1. Difference between == and equals() in Java, and how to override equals() and hashCode()

== Operator:

Compares references (memory locations) for objects, meaning it checks if two object references point to the same instance in memory.

Example:

String str1 = new String("Hello");

String str2 = new String("Hello");

System.out.println(str1 == str2); // false, different objects

equals() Method:

Compares the content of objects. If not overridden, equals() in the Object class behaves like ==.

Why Override equals() and hashCode(): In collections like HashMap, objects rely on these methods for equality checks and hashing. Consistent override ensures proper functionality.

Override Example:

class Employee {

private int id;

private String name;

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Employee employee = (Employee) o;

return id == employee.id && Objects.equals(name, employee.name);

}

@Override

public int hashCode() {

return Objects.hash(id, name);

}

}

2. Java Memory Model (JMM) and Garbage Collection (GC)

Java Memory Model (JMM):

Specifies how threads interact through memory and defines visibility, ordering, and atomicity of variables.

The volatile keyword, synchronized blocks, and atomic classes (like AtomicInteger) help achieve thread-safety as per JMM.

Garbage Collection:

Automatic memory management in JVM to reclaim memory from objects no longer in use.

Collectors: Java has several GC algorithms:

Serial GC: Single-threaded, suitable for small applications.

Parallel GC: Multi-threaded, suitable for high throughput.

CMS (Concurrent Mark-Sweep) GC: Low-pause GC.

G1 GC: Balanced for low-pause and throughput, recommended for large applications.

Generations: Java uses Young and Old generations to manage object lifecycles.

3. synchronized and volatile Keywords

synchronized:

Provides exclusive access to resources for a single thread.

Used on methods or blocks to ensure only one thread can execute them.

Example:

public synchronized void increment() {

count++;

}

Intrinsic Locks: synchronized uses intrinsic locks, making methods or blocks thread-safe.

volatile:

Ensures visibility across threads; changes to a volatile variable are immediately visible to other threads.

Does not prevent multiple threads from modifying the variable at the same time (non-atomic).

Example:

private volatile boolean isActive = true;

4. Difference between final, finally, and finalize()

final:

Used with classes, methods, or variables to prevent modification.

Final variables are constants, final methods cannot be overridden, and final classes cannot be subclassed.

finally:

A block in exception handling that executes whether an exception is thrown or not.

Example:

try {

// risky code

} finally {

// cleanup code, always executed

}

finalize():

A method called by GC on an object before it’s destroyed.

Rarely used in modern Java due to unpredictable behavior; better alternatives are AutoCloseable and try-with-resources.

5. HashMap Internal Working and Java 8 Enhancements

Bucket System: HashMap uses an array of buckets, where each bucket stores entries as linked lists (in Java 7) or red-black trees (in Java 8) for high-collision buckets.

Hashing: Key’s hashCode determines the bucket index. If multiple keys map to the same bucket, they form a collision chain.

Java 8 Improvement: Converts buckets with high collisions into balanced trees, reducing search time from O(n) to O(log n).

Example:

HashMap<String, Integer> map = new HashMap<>();

map.put("apple", 1);

map.put("banana", 2);

6. Difference Between ArrayList and LinkedList

ArrayList:

Backed by a resizable array, providing fast access via indices.

Inefficient for insertions/removals in the middle, as elements are shifted.

LinkedList:

Doubly-linked list structure, ideal for frequent insertions/removals.

Slower for random access since it requires traversal from the head or tail.

Example:

List<String> arrayList = new ArrayList<>();

List<String> linkedList = new LinkedList<>();

7. Common Design Patterns

Singleton: Ensures only one instance of a class.

Factory: Allows creation of objects without specifying the exact class.

Observer: Defines a one-to-many relationship, notifying dependent objects of changes.

Singleton Example:

class Singleton {

private static Singleton instance;

private Singleton() {}

public static Singleton getInstance() {

if (instance == null) instance = new Singleton();

return instance;

}

}

8. Java Streams vs. Traditional Iteration

Streams: Support functional programming, allowing operations like filter, map, and reduce. Streams are processed lazily, optimizing performance.

Traditional Iteration: Uses loops, requiring more boilerplate code.

Stream Example:

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

names.stream()

.filter(name -> name.startsWith("A"))

.forEach(System.out::println);

9. Java Exception Handling and Checked vs. Unchecked Exceptions

Checked Exceptions: Caught or declared in method signatures (e.g., IOException).

Unchecked Exceptions: Occur at runtime, derived from RuntimeException.

Example:

try {

throw new IOException("File error");

} catch (IOException e) {

e.printStackTrace();

}

10. Callable vs. Runnable

Runnable: Doesn’t return a result or throw checked exceptions.

Callable: Returns a result and can throw checked exceptions, useful for parallel tasks with results.

Example:

Callable<Integer> task = () -> {

return 123;

};

ExecutorService executor = Executors.newFixedThreadPool(1);

Future<Integer> future = executor.submit(task);

11. What are Comparable and Comparator interfaces, and how do you use them?

Comparable:

Used to define a natural order for objects.

Requires implementing the compareTo() method, which compares this object with the specified object.

Comparator:

Used to define multiple or custom ordering for objects.

Requires implementing the compare() method, comparing two objects provided as parameters.

Example:

class Employee implements Comparable<Employee> {

private int id;

private String name;

@Override

public int compareTo(Employee other) {

return Integer.compare(this.id, other.id); // Natural order by id

}

}

// Custom Comparator

Comparator<Employee> nameComparator = (e1, e2) -> e1.getName().compareTo(e2.getName());

12. Explain transient keyword and its usage in Java.

transient:

Prevents serialization of a field in an object. Transient fields are skipped during the serialization process.

Useful for sensitive information or fields that don't need to be saved.

Example:

class User implements Serializable {

private String username;

private transient String password; // Not serialized

}

13. What is the try-with-resources statement?

Introduced in Java 7, the try-with-resources statement automatically closes resources like files or sockets.

Resources used within the try block must implement AutoCloseable.

Example:

try (FileInputStream fis = new FileInputStream("file.txt")) {

// Read from file

} catch (IOException e) {

e.printStackTrace();

}

14. Explain the concept of Stream API in Java 8 with examples.

Stream API: Allows functional-style operations on collections of data. Streams support operations like filter, map, reduce, and are either intermediate (transforming) or terminal (producing a result).

Example:

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

List<String> filteredNames = names.stream()

.filter(name -> name.startsWith("A"))

.collect(Collectors.toList());

15. Difference between HashMap, TreeMap, and LinkedHashMap.

HashMap: Unordered collection with average O(1) time complexity for operations.

TreeMap: Sorted by keys (natural order or Comparator), provides O(log n) time complexity.

LinkedHashMap: Ordered by insertion or access order, preserving iteration order.

16. Explain Serialization and Deserialization in Java.

Serialization: Converts an object into a byte stream to save it or transfer it.

Deserialization: Converts the byte stream back into an object.

Example:

ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("data.obj"));

out.writeObject(new MyObject());

17. How does the volatile keyword work, and when should it be used?

Volatile:

Ensures visibility of changes to a variable across threads by always reading from main memory.

Should be used when a variable is accessed by multiple threads without needing synchronization.

18. Explain the Singleton design pattern with thread-safety.

Singleton:

Guarantees a single instance of a class, often using lazy initialization and double-checked locking for thread safety.

Example:

class Singleton {

private static volatile Singleton instance;

private Singleton() {}

public static Singleton getInstance() {

if (instance == null) {

synchronized (Singleton.class) {

if (instance == null) instance = new Singleton();

}

}

return instance;

}

}

19. Difference between stack and heap memory in Java.

Stack: Stores local variables and method calls, follows LIFO (Last In First Out).

Heap: Stores objects and class-level data, managed by Garbage Collector.

20. Explain method overloading and method overriding with examples.

Overloading: Same method name, different parameters in the same class.

Overriding: Same method signature in parent and child class, providing a specific implementation in the child class.

Example:

class Animal {

void sound() { System.out.println("Generic animal sound"); }

}

class Dog extends Animal {

@Override

void sound() { System.out.println("Bark"); }

}

---

21. What is reflection in Java, and what are its practical uses and downsides?

Reflection:

Allows inspection and modification of classes, methods, and fields at runtime.

Used in frameworks, testing, and dynamic instantiation, but can impact performance and break encapsulation.

Example:

Class<?> clazz = Class.forName("mypackage.MyClass");

Method method = clazz.getMethod("myMethod");

method.invoke(clazz.newInstance());

22. How does Java handle immutability, and why are String objects immutable?

Immutability:

Ensures the object's state cannot be changed after creation. In Java, String is immutable for security and performance reasons.

To make an immutable class, declare fields final and provide no setters.

Example:

final class Employee {

private final String name;

public Employee(String name) { this.name = name; }

public String getName() { return name; }

}

23. Explain method references in Java 8.

Method References:

A shorthand for lambda expressions that refer to methods by name.

Syntax: ClassName::methodName

Example:

List<String> names = Arrays.asList("Alice", "Bob");

names.forEach(System.out::println); // equivalent to name -> System.out.println(name)

24. Difference between StringBuilder and StringBuffer.

StringBuilder: Non-thread-safe, but faster for single-threaded environments.

StringBuffer: Thread-safe due to synchronization, making it slower.

25. How does Java’s memory model support multi-threading, and what is ReentrantLock?

JMM: Defines visibility and ordering rules, crucial for multi-threading.

ReentrantLock: Provides explicit locking with advanced features like fairness.

26. Explain Java annotations and custom annotations.

Annotations: Metadata for code that influences behavior at compile or runtime.

Custom Annotation Example:

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

public @interface MyAnnotation { String value(); }

27. Difference between abstraction and encapsulation.

Abstraction: Hides complexity; achieved through abstract classes and interfaces.

Encapsulation: Hides internal details; achieved through private fields and public methods.

28. What is the role of transient keyword?

Transient: Prevents serialization of fields, often used for sensitive data like passwords.

29. Explain dependency injection and its benefits.

Dependency Injection: Allows passing dependencies into a class rather than creating them within, enhancing testability and decoupling.

30. Difference between List, Set, and Map in Java Collections.

List: Ordered collection, allows duplicates.

Set: Unordered collection, no duplicates.

Map: Key-value pairs, no duplicates key