# Authentication and Authorization:

**Authentication** is the process of verifying who a user is trying to access.

While **Authorization** is the process of verifying what they have access to.

**Extra:**

Authentication verifies the identity of a user or service. Authorization determines their access rights.

Could you please share some details about the project's and the team I will be working with?

Is it possible for you to share more details about the project that I will be working on?

I was informed that I would receive a link for onboarding on the day before my joining date, but I have not received it yet. Could you please provide me with the link or further information about the onboarding process?

When can I expect the delivery of the company laptop that will be provided for my work?

Excuse me, could you please let me know who I should reach out to for further information?

May I ask why the job offer was cancelled? I have already left my previous company and currently do not have any other offers. What should be my next course of action?

My name is Rakesh Kumar. I have 2 years of experience as java developer. I am working as Software Engineer in financial domain in the company called APCFSS, Andhra Pradesh Center for Financial Systems and Services, at Vijayawada Andhra Pradesh. And this Company established by Government of AP to design, develop, implement and maintain the financial management system of AP and to help Government to implement appropriate e-governance solutions and good governance initiatives. I have worked on project like HERB, the full form is

Actually, the reason I left my previous organization is because my father and grandmother both required hospitalization for surgeries within a short period of time. As I'm the only one to take care of them because my mom alone cannot help them out, I needed to be present at the hospital to provide support. Unfortunately, this impacted my work progress due to the frequent visits required. In order to focus on my family during this difficult time, I made the decision to take a break from work until everything returned to normal.

**OOP - Object Oriented Programming**

1. Encapsulation
2. Polymorphism
3. Inheritance
4. Abstraction

# Encapsulation:

Encapsulation is an ability of objects to hide their internal characteristics (or state) and behaviour.

This can be achieved by using access modifiers by controlling access rights to other classes.

**Access Modifiers**

Access Modifiers are used to set the accessibility or scope or visibility of a data member (field, constructor, method or class).

1. default
2. public
3. private
4. protected

**default:** Whenever a specific access level is not specified, then it is assumed to be default. The scope of the default level is *within a package*.

**public:** Whenever the public access specifier is used, the data member will be *accessible all throughout the application.*

**protected:** We can access the protected modifier members *within in the same package and also from outside the package, but inheritance is must for accessing it from outside the package*. *We can access in child class by inheriting a parent class.*

**private:** We can access the private modifier member *only within the same class*. They are *not accessible from outside the class.*

*private and protected access modifiers are not allowed to use at class level (class and interface).*

# Inheritance:

Java Inheritance lets one class acquire the properties and attributes of another class.

The concept of inheritance in java is that new classes can be constructed on the top of older ones.

We can use the parent class’s methods and properties when we inherit from the current class.

We can also add additional fields and methods in the current class.

The main advantage of inheritance is code reusability and also method overriding (runtime [polymorphism](https://www.mygreatlearning.com/blog/polymorphism-in-java/)).

1. Single inheritance
2. Multi-level inheritance
3. Hierarchical Inheritance
4. Multiple inheritance
5. Hybrid Inheritance

Java supports only Single, Multilevel, and Hierarchical types of inheritance. Java does not support Multiple and Hybrid inheritance.

**Single Inheritance:**

Just one class is subject to this kind of inheritance. Simply one class extends to another class.

**Multi-Level Inheritance:**

A class extends to another class that is already extended from another class.

**Hierarchical Inheritance:**

A numerous subclasses derive from a single class.

This procedure makes dynamic polymorphism and code reuse possible (method overriding).

**Multiple Inheritance:**

Defining sub class from more than one base class is known as ‘Multiple Inheritance’

**Hybrid Inheritance:**

This is the combination of two or more inheritance types, such as Hierarchical Inheritance and Multiple Inheritance and Single Inheritance.

**super keyword** gives the reference of the super class object.

**this keyword** gives the reference of the current class object.

# Polymorphism:

Polymorphism is the ability of an object to take on different forms. In Java, polymorphism refers to the ability of a class to provide different implementations of a method, depending on the type of object that is passed.

Method Overloading and Method Overriding comes under the Polymorphism.

**Method overloading** is the process that can create multiple methods of the same name in the same class, but the number, types, and order of parameters and the return type of the methods are different.

**Method overriding** is the process if the subclass provides its definition to a method already present in the superclass, then that function in the base class is said to be overridden.

Polymorphism in Java can be classified into two types:

1. Static/Compile-Time Polymorphism
2. Dynamic/Runtime Polymorphism

**Static/Compile-Time Polymorphism**

The call to the method is resolved at compile-time only. Compile-Time polymorphism is achieved through Method Overloading.

**Dynamic/Runtime Polymorphism**

The call to the method is resolved dynamically at runtime rather than at compile-time. Runtime polymorphism is achieved through Method Overriding.

We can use parent class's reference to show runtime polymorphism

Method Overriding is done when a child or a subclass has a method with the same name, parameters, and return type as the parent or the superclass; then that function overrides the function in the superclass. In simpler terms

Also, it should be noted that runtime polymorphism can only be achieved through functions and not data members.

Overriding is done by using a reference variable of the superclass. The method to be called is determined based on the object which is being referred to by the reference variable. This is also known as Up-casting.

Up-casting takes place when the Parent class’s reference variable refers to the object of the child class.

Java 8 interface changes include static methods and default methods in interfaces. Prior to Java 8, we could have only method declarations in the interfaces. But from Java 8, we can have **default methods** and **static methods** in the interfaces.

*Now when a class implements an Interface, it is not mandatory to provide implementation for default methods of interface.*

IS-A relationship

HAS-A relationship

Composition is like holding and using another class object reference.

Coming to Aggregation, the main object and that usable object of other class are independent.

# Abstraction:

The creation of abstract classes without including the actual implementation of all the methods that aims to separate the implementation details of a class from behaviour.

We cannot instantiate an abstract class in Java because it is abstract. It is not complete, hence it cannot be used.

Abstract classes cannot be instantiated, but they can be subclassed. When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

Synchronization in java is the capability to control the access of multiple threads to any shared resource.

Thread synchronization is the concurrent execution of two or more threads that share critical resources. Threads should be synchronized to avoid critical resource use conflicts. Otherwise, conflicts may arise when parallel-running threads attempt to modify a common variable at the same time.

# Collections Framework

* A collection represents a group of objects known as its elements.
* Set, which is a collection that cannot contain duplicate elements.
* List, which is an ordered collection and can contain duplicate elements.
* Map, which is an object that maps keys to values and cannot contain duplicate keys.

All the classes and interfaces of collections framework are present in the java.util package.

**Collection Interface:**

Collection Interface is one of the root interfaces of the Collection Hierarchy.

The Collection interface is not directly implemented by any class.

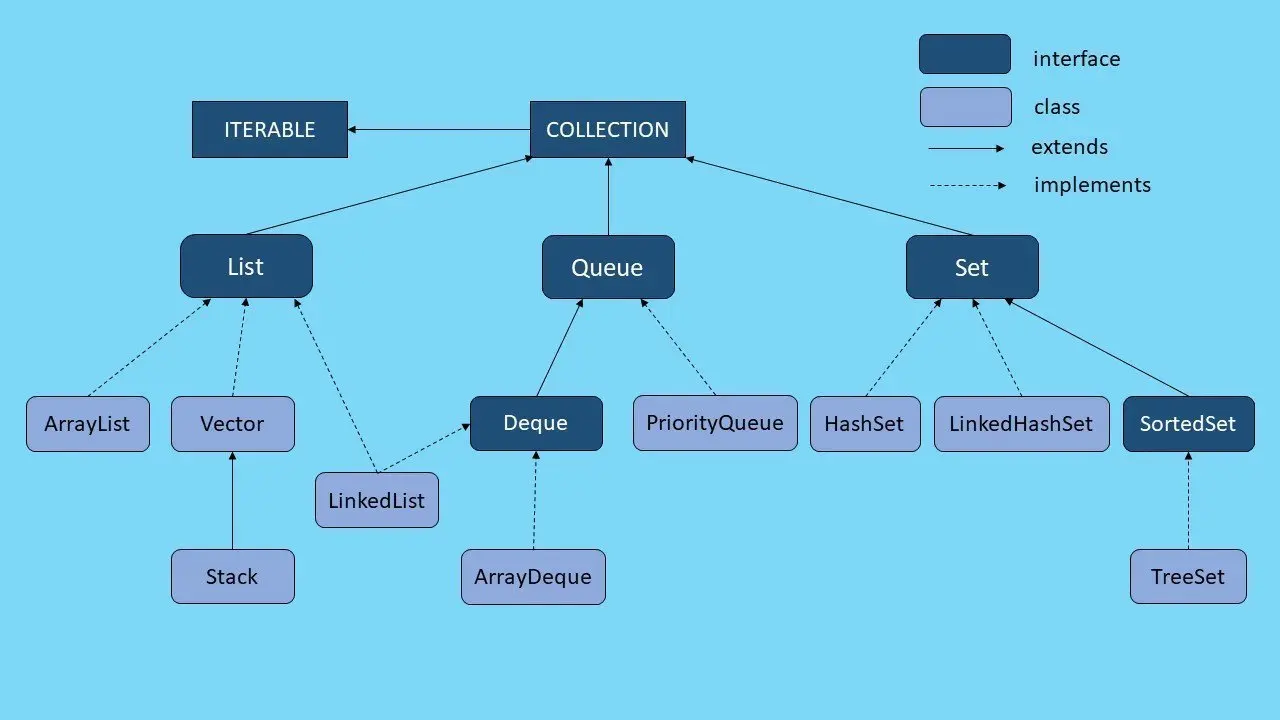
**Collections class:**

Collections class exclusively consists of static methods that operate on or return collections.

Collection interface extends the Iterable Interface.

Collection interface is extended by 3 interfaces

1. List<T>
2. Set<T>
3. Queue<T>



Based on some characteristics collections are classified

1. Type of data (Homogeneous or Heterogeneous).
2. Allowed Null objects.
3. Allowed duplicates.
4. Insertion Order.
5. Data Structure.

**List<T> Interface:**

List stores elements in sequencial order. Sequential order means the first element, followed by the second element, followed by the third element, and so on.

List interface is implemented by the classes

1. ArrayList
2. LinkedList
3. Vector (and Vector is extended by Stack class)

List allows Heterogeneous Objects.

List allows null Objects.

List follows index based process.

# ArrayList<T> class:

1. ArrayList stores heterogeneous data.
2. Allows Null Objects.
3. It allows duplicate Objects.
4. Insertion is preserved.
5. Data structure is growable size array.
6. Iterator and ListIterator Cursors can be used.

# LinkedList<T> class:

1. Heterogeneous Objects are allowed.
2. Allows Null Objects.
3. Insertion order is preserved.
4. Duplicate objects are allowed.
5. Data structure is Doubly Linked List.
6. Cursors – Iterator and ListIterator.

# Vector<T> class:

1. Heterogeneous Object are allowed.
2. Null Objects are allowed.
3. Insertion Order is preserved.
4. Duplicate objects are allowed.
5. Data structure is growable array.
6. Cursors – ListIterator and Iterator.

# Stack<T> class:

The Stack class represents a last-in-first-out (LIFO) stack of objects.

It extends class Vector with five operations that allow a vector to be treated as a stack.

The usual *push* and *pop* operations are provided, as well as a method to *peek* at the top item on the stack, a method to test for whether the stack is *empty*, and a method to *search* the stack for an item and discover how far it is from the top.

**HashSet<T>:**

1. Heterogeneous Objects are allowed.
2. Duplicates are not allowed.
3. Only one null object is allowed.
4. Data Structure is Hash Table.
5. Insertion Order is preserved.
6. Set is unidirectional so it supports only iterator.

# LinkHashSet<T>:

1. Heterogeneous Objects are allowed.
2. Duplicates are not allowed, even if we add if we add duplicate, there is no Compile Time Error, it will print only once.
3. Data structure is Hash Table and LinkedList
4. Insertion order is preserved.
5. Only one null object is allowed.
6. Set is unidirectional so it supports only iterator.

# TreeSet<T>:

1. Heterogeneous Object are not allowed, if we add we will get class cast exception.
2. Duplicates are not allowed even though if we add, there will be no Compile Time Error, it will add only once.
3. Only one null object is allowed.
4. Data structure is tree.
5. Insertion order is preserved (depends on hash code).
6. Set is unidirectional so it supports only iterator.

# String:

String is a sequence of characters.

In most of the programming languages, a String is an array of characters.

Java implements strings as objects of type String Class.

When you create a String that cannot be changed.

Once a String Object has been created, you cannot change the characters that comprise that string.

You can still perform all types of String operations, but each time you need an altered version of an existing String, a new String object will be created that contains the modifications.

The original string is left unchanged.

The strings within the objects of type String are unchangeable means that the contents of the String instance cannot be changed after it has been created. However, a variable declared as a String reference can be changed to point at some other String object at any time.

This approach is used because fixed or immutable strings can be implemented more efficiently than changeable ones.

For modifiable strings, Java provides:

1. StringBuffer
2. StringBuilder

Both hold strings those can be modified after they are created.

The String, StringBuffer and StringBuilder classes are defined in java.lang package.

All are defined final, which means that none of these classes may be subclassed.

All three implements the CharSequence interface.

**String Pool:**

String pool in java is pool of strings stored in Heap Memory.

**Design Patterns**

Fly Weight Design Pattern

MVC Design Pattern

MVC2 Design Pattern

Factory Design Pattern

Sigleton Design Pattern

DAO Design Pattern

DTO Design Pattern

Strategy Design Pattern

**Creational Design Patterns**

Creational Design Patterns provide solutions to instantiate an object in the best possible way for specific situations.

Factory Design Pattern

Singleton Design Pattern

Builder Design Pattern

Abstract Factory Design Pattern

Prototype Design pattern

# Factory Design pattern

**Factory Design pattern** is used when we have a super class with multiple subclasses and based on input, we need to return one of the subclass.

We implement a Factory class with a method that return the subclass object based on the input.

We can keep factory class singleton or we can keep the method that returns the subclass as static.

# Singleton design pattern

**Singleton design pattern** restricts the instantiation of a class and ensure that the class has only one instance, while providing a global access point to this instance.

# Builder Design pattern

When the object contains a lot of attributes some of the parameters might be optional but in factory pattern, we are forced to send all the parameters and optional parameters need to send as null.

**Builder Design pattern** provide a way to build the object step-by-step and provide a method that will actually return the final object.

# Prototype Design Pattern

**Prototype Design Pattern** provides a mechanism to copy the original object to a new object and then modify it according to our needs.

This pattern uses java cloning to the object.

However, whether to use the shallow or deep copy of the object properties depends on our requirements.

# intro

My name is Rakesh Kumar. I have 2 years of experience as a java developer. I am working as Software Engineer in the financial domain in the company called APCFSS, Andhra Pradesh Center for Financial Systems and Services, at Vijayawada Andhra Pradesh. And this Company was established by the Government of AP to design, develop, implement, and maintain the financial management system of AP and to help Government to implement appropriate e-governance solutions and good governance initiatives. I have worked on a project called HERB, the full form is H for Human Resources, E for Expenditures, R for Receipts, and B for Budget.  
Along with these modules, it has several modules like ESS (Employee Self Service), Pension Settlement, and more.  
  
We developed separate different contexts for each of these modules which can be accessed from a single login dashboard.  
Technologies used in these modules are JDK11, Spring Boot, Spring Data JPA for REST service, and PostgreSQL Database and for security, authentication, and authorization we used Spring Security and Keycloak OpenID Connect. For front-end development, we used React JS.  
  
I was involved mainly in the Expenditures module and Pension Settlement module and developed Document Management System (DMS).  
  
DMS context is about managing documents. In AWS, there is an object storage service called S3(Simple Storage Service). We used it for storing documents in buckets nothing but containers. I developed 5 REST services in this context 3 for uploading, downloading, and deleting documents in the paths specified for module type, for these the bucket and folder path in which they will be stored are defined in the application properties file and will be retrieved from that file and another 2 for uploading and downloading from or to the user-defined path in the S3 bucket, here the path will be taken from user input.   
  
And also developed some minor services in other modules like hitting the REST services like Aadhaar validating and eKyc services using RestTemplate, reading and writing excel sheets using Apache POI, and some SMS services(I worked with SMS Striker and CDAC and airtel service providers. They provide an URL and a template and a template ID for each message. They took these template messages and template ID and mobile as query strings in the URL).  
  
Coming to HERB functionality, in the Government of Andhra Pradesh the organization structure is like a group of organizations that comes under a DDOCODE(DDOCODE is nothing but a unique) likewise there are many DDOCODEs, and for each DDOCODE a person will be there known as DDO(Drawing and Disbursing Officer). His responsibility is to create a bill and draws the money and distribute it to the organizations which come under him.

# GIT

Git is open sourced distributed version control system.

It is used for tracking the project changes.

With the help of versioning system, the developer can identify who has made what changes and then run tests and fix bugs if any and then do necessary feature implementation.

In case of any unforeseen circumstances, the code can be reverted to any previously working version.

Git can be installed on any workstation and acts as client and server.

Every developer has a local copy of full version history of the project on their individual system.

# Optional class

The purpose of the class is to provide a type-level solution for representing optional values instead of *null* references.

It is used to deal with NullPointerException in java application. It provides the methods to easily check whether a variable has null value or not.

# Stream API

The Stream API is used to process collections of objects.

Streams don’t change the original data structure, they only provide the result as per the pipelined methods.

### What is the default value of the local variables?

**Local variables are not initialized to any default value**, whether they are of primitive types (such as int, double, boolean, etc.) or object reference types (such as String, ArrayList, etc.).

This means that if you declare a local variable but don't initialize it with a value, attempting to use that variable in any way (such as trying to read its value or passing it as an argument to a method) will result in a compilation error.

To avoid this, you **should always initialize local variables with an appropriate value before using them in your code**.

**what about global variables?**

In Java, **there is no concept of global variables** in the same way that there is in some other programming languages, such as C or C++. Instead, in Java, you can define variables at different levels of visibility depending on where you declare them.

The closest thing to a global variable in Java is a ***class variable***, which is also known as a static variable. A class variable is a variable that is declared at the class level and is shared by all instances of that class. When you declare a class variable, you use the "static" keyword, which indicates that the variable belongs to the class rather than to any particular instance of the class.

If you don't explicitly assign a value to a class variable, it will be initialized to the default value for its type (e.g. 0 for int, null for reference types).

**what if we dont specify static keyword?**

If you don't specify the ‘static’ keyword when declaring a variable in Java, it will be an ***instance variable*** or a ***member variable***, which means that each instance of the class will have its own copy of the variable.

Instance variables are initialized to their default values when an instance of the class is created. The default value for numeric types (such as int, double, etc.) is 0, and the default value for reference types (such as String, ArrayList, etc.) is null.

If you declare a class variable (a static variable) as “final”, it must be initialized when it is declared or in a static block, and its value cannot be changed afterwards.

In Java, when you declare a variable as “final”, it means that the variable's value cannot be changed once it is initialized. Therefore, the Java compiler enforces that all “final” variables must be initialized exactly once.

The reason for this is to ensure that the value of the “final” variable is known and consistent throughout the program's execution. If a “final” variable is not initialized, it would be in an undefined state and using it could result in unpredictable behavior or runtime errors. Additionally, initializing a “final” variable ensures that the value cannot be accidentally changed later in the program, which can help prevent bugs and make the code more robust.

### What will be the initial value of an object reference which is defined as an instance variable?

All instance object references are initialized to null in Java.

Is an object will be created to parent class when we are creating an object of sub class?

What is the use of calling super() in sub class?

Is it mandatory to call super() in subclass?

Composition and Aggregation?

real time examples for thread safe and non-thread safe in java

1. How is singleton bean created and then how that single bean autowired in different classes?