









1. INTERNAL WORKING OF HASH MAP


HashMap in Java

-  **Definition:** A HashMap in Java is a data structure that stores key-value pairs, allowing for Constant-time performance for basic operations like adding, retrieving, and removing elements. It's part of Java Collections Framework and implements the Map interface.

Internal Working:

-  **Hashing:** When you add a key-value pair to a HashMap, the key's hash code is calculated using its hashCode() method, representing its uniqueness.
-  **Bucket Allocation:** The hash code determines which bucket the pair will be stored in. Buckets hold multiple entries, and the number of buckets is set by the initial capacity of the HashMap.
-  **Collision Handling:** Since multiple keys can have the same hash code, collisions can occur. HashMap handles this by forming linked lists of entries with the same hash code.
-  **Entry Insertion:** New entries are added to the HashMap. If there's a collision, the new entry joins the linked list in the corresponding bucket.
-  **Entry Retrieval:** When you search for a value using a key, the key's hash code helps locate the correct bucket quickly. Then, the linked list in that bucket is traversed to find the matching entry.
-  **Resize and Rehash:** If the number of entries increases beyond a certain threshold (the load factor), the HashMap automatically doubles its number of buckets. All entries are rehashed and redistributed to new buckets during this process.
-  **Performance:** HashMap's performance relies on a well-distributed distribution of hash codes to minimize collisions. With proper implementation of hashCode() and equals() methods in key objects, HashMap offers constant-time complexity ($O(1)$) for most operations on average.

2. LOAD FACTOR

-  The load factor in Java is a parameter used in hash-based data structures like HashMap and HashSet. It determines when the data structure will resize its internal array. The default load factor

is 0.75, which means the data structure resizes when it is 75% full. Choosing a higher load factor reduces resizing frequency but may impact performance, while a lower load factor reduces memory usage but increases resizing operations. It's important to select an appropriate load factor based on your application's requirements.

3. FUNCTIONAL INTERFACE

- ✚ a functional interface in Java is an interface that has **exactly one abstract method**.
- ✚ It is used to facilitate functional programming by representing and passing behavior as a parameter.
- ✚ Functional interfaces are often used with lambda expressions and method references. Annotating a functional interface with the `@FunctionalInterface` annotation helps validate its requirements and serves as documentation.
- ✚ Java provides predefined functional interfaces in the `java.util.function` package, and they are widely used in the Stream API for performing functional-style operations on collections. Using functional interfaces allows for code reusability, modularity, and separation of concerns.

4. JAVA SINGLETON CLASS

- ✚ **What:** In object-oriented programming, a java singleton class is a class that can have only one object (an instance of the class) at a time. After the first time, if we try to instantiate the Java Singleton classes, the new variable also points to the first instance created.

Remember the key points while defining a class as a singleton class that is while designing a singleton class:

- Make a constructor private.
 - Write a static method that has the return type object of this singleton class. Here, the concept of Lazy initialization is used to write this static method.
 - Purpose of Singleton Class
- ✚ The primary purpose of a java Singleton class is to restrict the limit of the number of object creations to only one. This often ensures that there is access control to resources, for example, socket or database connection.
 - ✚ Memory space wastage does not occur with the use of the singleton class because it restricts instance creation. As the object creation will take place only once instead of creating it each time a new request is made.

5. INTERFACE

- ✚ Interfaces are a way to achieve abstraction in Java.

- ✚ Interfaces are collections of abstract methods and constants.
- ✚ Classes implement interfaces by providing implementations for their abstract methods.
- ✚ **(in java 8) Interfaces can have default and static methods**
- ✚ Interfaces can extend other interfaces.
- ✚ Interfaces can be used as marker interfaces.
- ✚ Interfaces can be functional interfaces with a single abstract method.
- ✚ In short, interfaces are a powerful tool that can be used to achieve abstraction, polymorphism, and multiple inheritance in Java.
- ✚ Here are some of the key benefits of using interfaces in Java:
 - ✚ Abstraction: Interfaces allow you to abstract away the implementation details of a class, making it easier to use and understand.
 - ✚ Polymorphism: Interfaces allow you to use polymorphism, which means that you can treat different objects of different types in the same way.
 - ✚ Multiple inheritance: Interfaces allow you to achieve multiple inheritance, which means that a class can inherit from multiple interfaces.

6. ABSTRACT CLASSES AND INTERFACES

- ✚ Abstract classes are similar to regular classes, but they can contain abstract methods, which are methods that do not have an implementation. This means that abstract classes cannot be instantiated, but they can be extended by other classes. Abstract classes are often used to provide a base class for other classes, or to define a common interface that multiple classes can implement.
- ✚ Interfaces are a special type of class that only contains abstract methods and constants. Interfaces **cannot be instantiated, but they can be implemented by other classes**. Interfaces are often used to define a contract that different classes must agree to follow. This can be useful for ensuring that different classes can interact with each other in a consistent way.

7. MARKER INTERFACES

- ✚ Interfaces in Java provide a way to achieve abstraction, encapsulation, and polymorphism. They allow for loose coupling between components and facilitate code reuse and modularity. Interfaces are extensively used in Java APIs and frameworks, enabling developers to define and implement different behavior while adhering to a common contract.

- ✚ marker interfaces in Java are used for adding metadata or indicating specific characteristics to classes at runtime. They enable conditional behavior, integration with frameworks, and serve as a form of documentation. Marker interfaces provide flexibility, extensibility, and compile-time validation in certain cases. They are lightweight and offer a way to enhance code readability and communicate intent.

8. THE STATIC KEYWORD

- ✚ In Java, the static keyword is used to define a member (variable or method) that belongs to the class itself, rather than to an instance of the class. It has the following implications:
- ✚ Static Variables: A static variable, also known as a class variable, is shared among all instances of a class. It is initialized only once when the class is loaded into memory and remains the same for all instances. Static variables are declared with the static keyword and can be accessed using the class name.
- ✚ Static Methods: A static method belongs to the class itself and can be invoked without creating an instance of the class. Static methods are useful for utility or helper methods that do not rely on instance-specific data. They can only directly access static variables or other static methods.
- ✚ Static Blocks: A static block is a block of code enclosed in curly braces and preceded by the static keyword. It is used to initialize static variables or perform other one-time operations when the class is loaded. Static blocks are executed before the execution of any instance methods or the creation of any instances.

9. OOPS CONCEPT IN JAVA

- ✚ Object-Oriented Programming (OOP) is a programming paradigm that organizes software design around objects, which can represent real-world entities or abstract concepts. Java is an object-oriented programming language, and it supports the core principles of OOP. Here are the key OOP concepts in Java:
- ✚ Class: A class is a blueprint or template that defines the properties (attributes) and behaviors (methods) of objects. It serves as a blueprint for creating objects.

- ✚ Object: An object is an instance of a class. It represents a specific entity or instance that possesses the attributes and behaviors defined by its class.
- ✚ Encapsulation: Encapsulation is the practice of hiding the internal details of an object and providing access to only the necessary information through methods. It helps in achieving data abstraction and protects data integrity.
- ✚ Inheritance: Inheritance is a mechanism that allows a class to inherit the properties and methods of another class. The class that inherits from another class is called a subclass (or derived class), and the class being inherited from is called a superclass (or base class). Inheritance promotes code reuse and supports the "is-a" relationship.
- ✚ Polymorphism: polymorphism in Java refers to the ability of an object to take on multiple forms. It allows objects of different classes to be treated as objects of a common superclass or interface. Polymorphism is achieved through method overriding and method overloading. It enables code reusability, modularity, and the ability to work with objects of different types in a unified manner.
- ✚ Abstraction: , abstraction in Java simplifies and highlights relevant characteristics of real-world entities. It is achieved through abstract classes and interfaces. Abstract classes provide a blueprint for subclasses, allowing for partial implementation. Interfaces define a set of rules for classes to implement, providing full abstraction. Abstraction promotes code modularity, reusability, and separation of concerns, helping manage complexity and design scalable software systems.
- ✚ Abstraction in Java, along with interfaces, is essential for code reusability, providing a base implementation that subclasses can inherit. Abstract classes also allow partial implementation, define common functionality, enable state and data management, offer flexibility in design, and support polymorphism and inheritance. Abstraction complements interfaces by adding more features and flexibility to object-oriented programming.
- ✚ Association: Association represents the relationship between two or more objects. It can be a one-to-one, one-to-many, or many-to-many relationship. Objects can interact with each other through associations.

- ✚ Composition: Composition is a stronger form of association where one class (the whole) contains objects of another class (the part). The objects' lifecycles are linked, and the part objects cannot exist independently of the whole.
- ✚ Aggregation: Aggregation is a weaker form of association where one class (the container) has a relationship with another class (the contained), but the contained objects can exist independently of the container.
- ✚ Overloading and Overriding: Overloading refers to the ability to have multiple methods with the same name but different parameter lists in a class. Overriding occurs when a subclass provides a different implementation of a method defined in its superclass.

10. ABSTRACTION IN JAVA

- ✚ Code Reusability: Abstract classes promote code reuse by providing a base implementation that subclasses can inherit.
- ✚ Common Functionality: Abstract classes define common methods, fields, and behavior that subclasses can inherit, ensuring consistency and modular code.
- ✚ Partial Implementation: Abstract classes allow for partial implementation of functionality, with subclasses overriding and extending abstract methods.
- ✚ Flexibility in Design: Abstraction classes strike a balance between concrete classes and interfaces, providing flexibility in class design.
- ✚ State and Data Management: Abstract classes can hold state and manage data within the class hierarchy.
- ✚ Disadvantages of Abstraction Classes (in short):
 - ✚ Limited Inheritance: Subclasses can only inherit from a single abstract class, unlike interfaces that support multiple inheritance.
 - ✚ Tighter Coupling: Subclasses are tightly coupled to the abstract class, which may introduce dependencies and constraints.
 - ✚ Limited Interface Implementation: Abstract classes can only be inherited, limiting their usage compared to interfaces that can be implemented by unrelated classes.

- ✚ Complexity: Abstract classes can add complexity, especially in complex inheritance hierarchies.

11. CAN WE CREATE AN OBJECT OF ABSTRACTION CLASS

- ✚ No, we cannot create objects of abstract classes directly in Java. Abstract classes are meant to be extended by subclasses, and they serve as a blueprint or template for those subclasses. The purpose of an abstract class is to provide common functionality and define abstract methods that must be implemented by its subclasses.

12. WHAT IS JAVA?

- ✚ Java is a high-level, object-oriented programming language developed by Sun Microsystems (now owned by Oracle). It is platform-independent, meaning that Java programs can run on any platform with a Java Virtual Machine (JVM).

13. WHAT ARE THE MAIN FEATURES OF JAVA?

- ✚ Object-oriented: Java supports the principles of encapsulation, inheritance, and polymorphism.
- ✚ Platform-independent: Java programs can run on any platform with a JVM.
- ✚ Robust: Java includes features like strong type checking, automatic memory management (garbage collection), and exception handling.
- ✚ Multi-threaded: Java supports concurrent programming with built-in thread support.
- ✚ Secure: Java includes security features to protect against vulnerabilities.

14. WHAT IS THE DIFFERENCE BETWEEN A CLASS AND AN OBJECT IN JAVA?

- ✚ A class is a blueprint or template that defines the properties and behaviors of objects. It defines the structure and behavior that objects of that class will have.
- ✚ An object is an instance of a class. It represents a specific entity created using the class blueprint and has its own state and behavior.

15. WHAT IS THE DIFFERENCE BETWEEN == AND .EQUALS() IN JAVA?

- ✚ The == operator is used for comparing primitive types and object references. It checks whether two variables refer to the same memory location.
- ✚ The .equals() method is used for comparing the content or values of objects. It is typically overridden by classes to provide a customized comparison.

16. CONTAINERS

- ✚ containers in Java are classes or interfaces that hold and manage multiple objects. They provide data structures for organizing and manipulating collections of data. Common container classes include List, Set, Map, Queue, and Stack, which offer different ways of storing and accessing elements. Containers in Java provide convenient methods for adding, removing, and manipulating objects within the data structure. They are part of the Java Collections Framework and facilitate efficient data management in Java programs.

17. WHAT ARE THE ACCESS MODIFIERS IN JAVA?

- ✚ Java has four access modifiers: public, private, protected, and default (no modifier).
- ✚ public: Accessible from anywhere.
- ✚ private: Accessible only within the same class.
- ✚ protected: Accessible within the same class, subclasses, and the same package.
- ✚ Default: Accessible within the same package.

18. WHAT IS THE DIFFERENCE BETWEEN FINAL, FINALLY, AND FINALIZE IN JAVA?

final:

- ✚ final is a keyword in Java used to define entities that cannot be changed or overridden.
- ✚ When applied to a variable, it indicates that the variable's value cannot be modified once assigned.
- ✚ When applied to a method, it signifies that the method cannot be overridden in subclasses.
- ✚ When applied to a class, it indicates that the class cannot be subclassed.

finally:

- ✚ finally is a block in Java's exception handling mechanism used to specify code that will be executed regardless of whether an exception is thrown or not.
- ✚ It is typically used in conjunction with try and catch blocks.
- ✚ The code within the finally block will always execute, even if an exception occurs or if a return statement is encountered.

- ✚ **finalize:**

- ✚ finalize is a method defined in the Object class, which is the root class for all Java classes.
- ✚ It is called by the garbage collector before reclaiming the memory occupied by an object that is eligible for garbage collection.
- ✚ The finalize method can be overridden in a class to perform cleanup operations or release resources before the object is garbage collected.
- ✚ However, the use of finalize is generally discouraged as it is not guaranteed to be called promptly or at all.

19. AUTHENTICATION AND AUTHORIZATION

- ✚ Authentication and authorization are two essential concepts in the realm of security and access control within a software system.
- ✚ Authentication:
 - ✚ Authentication is the process of verifying the identity of a user or entity accessing a system or resource. It ensures that the claimed identity is valid and matches the credentials provided. Authentication typically involves validating the authenticity of the user's credentials, such as username and password, against stored user information or an authentication service. The goal of authentication is to ensure that only legitimate and authorized users are granted access to the system.
- ✚ Common authentication mechanisms include:
 - ✚ Username and password authentication: Users provide a unique username and a corresponding password to prove their identity.
 - ✚ Multi-factor authentication (MFA): Users provide multiple pieces of evidence to authenticate themselves, such as a combination of passwords, biometrics, security tokens, or one-time verification codes.
 - ✚ Single sign-on (SSO): Users authenticate once and gain access to multiple interconnected systems or applications without providing credentials for each one individually.
- ✚ Authorization:
 - ✚ Authorization, on the other hand, is the process of granting or denying access permissions to authenticated users or entities based on their roles, privileges, or attributes. It determines what actions or resources a user is allowed to access, modify, or perform within the system.

Authorization is typically based on predefined access control rules or policies that are enforced by the system.

- ✚ Authorization mechanisms include:
- ✚ Role-based access control (RBAC): Users are assigned specific roles, and access permissions are associated with those roles. Users are granted access based on their assigned roles.
- ✚ Attribute-based access control (ABAC): Access decisions are based on various attributes or characteristics of the user, such as their job title, department, location, or other user-specific attributes.
- ✚ Access control lists (ACL): Specific access permissions are defined for individual users or groups, listing which resources they can access and what actions they can perform.

20. TRANSACTIONS

- ✚ transactions are used to ensure the atomicity, consistency, isolation, and durability (ACID) properties of a group of database operations
- ✚ Transactions are a way to group multiple database operations together as a single unit of work.
- ✚ Transactions ensure that either all operations within the transaction are successfully completed or none of them are applied.
- ✚ Transactions can be implemented in Java using the JDBC API, JPA, or frameworks like Spring and Hibernate.
- ✚ Transactions are important for ensuring data integrity, consistency, and reliability when working with databases.
- ✚ Here are some of the benefits of using transactions in Java:
- ✚ Data integrity: Transactions ensure that data is not corrupted or lost in case of an error.
- ✚ Consistency: Transactions ensure that the database is always in a consistent state.
- ✚ Reliability: Transactions ensure that database operations are performed reliably, even in the event of an error.
- ✚ Performance: Transactions can improve performance by grouping multiple database operations together.

21. THE TERMS "COLLECTION" AND "COLLECTIONS" IN JAVA REFER TO DIFFERENT CONCEPTS:

- ✚ Collection: Collection (note the singular form) is an interface in the java.util package that represents a group of objects. It is the root interface of the Java Collections Framework and defines the basic operations and behaviors that all collection classes should support, such as adding, removing, and iterating over elements.
- ✚ Collections: Collections (note the plural form) is a utility class in java.util package that provides various static methods for working with collections. It contains methods for performing common operations on collections, such as sorting, searching, and synchronizing. These utility methods can be used with any class implementing the Collection interface.

22. WHAT IS A CONCERT HASH MAP

23. HOW TO HIDE CLASS IN SPRING BOOT

24. CONCURRENTHASHMAP

- ✚ ConcurrentHashMap is a class in Java's java.util.concurrent package that provides a thread-safe implementation of the Map interface.
- ✚ It is designed to be used in multi-threaded environments where multiple threads can access and modify the map concurrently.
- ✚ Unlike the traditional HashMap class, which is not thread-safe and can cause issues when accessed by multiple threads simultaneously, ConcurrentHashMap ensures thread safety without the need for external synchronization.
- ✚ It achieves this by internally partitioning the map into segments, with each segment having its own lock. This allows concurrent read and write operations to different segments of the map to proceed simultaneously, improving performance.
- ✚ Key features of ConcurrentHashMap include:

- ✚ Thread safety: Multiple threads can perform read and write operations concurrently without the need for external synchronization.
- ✚ Scalability: The map is internally divided into segments, allowing multiple threads to operate on different segments concurrently, reducing contention and improving performance.
- ✚ High-performance operations: `ConcurrentHashMap` provides efficient operations for common map operations, such as `put`, `get`, and `remove`, even under concurrent access.
- ✚ Iteration: Iterating over the `ConcurrentHashMap`'s key-value pairs provides a weakly consistent snapshot of the map at the time of iteration, without throwing concurrent modification exceptions.

25. HASHTABLE IS THERE WHY WE NEED CONCURRENTHASHMAP

- ✚ Yes, the `Hashtable` class in Java is another implementation of the `Map` interface that provides thread safety. Both `Hashtable` and `ConcurrentHashMap` offer thread-safe operations, but there are some key differences between them that make `ConcurrentHashMap` preferable in most cases.
- ✚ **Performance:** `ConcurrentHashMap` typically offers better performance than `Hashtable` in concurrent scenarios. `Hashtable` achieves thread safety by synchronizing the entire map, which can lead to contention and decreased performance when multiple threads access the map concurrently. On the other hand, `ConcurrentHashMap` partitions the map into segments, allowing concurrent access to different segments and reducing contention.
- ✚ **Granularity of Locking:** `ConcurrentHashMap` provides fine-grained locking at the segment level, which means multiple threads can read and write to different segments concurrently. In contrast, `Hashtable` uses a single lock for the entire map, which allows only one thread to access the map at a time, even for unrelated keys.
- ✚ **Iteration:** Iterating over a `ConcurrentHashMap` does not require locking the entire map, whereas iterating over a `Hashtable` requires acquiring the lock on the entire map, blocking other threads from accessing it. This can impact the overall scalability and performance of concurrent applications.

- + Null Values: Hashtable does not allow null values or null keys, whereas ConcurrentHashMap permits null values and null keys (though it is generally recommended to avoid using null as a key).

26. WHAT ARE THE DIFFERENT TYPES OF COLLECTIONS AVAILABLE IN JAVA?

- + In Java, the Collection Framework provides several types of collections. The main interfaces that represent these collections are:
 - + **List**: A list is an ordered collection that allows duplicate elements. The main implementing classes are ArrayList, LinkedList, and Vector.
 - + **Set**: A set is a collection that does not allow duplicate elements. The main implementing classes are HashSet, LinkedHashSet, and TreeSet.
 - + **Queue**: A queue is a collection that follows the FIFO (First-In-First-Out) principle. The main implementing classes are LinkedList, PriorityQueue, and ArrayDeque.
 - + **Map**: A map is a collection that stores key-value pairs. The main implementing classes are HashMap, LinkedHashMap, TreeMap, and Hashtable.
- + In addition to these core interfaces, there are other specialized interfaces and classes available in the Collection Framework, such as:
 - + **SortedSet**: An interface that extends Set and maintains its elements in sorted order.
 - + **SortedMap**: An interface that extends Map and maintains its key-value pairs in sorted order.

- ✚ **Deque**: An interface that extends Queue and provides additional operations at both ends of the queue.
- ✚ **NavigableSet**: An interface that extends SortedSet and provides navigation methods, such as finding elements based on their position.
- ✚ **NavigableMap**: An interface that extends SortedMap and provides navigation methods, such as finding entries based on their position.

27. WHAT IS THE DIFFERENCE BETWEEN ARRAYLIST AND LINKEDLIST? WHEN WOULD YOU USE EACH ONE?

- ✚ ArrayList:
 - ✚ Uses a dynamic array to store elements.
 - ✚ Provides constant-time $O(1)$ random access to elements by index.
 - ✚ Performs slower for frequent insertions or deletions in the middle of the list.
 - ✚ Has a slightly higher memory overhead.
- ✚ LinkedList:
 - ✚ Uses a doubly linked list to store elements.
 - ✚ Requires iterating through the list from the beginning or end to reach a specific element, resulting in $O(n)$ time complexity for random access by index.
 - ✚ Performs faster for frequent insertions or deletions, especially at the beginning or middle of the list.
 - ✚ Has a higher memory overhead due to the additional memory needed to maintain the links between elements.
- ✚ Which one should you use?
- ✚ Use ArrayList when:
 - ✚ You need fast random access to elements by index.
 - ✚ You perform more read operations than write operations.
 - ✚ You have a relatively stable or predictable list size, as ArrayList requires occasional resizing.
- ✚ Use LinkedList when:

- ✚ You frequently insert or remove elements, especially at the beginning or middle of the list.
- ✚ You don't require frequent random access to elements by index.
- ✚ You have a dynamic or changing list size.

28. DIFFERENCE BETWEEN TREESET AND LINKEDHASHSET

✚ TreeSet:

- ✚ Internally uses a balanced binary search tree to store elements.
- ✚ Provides a sorted order for its elements.
- ✚ Does not allow duplicate elements.
- ✚ Provides efficient operations for searching, insertion, and deletion of elements.

✚ LinkedHashSet:

- ✚ Internally uses a combination of a hash table and a linked list to store elements.
- ✚ Maintains the order of elements based on their insertion.
- ✚ Does not allow duplicate elements.
- ✚ Provides efficient operations for element access, insertion, and deletion.
- ✚ Which one should you use?

✚ TreeSet: If you need elements to be sorted and efficient searching is important, then TreeSet is a good choice.

✚ LinkedHashSet: If you want to maintain the order of insertion and need fast access to elements, then LinkedHashSet is a good choice.

29. WHAT IS THE DIFFERENCE BETWEEN HASHSET AND TREESET? WHEN WOULD YOU USE EACH ONE?

✚ HashSet:

- ✚ Does not maintain any specific order of elements.
- ✚ Provides faster performance for basic operations like add, remove, and contains (on average) due to hashing.
- ✚ Can be used when fast element lookup and uniqueness are required but specific order is not important.

✚ TreeSet:

- ✚ Keeps elements sorted.
- ✚ Provides slower performance for basic operations like add, remove, and contains but provides efficient operations for maintaining sorted order.
- ✚ Can be used when elements need to be sorted and efficient retrieval of elements in a specific order is important.

30. WHAT IS THE DIFFERENCE BETWEEN HASHMAP AND CONCURRENTHASHMAP? WHEN WOULD YOU USE EACH ONE?

- ✚ Thread-safety: HashMap is not thread-safe, while ConcurrentHashMap is designed to be thread-safe.
- ✚ Performance: HashMap provides better performance in single-threaded environments, while ConcurrentHashMap is optimized for concurrent access and performs better in highly concurrent scenarios.
- ✚ Iteration: HashMap may throw ConcurrentModificationException if modified during iteration, while ConcurrentHashMap supports safe iteration even during modifications.
- ✚ Null Values and Keys: Both HashMap and ConcurrentHashMap allow null values and keys.
- ✚ Use HashMap when you are working in a single-threaded environment or when thread-safety is managed externally, and you don't require concurrent access to the map.
- ✚ Use ConcurrentHashMap when you need thread-safe access to the map, multiple threads will concurrently access and modify the map, or you want better performance in highly concurrent scenarios.

31. HOW DOES THE JAVA COLLECTION FRAMEWORK HANDLE DUPLICATE ELEMENTS?

✚ In short, the Java Collection Framework handles duplicate elements as follows:

- ✚ List implementations allow duplicate elements and preserve their order of insertion.
- ✚ Set implementations do not allow duplicate elements and ensure uniqueness.
- ✚ Queue implementations can handle duplicate elements and maintain their order.
- ✚ Map implementations do not allow duplicate keys, but they can have duplicate values.

32. CAN YOU EXPLAIN THE CONCEPT OF FAIL-FAST AND FAIL-SAFE ITERATORS IN JAVA COLLECTIONS?

✚ Certainly! In Java collections, the terms "fail-fast" and "fail-safe" refer to different iterator behaviors when the underlying collection is modified during iteration:

✚ Fail-Fast Iterators:

- ✚ Fail-fast iterators are the default iterator implementation in most Java collections, including ArrayList, HashSet, HashMap, and others.
- ✚ If a collection is modified structurally (i.e., elements are added or removed) while an iterator is iterating over it, a fail-fast iterator will throw a ConcurrentModificationException.
- ✚ This behavior is a safety mechanism to detect concurrent modifications and avoid potential inconsistencies or data corruption.
- ✚ Fail-fast iterators are designed to provide quick and explicit feedback when the collection's state is modified during iteration.
- ✚ Fail-Safe Iterators:

- ✚ Fail-safe iterators are used in some concurrent collections, such as ConcurrentHashMap and CopyOnWriteArrayList.
- ✚ Unlike fail-fast iterators, fail-safe iterators operate on a snapshot of the collection taken at the time of iteration.
- ✚ If the underlying collection is modified during iteration, fail-safe iterators do not throw a ConcurrentModificationException.

- ✚ Instead, they work on the original copy of the collection, unaffected by modifications, ensuring a consistent iteration.
- ✚ However, fail-safe iterators might not reflect the most recent modifications made to the collection after the iterator was created.
- ✚ Usage scenarios:

- ✚ Fail-fast iterators are commonly used when you want to detect and handle concurrent modifications, ensuring data integrity. They help identify potential issues early.
- ✚ Fail-safe iterators are useful when you need to iterate over a collection while allowing concurrent modifications. They provide a consistent view of the collection at the time of iteration, but might not reflect the latest changes.

33. HOW DOES THE COMPARETO() METHOD WORK IN THE COMPARABLE INTERFACE?

- ✚ the compareTo() method in the Comparable interface is used for comparing objects for natural ordering. It returns an integer value indicating the ordering relationship between the objects:
- ✚ Negative value: this object is less than the compared object.
- ✚ Zero: this object is equal to the compared object.
- ✚ Positive value: this object is greater than the compared object.
- ✚ The compareTo() method is commonly used for sorting objects in a natural order and must be implemented by classes that implement the Comparable interface. It allows objects to be compared based on their intrinsic properties, providing a way to define their relative positions in sorted sequences.

34. WHAT IS THE PURPOSE OF THE HASHCODE() AND EQUALS() METHODS IN JAVA COLLECTIONS?

- ✚ the purpose of the hashCode() and equals() methods in Java collections is as follows:
- ✚ hashCode(): The hashCode() method generates a unique integer value (hash code) for an object. It is used by hash-based collections to determine the storage location of an object. Objects that are equal according to equals() must have the same hash code.

- ✚ equals(): The equals() method compares the equality of two objects. It is used by collections to determine if an object already exists in the collection. By default, it compares object references, but it is often overridden to provide custom equality comparison based on object attributes.
- ✚ Together, these methods allow collections to handle object uniqueness, searching, and retrieval efficiently. They ensure that objects are correctly stored and retrieved in hash-based collections and enable proper equality comparison between objects.

35. EXPLAIN THE CONCEPT OF A SYNCHRONIZED COLLECTION AND WHEN IT IS NECESSARY TO USE IT.

36. WHAT IS THE ROLE OF THE COLLECTIONS CLASS IN JAVA?

- ✚ the Collections class in Java provides utility methods for working with collections. It offers algorithms for sorting, searching, and manipulating collections. It also provides methods for creating immutable and synchronized collections, as well as empty and singleton collections. Additionally, it offers type-safe collections and various other utility methods for common collection operations. The Collections class serves as a central hub for collection-related functionalities in Java.

37. HOW CAN YOU SORT ELEMENTS IN A COLLECTION? EXPLAIN THE DIFFERENT WAYS TO ACHIEVE SORTING.

- ✚ Using the Collections.sort() Method:

- ✚ Implementing the Comparable Interface:
- ✚ Using a Custom Comparator:

- ✚ Using Stream API:
- ✚ Using Sorted Collections:

38. WHAT IS THE DIFFERENCE BETWEEN ARRAYLIST AND VECTOR? WHICH ONE IS PREFERRED IN A MULTI-THREADED ENVIRONMENT?

✚ In short, the difference between ArrayList and Vector in Java is as follows:

✚ Thread Safety:

✚ ArrayList is not thread-safe, meaning it is not synchronized and can lead to data corruption or inconsistencies in a multi-threaded environment.

✚ Vector is thread-safe as it provides synchronized methods, ensuring safe access and modification of elements in a multi-threaded environment.

✚ Performance:

✚ ArrayList performs better in a single-threaded environment due to its non-synchronized nature.

✚ Vector incurs a performance overhead due to the synchronization mechanism, making it slower than ArrayList in most cases.

✚ Growth Rate:

✚ ArrayList by default increases its capacity by 50% when it needs to grow its size.

✚ Vector by default doubles its capacity when it needs to grow its size.

✚ Legacy:

✚ ArrayList is part of the Java Collections Framework introduced in Java 1.2.

✚ Vector existed prior to the Collections Framework and is considered a legacy class.

39. HOW DOES THE JAVA COLLECTION FRAMEWORK HANDLE NULL ELEMENTS?

✚ In short, the Java Collection Framework handles null elements in the following ways:

- ✚ List Implementations (e.g., ArrayList, LinkedList):

- ✚ Lists can contain null elements.

- ✚ Set Implementations (e.g., HashSet, TreeSet):

- ✚ Most set implementations allow a single null element.

HashSet allows one null element, as it uses hashing for element storage.

- ✚ TreeSet does not allow null elements, as it uses a natural ordering or custom comparator that cannot handle null.

- ✚ Map Implementations (e.g., HashMap, TreeMap):

- ✚ HashMap can have a single null key and multiple null values.

- ✚ TreeMap does not allow null keys but can have null values.

- ✚ Null keys or values can be handled by using appropriate methods like put(), get(), and remove().

40. WHAT IS THE PURPOSE OF THE ITERATOR INTERFACE, AND HOW IS IT USED?

- ✚ The **purpose** of the Iterator interface in Java is to provide a way to iterate over elements in a collection, allowing sequential access to the elements and performing operations like retrieval and removal. In short, the Iterator interface serves the following purposes:

- ✚ Iterating over Elements: The Iterator interface allows you to iterate over the elements of a collection, one at a time, without exposing the underlying implementation details of the collection.

- ✚ Retrieving Elements: It provides methods like next() to retrieve the next element in the iteration sequence. The iterator keeps track of the current position and moves to the next element upon each call to next().

- ✚ Removing Elements: The Iterator interface provides a method called remove() to remove the current element from the underlying collection during the iteration. This allows you to safely remove elements while iterating without causing concurrent modification issues.

- ✚ StringBuilder and StringBuffer are classes in Java that provide mutable sequences of characters. They are used when there is a need to manipulate strings efficiently, especially when frequent modifications or concatenations are required.
- ✚ The main purpose of StringBuilder and StringBuffer is to overcome the immutability of the String class. While String objects are immutable, meaning their values cannot be changed once created, StringBuilder and StringBuffer allow for in-place modifications of strings.

41. THE KEY DIFFERENCES BETWEEN STRINGBUILDER AND STRINGBUFFER ARE:

- ✚ Thread Safety: StringBuilder is not thread-safe, meaning it is not synchronized and is not suitable for use in multi-threaded environments where multiple threads may concurrently access or modify the same object. On the other hand, StringBuffer is thread-safe as it provides synchronized methods, making it safe for concurrent access by multiple threads.
- ✚ Performance: StringBuilder is generally faster than StringBuffer because it does not incur the overhead of synchronization. However, the difference in performance is typically insignificant unless there is heavy concurrent access. In such cases, StringBuilder is preferred for better performance.
- ✚ Both StringBuilder and StringBuffer provide similar methods for appending, inserting, deleting, and modifying characters in a string. These methods allow for efficient manipulation of string data without creating multiple intermediate string objects.

Sure! Here are some common Core Java interview questions and answers:

42. Q1. WHAT IS THE DIFFERENCE BETWEEN JDK, JRE, AND JVM?

A1.

- ✚ JDK (Java Development Kit) is a software development kit that provides tools for developing and running Java applications. It includes the JRE, compiler, debugger, and other development tools.
- ✚ JRE (Java Runtime Environment) is the environment in which Java applications are executed. It includes the JVM and necessary libraries to run Java programs.
- ✚ JVM (Java Virtual Machine) is an abstract machine that executes Java bytecode. It provides platform independence by converting bytecode into machine-specific instructions.

43. Q2. WHAT IS THE DIFFERENCE BETWEEN ABSTRACT CLASS AND INTERFACE?

A2.

- ✚ An abstract class is a class that cannot be instantiated and may contain both abstract and non-abstract methods. It can have instance variables and constructors. Subclasses extend the abstract class and provide implementations for the abstract methods.
- ✚ An interface is a contract that defines a set of methods that a class must implement. It cannot have instance variables or constructors. Classes implement interfaces and provide implementations for all the methods defined in the interface.

44. Q3. WHAT IS THE DIFFERENCE BETWEEN CHECKED AND UNCHECKED EXCEPTIONS?

A3.

- ✚ Checked exceptions are checked at compile-time and must be either caught or declared in the method signature using the throws keyword. Examples include IOException and SQLException.
- ✚ Unchecked exceptions (RuntimeExceptions) are not checked at compile-time and do not require explicit handling. Examples include NullPointerException and ArrayIndexOutOfBoundsException.

45. Q4. WHAT IS METHOD OVERLOADING AND METHOD OVERRIDING?

A4.

- ✚ Method overloading is when multiple methods with the same name but different parameters exist in a class. They are differentiated based on the number, type, or order of parameters.
- ✚ Method overriding is when a subclass provides a different implementation for a method that is already defined in its superclass. The method in the subclass must have the same name, return type, and parameters as the method in the superclass.

46. Q5. WHAT IS THE DIFFERENCE BETWEEN FINAL, FINALLY, AND FINALIZE?

A5.

- ✚ final is a keyword used to declare a variable, method, or class that cannot be changed or overridden.
- ✚ finally is a block used in exception handling to ensure that a section of code is always executed, regardless of whether an exception is thrown or caught.
- ✚ finalize is a method defined in the Object class that is called by the garbage collector before an object is destroyed. It can be overridden to perform cleanup operations before the object is garbage collected.

47. WHAT IS THREAD IN JAVA

- ✚ A thread in Java is like a separate worker that can perform tasks simultaneously with other threads in a program. It allows different parts of the program to run at the same time, completing their tasks independently. Think of threads as individual workers in a factory, each working on their assigned task simultaneously. This

concurrent execution improves the efficiency and responsiveness of the program by utilizing the available resources more effectively.

- ✚ A thread is a path or direction that is taken when a program is being executed.
- ✚ Every program has at least one thread, which is known as the main thread.
- ✚ Threads allow a program to operate more efficiently by doing multiple things at the same time.
- ✚ There are two ways to create a thread in Java: extending the Thread class or implementing the Runnable interface.
- ✚ Once you have created a thread, you need to start it by calling the start() method on the thread object.

48. THREAD LIFE CYCLE

- ✚ New: The thread is created but has not yet started.
- ✚ Runnable: The thread is ready to run and waiting for its turn to be scheduled.
- ✚ Running: The thread is actively executing its code.
- ✚ Blocked/Waiting: The thread is temporarily paused and waiting for a certain condition to be satisfied.
- ✚ Terminated: The thread has completed its execution or has been explicitly terminated.

49. TYPE OF THREAD IN JAVA

- ✚ User Threads: User threads are the regular threads created by the application. They are created and controlled by the user and are used to perform various tasks and operations in the program. User threads can be created explicitly by extending the Thread class or implementing the Runnable interface.
- ✚ Daemon Threads: Daemon threads are background threads that provide support to user threads. They are considered to be non-essential and do not prevent the JVM from exiting when all user threads have completed. Daemon

threads are typically used for tasks such as garbage collection or monitoring, where they run in the background and perform certain operations without interfering with the main functionality of the application.

- ✚ Both user threads and daemon threads can be created in Java, but the main distinction is that user threads are considered essential threads that keep the application running, while daemon threads are non-essential threads that provide supporting functionality.

50. SYNCHRONIZATION

- ✚ Synchronization in Java ensures that only one thread can access a critical section of code or shared resource at a time. It prevents data inconsistencies and race conditions. Using the synchronized keyword, threads take turns executing synchronized methods or code blocks. This maintains data integrity and coordination among threads in multi-threaded applications.

MySQL interview questions:

- *What is MySQL?*

MySQL is an open-source relational database management system (RDBMS) that is widely used for managing and storing data. It provides a powerful and scalable platform for various applications and supports SQL (Structured Query Language) for interacting with the database.

- *What are the different data types supported by MySQL?*

MySQL supports various data types such as numeric types (e.g., INT, DECIMAL), string types (e.g., VARCHAR, CHAR), date and time types (e.g., DATE, DATETIME), and more. It also supports user-defined data types.

- *What is the difference between CHAR and VARCHAR data types in MySQL?*

The CHAR data type is used for fixed-length character strings, while the VARCHAR data type is used for variable-length character strings. CHAR requires a fixed amount of storage space, whereas VARCHAR uses only the required amount based on the actual data length.

- *Explain the difference between primary key and foreign key in MySQL.*

A primary key is a column or a combination of columns that uniquely identifies each row in a table. It ensures the uniqueness and integrity of the data in the table. A foreign key is a column that establishes a relationship between two tables. It references the primary key of another table to enforce referential integrity.

- *What is the purpose of the SELECT statement in MySQL?*

The SELECT statement is used to retrieve data from one or more tables in the database. It allows you to specify the columns you want to retrieve, filter the data using conditions, sort the results, and perform various operations like aggregations and joins.

- *How do you join tables in MySQL?*

In MySQL, you can join tables using the JOIN keyword. The common types of joins are INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN. Joins are performed based on matching values in specified columns between the tables.

- *What are the different types of indexes in MySQL?*

MySQL supports several types of indexes, including primary key indexes, unique indexes, and regular indexes. Primary key indexes ensure uniqueness and fast data retrieval. Unique indexes enforce uniqueness but allow null values. Regular indexes improve query performance by creating a sorted data structure.

- *What is the difference between DELETE and TRUNCATE commands in MySQL?*

The DELETE command is used to remove specific rows from a table based on certain conditions, while the TRUNCATE command is used to remove all rows from a table. DELETE is a logged operation and can be rolled back, whereas TRUNCATE is not logged and cannot be rolled back.

- *How do you backup and restore a MySQL database?*

MySQL provides several methods to backup and restore a database, including using the mysqldump command-line tool, using MySQL Workbench, or using the backup and restore functions provided by hosting providers. These methods allow you to create a backup file of the database and restore it when needed.

- *Explain the ACID properties in the context of MySQL transactions.*

ACID stands for Atomicity, Consistency, Isolation, and Durability. In the context of MySQL transactions, these properties ensure that database transactions are processed reliably and maintain data integrity. Atomicity ensures that a transaction is treated as a single, indivisible unit. Consistency ensures that the database remains in a consistent state before and after the transaction. Isolation ensures that concurrent transactions do not interfere with each other. Durability ensures that once a transaction is committed, its changes are permanently saved even in the event of system failure.

- *What is normalization in MySQL? Why is it important?*

Normalization is the process of organizing data in a database to eliminate redundancy and dependency issues. It involves breaking down a table into multiple smaller tables and establishing relationships between

- *difference between an outer join and an inner join*

The main *difference between an outer join and an inner join* lies in the way they handle unmatched rows between the joined tables. Here's a brief explanation of both:

Inner Join:

An inner join returns only the rows that have matching values in both tables being joined.

It combines rows from two or more tables based on a related column between them.

Only the matching rows are included in the result set.

Outer Join:

An outer join returns all the rows from one table and the matching rows from the other table(s).

It includes unmatched rows from one or both tables in the result set.

There are three types of outer joins: left outer join, right outer join, and full outer join.

Left Outer Join:

Returns all the rows from the left (or first) table and the matching rows from the right (or second) table.

If there is no match in the right table, it includes null values for the right table's columns.

Right Outer Join:

Returns all the rows from the right (or second) table and the matching rows from the left (or first) table.

If there is no match in the left table, it includes null values for the left table's columns.

Full Outer Join:

Returns all the rows from both tables, including unmatched rows from both sides.

If there is no match in either table, it includes null values for the columns of the table without a match.

Wrapper classs

Wrapper classes in Java are used to convert primitive data types into objects. They provide additional functionalities and methods for working with primitive values. Each primitive type has a corresponding wrapper class. Wrapper classes are useful when objects are required, such as when working with collections or using object-oriented features. They can also store null values, unlike primitive types.

Bucket id and Bucket number

- **Bucket ID or Bucket Number in Hash Tables or Hash Maps:** It refers to the index or location within a data structure where an element or key-value pair is stored based on its hash code.
- **Bucket ID or Bucket Number in Error Reporting or Crash Dumps:** It is a unique identifier assigned to a specific error or crash instance, allowing for classification and tracking of different occurrences for analysis and troubleshooting purposes.

POJO class in Java

a POJO (Plain Old Java Object) class in Java is a simple Java class that encapsulates data and provides getter and setter methods for accessing and modifying that data.

It follows specific guidelines, such as having private fields, public getter and setter methods, a default constructor, and no dependency on specific frameworks or libraries.

POJO classes are lightweight data models used in various applications and frameworks for data encapsulation and transfer.

conjunction