**Java**

1) 15 interfaces in Java:

java.lang (Core language features)

1. Cloneable

2. Comparable

3. Iterable

4. Runnable

5. CharSequence

6. AutoCloseable

Collection Framework (java.util)

Collection Interfaces:

7. Collection

8. List

9. Set

10. Queue

11. Deque

12. Map

13. SortedSet

14. SortedMap

Iterators:

15. Iterator

16. ListIterator

Input/Output Interfaces ([java.io](http://java.io/))

Used for data input and output operations.

17. Serializable

18. DataInput

19. DataOutput

20. Externalizable

21. Flushable

22. Closeable

2. How many child classes could be in super class?

There is ***no restriction on how many child classes a superclass*** can have in Java. However, the use of inheritance should be guided by pragmatic design factors such as performance and maintainability.

3. Private class used in java or not?

**Private nested classes** are allowed and can be used when you want the class to be accessible only within the enclosed class.

**Private top-level classes** are **not allowed** in Java.

4. Difference between collection Interface and collections class?

**collection Interface:**

A collection represents a group of objects known as elements.

The Collection interface is the **root interface** of the **Java Collections Framework**.

**Common Implementations of Collection Interface:** List, Set, Queue etc.

**Key Methods in collection:**

1. Adds,
2. removes specific/all,
3. check if contains specified element,
4. check collection is empty or not,
5. get size of collection,
6. Iterating over the collection's elements.

Collection<String> collection = new ArrayList<String>();

* **Collection<String>**: This specifies that the collection will hold String objects. It's a generic type, which means you can specify the type of objects that the collection will store.
* **new ArrayList<>()**: The diamond operator (<>) is used here to let the compiler infer the type of the ArrayList from the declaration on the left-hand side. The type String is already specified in Collection<String>, so the compiler can automatically deduce that ArrayList<String> is the correct type for the right-hand side.

**collections class:** The Collections class is a **utility class** (java.util) that provides static methods for operating on collections

**Key Methods in collection:**

1. Sorting,
2. reversing,
3. finding the maximum or minimum elements
4. Shuffles the elements in a list
5. Adds elements to a collection
6. Creates an unmodifiable list

5. Dictionary is there in Java or Not?

While ***Dictionary***is still available in the Java API, it is **rarely used** in modern projects. Most developers ***now use Map*** or one of its popular implementations ***(like HashMap or TreeMap)***. You may encounter Dictionary in legacy systems or older codebases, but it is not used in new development.

6.Overview of Static Methods in the Collections Class:

static methods (without having to create an instance of the Collections class) to operate on collections.

sorting, reversing, and shuffling elements, searching for elements.

manipulate and interact with Collection objects without having to create an instance of the Collections class (since all the methods are static).

7.What is Thread in java?

Example: A web server handling multiple user requests. Each user request can be handled by a different thread, so the server can manage several requests at once, improving throughput.

Concurrency refers to the ability of a system to handle multiple tasks at the same time. This does not necessarily mean that tasks are executed simultaneously, but that they can be managed and executed in overlapping periods.

8., TreeSet, and TreeMap?

part of the java.util package.

commonly used to store collections of elements.

1. Hashset -> backed by HashMap to store elements(Unorder,No duplicate,allow null)

Add, Remove, Contains O(1)

2. Treeset -> backed by TreeMap and use Red-Black Tree to store elements (sorted,no duplicate,allow null)

Add, Remove, Contains (O(log n))

3. LinkedHashSet -> stored in insertion order (insertion order,one null value, no sort.

Add, remove, contains O(1).

9.Vector vs ArrayList?

Vector is a legacy class that should generally be avoided in modern Java programming due to its performance overhead, particularly its synchronization.

If thread safety is not a concern, prefer using ArrayList for most use cases.

If thread safety is required, consider using CopyOnWriteArrayList or manually synchronizing an ArrayList using Collections.synchronizedList() for better performance and flexibility.

Vector vs ArrayList:

**Feature**  **Vector**  **ArrayList**

Thread Safety Synchronized, thread-safe by default Not synchronized, not thread-safe by default

Growth Factor Doubles in size when full (default) Increases by 50% (default)

Performance Slower due to synchronization Faster in most cases

Usage Legacy use cases, thread safet Modern use cases, general-purpose list

Introduced in Java 1.0 Java 1.2 (JDK 1.2)

10.Generics?

Example with generics: ArrayList<**String**> list = new ArrayList<>(); will only allow String elements.

11.Array vs ArrayList?

Arrays are simpler, faster, and memory-efficient, but they have a fixed size and cannot be resized.

ArrayList is more flexible, dynamic, and provides more methods and functionality, but has additional memory overhead and performance costs due to resizing.

12.throw and throws - try and catch?

In modern Java programming, throw and throws still play a crucial role in handling exceptions, and try and catch are used together with them. However, they serve different purposes in the exception-handling mechanism.

In Java, the combination of **throw, throws, try, and catch creates a structured exception-handling** flow. try and catch are used together for handling exceptions when they occur, while throw and throws are used to declare and propagate exceptions.

throw: We use it inside the method (readFile) to explicitly throw an exception.

throws: We declare throws IOException in the method signature of readFile to indicate that it may throw an exception.

try and catch: We wrap the method call readFile("") inside a try block and handle the exceptions using catch.

finally: Regardless of whether an exception occurred or not, the code in the finally block is executed, making it ideal for cleanup operations (like closing files, releasing resources, etc.).

This demonstrates how exception handling works in Java using throw, throws, try, catch, and finally to manage and handle different types of exceptions effectively.