Java OOPS :

Difference between ArrayList and LinkedList in java :

ArrayList internally uses a dynamic array to store the element. Manipulation in ArrayList is slow compared to linkedList since insertion and deletion requires shifting of elements internally in memory. When dataset is large and the main operation is write, linkedList is preferred ArrayLists is faster at reading data since data can be directly accessed via their indice.

LinkedList in java uses a **doubly linked list,** which means each element contains address of its prev and next element. LinkedList is preferred when dataset is large and write operation is maximum.

A default size of 10 is allocated to arrayList if not initialized but linkedList is Null.

Both are extended from the collection framework in java.util

Difference between == and .equals

== is an operator and is used for comparison of objects. If 2 objects are created and we want to check if they reside in the same memory location, we can use ==

.equals is a method and it is used to compare 2 strings to check whether all the characters of both the strings match

Exceptions in Java

Exceptions are objects that describe an error or an unusual condition that has occurred in a program. They are classified into three main categories: checked exceptions, unchecked exceptions, and errors.

1. **Checked Exceptions:**

   Exceptions that are checked at CompileTime
   Eg : ClassNotFoundException

2. **Unchecked Exceptions:**

   They are not checked during the compile time but rather at runtime.
   Eg : NullPointerException,ArrayIndexOutOfBounds

3. **Errors** :
   Eg : OutOfMemoryError , StackOverFlowError, VirtualMachineError

Difference between final, finally and finalize :

Final vs. Finally vs. Finalize: `final` keyword in Java is used to declare constants, prevent method overriding, and inheritance. `finally` block follows `try` block and executes regardless of exceptions. `finalize()` is a method in `Object` class used for garbage collection, deprecated in Java 9.

Explain about servlets :

Servlets are Java classes that run on a web server and act as a middle layer between client requests (typically web browsers) and server responses (typically web pages). They are part of the Java EE (Enterprise Edition) platform and are used to extend the capabilities of servers that host applications accessed via a request-response programming model.

Platform Independence: Servlets are written in Java, which is platform-independent, ensuring that they can run on any server that supports the Java platform.
Performance: Servlets execute within the web server's process, offering better performance compared to CGI scripts that spawn new processes for each request.
Scalability: Servlets can handle multiple requests concurrently, making them suitable for high-traffic applications.
Robustness: With Java's strong memory management and exception handling, servlets are less prone to memory leaks and cras

Explain the MVC (Model-View-Controller) architecture and how it is implemented in Java EE applications.

The Model-View-Controller (MVC) architecture is a design pattern used in software engineering to separate an application into three interconnected components, each with a distinct responsibility.

1. Model

- **What it is**: The part of the application that handles the data.
- **What it does**: Manages the data, updates it, and retrieves it.
- **Example**: A class that represents a user in a system and contains their information like name, email, etc.

2. View

- **What it is**: The part of the application that the user interacts with.
- **What it does**: Displays the data and sends user actions to the Controller.
- **Example**: A web page or a mobile app screen showing a user's profile information.

3. Controller

- **What it is**: The part of the application that handles user input and interactions.
- **What it does**: Takes input from the View, processes it, updates the Model, and decides what to show in the View.
- **Example**: A function that runs when a user clicks a button to update their profile.

How They Work Together

1. **User Interaction**: The user interacts with the View (e.g., clicks a button).
2. **Controller Response**: The Controller receives the click, processes it, and updates the Model (e.g., changes the user's profile data).
3. **Model Update**: The Model changes the data (e.g., saves the new profile data).
4. **View Update**: The View updates to show the new data (e.g., shows the updated profile).

Benefits

- **Organized Code**: Keeps the code clean and separated by function.
- **Easier Maintenance**: Makes it easier to update parts of the application without breaking everything.
- **Better Testing**: Allows for easier testing of each part separately.

Example

Imagine an online store:

- **Model**: Manages products, prices, and user data.
- **View**: Web pages that show the products and prices.
- **Controller**: Handles adding products to the cart, checking out, etc.

In summary, MVC is a way to organize code into three parts: the Model (data), the View (user interface), and the Controller (logic), making it easier to manage, update, and test the application.

Difference between stateless and stateful session beans.

Stateless Session Beans	Stateful Sessions Beans
Are pooled in memory, to save the overhead of creating a bean every time one is needed. WebLogic Server uses a bean instance when needed and puts it back in the pool when the work is complete. Stateless sessions beans provide faster performance than stateful beans.	Each client creates a new instance of a bean, and eventually removes it. Instances may be passivated to disk if the cache fills up. An application issues an `ejbRemove()` to remove the bean from the cache. Stateful sessions beans do not perform as well as stateless sessions beans.
Have no identity and no client association; they are anonymous.	Are bound to particular client instances.Each bean has an implicit identity. Each time a client interacts with a stateful session bean during a session, it is the same object.
Do not persist. The bean has no state between calls.	Persist. A stateful session bean's state is preserved for the duration of a session.

Stateful Session EJB Creation

No stateful session EJB instances exist in WebLogic Server at startup. Before a client begins accessing a stateful session bean, it creates a new bean instance to use during its session with the bean. When the session is over the instance is destroyed. While the session is in progress, the instance is cached in memory.

Choosing Between Stateless and Stateful Beans

Stateless session beans are a good choice if your application does not need to maintain state for a particular client between business method calls. WebLogic Server is multi-threaded, servicing multiple clients simultaneously. With stateless session beans, the EJB container is free to use any available, pooled bean instance to service a client request, rather than reserving an instance for each client for the duration of a session. This results in greater resource utilization, scalability and throughput.

Stateless session beans are preferred for their light-weight implementation. They are a good choice if your application's beans perform autonomous, distinct tasks without bean-to-bean interaction.

Stateful session beans are a good choice if you need to preserve the bean's state for the duration of the session.

For examples of applications of stateless and stateful session beans, see Stateless Session Beans and Stateful Session Beans.

1. **Reverse a String using String builder :**

```
String x = “Value1”;
StringBuilder x1 = new StringBuilder();
x1.append(x);
x2.reverse();
```

2. **Find max element from array**

Java 8 provides streams.

We can use the max method

For eg from java.util.Arrays

```
Int[] x= new int[5]{1,2,3,4,5,};
Int max = Arrays.stream(x).max().getAsInt();
```

String

1. **Immutable:** Once created, the content of a `String` object cannot be changed. Any operation that seems to modify a `String` actually creates a new `String` object. This immutability ensures thread safety and allows strings to be used as constants.
2. **Performance:** Due to immutability, operations like concatenation (`+` operator or `concat()` method), substring extraction, and conversion to uppercase or lowercase involve creating new `String` objects. This can lead to inefficiency when performing a large number of such operations.
3. **Usage:** `String` is suitable when the content of the string will not change frequently, such as storing constants, representing fixed strings in the program, or when thread safety is required without explicit synchronization.

StringBuffer and StringBuilder

1. **Mutable:** Both `StringBuffer` and `StringBuilder` are mutable, meaning their content can be changed after creation. They provide methods to modify the content of the string directly without creating new objects, which can improve performance for operations involving frequent modifications.
2. **Synchronization:** `StringBuffer` is synchronized, meaning it is thread-safe. This synchronization ensures that multiple threads can safely manipulate a `StringBuffer` object concurrently. On the other hand, `StringBuilder` is not synchronized, making it faster but not thread-safe.
3. **Performance:** Use `StringBuilder` for better performance in single-threaded environments where synchronization is not needed. `StringBuilder` is generally faster than `StringBuffer` because it is not synchronized. `StringBuffer` is preferable in multi-threaded scenarios or when thread safety is required.

Java Collections :

Java Collections is a framework that provides a set of classes and interfaces to store and manipulate a group of objects. It is part of the Java Standard Edition (Java SE) and is found in the `java.util` package. Here are some key points about Java Collections:

The java collections has several **interfaces**

`Collection`: The root interface of the collection hierarchy. All the methods are taken from this interface as every other interface extends this interface except **Map** and **HashMap.**

`List`: An ordered collection (also known as a sequence). It allows duplicate elements. Examples are `ArrayList`, `LinkedList`, and `Vector`.

`Set`: A collection that does not allow duplicate elements. Examples are `HashSet`, `LinkedHashSet`, and `TreeSet`.

`Queue`: A collection used to hold multiple elements prior to processing. Examples are `PriorityQueue` and `LinkedList`.

`Map`: An object that maps keys to values. A map cannot contain duplicate keys. Examples are `HashMap`, `TreeMap`, and `LinkedHashMap`

It also has several **Classes**

`ArrayList`: A resizable array(Dynamic) implementation of the `List` interface.
`LinkedList`: A doubly linked list implementation of the `List` and `Queue` interfaces.
`HashSet`: A hash table-backed implementation of the `Set` interface.
`TreeSet`: A NavigableSet implementation based on a TreeMap.
`HashMap`: A hash table-based implementation of the `Map` interface.
`TreeMap`: A Red-Black tree-based implementation of the `NavigableMap` interface

Why collections?

1. Ease of use
2. Reusability
3. Performance
4. Consistency

Eg: Create an ArrayList for students. Perform operations like add,addall , remove,removeall

When to use ArrayList and LinkedList?

ArrayList can be used for scenarios where reading is extensive than writing as elements can be directly accessed via their indice.

LinkedList can be used for scenarios where writing is extensive than reading as elements dont have to be shifted in memory when inserting or deleting an element. Only the address has the be updated in a node.

We can use Collections.sort() to sort an arrayList

What are Vectors

Vectors are just like array list and they extend from the collection interface.

But vectors are threadsafe meaning vectors are synchronized Multiple threads can safely access a vector. ArrayList is not threadsafe

This also means that vector is slower than arrayList as synchronization means it may cause overheads and therefore vectors are slower than arrayList.

When adding vectors, it doubles itself in size whereas ArrayList increases its size only by 50%

Lambda Expression in Java :

Lambda was introduced in java 8 and its purpose is to represent single abstract methods of an interface in a concise way.

You dont have to write everything from defining the object then writing the concrete body.

Instead

()->{};

Runnable r = () ->{};

Serialization In Java :

Serialization in java is the process of converting an object into a byte stream, therby making it possible to save the objects state to a file or transmit it over a network.

Reverse is known as deserialization .

How is it implemented in java.

Serialization is implemented in java using the Serializable Interface.

Any class that implements the serializable class will be serialized and all the contents inside the class will be serialized during runtime. This interface is a markerInterface.

ObjectOutputStream is used to write an object to an OutputStream
ObjectInputStream is used to read an object from InputStream

Serial Version UID is a unique identifier for a class that is serialized/ If it is not declared, jvm will generate one during runtime

Anything that you dont want to be serialized will be marked transient