It is not an implementation. It is only a Java specification. | Hibernate is an implementation of JPA. |
| It is a standard API that permits to perform database operations. | It is used in mapping Java data types with SQL data types and database tables. |
| As an object-oriented query language, it uses **Java Persistence Query Language (JPQL)** to execute database operations. | As an object-oriented query language, it uses **Hibernate Query Language (HQL)** to execute database operations. |
| To interconnect with the entity manager factory for the persistence unit,ituses **EntityManagerFactory**  interface. Thus, it gives an entity manager. | To create Session instances, it uses **SessionFactory** interface. |
| To make, read, and remove actions for instances of mapped entity classes, it uses **EntityManager** interface.  This interface interconnect**s** with the persistence condition. | To make, read, and remove actions for instances of mapped entity classes, it uses **Session**interface. It acts as a runtime interface between a Java application and Hibernate. |

1. Entity lifecycle

* Transient/New State

When an entity object is initially create, its state is new. In this state the object is not yet associated with an Entity Manager and has no representation in the database.

* Managed State

This is when the entity is persisted to the database via an Entity Manager persist() method which must be invoked within an active transaction.

* Detached

This is when the entity is disconnected from the entity manager

* Removed

Entity can be marked as removed by using the entityManager remove() method within an active transaction. The entity object changes from managed state to remove state and physically deleted from the database during commit.

1. Database Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

* [NOT NULL](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [UNIQUE](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [FOREIGN KEY](https://www.w3schools.com/sql/sql_foreignkey.asp) - Prevents actions that would destroy links between tables
* [CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that the values in a column satisfies a specific condition
* [DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified
* [CREATE INDEX](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

1. Relationships

* One to One
* One to Many
* Many to One
* Many to Many

1. Transactions

A SQL transaction is **a grouping of one or more SQL statements that interact with a database**. A transaction in its entirety can commit to a database as a single logical unit or rollback (become undone) as a single logical unit. In SQL, transactions are essential for maintaining database integrity.

1. Indexes

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

A table or view can contain the following types of indexes:

* Clustered
  + Clustered indexes sort and store the data rows in the table or view based on their key values. These are the columns included in the index definition. There can be only one clustered index per table, because the data rows themselves can be stored in only one order.
  + The only time the data rows in a table are stored in sorted order is when the table contains a clustered index. When a table has a clustered index, the table is called a clustered table. If a table has no clustered index, its data rows are stored in an unordered structure called a heap.
* Nonclustered
  + Nonclustered indexes have a structure separate from the data rows. A nonclustered index contains the nonclustered index key values and each key value entry has a pointer to the data row that contains the key value.

1. Joins

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

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

1. Queries
2. Locking

**Microservices**

1. Different architectures
2. Why microservice architecture is the best for big projects
3. Challenges

* Sizing

Solution: You can choose per feature. E.g Cards, accounts, loans

* Deployment

How do we deploy all the 100s of microservices?

**Solution**: use docker compose to run all your images/services with only one command (docker-compose up).

**If services were stopped, you and start all of them at the same time. docker start fee e** // You can run one command to deploy all container and specify.

* Portability

How do we move the 100s of services between environments

**Solution**: The same docker image you can use in different environments. You don’t need to install the project dependencies, you just download the docker image, install docker and run/convert the image to a docker container.

* Scalability

How do we scale our applications based on requirements

**Solution**: docker run -p 8081:8080 lwazi/accounts //can create another instance of the same service and provide difference port number. With virtual machine, you need to create a virtual machine, set up the environment and run another instance. You will have to buy all the deferent virtual machines but with docker you will use/run the the different container instances

1. What is docker image

Docker image is package which has all the dependencies that your application need (java, spring boot libraries, business logic). With one image you can create multiple containers (instances) with different port numbers. These containers run in their own isolated environment and they don’t need operating systems unlike virtual machines.

1. What is a container

Is a environment that allows us to build and run software packages. These packages include the code and all dependencies to run the code. We call these package container images.

1. What is Docker

Docker is a tool that allows applications to be packaged with all dependencies and ran wherever wanted

1. Create docker image

* Option1 (Docker file)
* Create a docker file inside your project folders
* docker images //check if you don’t have images
* docker build . -t lwazi/accounts //create image
* Option2 (Build-packs)
* What are build-packs

Buildpacks are provided by spring-boot and the purpose of build-packs is that you don’t have to write a docker file definition. Build-packs detects all the dependencies you have in your application, business logic and base on that, they will create a docker image without you creating a docker file.

* Sas
* as

1. Create docker container from docker image

Docker run -p 8080:8080 lwazi/accounts

* Docker ps //check all containers that are running
* Docker run -p 8081:8080 lwazi/accounts //can create another instance of the same service and provide different port number

1. Docker

* **docker images** //check if you don’t have images
* **docker build . -t lwazi/accounts** //create image
* **Docker ps** //check all containers that are running
* **Docker ps -a** //returns all containers present in the system/server
* **Docker run -p 8090:8080 lwazi/accounts** //creates docker container from the docker images
* **Docker run -p 8081:8080 lwazi/accounts**  //can create another instance of the same service/container and provide different port number.
* **Docker run -d -p 8082:8080 lwazi/accounts**  //run container without see/returning logs.
* **Docker logs fc** //checks the current logs of the specified container. “fc” is the container id.
* **Docker logs -f fc** //returns latest and new container logs.”-f” means follow.
* **docker stop fc** //gradually (takes some time) stops the running the container. It will shutdown the container first then stops the container. Check the logs.
* **docker kill fc** //immediately stops/kill the running container. Does not wait to shutdown the container.
* **docker start fee e** //deploy both containers at the same time
* **docker container pause fc** // pause the container so that it does not take any requests.
* **Docker container unpause fc** //un-pause the container
* **Docker container inspect fc** //returns container details
* **Docker stats** // returns the stats of all running containers e.g CPU used.
* **Docker rm fc** // removes/deletes the container
* **Docker push docker.io/lwazi/accounts** // pushes image to remote repo so that someone else can download and run it.
* **Docker compose**

With docker compose you can create a yml file with all the services/images defined in it. You can run a single command to execute to yml file which will run all specified images.

* docker-compse up //this will read the file and execute and run all specified images in the yml file.
* **Cd**

JMS (Java Message Serve)

JMS is a java messaging oriented middleware API for sending messages between applications.

JMS Components

* JMS Provider – Is the JMS middleware API
* JMS clients – Java applications that produces or receive messages
* JMS Producers – application that creates the and send the message
* JMS Consumer - application that received the message
* JMS Application – System composed of many JMS clients and one JMS provider.

Messaging Models

|  |  |
| --- | --- |
| PTP (Point to Point) | Publish-Subscribe |
| Each message has only one consumer | Each message can have multiple consumers. |
| Messages are first sent to the Queue | Messages are first published to the Topic |
| A sender and a receiver of the message have no timing dependencies. The receiver can fetch the message whether or not it was running when the client sent the message | Publishers and Subscribers have a timing dependency. |
| The receiver acknowledges the successful processing of a message | Does not provide acknoledgement |
|  |  |

IOC

Collections

Tread

Override and Overload

Principles of programming

SOAP vs REST

**Difference private, protected, public and default**

**JAVA JMS**

**How to create an API from scratch to the end**

**difference between clean install and clean deploy**

**have you used Hybernate and how?**

**What are the basic fundamental concepts of programming?**

**SOLID is an acronym for five other class-design principles**