# Java, Spring Boot, Microservices, and Angular Interview Preparation Guide

August 2025

## Contents

## Introduction

This document provides answers and examples for Java, Spring Boot, Microservices, and Angular interview questions, organized by topic. Each question includes a beginner-friendly explanation and a practical example, suitable for interview preparation. Additional common questions are included at the end.

## 1 Core Java & OOPs

### 1.1 Difference between String, StringBuilder, and StringBuffer

**Answer:** String is immutable, creating new objects for modifications. StringBuilder is mutable, non-thread-safe, and fast. StringBuffer is mutable, thread-safe, but slower due to synchronization.

```java
public class StringExample {
  public static void main(String[] args) {
    String str = "Hello";
    str += " World"; // New object
    System.out.println(str); // Hello World
    StringBuilder sb = new StringBuilder("Hello");
    sb.append(" World");
    System.out.println(sb); // Hello World
    StringBuffer sbf = new StringBuffer("Hello");
    sbf.append(" World");
    System.out.println(sbf); // Hello World
  }
}
```

### 1.2 Difference between .equals() method and == operator

**Answer:** == compares object references or primitive values. .equals() compares object content, customizable for classes.

```
public class EqualsExample {
  public static void main(String[] args) {
    String s1 = new String("Hello");
    String s2 = new String("Hello");
    System.out.println(s1 == s2); // false
    System.out.println(s1.equals(s2)); // true
    String s3 = "Hello";
    String s4 = "Hello";
    System.out.println(s3 == s4); // true
  }
}
```

### 1.3 Difference between method overloading and method overriding

**Answer:** Overloading uses different parameters in the same class (compile-time). Overriding redefines a method in a subclass (runtime).

```
class Animal {
  void sound() { System.out.println("Animal sound"); }
}
class Dog extends Animal {
  void sound() { System.out.println("Dog barks"); } // Override
  void sound(String type) { System.out.println("Dog " + type); } //
     Overload
}
public class PolymorphismExample {
  public static void main(String[] args) {
    Dog dog = new Dog();
    dog.sound(); // Dog barks
    dog.sound("growls"); // Dog growls
  }
}
```

### 1.4 What is Singleton class?

**Answer:** Ensures one instance with global access, used for shared resources.

### 1.5 Create Singleton class

**Answer:** Uses private constructor, static instance, and static access method.

```
public class Singleton {
  private static Singleton instance;
  private Singleton() {}
  public static synchronized Singleton getInstance() {
    if (instance == null) instance = new Singleton();
    return instance;
  }
  public static void main(String[] args) {
    Singleton s1 = Singleton.getInstance();
    Singleton s2 = Singleton.getInstance();
    System.out.println(s1 == s2); // true
  }
```

```
}
```

### 1.6 What is thread safety and how do you ensure thread-safe classes?

**Answer:** Thread safety prevents data corruption in multi-threaded environments using synchronization, immutability, or thread-safe classes.

```
public class ThreadSafeCounter {
  private int count = 0;
  public synchronized void increment() { count++; }
  public synchronized int getCount() { return count; }
  public static void main(String[] args) throws InterruptedException {
    ThreadSafeCounter counter = new ThreadSafeCounter();
    Runnable task = () -> { for (int i = 0; i < 1000; i++) counter.
        increment(); };
    Thread t1 = new Thread(task); Thread t2 = new Thread(task);
    t1.start(); t2.start(); t1.join(); t2.join();
    System.out.println(counter.getCount()); // 2000
  }
}
```

### 1.7 How does HashMap work internally?

**Answer:** Uses a hash table with buckets. Keys' hashCode() determines bucket index; collisions use linked lists or trees (Java 8+).

```
import java.util.HashMap;
public class HashMapExample {
  public static void main(String[] args) {
    HashMap<String, Integer> map = new HashMap<>();
    map.put("A", 1);
    map.put("B", 2);
    System.out.println(map.get("A")); // 1
  }
}
```

### 1.8 Difference: HashMap vs Hashtable vs ConcurrentHashMap

**Answer:** HashMap: non-thread-safe, allows nulls. Hashtable: thread-safe, no nulls. ConcurrentHashMap: thread-safe, concurrent access, no nulls.

```
import java.util.*;
public class MapComparison {
  public static void main(String[] args) {
    HashMap<String, Integer> hm = new HashMap<>();
    hm.put(null, 1);
    System.out.println(hm); // {null=1}
    Hashtable<String, Integer> ht = new Hashtable<>();
    ConcurrentHashMap<String, Integer> chm = new ConcurrentHashMap<>();
  }
}
```

### 1.9 How does HashMap work with Employee object as key?

**Answer:** Uses Employee's hashCode() and equals() for bucket placement and collision resolution.

```java
import java.util.HashMap;
class Employee {
  int id; String name;
  Employee(int id, String name) { this.id = id; this.name = name; }
  @Override public int hashCode() { return id * 31 + name.hashCode(); }
  @Override public boolean equals(Object obj) {
    if (!(obj instanceof Employee)) return false;
    Employee other = (Employee) obj;
    return id == other.id && name.equals(other.name);
  }
}
public class HashMapEmployee {
  public static void main(String[] args) {
    HashMap<Employee, String> map = new HashMap<>();
    Employee e1 = new Employee(1, "Alice");
    map.put(e1, "Developer");
    System.out.println(map.get(new Employee(1, "Alice"))); // Developer
  }
}
```

### 1.10 What is immutability and how does it help in concurrency?

**Answer:** Immutability prevents state changes, ensuring thread safety without synchronization.

```java
public final class ImmutableClass {
  private final int value;
  public ImmutableClass(int value) { this.value = value; }
  public int getValue() { return value; }
  public static void main(String[] args) {
    ImmutableClass obj = new ImmutableClass(42);
    System.out.println(obj.getValue()); // 42
  }
}
```

### 1.11 What is volatile and synchronized?

**Answer:** volatile ensures variable visibility; synchronized ensures mutual exclusion.

```java
public class VolatileSynchronized {
  private volatile boolean running = true;
  public synchronized void update() { running = false; }
  public static void main(String[] args) throws InterruptedException {
    VolatileSynchronized vs = new VolatileSynchronized();
    new Thread(() -> { while (vs.running) {} System.out.println("Stopped");
        }).start();
    Thread.sleep(1000);
    vs.update();
  }
}
```

### 1.12 How many ways can an object be created in Java?

**Answer:** Using new, Class.forName(), clone(), deserialization, factory methods.

```java
public class ObjectCreation {
  public static void main(String[] args) throws Exception {
    ObjectCreation obj1 = new ObjectCreation();
    ObjectCreation obj2 = (ObjectCreation) Class.forName("ObjectCreation").
        newInstance();
  }
}
```

### 1.13 What is the use of ResponseEntity?

**Answer:** Represents HTTP response with status, headers, and body.

```java
import org.springframework.http.*;
import org.springframework.web.bind.annotation.*;
@RestController
public class ResponseEntityExample {
  @GetMapping("/user")
  public ResponseEntity<String> getUser() {
    return new ResponseEntity<>("User found", HttpStatus.OK);
  }
}
```

### 1.14 What is meant by functional interfaces?

**Answer:** Interfaces with one abstract method, used with lambdas.

```java
@FunctionalInterface
interface MyFunction { void apply(String s); }
public class FunctionalInterfaceExample {
  public static void main(String[] args) {
    MyFunction func = s -> System.out.println(s);
    func.apply("Hello"); // Hello
  }
}
```

### 1.15 How do functional interfaces work?

**Answer:** Enable functional programming via lambda expressions.

```java
import java.util.function.Consumer;
public class FunctionalInterfaceWork {
  public static void main(String[] args) {
    Consumer<String> consumer = s -> System.out.println(s);
    consumer.accept("Hello Consumer"); // Hello Consumer
  }
}
```

### 1.16 What is dependency injection and its types?

**Answer:** Provides dependencies externally. Types: constructor, setter, field injection.

```java
import org.springframework.stereotype.Component;
import org.springframework.beans.factory.annotation.Autowired;
@Component
class Service { public void serve() { System.out.println("Serving"); } }
@Component
```

```
class Client {
  private final Service service;
  @Autowired
  public Client(Service service) { this.service = service; }
  public void doWork() { service.serve(); }
}
```

### 1.17 Difference between IOC and Dependency Injection

**Answer:** IoC inverts control to a framework; DI is a way to achieve IoC.

```
// See above (Question 16)
```

### 1.18 How many ways to achieve dependency injection and which is best?

**Answer:** Constructor, setter, field injection. Constructor is best for explicit dependencies.

```
// See Question 16
```

### 1.19 @Primary vs @Qualifier — which takes priority?

**Answer:** @Primary sets default bean; @Qualifier overrides it.

```
import org.springframework.stereotype.Component;
import org.springframework.beans.factory.annotation.*;
interface Service {}
@Component @Primary
class DefaultService implements Service {}
@Component @Qualifier("special")
class SpecialService implements Service {}
@Component
class Client {
  @Autowired @Qualifier("special") Service service;
}
```

### 1.20 How to create beans manually?

**Answer:** Use @Bean in a @Configuration class.

```
import org.springframework.context.annotation.*;
@Configuration
public class AppConfig {
  @Bean
  public Service myService() { return new Service(); }
}
```

### 1.21 What is the difference between @RestController and @Controller?

**Answer:** @Controller returns view names; @RestController returns data (e.g., JSON).

```
import org.springframework.stereotype.*;
import org.springframework.web.bind.annotation.*;
@Controller
public class MyController {
  @GetMapping("/view")
```

```
  public String getView() { return "view"; }
}
@RestController
public class MyRestController {
  @GetMapping("/api")
  public String getData() { return "Data"; }
}
```

### 1.22 What is @SpringBootApplication annotation?

**Answer:** Combines @EnableAutoConfiguration, @ComponentScan, @Configuration.

```
import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
@SpringBootApplication
public class Application {
  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }
}
```

### 1.23 What are stereotype annotations in Spring Boot?

**Answer:** @Component, @Service, @Repository, @Controller mark classes for scanning.

```
import org.springframework.stereotype.*;
@Service
public class MyService {
  public void serve() { System.out.println("Service"); }
}
```

### 1.24 What is path variable?

**Answer:** @PathVariable extracts URL path values.

```
import org.springframework.web.bind.annotation.*;
@RestController
public class PathVariableExample {
  @GetMapping("/user/{id}")
  public String getUser(@PathVariable int id) {
    return "User ID: " + id;
  }
}
```

### 1.25 What is the default server in Spring Boot?

**Answer:** Tomcat.

```
// Run Application.java to start Tomcat
```

### 1.26 What is the default port in Spring Boot?

**Answer:** 8080.

```
// Access http://localhost:8080
```

## 2 Java 8 Features

### 2.1 Java 8 features used in your project

**Answer:** Lambda expressions, Stream API, Optional, default/static methods, forEach.

```
import java.util.*;
public class Java8Features {
  public static void main(String[] args) {
    List<String> list = Arrays.asList("A", "B");
    list.forEach(s -> System.out.println(s)); // A, B
  }
}
```

### 2.2 What is a lambda expression?

**Answer:** Concise function representation: (params) -> expression.

```
import java.util.*;
public class LambdaExample {
  public static void main(String[] args) {
    List<String> list = Arrays.asList("A", "B");
    list.forEach(s -> System.out.println(s)); // A, B
  }
}
```

### 2.3 Write code using lambda expression

```
import java.util.*;
public class LambdaCode {
  public static void main(String[] args) {
    List<Integer> numbers = Arrays.asList(1, 2, 3);
    numbers.forEach(n -> System.out.println(n * 2)); // 2, 4, 6
  }
}
```

### 2.4 What is Stream API?

**Answer:** Processes collections functionally with operations like filter, map, collect.

```
import java.util.*;
import java.util.stream.*;
public class StreamExample {
  public static void main(String[] args) {
    List<Integer> numbers = Arrays.asList(1, 2, 3, 4);
    List<Integer> evens = numbers.stream()
      .filter(n -> n % 2 == 0)
      .collect(Collectors.toList());
    System.out.println(evens); // [2, 4]
  }
}
```

### 2.5 Intermediate vs terminal operations in streams

**Answer:** Intermediate (lazy, e.g., filter, map); terminal (triggers, e.g., collect, forEach).

```
import java.util.*;
public class StreamOperations {
  public static void main(String[] args) {
    List<Integer> numbers = Arrays.asList(1, 2, 3);
    numbers.stream().filter(n -> n > 1).forEach(System.out::println); // 2,
        3
  }
}
```

### 2.6 List of intermediate and terminal methods

**Answer:** Intermediate: filter, map, sorted, distinct. Terminal: collect, forEach, reduce, count.

```
import java.util.*;
import java.util.stream.*;
public class StreamMethods {
  public static void main(String[] args) {
    List<Integer> numbers = Arrays.asList(1, 2, 2, 3);
    long count = numbers.stream().distinct().count(); // 3
    System.out.println(count);
  }
}
```

### 2.7 Why Java introduced default and static methods in interfaces

**Answer:** Default: Add methods without breaking implementations. Static: Utility methods.

```
interface MyInterface {
  default void defaultMethod() { System.out.println("Default"); }
  static void staticMethod() { System.out.println("Static"); }
}
class MyClass implements MyInterface {}
public class InterfaceExample {
  public static void main(String[] args) {
    new MyClass().defaultMethod(); // Default
    MyInterface.staticMethod(); // Static
  }
}
```

### 2.8 How to create an immutable class in Java

**Answer:** Use final class, final fields, no setters, deep copy for mutable objects.

```
public final class ImmutableClass {
  private final int value;
  public ImmutableClass(int value) { this.value = value; }
  public int getValue() { return value; }
  public static void main(String[] args) {
    ImmutableClass obj = new ImmutableClass(42);
    System.out.println(obj.getValue()); // 42
  }
}
```

### 2.9 How to use groupingBy() in streams

**Answer:** Groups elements by a classifier.

```
import java.util.*;
import java.util.stream.*;
public class GroupingByExample {
  public static void main(String[] args) {
    List<String> names = Arrays.asList("Alice", "Bob", "Adam");
    Map<Character, List<String>> grouped = names.stream()
      .collect(Collectors.groupingBy(s -> s.charAt(0)));
    System.out.println(grouped); // {A=[Alice, Adam], B=[Bob]}
  }
}
```

### 2.10 How to create an Optional of an employee object

**Answer:** Wraps object to handle null cases.

```
import java.util.Optional;
class Employee {
  String name;
  Employee(String name) { this.name = name; }
}
public class OptionalExample {
  public static void main(String[] args) {
    Optional<Employee> emp = Optional.of(new Employee("Alice"));
    System.out.println(emp.get().name); // Alice
  }
}
```

### 2.11 Use Java 8 streams to remove duplicates from a list

```
import java.util.*;
import java.util.stream.*;
public class RemoveDuplicates {
  public static void main(String[] args) {
    List<Integer> numbers = Arrays.asList(1, 2, 2, 3);
    List<Integer> unique = numbers.stream().distinct().collect(Collectors.
       toList());
    System.out.println(unique); // [1, 2, 3]
  }
}
```

### 2.12 Find 3rd highest salary using Java 8 streams

```
import java.util.*;
import java.util.stream.*;
class Employee {
  double salary;
  Employee(double salary) { this.salary = salary; }
  double getSalary() { return salary; }
}
public class ThirdHighestSalary {
  public static void main(String[] args) {
    List<Employee> employees = Arrays.asList(
```

```
      new Employee(50000), new Employee(70000), new Employee(60000)
    );
    double thirdHighest = employees.stream()
      .map(Employee::getSalary)
      .distinct()
      .sorted(Comparator.reverseOrder())
      .skip(2)
      .findFirst()
      .orElse(0.0);
    System.out.println(thirdHighest); // 0.0 (if <3 salaries)
  }
}
```

# 3 Spring Boot & Microservices

## 3.1 How do you configure different environments in Spring Boot?

**Answer:** Use application-{profile}.properties and spring.profiles.active.

```
# application-dev.properties
server.port=8081
```

```
import org.springframework.context.annotation.*;
@Profile("dev")
@Component
public class DevConfig {}
```

## 3.2 What is a Spring Boot profile and how did you use it?

**Answer:** Profiles activate environment-specific configurations.

```
# application.properties
spring.profiles.active=dev
```

## 3.3 What is exception handling and what is an advisor?

**Answer:** Exception handling manages errors; advisor combines AOP advice and pointcut.

```
import org.springframework.http.*;
import org.springframework.web.bind.annotation.*;
@RestController
public class ExceptionController {
  @ExceptionHandler(NullPointerException.class)
  public ResponseEntity<String> handleNPE() {
    return new ResponseEntity<>("Error", HttpStatus.BAD_REQUEST);
  }
}
```

## 3.4 How does @Transactional annotation work?

**Answer:** Manages transactions, ensuring atomicity.

```java
import org.springframework.stereotype.*;
import org.springframework.transaction.annotation.*;
@Service
public class UserService {
  @Transactional
  public void saveUser() {}
}
```

### 3.5 What are the design patterns used in microservices?

**Answer:** Circuit Breaker, API Gateway, Service Discovery, Event-Driven, CQRS.

### 3.6 Explain each design pattern and when to use them

**Answer:** Circuit Breaker: Prevents failures. API Gateway: Routing/security. Service Discovery: Dynamic location. Event-Driven: Loose coupling. CQRS: Complex domains.

```java
import io.github.resilience4j.circuitbreaker.annotation.*;
@Service
public class MyService {
  @CircuitBreaker(name = "myService")
  public String callApi() { return "Success"; }
}
```

### 3.7 What is microservices architecture?

**Answer:** Small, independent services communicating via APIs.

```java
import org.springframework.web.bind.annotation.*;
@RestController
public class UserController {
  @GetMapping("/users")
  public String getUsers() { return "User List"; }
}
```

### 3.8 Why microservices?

**Answer:** Scalability, independent deployment, fault isolation.

### 3.9 How do microservices communicate with each other?

**Answer:** Via REST, message queues, or gRPC.

```java
import org.springframework.web.client.RestTemplate;
public class Communication {
  public static void main(String[] args) {
    RestTemplate restTemplate = new RestTemplate();
    String response = restTemplate.getForObject("http://other-service/users
        ", String.class);
  }
}
```

### 3.10 Difference between synchronous and asynchronous communication

**Answer:** Synchronous: Blocking (REST). Asynchronous: Non-blocking (Kafka).

### 3.11 When to use synchronous vs asynchronous communication

**Answer:** Synchronous for immediate responses; asynchronous for decoupled systems.

```
import org.springframework.kafka.annotation.*;
@Service
public class KafkaConsumer {
  @KafkaListener(topics = "myTopic")
  public void consume(String message) {
    System.out.println(message);
  }
}
```

### 3.12 What is Kafka and how have you implemented it?

**Answer:** Distributed messaging system for event-driven architectures.

```
import org.springframework.kafka.core.*;
import org.springframework.stereotype.*;
@Service
public class KafkaProducer {
  @Autowired
  private KafkaTemplate<String, String> kafkaTemplate;
  public void sendMessage(String msg) {
    kafkaTemplate.send("myTopic", msg);
  }
}
```

### 3.13 How have you used Spring Security in your project?

**Answer:** Secures applications via authentication/authorization.

```
import org.springframework.context.annotation.*;
import org.springframework.security.config.annotation.web.builders.*;
@Configuration
@EnableWebSecurity
public class SecurityConfig {
  @Bean
  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
      Exception {
    http.authorizeRequests().anyRequest().authenticated().and().httpBasic()
        ;
    return http.build();
  }
}
```

### 3.14 What is JWT security and how have you used it?

**Answer:** Token-based authentication using JSON Web Tokens.

```
import io.jsonwebtoken.*;
public class JwtExample {
  public String generateToken(String username) {
    return Jwts.builder()
      .setSubject(username)
      .signWith(SignatureAlgorithm.HS512, "secret")
      .compact();
  }
```

```
}
```

### 3.15 What are the things to consider while developing REST APIs?

**Answer:** RESTful principles, versioning, error handling, security, documentation.

```
import org.springframework.http.*;
import org.springframework.web.bind.annotation.*;
@RestController
@RequestMapping("/api/v1")
public class ApiController {
  @GetMapping("/users")
  public ResponseEntity<List<String>> getUsers() {
    return ResponseEntity.ok(Arrays.asList("Alice", "Bob"));
  }
}
```

### 3.16 Difference between PUT, POST, and PATCH

**Answer:** POST creates, PUT updates entire resource, PATCH updates partially.

```
import org.springframework.web.bind.annotation.*;
@RestController
public class UserController {
  @PostMapping("/users") public String createUser() { return "Created"; }
  @PutMapping("/users/{id}") public String updateUser(@PathVariable int id)
      { return "Updated"; }
  @PatchMapping("/users/{id}") public String patchUser(@PathVariable int id
      ) { return "Patched"; }
}
```

### 3.17 API versioning strategies

**Answer:** URI versioning, query parameters, headers, media types.

```
import org.springframework.web.bind.annotation.*;
@RestController
@RequestMapping("/api/v1")
public class VersionedController {
  @GetMapping("/users")
  public String getUsers() { return "Version 1"; }
}
```

### 3.18 What is idempotency and why is it critical?

**Answer:** Ensures multiple identical requests have the same effect, critical for reliability.

```
import org.springframework.web.bind.annotation.*;
@RestController
public class IdempotentController {
  @PutMapping("/users/{id}")
  public ResponseEntity<String> updateUser(@PathVariable int id) {
    return ResponseEntity.ok("Updated");
  }
}
```

### 3.19 How do you achieve idempotency in microservices?

**Answer:** Use unique request IDs or design idempotent operations.

```
import java.util.*;
@Service
public class IdempotentService {
  private Set<String> processedIds = new HashSet<>();
  public String process(String requestId) {
    if (processedIds.contains(requestId)) return "Already processed";
    processedIds.add(requestId);
    return "Processed";
  }
}
```

### 3.20 Securing APIs using JWT and OAuth2

**Answer:** JWT for authentication, OAuth2 for authorization.

```
// See JWT example (Question 14)
```

### 3.21 Validating incoming payloads with annotations

**Answer:** Use @Valid and Bean Validation annotations.

```
import javax.validation.constraints.*;
public class User {
  @NotNull private String name;
}
import org.springframework.web.bind.annotation.*;
import javax.validation.*;
@RestController
public class UserController {
  @PostMapping("/users")
  public ResponseEntity<String> createUser(@Valid @RequestBody User user) {
    return ResponseEntity.ok("Valid");
  }
}
```

### 3.22 What are the steps to test Spring Boot microservices applications?

**Answer:** Unit tests, integration tests, mock external services, test REST APIs.

```
import org.junit.jupiter.api.*;
import org.springframework.boot.test.context.*;
import org.springframework.boot.test.web.client.*;
@SpringBootTest
public class UserControllerTest {
  @Autowired private TestRestTemplate restTemplate;
  @Test
  public void testGetUsers() {
    ResponseEntity<String> response = restTemplate.getForEntity("/users",
        String.class);
    Assertions.assertEquals(HttpStatus.OK, response.getStatusCode());
  }
}
```

### 3.23 How to handle exceptions in Spring Boot

**Answer:** Use @ExceptionHandler or @ControllerAdvice.

```
import org.springframework.http.*;
import org.springframework.web.bind.annotation.*;
@ControllerAdvice
public class GlobalExceptionHandler {
  @ExceptionHandler(Exception.class)
  public ResponseEntity<String> handleException(Exception e) {
    return new ResponseEntity<>("Error: " + e.getMessage(), HttpStatus.
        INTERNAL_SERVER_ERROR);
  }
}
```

### 3.24 How to create global exceptions and what annotations are used

**Answer:** Use @ControllerAdvice and @ExceptionHandler.

```
// See above
```

### 3.25 How to exclude classes from component scan

**Answer:** Use exclude or excludeFilters in @ComponentScan.

```
import org.springframework.context.annotation.*;
@ComponentScan(basePackages = "com.example", excludeFilters = @Filter(type
    = FilterType.ASSIGNABLE_TYPE, classes = MyClass.class))
@Configuration
public class AppConfig {}
```

### 3.26 How does component scan work?

**Answer:** Scans packages for stereotype-annotated classes, registering them as beans.

```
import org.springframework.stereotype.*;
@Component
public class MyComponent {}
```

### 3.27 What is the most challenging task you've done?

**Answer:** Optimizing a microservice for high load with caching.

```
import org.springframework.cache.annotation.*;
@Service
public class OptimizedService {
  @Cacheable("data")
  public String getData() { return "Data"; }
}
```

### 3.28 What are the top 3 performance bottlenecks in microservices?

**Answer:** Network latency, database queries, resource management.

```
import org.springframework.data.jpa.repository.*;
public interface UserRepository extends JpaRepository<User, Long> {
  @Query("SELECT u FROM User u WHERE u.active = true")
  List<User> findActiveUsers();
}
```

### 3.29 How do you monitor microservices?

**Answer:** Use Prometheus, Grafana, or Spring Actuator.

```
# application.properties
management.endpoints.web.exposure.include=*
```

### 3.30 How do you ensure system resiliency under high load?

**Answer:** Use circuit breakers, retries, load balancing, caching.

```
// See Circuit Breaker (Question 6)
```

### 3.31 What is centralized configuration and secrets management?

**Answer:** Centralized configuration (Spring Cloud Config); secrets (Vault).

```
# application.properties
spring.config.import=configserver:http://config-server
```

### 3.32 What is service discovery (Eureka/Consul)?

**Answer:** Dynamically locates services.

```
import org.springframework.cloud.netflix.eureka.*;
@SpringBootApplication
@EnableEurekaClient
public class Application {}
```

### 3.33 Inter-service communication: Feign vs RestTemplate vs WebClient

**Answer:** RestTemplate (synchronous), WebClient (reactive), Feign (declarative).

```
import feign.*;
@FeignClient(name = "user-service")
interface UserClient {
  @GetMapping("/users")
  List<String> getUsers();
}
```

### 3.34 Circuit Breaker and Retry: Resilience4j

**Answer:** Circuit Breaker prevents failures; Retry attempts failed operations.

```
import io.github.resilience4j.circuitbreaker.annotation.*;
import io.github.resilience4j.retry.annotation.*;
@Service
public class ResilientService {
  @CircuitBreaker(name = "myService")
```

```
  @Retry(name = "myService")
  public String call() { return "Success"; }
}
```

### 3.35 Blue-green vs canary deployments

**Answer:** Blue-Green: Two environments, switch traffic. Canary: Gradual rollout.

### 3.36 Handling version mismatch between services

**Answer:** Use API versioning or backward compatibility.

```
// See API versioning (Question 17)
```

### 3.37 What are Maven commands you use daily?

**Answer:** mvn clean install, mvn test, mvn package.

```
mvn clean install
```

### 3.38 Have you used Docker? Benefits and challenges?

**Answer:** Benefits: consistency, portability. Challenges: resource overhead.

```
FROM openjdk:11
COPY target/myapp.jar /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

## 4 Spring & Bean Lifecycle

### 4.1 Bean lifecycle and Spring container

**Answer:** Creation, dependency injection, initialization, use, destruction.

```
import javax.annotation.*;
@Component
public class MyBean {
  @PostConstruct
  public void init() { System.out.println("Initialized"); }
}
```

### 4.2 Use of @Component, @Service, @Repository

**Answer:** @Component: generic, @Service: business logic, @Repository: data access.

```
import org.springframework.stereotype.*;
@Repository
public class UserRepository {}
```

### 4.3 Role of @ComponentScan, @Configuration, @Bean

**Answer:** @ComponentScan scans beans, @Configuration defines config, @Bean creates beans.

```
import org.springframework.context.annotation.*;
@Configuration
@ComponentScan("com.example")
public class AppConfig {
  @Bean
  public MyBean myBean() { return new MyBean(); }
}
```

### 4.4 What is Spring Boot auto-configuration?

**Answer:** Automatically configures beans based on dependencies.

```
import org.springframework.boot.autoconfigure.*;
@SpringBootApplication
public class Application {}
```

## 5 JPA & Database

### 5.1 What is Fetch Type (Lazy vs Eager Loading)?

**Answer:** Lazy loads data on demand; Eager loads immediately.

```
import javax.persistence.*;
@Entity
public class User {
  @OneToMany(fetch = FetchType.LAZY)
  private List<Order> orders;
}
```

### 5.2 Lazy vs Eager loading — real-time use cases

**Answer:** Lazy for large datasets; Eager for small, frequent data.

### 5.3 Complex entity relationship experience

**Answer:** Managed @OneToMany, @ManyToMany with cascading.

```
import javax.persistence.*;
@Entity
public class Department {
  @OneToMany(mappedBy = "department")
  private List<Employee> employees;
}
```

### 5.4 What is the N+1 query problem?

**Answer:** Multiple queries for related data due to lazy loading.

```
List<User> users = repository.findAll();
for (User u : users) { u.getOrders().size(); }
```

### 5.5 How to optimize N+1 using Spring Data JPA

**Answer:** Use @EntityGraph or JOIN FETCH.

```
import org.springframework.data.jpa.repository.*;
public interface UserRepository extends JpaRepository<User, Long> {
  @EntityGraph(attributePaths = {"orders"})
  List<User> findAll();
}
```

### 5.6 Difference between get() and load() in Hibernate

**Answer:** get() loads immediately, returns null if not found; load() uses proxy, throws exception if not found.

```
import org.hibernate.*;
public class HibernateExample {
  public static void main(String[] args) {
    Session session = sessionFactory.getCurrentSession();
    User user = session.get(User.class, 1L);
    User proxy = session.load(User.class, 1L);
  }
}
```

### 5.7 Writing optimized JPQL and Criteria queries

```
import org.springframework.data.jpa.repository.*;
public interface UserRepository extends JpaRepository<User, Long> {
  @Query("SELECT u FROM User u WHERE u.age > ?1")
  List<User> findByAgeGreaterThan(int age);
}
```

### 5.8 Transaction management: @Transactional deep dive

**Answer:** Controls transaction boundaries, rolls back on exceptions.

```
import org.springframework.transaction.annotation.*;
@Transactional(rollbackOn = Exception.class)
public void saveUser(User user) {}
```

## 6 Testing & Mocking

### 6.1 Differences between stubbing and mocking

**Answer:** Stubbing provides canned responses; mocking verifies interactions.

```
import org.junit.jupiter.api.*;
import static org.mockito.Mockito.*;
public class MockTest {
  @Test
  public void testMock() {
    List<String> mockedList = mock(List.class);
    when(mockedList.get(0)).thenReturn("Stubbed");
    verify(mockedList).get(0);
  }
}
```

### 6.2 Why do we need both approaches?

**Answer:** Stubbing for outputs, mocking for behavior verification.

### 6.3 What is Spy in Mockito and when to use it?

**Answer:** Wraps real object, allows real calls unless overridden.

```
import org.junit.jupiter.api.*;
import static org.mockito.Mockito.*;
public class SpyTest {
  @Test
  public void testSpy() {
    List<String> list = new ArrayList<>();
    List<String> spy = spy(list);
    spy.add("test");
    verify(spy).add("test");
  }
}
```

### 6.4 How to write JUnit test cases for static methods

**Answer:** Use PowerMock or refactor to instance methods.

```
public class StaticUtil {
  public static String getName() { return "Test"; }
}
import org.junit.jupiter.api.*;
public class StaticTest {
  @Test
  public void testStatic() {
    Assertions.assertEquals("Test", StaticUtil.getName());
  }
}
```

### 6.5 What is the use of Mockito framework?

**Answer:** Creates mocks/spies for dependency-free testing.

```
import org.junit.jupiter.api.*;
import static org.mockito.Mockito.*;
public class MockitoTest {
  @Test
  public void testMockito() {
    Service service = mock(Service.class);
    when(service.getData()).thenReturn("Mocked");
    Assertions.assertEquals("Mocked", service.getData());
  }
}
```

## 7 Coding & Problem Solving

### 7.1 Reverse a string by preserving word position

```java
public class StringReverse {
  public static String reverseWords(String s) {
    String[] words = s.split(" ");
    StringBuilder result = new StringBuilder();
    for (String word : words) {
      result.append(new StringBuilder(word).reverse()).append(" ");
    }
    return result.toString().trim();
  }
  public static void main(String[] args) {
    System.out.println(reverseWords("Hello World")); // olleH dlroW
  }
}
```

### 7.2 Remove duplicates from string/array/list

```java
import java.util.*;
import java.util.stream.*;
public class RemoveDuplicates {
  public static void main(String[] args) {
    List<Integer> numbers = Arrays.asList(1, 2, 2, 3);
    List<Integer> unique = numbers.stream().distinct().collect(Collectors.
       toList());
    System.out.println(unique); // [1, 2, 3]
  }
}
```

### 7.3 Longest substring without repeating characters

```java
import java.util.*;
public class LongestSubstring {
  public static int lengthOfLongestSubstring(String s) {
    Set<Character> set = new HashSet<>();
    int max = 0, i = 0, j = 0;
    while (j < s.length()) {
      if (!set.contains(s.charAt(j))) {
        set.add(s.charAt(j++));
        max = Math.max(max, set.size());
      } else {
        set.remove(s.charAt(i++));
      }
    }
    return max;
  }
  public static void main(String[] args) {
    System.out.println(lengthOfLongestSubstring("abcabcbb")); // 3
  }
}
```

### 7.4 Check if two strings/numbers are palindrome

```java
public class Palindrome {
  public static boolean isPalindrome(String s) {
```

```
    s = s.toLowerCase().replaceAll("[^a-z0-9]", "");
    int i = 0, j = s.length() - 1;
    while (i < j) {
      if (s.charAt(i++) != s.charAt(j--)) return false;
    }
    return true;
  }
  public static void main(String[] args) {
    System.out.println(isPalindrome("A man, a plan, a canal: Panama")); //
       true
  }
}
```

### 7.5 Check if two strings/numbers are anagram

```
public class Anagram {
  public static boolean isAnagram(String s1, String s2) {
    if (s1.length() != s2.length()) return false;
    int[] count = new int[26];
    for (char c : s1.toCharArray()) count[c - 'a']++;
    for (char c : s2.toCharArray()) count[c - 'a']--;
    for (int c : count) if (c != 0) return false;
    return true;
  }
  public static void main(String[] args) {
    System.out.println(isAnagram("listen", "silent")); // true
  }
}
```

### 7.6 Sort an array/list/string

```
import java.util.*;
public class SortExample {
  public static void main(String[] args) {
    List<Integer> list = Arrays.asList(3, 1, 2);
    Collections.sort(list);
    System.out.println(list); // [1, 2, 3]
  }
}
```

### 7.7 Count occurrence of characters in a string

```
import java.util.*;
import java.util.stream.*;
public class CharCount {
  public static Map<Character, Integer> countChars(String s) {
    return s.chars()
      .mapToObj(c -> (char) c)
      .collect(Collectors.groupingBy(c -> c, Collectors.counting()))
      .entrySet().stream()
      .collect(Collectors.toMap(Map.Entry::getKey, e -> e.getValue().
         intValue()));
  }
  public static void main(String[] args) {
```

```
    System.out.println(countChars("hello")); // {h=1, e=1, l=2, o=1}
  }
}
```

### 7.8 Print duplicate characters in a string

```
import java.util.*;
import java.util.stream.*;
public class DuplicateChars {
  public static void printDuplicates(String s) {
    Map<Character, Long> map = s.chars()
      .mapToObj(c -> (char) c)
      .collect(Collectors.groupingBy(c -> c, Collectors.counting()));
    map.entrySet().stream()
      .filter(e -> e.getValue() > 1)
      .forEach(e -> System.out.println(e.getKey()));
  }
  public static void main(String[] args) {
    printDuplicates("hello"); // l
  }
}
```

### 7.9 Print only special characters in a string

```
public class SpecialChars {
  public static void printSpecial(String s) {
    s.chars()
      .mapToObj(c -> (char) c)
      .filter(c -> !Character.isLetterOrDigit(c))
      .forEach(System.out::println);
  }
  public static void main(String[] args) {
    printSpecial("Hello!@#"); // !, @, #
  }
}
```

### 7.10 Reverse the given string

```
public class ReverseString {
  public static String reverse(String s) {
    return new StringBuilder(s).reverse().toString();
  }
  public static void main(String[] args) {
    System.out.println(reverse("Hello")); // olleH
  }
}
```

### 7.11 Print difference between two strings

```
public class StringDiff {
  public static void printDiff(String s1, String s2) {
    for (int i = 0; i < Math.min(s1.length(), s2.length()); i++) {
      if (s1.charAt(i) != s2.charAt(i)) {
```

```
        System.out.println("Diff at " + i + ": " + s1.charAt(i) + " vs " +
            s2.charAt(i));
      }
    }
  }
  public static void main(String[] args) {
    printDiff("hello", "heelo"); // Diff at 2: l vs e
  }
}
```

### 7.12 Count words in a string

```
public class WordCount {
  public static int countWords(String s) {
    return s.trim().split("\\s+").length;
  }
  public static void main(String[] args) {
    System.out.println(countWords("Hello World")); // 2
  }
}
```

### 7.13 Find second highest element in an array

```
public class SecondHighest {
  public static int findSecondHighest(int[] arr) {
    int max = Integer.MIN_VALUE, second = Integer.MIN_VALUE;
    for (int num : arr) {
      if (num > max) {
        second = max; max = num;
      } else if (num > second && num != max) {
        second = num;
      }
    }
    return second;
  }
  public static void main(String[] args) {
    System.out.println(findSecondHighest(new int[]{5, 3, 8, 1})); // 5
  }
}
```

### 7.14 Find common elements of two arrays

```
import java.util.*;
import java.util.stream.*;
public class CommonElements {
  public static List<Integer> findCommon(int[] arr1, int[] arr2) {
    Set<Integer> set = Arrays.stream(arr1).boxed().collect(Collectors.toSet
        ());
    return Arrays.stream(arr2).filter(set::contains).boxed().collect(
        Collectors.toList());
  }
  public static void main(String[] args) {
    System.out.println(findCommon(new int[]{1, 2, 3}, new int[]{2, 3, 4}));
        // [2, 3]
```

```
  }
}
```

### 7.15 Decode string like a2b3c1 to aabbbc

```
public class DecodeString {
  public static String decode(String s) {
    StringBuilder result = new StringBuilder();
    for (int i = 0; i < s.length(); i += 2) {
      char c = s.charAt(i);
      int count = Character.getNumericValue(s.charAt(i + 1));
      result.append(String.valueOf(c).repeat(count));
    }
    return result.toString();
  }
  public static void main(String[] args) {
    System.out.println(decode("a2b3c1")); // aabbbc
  }
}
```

### 7.16 Check if a number is prime

```
public class PrimeCheck {
  public static boolean isPrime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i <= Math.sqrt(n); i++) {
      if (n % i == 0) return false;
    }
    return true;
  }
  public static void main(String[] args) {
    System.out.println(isPrime(7)); // true
  }
}
```

### 7.17 Generate Fibonacci series

```
public class Fibonacci {
  public static void printFibonacci(int n) {
    int a = 0, b = 1;
    for (int i = 0; i < n; i++) {
      System.out.print(a + " ");
      int next = a + b;
      a = b; b = next;
    }
  }
  public static void main(String[] args) {
    printFibonacci(5); // 0 1 1 2 3
  }
}
```

### 7.18 Find factorial of a number

```
public class Factorial {
  public static long factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
  }
  public static void main(String[] args) {
    System.out.println(factorial(5)); // 120
  }
}
```

### 7.19 Find min and max element in an array

```
public class MinMax {
  public static int[] findMinMax(int[] arr) {
    int min = arr[0], max = arr[0];
    for (int num : arr) {
      min = Math.min(min, num);
      max = Math.max(max, num);
    }
    return new int[]{min, max};
  }
  public static void main(String[] args) {
    int[] result = findMinMax(new int[]{3, 1, 4, 2});
    System.out.println("Min: " + result[0] + ", Max: " + result[1]); // Min
        : 1, Max: 4
  }
}
```

### 7.20 Find max repeated word in a sentence

```
import java.util.*;
import java.util.stream.*;
public class MaxRepeatedWord {
  public static String findMaxRepeated(String s) {
    Map<String, Long> map = Arrays.stream(s.split("\\s+"))
      .collect(Collectors.groupingBy(w -> w, Collectors.counting()));
    return map.entrySet().stream()
      .max(Map.Entry.comparingByValue())
      .map(Map.Entry::getKey)
      .orElse("");
  }
  public static void main(String[] args) {
    System.out.println(findMaxRepeated("hello world hello")); // hello
  }
}
```

### 7.21 Rotate array from k=2

```
public class RotateArray {
  public static void rotate(int[] arr, int k) {
    k = k % arr.length;
    reverse(arr, 0, arr.length - 1);
```

```
    reverse(arr, 0, k - 1);
    reverse(arr, k, arr.length - 1);
  }
  private static void reverse(int[] arr, int start, int end) {
    while (start < end) {
      int temp = arr[start];
      arr[start++] = arr[end];
      arr[end--] = temp;
    }
  }
  public static void main(String[] args) {
    int[] arr = {1, 2, 3, 4, 5};
    rotate(arr, 2);
    System.out.println(Arrays.toString(arr)); // [4, 5, 1, 2, 3]
  }
}
```

### 7.22 Merge characters from two strings alternately

```
public class MergeStrings {
  public static String mergeAlternately(String s1, String s2) {
    StringBuilder result = new StringBuilder();
    int i = 0;
    while (i < s1.length() || i < s2.length()) {
      if (i < s1.length()) result.append(s1.charAt(i));
      if (i < s2.length()) result.append(s2.charAt(i));
      i++;
    }
    return result.toString();
  }
  public static void main(String[] args) {
    System.out.println(mergeAlternately("abc", "pqr")); // apbqcr
  }
}
```

### 7.23 Find max repeated word in a sentence

```
// See Question 20
```

### 7.24 Modify and improve given code

**Answer:** Refactor for readability, efficiency, error handling.

```
public class BadCode {
  public void print(String s) { System.out.println(s); }
}
public class ImprovedCode {
  public void print(String s) {
    if (s == null) throw new IllegalArgumentException("Input cannot be null
        ");
    System.out.println(s.trim());
  }
}
```

### 7.25 Find a file in a subdirectory

```java
import java.nio.file.*;
import java.io.*;
public class FileSearch {
  public static void findFile(Path dir, String fileName) throws IOException
      {
    Files.walk(dir)
      .filter(p -> p.getFileName().toString().equals(fileName))
      .forEach(System.out::println);
  }
  public static void main(String[] args) throws IOException {
    findFile(Paths.get("."), "test.txt");
  }
}
```

### 7.26 Write a method to fetch employee details

```java
import org.springframework.http.*;
import org.springframework.web.bind.annotation.*;
@RestController
public class EmployeeController {
  @GetMapping("/employees/{id}")
  public ResponseEntity<Employee> getEmployee(@PathVariable Long id) {
    Employee emp = new Employee(id, "Alice");
    return ResponseEntity.ok(emp);
  }
}
class Employee {
  Long id; String name;
  Employee(Long id, String name) { this.id = id; this.name = name; }
}
```

## 8 Angular & Frontend

### 8.1 What is ng-content and how does content projection work?

**Answer:** ng-content projects content from parent to child components.

```html
<!-- child.component.html -->
<ng-content></ng-content>
<!-- parent.component.html -->
<app-child><h1>Projected Content</h1></app-child>
```

### 8.2 Difference between @ViewChild and @ContentChild

**Answer:** @ViewChild accesses template elements; @ContentChild accesses projected content.

```typescript
import { Component, ContentChild, ViewChild, ElementRef } from '@angular/
    core';
@Component({
  selector: 'app-child',
  template: '<ng-content></ng-content>'
})
```

```
export class ChildComponent {
  @ContentChild('content') content: ElementRef;
}
@Component({
  selector: 'app-parent',
  template: '<app-child><div #content>Content</div></app-child>'
})
export class ParentComponent {
  @ViewChild('view') view: ElementRef;
}
```

### 8.3 Explain Dependency Injection in Angular

**Answer:** Provides dependencies via constructor injection.

```
import { Injectable, Component } from '@angular/core';
@Injectable()
class MyService {}
@Component({})
export class MyComponent {
  constructor(private service: MyService) {}
}
```

### 8.4 How does Angular load dynamic components?

**Answer:** Uses ComponentFactoryResolver and ViewContainerRef.

```
import { Component, ViewChild, ViewContainerRef, ComponentFactoryResolver }
    from '@angular/core';
@Component({
  selector: 'app-dynamic',
  template: '<ng-container #container></ng-container>'
})
export class DynamicComponent {
  @ViewChild('container', { read: ViewContainerRef }) container:
     ViewContainerRef;
  constructor(private resolver: ComponentFactoryResolver) {}
  ngOnInit() {
    const factory = this.resolver.resolveComponentFactory(
       MyDynamicComponent);
    this.container.createComponent(factory);
  }
}
```

### 8.5 What is the PipeTransform interface?

**Answer:** Defines custom pipes for data transformation.

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({ name: 'myPipe' })
export class MyPipe implements PipeTransform {
  transform(value: string): string {
    return value.toUpperCase();
  }
}
```

### 8.6 How does Angular bootstrapping work via AppModule?

**Answer:** AppModule defines root module, bootstrapped via platformBrowserDynamic.

```
import { NgModule } from '@angular/core';
@NgModule({
  declarations: [AppComponent],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

### 8.7 What is an HTTP interceptor?

**Answer:** Intercepts HTTP requests/responses for headers, error handling.

```
import { Injectable, HttpInterceptor, HttpRequest, HttpHandler } from '
    @angular/common/http';
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler) {
    const authReq = req.clone({ setHeaders: { Authorization: 'Bearer token'
        } });
    return next.handle(authReq);
  }
}
```

### 8.8 Common use cases of HTTP interceptors

**Answer:** Adding tokens, logging, error handling.

### 8.9 What is GraphQL and how does it compare to REST?

**Answer:** GraphQL: Query-based, flexible. REST: Fixed endpoints.

```
query {
  user(id: 1) {
    name
  }
}
```

### 8.10 Use Apollo in Angular to fetch GraphQL data

```
import { Component } from '@angular/core';
import { Apollo } from 'apollo-angular';
import gql from 'graphql-tag';
@Component({})
export class UserComponent {
  constructor(private apollo: Apollo) {
    this.apollo.query({ query: gql`{ user(id: 1) { name } }` })
      .subscribe(result => console.log(result.data));
  }
}
```

### 8.11 Differences: BehaviorSubject, Subject, ReplaySubject

**Answer:** Subject: No initial value. BehaviorSubject: Initial value, emits last. ReplaySubject: Replays multiple values.

```
import { BehaviorSubject } from 'rxjs';
const subject = new BehaviorSubject('initial');
subject.subscribe(v => console.log(v)); // initial
subject.next('new'); // new
```

### 8.12 Explain: switchMap, mergeMap, concatMap, exhaustMap

**Answer:** switchMap cancels previous, mergeMap runs concurrently, concatMap sequential, exhaustMap ignores new until complete.

```
import { of } from 'rxjs';
import { switchMap } from 'rxjs/operators';
of(1, 2, 3).pipe(
  switchMap(id => of(`User ${id}`))
).subscribe(console.log); // User 3
```

### 8.13 Design a semantic HTML navigation menu

```
<nav>
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/about">About</a></li>
  </ul>
</nav>
```

### 8.14 List 5 semantic HTML tags and their uses

**Answer:** header (page header), nav (navigation), main (main content), article (independent content), footer (page footer).

### 8.15 Fetch API data and display in a table

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';
@Component({
  template: `
    <table>
      <tr *ngFor="let user of users">
        <td>{{ user.name }}</td>
      </tr>
    </table>
  `
})
export class UserTableComponent {
  users: any[] = [];
  constructor(private http: HttpClient) {
    this.http.get('https://api.example.com/users').subscribe(data => this.
        users = data);
  }
}
```

### 8.16 Search by name using Reactive Form

```
import { Component } from '@angular/core';
import { FormGroup, FormControl } from '@angular/forms';
@Component({
  template: `
    <form [formGroup]="form">
      <input formControlName="name">
    </form>
    <div *ngFor="let user of filteredUsers">{{ user.name }}</div>
  `
})
export class SearchComponent {
  form = new FormGroup({ name: new FormControl('') });
  users = [{ name: 'Alice' }, { name: 'Bob' }];
  filteredUsers = this.users;
  ngOnInit() {
    this.form.get('name').valueChanges.subscribe(value => {
      this.filteredUsers = this.users.filter(u => u.name.includes(value));
    });
  }
}
```

### 8.17 Error handling on API failure

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { of } from 'rxjs';
import { catchError } from 'rxjs/operators';
@Component({})
export class ErrorComponent {
  constructor(private http: HttpClient) {
    this.http.get('https://api.example.com/users')
      .pipe(catchError(err => of([])))
      .subscribe(data => console.log(data));
  }
}
```

### 8.18 Flatten a nested array

```
const nested = [1, [2, 3], [4, [5]]];
const flat = nested.flat(Infinity);
console.log(flat); // [1, 2, 3, 4, 5]
```

## 9 Miscellaneous

### 9.1 What do you know about ISO8583?

**Answer:** Standard for financial transaction messaging.

### 9.2 How do Angular applications interact with backend APIs?

**Answer:** Via HttpClient for REST or Apollo for GraphQL.

```
// See Question 15, Section 8
```

### 9.3 How do you clone code from Git and commit changes?

**Answer:** Clone: git clone. Commit: git add, git commit, git push.

```
git clone https://github.com/repo.git
git add .
git commit -m "Add feature"
git push
```

### 9.4 How to resolve merge conflicts?

**Answer:** Edit conflicting files, mark resolved, commit.

```
git pull
# Resolve conflicts
git add .
git commit
```

### 9.5 Daily Git commands you use

**Answer:** git pull, git add, git commit, git push, git status.

## 10 Additional Frequently Asked Questions

### 10.1 What is the difference between checked and unchecked exceptions?

**Answer:** Checked: Compile-time, must handle (e.g., IOException). Unchecked: Runtime, optional (e.g., NullPointerException).

```
public class ExceptionExample {
  public static void main(String[] args) {
    try {
      new FileReader("file.txt");
    } catch (IOException e) {
      System.out.println("Checked: " + e);
    }
  }
}
```

### 10.2 What is the purpose of @Autowired annotation in Spring?

**Answer:** Injects dependencies automatically.

```
import org.springframework.stereotype.*;
import org.springframework.beans.factory.annotation.*;
@Component
class MyComponent {
  @Autowired
  private MyService service;
}
```

### 10.3 What is the Circuit Breaker pattern in microservices?

**Answer:** Prevents cascading failures by stopping requests to failing services.

```
import io.github.resilience4j.circuitbreaker.annotation.*;
@Service
public class CircuitBreakerService {
  @CircuitBreaker(name = "myService")
  public String call() { return "Success"; }
}
```