**1. Understanding REST API**

**Q: Explain RESTful APIs in depth and how they differ from traditional web services like SOAP.**  
**A:** REST (Representational State Transfer) is an architectural style that uses HTTP methods to perform CRUD operations. Unlike SOAP, which relies on XML and strict contracts, REST is lightweight, uses JSON or XML, and is stateless, meaning each request from a client contains all necessary information. RESTful APIs follow REST constraints such as uniform interface, statelessness, cacheability, client-server architecture, layered system, and optional code-on-demand.

**2. REST Principles**

**Q: What are the six REST architectural constraints?**  
**A:**

1. **Stateless:** Each request must contain all necessary information; the server does not store session data.
2. **Client-Server Architecture:** Separation of concerns between UI and backend logic.
3. **Cacheability:** Responses should define whether they are cacheable to optimize performance.
4. **Uniform Interface:** Consistent resource identification (e.g., using URIs) and standardized methods (GET, POST, etc.).
5. **Layered System:** APIs can have multiple layers (e.g., security, load balancing) without affecting clients.
6. **Code on Demand (optional):** Servers can send executable code (like JavaScript) to clients.

**3. Asynchronous REST API Calls**

**Q: How can one REST endpoint call another REST API asynchronously?**  
**A:**

* **Using CompletableFuture in Java:**

java

CopyEdit

@Async

public CompletableFuture<String> callAnotherAPI() {

RestTemplate restTemplate = new RestTemplate();

String response = restTemplate.getForObject("https://example.com/api", String.class);

return CompletableFuture.completedFuture(response);

}

* **Using WebClient in Spring WebFlux (non-blocking):**

java

CopyEdit

WebClient webClient = WebClient.create();

Mono<String> response = webClient.get()

.uri("https://example.com/api")

.retrieve()

.bodyToMono(String.class);

**4. Action Triggers in Messaging Tools**

**Q: How are actions triggered in messaging tools like Kafka and RabbitMQ?**  
**A:**

* **Kafka:** Consumers subscribe to topics and process messages when they arrive.
* **RabbitMQ:** Uses exchanges, queues, and bindings to route messages based on routing keys.
* **Trigger Mechanism:**
  + Events/messages are pushed into a queue/topic.
  + Listeners (consumers) pick up the event asynchronously.
  + Actions are executed based on message payload (e.g., updating a database, sending notifications).

**5. Design Patterns**

**Q: Which design patterns are commonly used in microservices?**  
**A:**

* **Singleton** (for instance management)
* **Factory Pattern** (for object creation)
* **Observer Pattern** (event-driven systems)
* **Circuit Breaker** (prevents cascading failures)
* **API Gateway Pattern** (manages multiple services)
* **Strangler Pattern** (for migrating monoliths to microservices)

**6. ArrayList**

**Q: What is an ArrayList, and how does it differ from a simple array?**  
**A:**

* **ArrayList** is a dynamic array that grows automatically, whereas arrays have a fixed size.
* ArrayList provides methods like add(), remove(), and contains(), which are not available in standard arrays.

**7. ArrayList vs. LinkedList**

**Q: How do ArrayList and LinkedList differ in performance?**  
**A:**

| **Feature** | **ArrayList** | **LinkedList** |
| --- | --- | --- |
| Access Speed | Fast (O(1)) | Slow (O(n)) |
| Insertion/Deletion | Slow (O(n)) | Fast (O(1) at head/tail) |
| Memory Usage | Less (contiguous memory) | More (extra memory for pointers) |

**8. Threads**

**Q: What are threads, and how do they differ from processes?**  
**A:**

* A **process** is an independent execution unit with its own memory space.
* A **thread** is a lightweight sub-process that shares memory with other threads in the same process.

**9. Spring Boot**

**Q: What are the key features of Spring Boot?**  
**A:**

* Auto-configuration
* Embedded servers (Tomcat, Jetty)
* Microservices support
* Spring Boot Starter dependencies
* Spring Boot Actuator for monitoring

**10. Methods of REST API**

**Q: List and explain the HTTP methods used in REST APIs.**  
**A:**

* **GET:** Fetch data
* **POST:** Create a resource
* **PUT:** Update a resource
* **PATCH:** Partial update
* **DELETE:** Remove a resource

**11. POST vs. PUT**

**Q: Explain the difference between POST and PUT.**  
**A:**

* **POST:** Used for creating new resources; it is not idempotent.
* **PUT:** Used for updating or replacing an existing resource; it is idempotent.

**12. Spring JPA**

**Q: What is Spring Data JPA?**  
**A:** Spring Data JPA simplifies database access by providing a repository abstraction, reducing the need for boilerplate CRUD operations.

**13. One-to-Many and Many-to-One Mapping**

**Q: How do you implement One-to-Many and Many-to-One mappings in JPA?**  
**A:**

* **One-to-Many:**

java

CopyEdit

@OneToMany(mappedBy = "department")

private List<Employee> employees;

* **Many-to-One:**

java

CopyEdit

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

**14. ArrayList vs. HashMap**

**Q: How does ArrayList differ from HashMap?**  
**A:**

| **Feature** | **ArrayList** | **HashMap** |
| --- | --- | --- |
| Data Structure | Dynamic array | Key-value pair |
| Search Time | O(n) | O(1) |
| Ordering | Maintains insertion order | No order |

**15. Multithreading**

**Q: How do you implement multithreading in Java?**  
**A:**

* **Extending Thread Class:**

java

CopyEdit

class MyThread extends Thread {

public void run() {

System.out.println("Thread running...");

}

}

* **Implementing Runnable:**

java

CopyEdit

class MyRunnable implements Runnable {

public void run() {

System.out.println("Runnable thread running...");

}

}

**16. Microservices Versioning**

**Q: How do you handle versioning in microservices?**  
**A:**

* **URI Versioning:** /api/v1/resource
* **Header Versioning:** Accept: application/vnd.example.v1+json
* **Query Parameter Versioning:** ?version=1
* **Content Negotiation:** MIME-type based

**17. Deadlock in Threads**

**Q: What is a deadlock, and how do you prevent it?**  
**A:**  
A **deadlock** occurs when two or more threads are waiting for each other to release locks.

**Prevention Strategies:**

* **Avoid Nested Locks:** Acquire locks in a consistent order.
* **Use Try-Lock:** Use tryLock() instead of synchronized to avoid indefinite waiting.
* **Timeouts:** Use timeouts to prevent blocking indefinitely.
* **Deadlock Detection:** Use tools like Java Flight Recorder.