**Q1. Difference between HashMap and ConcurrentHashMap?**  
👉 HashMap is not thread-safe. Concurrent modifications lead to ConcurrentModificationException.  
👉 ConcurrentHashMap uses segment/stripe locking (Java 8 → CAS + synchronized blocks) allowing concurrent reads and controlled writes.

**Q2. How do you avoid deadlocks in Java?**  
👉 Strategies:

* Lock ordering (always acquire locks in a fixed sequence).
* Use tryLock with timeout.
* Minimize lock scope.
* Prefer concurrent data structures over manual synchronization.

**Q3. Difference between parallelStream() and stream()?**  
👉 stream() is sequential, runs on single thread.  
👉 parallelStream() uses ForkJoinPool, splits workload across multiple threads.  
⚠️ Best for CPU-bound tasks, not for I/O-heavy workloads.

**Q4. How do you implement retry with exponential backoff in Java?**  
👉 Use ScheduledExecutorService or libraries like Resilience4j. Example: retry after 1s, 2s, 4s, … until max retries.

**Q5. REST API idempotency – how to design PUT vs POST?**  
👉 POST: create (non-idempotent).  
👉 PUT: create/update (idempotent – same request multiple times = same result).  
👉 Use Idempotency-Key header for banking/payment APIs.

**Section 2 – Spark**

**Q6. RDD vs DataFrame vs Dataset?**  
👉 RDD: low-level, type-unsafe.  
👉 DataFrame: high-level API, optimized by Catalyst, untyped.  
👉 Dataset: combines RDD (type safety) + DataFrame (optimizations).

**Q7. How do you handle skewed data in Spark?**  
👉 Techniques:

* Salting keys.
* Repartitioning/shuffle partition tuning.
* Use broadcast joins for small tables.
* Skew join optimization in Spark SQL.

**Q8. What is checkpointing in Spark Streaming?**  
👉 Saves state + metadata to HDFS (or other reliable storage).  
👉 Used for recovery after failures.  
👉 Two types: Metadata checkpointing, Data checkpointing.

**Q9. How do you achieve fault tolerance in Spark Streaming?**  
👉 Input data must be replayable (Kafka/Flume).  
👉 Spark uses lineage + checkpointing to recompute lost partitions.

**Q10. Spark job optimization techniques?**

* Cache/persist intermediate results.
* Minimize shuffles (use mapPartitions, reduceByKey).
* Use partition pruning, predicate pushdown.
* Avoid UDFs when SQL functions exist.

**Section 3 – Hadoop Ecosystem**

**Q11. HDFS block size and why large?**  
👉 Default: 128MB/256MB. Large blocks → fewer seeks, better throughput.

**Q12. NameNode vs DataNode?**  
👉 NameNode: stores metadata.  
👉 DataNode: stores actual blocks.

**Q13. Hive – external vs managed table?**  
👉 Managed: Hive controls lifecycle (drop = delete data).  
👉 External: Hive stores only metadata, data remains even if dropped.

**Q14. ORC vs Parquet?**  
👉 ORC: optimized for Hive, good compression, lightweight metadata.  
👉 Parquet: language-agnostic, nested data support, better for Spark.

**Q15. HBase row-key design – best practices?**  
👉 Short, unique, evenly distributed (avoid hotspotting).  
👉 Reverse timestamps for time-series data.

**Section 4 – Messaging Systems**

**Q16. Kafka vs RabbitMQ difference?**  
👉 Kafka: distributed log, high throughput, partitioned, replay capability.  
👉 RabbitMQ: traditional broker, push-based, supports complex routing.

**Q17. How does Kafka ensure ordering?**  
👉 Messages are ordered within a partition.  
👉 To guarantee ordering → all related keys must go to the same partition.

**Q18. How do you handle offset management in Kafka?**  
👉 Auto commit vs manual commit.  
👉 Best practice: commit offset **after processing** the message.

**Q19. Kafka “exactly once semantics” – how?**  
👉 Idempotent producer (enable.idempotence=true).  
👉 Transactions API for producer-consumer pipeline.

**Q20. What if Kafka broker goes down?**  
👉 ISR (in-sync replicas) ensures another replica is promoted as leader.  
👉 Producer/consumer retries with leader election.

**Section 5 – DevOps / CI-CD / TDD**

**Q21. Maven vs SBT vs Ant?**  
👉 Maven: convention over configuration, XML based.  
👉 SBT: Scala-based, better for Spark/Scala projects.  
👉 Ant: older, no dependency management.

**Q22. Jenkins pipeline stages for Spark job?**

1. Checkout code (Git).
2. Compile & run unit tests.
3. Build JAR with Maven.
4. Run integration tests on staging cluster.
5. Deploy to production (submit via spark-submit).

**Q23. Git – difference between merge and rebase?**  
👉 Merge: keeps branch history.  
👉 Rebase: linearizes history (cleaner, but dangerous if misused).

**Q24. JUnit 5 improvements over JUnit 4?**  
👉 @ParameterizedTest, @Nested, @DisplayName, better assertions.

**Q25. How do you ensure test coverage for streaming apps?**  
👉 Use **embedded Kafka** for testing.  
👉 Use MemoryStream in Spark Structured Streaming.  
👉 Validate outputs against expected results.

**Section 6 – Scenario/Case Studies**

**Q26. A Spark Streaming job is lagging behind real-time. What do you check?**

* Check Kafka consumer lag (kafka-consumer-groups.sh).
* Check batch interval & processing time.
* Increase parallelism (spark.streaming.kafka.maxRatePerPartition).
* Optimize transformations (reduce shuffles).

**Q27. You get OutOfMemoryError in Spark. How to debug?**

* Increase executor memory.
* Enable Kryo serialization.
* Use persist(StorageLevel.DISK\_ONLY) for large datasets.
* Avoid wide transformations.

**Q28. You have 1TB JSON logs in HDFS, query is slow. How to improve?**  
👉 Convert JSON → ORC/Parquet (columnar, compressed).  
👉 Partition by date/time.  
👉 Use vectorized queries in Hive/Spark.

**Q29. Kafka consumer is reprocessing same messages after restart. Why?**  
👉 Offsets not committed properly.  
👉 Solution: commit offsets **after** message is processed.

**Q30. Spark job with shuffle stage takes too long. How to optimize?**  
👉 Increase shuffle partitions.  
👉 Use map-side combine.  
👉 Avoid skew by salting.

**Section 7 – Behavioral / Banking Domain**

**Q31. Tell me about a time you optimized a big data pipeline.**  
👉 Example: Reduced Spark shuffle time by tuning partitions + using broadcast joins → improved SLA by 40%.

**Q32. How do you ensure data security in Big Data pipelines?**  
👉 Encrypt data at rest (HDFS TDE) & in transit (TLS).  
👉 Mask PII fields before ingestion.  
👉 Use Ranger/Atlas for access control & lineage.

**Q33. How do you ensure regulatory compliance (GDPR/CCPA)?**  
👉 Right-to-erasure implementation (delete user data from HDFS/Hive).  
👉 Audit logs for all access.

**✅ Quick Tips Before Monjin Video Interview**

* Monjin = **scenario-based + coding** → expect to **share screen & code live** (Java + Spark + Kafka basics).
* Keep 2–3 **real project examples** (data pipeline, Kafka ingestion, Spark optimization).
* Stress on **performance tuning, scalability, resilience** – Synechron’s banking clients need this.
* Keep **buzzwords handy**: "idempotency", "backpressure handling", "predicate pushdown", "exactly-once semantics".