GLOBAL LOGIC

Section A: Core Java (1–20)

1. Explain polymorphism with examples.

Answer:
"Polymorphism means one thing with many forms. In Java, it happens in two ways: method overloading at compile time, and method overriding at runtime."

Example:

```
class Shape {
   void draw(){ System.out.println("Drawing shape"); }
}
class Circle extends Shape {
   @Override
   void draw(){ System.out.println("Drawing circle"); }
}
// Overloading
void print(int a) {}
void print(String s) {}
```

---

2. Abstract class vs Interface.

Answer:
"Abstract classes can have both abstract and concrete methods, and fields. Interfaces only define contracts, but from Java 8 they support default and static methods. I use abstract class when I need shared code, and interfaces when I need multiple inheritance of type."

Example:

```
abstract class Animal { abstract void sound(); }
interface Pet { void play(); }
```

---

3. How does Java achieve platform independence?

Answer:
"Java code is compiled into bytecode, and the JVM executes this bytecode on any OS. That's why we say Java is 'write once, run anywhere.'"

Example:

```
javac Hello.java   # produces Hello.class (bytecode)
java Hello         # JVM runs it on any OS
```

---

4. Difference between `==` and `.equals()`.

Answer:
"`==` checks reference equality, `.equals()` checks logical equality."

Example:

```
String s1 = new String("hello");
String s2 = new String("hello");
System.out.println(s1 == s2);      // false
System.out.println(s1.equals(s2)); // true
```

---

5. What is a marker interface?

Answer:
"A marker interface has no methods. It just tells JVM something about the class. For example, `Serializable` marks a class for serialization."

Example:

```
class Student implements java.io.Serializable {}
```

---

6. How does HashMap work internally?

Answer:
"HashMap stores data in buckets using the key's hashCode. Collisions are handled by linked list or red-black tree from Java 8."

Example:

```
Map<Integer, String> map = new HashMap<>();
map.put(1, "A");
map.put(2, "B");
```

---

7. HashMap vs ConcurrentHashMap.

Answer:
"HashMap is not thread-safe. ConcurrentHashMap is thread-safe and uses segment/bucket-level locking for better performance."

Example:

```
ConcurrentHashMap<Integer, String> cmap = new ConcurrentHashMap<>();
```

---

8. ArrayList vs LinkedList.

Answer:
“ArrayList is backed by array → fast random access. LinkedList is backed by doubly linked list → faster insertions/deletions.”

Example:

```
List<Integer> a = new ArrayList<>();
List<Integer> l = new LinkedList<>();
```

---

9. Fail-fast vs Fail-safe iterators.

Answer:
“Fail-fast iterators throw `ConcurrentModificationException` if collection is modified, while fail-safe iterators work on a copy.”

Example:

```
Iterator<Integer> it = new ArrayList<>(List.of(1,2,3)).iterator(); // fail-
fast
Iterator<Integer> it2 = new
CopyOnWriteArrayList<>(List.of(1,2,3)).iterator(); // fail-safe
```

---

10. How does TreeMap maintain order?

Answer:
“TreeMap uses a Red-Black tree. Keys are stored in sorted order, either natural or custom comparator.”

Example:

```
TreeMap<Integer,String> t = new TreeMap<>();
t.put(2,"B"); t.put(1,"A");
System.out.println(t.keySet()); // [1,2]
```

---

11. Checked vs Unchecked exceptions.

Answer:
“Checked exceptions are handled at compile time, like IOException. Unchecked are runtime errors, like NullPointerException.”

Example:

```
// Checked
try { new FileReader("abc.txt"); } catch(IOException e){}
// Unchecked
int a=5/0; // ArithmeticException
```

---

12. Custom exception.

Answer:
“I create it by extending Exception or RuntimeException.”

Example:

```
class InvalidAgeException extends Exception {
   public InvalidAgeException(String msg){ super(msg); }
}
```

---

13. How do you handle deadlock?

Answer:
“I avoid nested locks, or use tryLock with timeout to prevent deadlock.”

Example:

```
ReentrantLock lock = new ReentrantLock();
if(lock.tryLock(1000, TimeUnit.MILLISECONDS)) { ... }
```

---

14. Runnable vs Callable.

Answer:
“Runnable doesn’t return a value, Callable returns a result and can throw checked exceptions.”

Example:

```
ExecutorService ex = Executors.newSingleThreadExecutor();
Future<Integer> f = ex.submit(() -> 5+10);
System.out.println(f.get()); // 15
```

---

15. Synchronized block vs method.

Answer:
“Synchronized method locks the whole method. A synchronized block locks only a portion.”

Example:

```
synchronized void foo(){}
void bar(){ synchronized(this){ ... } }
```

---

16. Lambda expression.

Answer:
“Lambda is shorthand for functional interfaces. It makes code concise.”

Example:

```
List<Integer> list = Arrays.asList(1,2,3);
list.forEach(x -> System.out.println(x));
```

---

17. Streams API.

Answer:
“Streams allow functional-style processing of collections.”

Example:

```
List<String> names = List.of("Alice","Bob","Alex");
List<String> result = names.stream()
   .filter(s -> s.startsWith("A"))
   .map(String::toUpperCase)
   .toList(); // ["ALICE","ALEX"]
```

---

18. Optional.

Answer:
“Optional avoids null pointer exceptions by wrapping values.”

Example:

```
Optional<String> name = Optional.of("John");
System.out.println(name.orElse("Default"));
```

---

19. map() vs flatMap().

Answer:
“map transforms each element, flatMap flattens nested structures.”

Example:

```
List<List<Integer>> list = List.of(List.of(1,2), List.of(3,4));
list.stream().flatMap(l -> l.stream()).forEach(System.out::println); // 1 2 3
4
```

---

20. Functional interface.

Answer:
"A functional interface has only one abstract method, used with lambdas."

Example:

```
@FunctionalInterface
interface MyFunc { void run(); }
MyFunc f = () -> System.out.println("Hello");
f.run();
```

Section B: SQL/PostgreSQL (21–30)

---

21. Second highest salary.

Answer:
"I use subquery to exclude the max."

Example:

```
SELECT MAX(salary)
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```

---

22. INNER JOIN vs LEFT JOIN.

Answer:
"INNER JOIN returns only matches. LEFT JOIN returns all from left, even if no match."

Example:

```
-- Inner join
SELECT e.name, d.dept_name
FROM emp e INNER JOIN dept d ON e.dept_id=d.id;

-- Left join
SELECT e.name, d.dept_name
FROM emp e LEFT JOIN dept d ON e.dept_id=d.id;
```

---

23. EXISTS vs IN.

Answer:
“IN checks values in a list. EXISTS checks if rows exist in subquery.”

Example:

```
SELECT name FROM emp WHERE dept_id IN (SELECT id FROM dept);
SELECT name FROM emp WHERE EXISTS (SELECT 1 FROM dept d WHERE
e.dept_id=d.id);
```

---

24. Duplicate salaries.

Answer:
“I use GROUP BY and HAVING.”

Example:

```
SELECT salary, COUNT(*)
FROM employees
GROUP BY salary
HAVING COUNT(*)>1;
```

---

25. Window functions.

Answer:
“Window functions compute values across rows.”

Example:

```
SELECT name, salary,
ROW_NUMBER() OVER(PARTITION BY dept ORDER BY salary DESC) as rnk
FROM employees;
```

---

26. Departments with >5 employees.

Answer:
“I use GROUP BY with HAVING.”

Example:

```
SELECT dept_id, COUNT(*)
FROM employees
```

```
GROUP BY dept_id
HAVING COUNT(*)>5;
```

---

27. HAVING vs WHERE.

Answer:
"WHERE filters rows before grouping, HAVING filters after grouping."

Example:

```
-- WHERE
SELECT * FROM emp WHERE salary>50000;
-- HAVING
SELECT dept_id, AVG(salary) FROM emp GROUP BY dept_id HAVING
AVG(salary)>50000;
```

---

28. Nth highest salary.

Answer:
"I can use OFFSET with LIMIT."

Example:

```
SELECT DISTINCT salary
FROM employees
ORDER BY salary DESC
OFFSET 2 LIMIT 1; -- 3rd highest
```

---

29. UNION vs UNION ALL.

Answer:
"UNION removes duplicates, UNION ALL keeps them."

Example:

```
SELECT name FROM emp1
UNION
SELECT name FROM emp2;   -- no duplicates
```

---

30. Indexes in PostgreSQL.

Answer:
"Indexes work like a book index: they speed up searches but slow inserts/updates slightly."

Example:

```
CREATE INDEX idx_emp_name ON employees(name);
```

Perfect — I'll extend the **scripted answers with examples** so you can respond with both **concept + short code snippet / query / real-life analogy**. That way, you'll sound both clear and practical in interviews.

Since examples will make it long, I'll continue with **Sections C, D & E (DSA + Spring Boot + HR)**, including **short code/query snippets where applicable**.

Section C: Coding / DSA Scripted Responses

31. Check if a string is a palindrome.

Answer:
"A palindrome reads the same forwards and backwards. I compare characters from both ends."
Example:

```
String s = "madam";
int i=0, j=s.length()-1;
boolean isPal = true;
while(i<j){
    if(s.charAt(i)!=s.charAt(j)){ isPal=false; break; }
    i++; j--;
}
// Result: true
```

---

32. Maximum subarray sum (Kadane's).

Answer:
"Kadane's algorithm keeps track of the current sum and resets it if it becomes negative."
Example:

```
int[] arr = {-2,1,-3,4,-1,2,1,-5,4};
int maxSum = arr[0], curr = arr[0];
for(int i=1;i<arr.length;i++){
    curr = Math.max(arr[i], curr+arr[i]);
    maxSum = Math.max(maxSum, curr);
}
// maxSum = 6
```

---

33. Longest substring without repeating characters.

Answer:
“I use sliding window with a set or map.”
Example:

```
String s = "abcabcbb";
Set<Character> set = new HashSet<>();
int left=0, max=0;
for(int right=0; right<s.length(); right++){
    while(set.contains(s.charAt(right))){
        set.remove(s.charAt(left++));
    }
    set.add(s.charAt(right));
    max = Math.max(max, right-left+1);
}
// max = 3 ("abc")
```

---

34. Binary search.

Answer:
“Binary search halves the search space until the target is found.”
Example:

```
int[] arr = {1,3,5,7,9};
int target=7, low=0, high=arr.length-1;
while(low<=high){
   int mid = (low+high)/2;
   if(arr[mid]==target) return mid;
   else if(arr[mid]<target) low=mid+1;
   else high=mid-1;
}
// returns index 3
```

---

35. Median of two sorted arrays.

Answer:
“I merge until the middle is reached, or use binary partition approach.”
Example (merge):

```
int[] a={1,3}, b={2};
int[] c=new int[a.length+b.length];
System.arraycopy(a,0,c,0,a.length);
System.arraycopy(b,0,c,a.length,b.length);
Arrays.sort(c);
// median = 2
```

---

36. Anagrams check.

Answer:
“I sort both strings and compare, or use frequency count.”
Example:

```
String s1="listen", s2="silent";
char[] c1=s1.toCharArray(), c2=s2.toCharArray();
Arrays.sort(c1); Arrays.sort(c2);
boolean isAnagram = Arrays.equals(c1,c2); // true
```

---

37. LRU Cache.

Answer:
“I use LinkedHashMap with access-order.”
Example:

```
class LRU<K,V> extends LinkedHashMap<K,V>{
    private int capacity;
    LRU(int capacity){ super(capacity,0.75f,true); this.capacity=capacity; }
    protected boolean removeEldestEntry(Map.Entry<K,V> eldest){
        return size()>capacity;
    }
}
// Auto removes oldest
```

---

38. Reverse linked list.

Answer:
“Iterative approach uses 3 pointers: prev, curr, next.”
Example:

```
ListNode prev=null, curr=head;
while(curr!=null){
    ListNode next=curr.next;
    curr.next=prev;
    prev=curr; curr=next;
}
// prev is new head
```

---

39. First non-repeating character.

Answer:
“I use frequency map, then scan.”
Example:

```
String s="swiss";
Map<Character,Integer> map=new HashMap<>();
for(char c: s.toCharArray()) map.put(c,map.getOrDefault(c,0)+1);
```

```
for(char c: s.toCharArray()){
    if(map.get(c)==1){ System.out.println(c); break; }
}
// Output: w
```

---

40. Majority element.

Answer:
"I use Boyer-Moore voting algorithm."
Example:

```
int[] arr={3,3,4,2,3,3,5};
int count=0, candidate=0;
for(int num:arr){
    if(count==0) candidate=num;
    count += (num==candidate)?1:-1;
}
// candidate=3
```

Section D: Spring Boot & Full Stack

41. @Controller vs @RestController vs @Service.

Answer:
"@Controller returns views, @RestController returns JSON/XML directly, @Service marks business logic layer."
Example:

```
@RestController
class MyApi {
   @GetMapping("/hello")
   public String hello(){ return "Hello World"; }
}
```

---

42. Dependency injection.

Answer:
"Instead of creating objects manually, Spring injects them. This makes code loosely coupled."
Example:

```
@Service
class UserService {}

@RestController
class UserController {
   @Autowired UserService service;
```

```
}
```

---

43. @Autowired vs constructor injection.

Answer:
"@Autowired injects by field/setter, but constructor injection is preferred for immutability."
Example:

```
class UserController {
   private final UserService service;
   public UserController(UserService service){ this.service=service; }
}
```

---

44. Connect Spring Boot to PostgreSQL.

Answer:
"I configure application.properties."
Example:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/mydb
spring.datasource.username=postgres
spring.datasource.password=secret
spring.jpa.hibernate.ddl-auto=update
```

---

45. REST API lifecycle.

Answer:
"A client sends request → DispatcherServlet → Controller → Service → DAO → DB → response returned as JSON."

---

46. Spring Boot starters.

Answer:
"Starters are pre-defined dependencies for common tasks. For example, `spring-boot-starter-web` includes Tomcat, Spring MVC, and JSON support."

---

47. HTTP Methods.

Answer:
"GET fetches data, POST creates, PUT updates, DELETE removes."
Example:

```
@PostMapping("/users")  // create user
@GetMapping("/users/{id}") // fetch user
```

---

48. CORS.

Answer:
"CORS allows cross-origin requests. I can enable it in controller or globally."
Example:

```
@CrossOrigin(origins="*")
@GetMapping("/data") public String data(){ return "ok"; }
```

---

49. Monolithic vs Microservices.

Answer:
"Monolithic is one big deployable, microservices are independent small services communicating via APIs. Microservices scale better."

---

50. Exception handling in REST.

Answer:
"I use @ControllerAdvice with @ExceptionHandler."
Example:

```
@ControllerAdvice
class GlobalException {
  @ExceptionHandler(Exception.class)
  public ResponseEntity<String> handle(Exception e){
     return ResponseEntity.status(500).body(e.getMessage());
  }
}
```

Section E: HR / Managerial

(These should sound natural, not scripted too much.)

51. Tell me about yourself.

Answer:
"I'm [Your Name], I've worked on Java, SQL, and Spring Boot projects. I'm comfortable in backend development, but also familiar with frontend basics. I enjoy solving coding challenges and building efficient applications. I recently cleared Round 1, and I'm excited to showcase my full-stack skills."

---

52. Why GlobalLogic?

Answer:
"GlobalLogic is known for combining engineering with design and innovation. I like that it works with global clients, and it will give me exposure to both technical and client-facing responsibilities."

---

53. Technical challenge under pressure.

Answer:
"In college, I worked on a project where the database kept crashing due to unoptimized queries. I analyzed and rewrote them using indexes and joins, which reduced execution time significantly. It taught me how optimization matters."

---

54. Teammate missing deadlines.

Answer:
"I would first discuss privately to understand if they're struggling. If needed, I'd offer help, and if deadlines are still at risk, I'd escalate politely to the manager."

---

55. Project using Java + SQL.

Answer:
"I built a Student Management System using Spring Boot and PostgreSQL. REST APIs handled student CRUD operations, data was stored in Postgres, and I used React for the frontend."

---

56. Staying updated.

Answer:
“I follow Java official documentation, read blogs, and practice on platforms like LeetCode and HackerRank. I also experiment with new frameworks in small projects.”

---

57. Strengths and weaknesses.

Answer:
“My strength is problem-solving and quick learning. My weakness was sometimes over-focusing on details, but I’ve been improving by managing my time better.”

---

58. New technology learning.

Answer:
“I would start with official docs, small POCs, and practice. For example, I taught myself Spring Boot by building a simple CRUD API before using it in a bigger project.”

---

59. Client disagrees with you.

Answer:
“I’d listen carefully to their perspective, provide technical reasoning with pros/cons, and if they still prefer their approach, I’d adapt while documenting risks.”

---

60. 3-year plan.

Answer:
“I see myself growing as a full-stack engineer, leading small teams, and contributing to scalable applications while continuously learning emerging technologies.”