Java feature:

Oops

Platform independent : as it worked on write once and run anywhere , WORA principle , as the java code is compiled in byte code and can be run into any any device who is equipped with the jvm, make it highly portable .

Easy to learn: as it remove complex things like pointer and multiple inheritance , it makes easier to beginner to choose to learn

Robust

Highly secure : as it provide bytecode verification and like others thing , it makes it highly secure

Multithreaded

Distributed: its designed for the distributed environment

Heap vs stack in java

**Stack Memory**

* **Purpose:** Stores method calls, local variables, and references to objects in the heap.
* **Structure:** Last-In-First-Out (LIFO).
* **Scope:** Each thread has its own stack.
* **Lifecycle:** Memory is allocated when a method is invoked and deallocated when the method ends.
* **Speed:** Very fast due to its simple structure.
* **Error Risk:** Can throw StackOverflowError if too many nested calls occur.

**Heap Memory**

* **Purpose:** Stores all Java objects and class instances.
* **Structure:** Shared across all threads.
* **Lifecycle:** Objects remain in memory until they're no longer referenced and garbage collected.
* **Speed:** Slower than stack due to dynamic allocation and garbage collection.
* **Error Risk:** Can throw OutOfMemoryError if memory is exhausted.

Access specifier :

Priovate : accessible in same class

Default: same class and same package

Protected: same classand subclass fromanother package

Public : accessed from anywhere

20. Explain the elements in public static void main .

The public static void main(String[] args) method is the **entry point** of any Java application. Each keyword in this method has a specific role:

**Breakdown of Elements**

1. **public** → Access modifier that allows the method to be called from anywhere.
2. **static** → Allows the method to be called without creating an instance of the class.
3. **void** → Specifies that the method does not return any value.
4. **main** → The name of the method that JVM looks for to start execution.
5. **String[] args** → An array of command-line arguments passed to the program.

22. What is the difference between instance and local variable?

**Instance Variable**

* **Declared inside a class but outside any method.**
* **Scope:** Available throughout the class and accessible by all methods.
* **Lifecycle:** Exists as long as the object exists.
* **Default Values:** Gets default values (e.g., 0 for integers, null for objects).

**Local Variable**

* **Declared inside a method, constructor, or block.**
* **Scope:** Limited to the method or block where it's declared.
* **Lifecycle:** Created when the method starts and destroyed when it ends.
* **Default Values:** Must be initialized before use.

24. Difference between System.out, system.err, System.in?

1. **Standard Input:** This is used to feed the data to user's program and usually a keyboard is used as a standard input stream and represented as System.in.
2. **Standard Output:** This is used to output the data produced by the user's program and usually a computer screen is used for standard output stream and represented as System.out.
3. **Standard Error:** This is used to output the error data produced by the user's program and usually a computer screen is used for standard error stream and represented as System.err.

why strings are immuatable in java

1. **Security**

Strings are commonly used to represent sensitive things like file paths, network connections, and database credentials. If a string could be changed after being created, malicious code could alter it and gain unintended access. Immutability makes this impossible.

2. **Hashing & Collections**

Strings are often used as keys in hash-based collections like HashMap or HashSet. Since the hash code of a string is based on its contents, if a string could change, it would break the hashing mechanism. Immutable strings ensure stable behavior in these collections.

3. **Thread Safety**

In multithreaded environments, shared data can lead to bugs if multiple threads modify it at the same time. Since strings can't change, they are naturally thread-safe—no synchronization is needed to use them concurrently.

4. **Performance Optimization via String Pooling**

Java maintains a **String pool** in memory. When you create a string like "Hello", it checks if that value already exists in the pool. If it does, it reuses the same object rather than creating a new one. This saves memory and improves performance—but only works because strings are immutable.

1. can we throw any exception from finally block?

Yes , we can , but generally not preferred.

1. do we have any alternative for the usage of finally?

Yes , try with resources

**How It Works**

Instead of manually closing resources in a finally block, Java’s **try-with-resources** automatically calls .close() on any object that implements **AutoCloseable or Closeable**.

Example: Reading a File Using Try-With-Resources

try (BufferedReader reader = new BufferedReader(new FileReader("example.txt"))) {

System.out.println(reader.readLine()); // Reads first line of the file

} catch (IOException e) {

System.out.println("Error: " + e.getMessage());

}

What will happen if we apply method override in same class

You will get the compilation error .

How to define variable in Interface ( What will be keyword for declaring variable in interface )

In Java, **variables in an interface are always implicitly public, static, and final**. You don’t need to specify these modifiers explicitly, but doing so improves code clarity

interface ExampleInterface {

int CONSTANT\_VALUE = 100; // Implicitly public, static, and final

}

What is difference between Wait and Stop

Wait : it used to hault the execution of any running thread , and wait for notify and notifyAll method

Its in object class

Stop: while this method used to terminate the execution.its in thread class.

6. What are the main interfaces in the Java Collection Framework?

1. **Collection**

* The root interface for most of the framework.
* Subinterfaces include List, Set, and Queue.

2. **List**

* An **ordered collection** (also known as a sequence).
* Allows duplicate elements.
* Examples: ArrayList, LinkedList, Vector.

3. **Set**

* A collection that **does not allow duplicates**.
* Examples: HashSet, TreeSet, LinkedHashSet.

**Queue**

* Designed for **holding elements prior to processing**, typically in a FIFO order.
* Examples: PriorityQueue, ArrayDeque.

5. **Deque**

* A double-ended queue that allows elements to be added or removed from either end.
* Extends Queue.

2. Provide a small example for Comparable and Comparator in Java.

List<User> sorted = users.stream()

.sorted() // Uses compareTo

.collect(Collectors.toList());

List<User> sortedByName = users.stream()

.sorted(Comparator.comparing(User::getName))

.collect(Collectors.toList());

4. Where are you using Functional Interfaces?

Functional Interfaces in Action

1. **Comparator Interface**
   * Comparator is a **functional interface** because it has a single abstract method: compare(T o1, T o2).
   * When you write something like:

Comparator<User> byName = Comparator.comparing(User::getName);

 **Streams API + Lambdas**  
Whenever you see a method like .sorted(), .filter(), .map(), etc., you're most likely using functional interfaces such as:

* Function (e.g., for .map(x -> x.toUpperCase()))
* Predicate (e.g., for .filter(x -> x.startsWith("A")))
* Consumer (e.g., for .forEach(System.out::println))

List<String> names = List.of("Alice", "Bob", "Charlie");

List<String> filtered = names.stream()

.filter(name -> name.startsWith("A")) // Predicate<String>

.map(String::toUpperCase) // Function<String, String>

.collect(Collectors.toList());

1. How do you interrupt a thread?

To interrupt a thread in Java, you use the interrupt() method provided by the Thread class. This is especially useful when you want to signal a thread to stop what it's doing—like breaking out of a sleep(), wait(), or a long-running loop.

Diff between concurrent list and synchronized list

Synchronized List

* Created using Collections.synchronizedList()
* Thread-safe by synchronizing every method call
* Single lock on the entire list—only one thread can access it at a time
* Manual synchronization is needed during iteration
* Performance bottleneck in high-concurrency scenarios due to coarse-grained locking

Concurrent List (CopyOnWriteArrayList)

* Thread-safe without external synchronization
* Uses a copy-on-write strategy: every write (add/remove) creates a new copy of the list
* Safe iteration without locking—even while other threads modify the list
* Best for read-heavy workloads where writes are infrequent

16. What is concurrency in java?

**Concurrency in Java** refers to the ability of a program to execute multiple tasks at the same time. It’s a powerful feature that allows Java applications to make better use of system resources—especially on multi-core processors

In Java

Java supports concurrency primarily through:

* **Threads**: Lightweight units of execution within a program.
* **Runnable / Callable**: Interfaces that define tasks to be run by threads.
* **Executor Framework**: Manages thread pools and task execution.
* **java.util.concurrent package**: Provides high-level concurrency utilities like ExecutorService, Future, CountDownLatch, Semaphore, etc.

hashmap vs concurrent hashmap

hashmap: , not thread satfy need to apply synchronization manually , allows one null as a key , and n number of null as a value ,suitable for single threaded environment .

conc hashmap: thread safety , use internal locking,does not allow null key and value , better suited for concurrent access.

9. Difference REST API and SOAP

**1. Architecture Style vs Protocol**

* **REST (Representational State Transfer):** Architectural style that uses standard HTTP methods (GET, POST, PUT, DELETE).
* **SOAP (Simple Object Access Protocol):** A strict protocol that uses XML for messaging, typically over HTTP but also other protocols like SMTP.

📄 **2. Data Format**

* **REST:** Flexible—supports JSON, XML, HTML, or even plain text.
* **SOAP:** Only XML (more verbose and rigid).

🛠️ **3. Flexibility & Ease**

* **REST:** Lightweight, simple to implement, easily consumed by browsers and mobile apps.
* **SOAP:** Heavyweight—requires more setup and tools (WSDL, XML Schema).

🔒 **4. Security**

* **REST:** Relies on HTTPS and can implement OAuth or JWT for token-based security.
* **SOAP:** Has built-in security via WS-Security (encryption, signing, authentication).

🧪 **5. Use Case Suitability**

* **REST:** Ideal for public APIs, mobile apps, and CRUD operations.
* **SOAP:** Better for enterprise-level services like banking, payment gateways where strict security and transactional support are needed.

📘 **6. Standardization**

* **REST:** Less standardized; interpretation varies by developer.
* **SOAP:** Highly standardized with formal contracts via WSDL.

🧩 Real-World Analogy

Think of **REST** like ordering food at a restaurant from a menu—simple, quick, pick and go. **SOAP** is more like a formal banquet—structured, with invitations, protocols, and a dress code.

Would you like to see how the same API might be implemented in both styles side by side?

** REST:** Lightweight, simple to implement, easily consumed by browsers and mobile apps.

* **SOAP:** Heavyweight—requires more setup and tools (WSDL, XML Schema).

🔒 **4. Security**

* **REST:** Relies on HTTPS and can implement OAuth or JWT for token-based security.
* **SOAP:** Has built-in security via WS-Security (encryption, signing, authentication).

🧪 **5. Use Case Suitability**

* **REST:** Ideal for public APIs, mobile apps, and CRUD operations.
* **SOAP:** Better for enterprise-level services like banking, payment gateways where strict security and transactional support are needed.

1. What are the different parts of a payload in an HTTP request?

url , header , requestBody , response body

The lifecycle of a Spring bean consists of the following phases, which are listed below

* **Container Started:**The Spring IoC container is initialized.
* **Bean Instantiated:** The container creates an instance of the bean.
* **Dependencies Injected:**The container injects the dependencies into the bean.
* **Custom init() method:** If the bean implements InitializingBean or has a custom initialization method specified via @PostConstruct or init-method.
* **Bean is Ready:**The bean is now fully initialized and ready to be used.
* **Custom utility method:** This could be any custom method you have defined in your bean.
* **Custom destroy() method:** If the bean implements DisposableBean or has a custom destruction method specified via @PreDestroy or destroy-method, it is called when the container is shutting down

How would ensures high availability in distributed java application:

Use different load balancers tenchnique like nginx and microservices load balancer

Use api gateway for the centralized access

Use microservices along with its different patterns like circuit breaker and other

Use event driven architecture to reduce load.

Use monitoring tools like Prometheus , grafna and actuator

Use archestration and dockerization.

how would you acces millions of request per seconds in java spring boot

we can use faster dbs like pg and maria and mongo db

we can inmplement reactive programming and asynchronous approach

can use load balancer and other techniques.

Perform monitoring

Use batch processing

diff between controller advice and restcontroller advice

same as like controller and restcontroller

key component of spring boot:

treeset vs linked hashset

treeset: no insertion order maintain

does not allow duplicates

does not allow null values

slower than linkedhashset

automatically sort element in natural order

linkedhashset:

insertion order preserved

no duplicates allowed

only one null value allows

faster than treeset

advantages and disadvantages of microservices

advantages:

less pressure on project

each team can acquire and own different service and take full authority

better fault isolation , cause affect in one cant affect in another if we make arrangements for that

we can independently update and deploy it

disadvantage

project structure becomes complex

testing will be hard , and some time , finding where actual , bug occurs , some time this also become hard

as here each services can have different db , so managing it becomes hard some time

it can be costly if not managed properly.

Double check block

Here checking of core logic twice

public static Singleton getInstance() {

if (instance == null) { // First check (no locking)

synchronized (Singleton.class) {

if (instance == null) { // Second check (with locking)

instance = new Singleton();

}

}

}

return instance;

why comparable isn’t count as a functional interface ?

first of all it has only one abstract method like compareTo and one default method like reversed() , but the main fact is like that its not meant for the behavioural like lambda , it like for natural ordering

pass by value and pass by reference

pass by value:

java is always pass by value means copy of variable value is passed to the method

pass by reference :

inplace of copy of object value , the actual reference of object is passed at method calling , which was supported in c++.

Key annotation is used in serialization and desererialization

jsonProperty: used to maps to field

jsoninclude: used to tell which property should be imported and having full control over it

jsonIgnore : it used to exclude fields

jsonCreator:used to marks method as a constructor for deserelization.

jsonDeserialized and jsonSerialized: it used for providing custom logic for the serialization and deserialization

how transaction managed by spring boot

using @Transactional annotation also by providing rollback rules

@Transactional(rollbackFor = Exception.class)

// Will roll back even for checked exceptions

And extending jpaRepository will automatically managed transaction internally

what happen if we dont declare the main method static

compilation will be , but execution fails , cause to call main method , jvm needs to create instance of class , to call method , but jvm not does that.

are wrapper class final and immutable

yes.

suppose you need to send a java object over a network in distributed application , what challenges will you face and how would you tackle

if we want to send the java object over a distributed network , we need to serialize it .

we need to send it along with encryption if it hold some valuable information ,

along with the serial uuid .

and for that we should mostly use DTOs.