Java + Spring Boot + Microservices Interview Notes

# How can we make sure an object is singleton while deserializing the object?

To ensure a singleton during deserialization, implement `readResolve()` method in your class:  
  
```java  
public class Singleton implements Serializable {  
 private static final Singleton INSTANCE = new Singleton();  
  
 private Singleton() {}  
  
 public static Singleton getInstance() {  
 return INSTANCE;  
 }  
  
 protected Object readResolve() {  
 return INSTANCE;  
 }  
}  
```  
  
Follow-up:  
- What if `readResolve()` is not implemented?  
- How does serialization break the Singleton pattern?

# How can we secure APIs using Gateway design pattern?

To secure APIs using an API Gateway:  
  
1. Use Authentication & Authorization at the gateway level (e.g., JWT tokens).  
2. Validate tokens using Spring Security or a filter.  
3. Configure rate limiting, logging, and IP whitelisting.  
  
Example with Spring Cloud Gateway:  
```yaml  
filters:  
 - name: JwtAuthenticationFilter  
 args:  
 tokenHeader: Authorization  
```  
  
Follow-up:  
- What is the advantage of putting auth logic at the gateway instead of individual services?

# How does Spring Security work?

Spring Security is a framework that handles authentication, authorization, and protection against common attacks (CSRF, CORS, etc.).  
  
Key Components:  
- SecurityFilterChain  
- AuthenticationManager  
- UserDetailsService  
  
```java  
@Bean  
SecurityFilterChain security(HttpSecurity http) throws Exception {  
 return http.csrf().disable()  
 .authorizeHttpRequests(auth -> auth.anyRequest().authenticated())  
 .httpBasic().and().build();  
}  
```  
  
Follow-up:  
- How to customize UserDetailsService?  
- How to add JWT filters in Spring Security?

# What are intermediate and terminal operators in Java Stream API?

- Intermediate Operators: `map()`, `filter()`, `sorted()`, `distinct()` (lazy, return Stream)  
- Terminal Operators: `collect()`, `forEach()`, `count()`, `reduce()` (trigger evaluation)  
  
Example:  
```java  
list.stream().filter(x -> x > 10).map(x -> x \* 2).collect(Collectors.toList());  
```  
  
Follow-up:  
- What is lazy evaluation in Streams?

# What is the Executor Framework and CompletableFuture?

Executor Framework is used to run tasks asynchronously.  
  
Example:  
```java  
ExecutorService service = Executors.newFixedThreadPool(10);  
service.submit(() -> System.out.println("Task executed"));  
```  
  
`CompletableFuture` helps write async, non-blocking code.  
```java  
CompletableFuture.supplyAsync(() -> "Hello")  
 .thenApply(String::toUpperCase)  
 .thenAccept(System.out::println);  
```  
  
Follow-up:  
- What is the difference between Future and CompletableFuture?

# Other Questions (To Be Expanded)

- How to secure API?  
- Stream API advantage over regular  
- Lazy and eager evaluation explanation and benefits  
- Microservices: Communication, Feign Client, Token Authentication  
- Query optimization for production  
- Roman number conversion for 435 and 65 (Java code)  
- Stream API for 5th highest salary and SQL version  
- Connect two databases in Spring Boot  
- Dependency injection for two DBs and routing  
- Selective class inclusion in libraries  
- Building high-level microservice architecture  
- Abstracting many services and service discovery  
- Complex transaction scenarios in microservices  
- HTTPS in Spring Boot  
- Memory leak detection and prevention  
- Deployment to GCP: Manual and CI/CD  
- API Gateway, gRPC, Messaging services  
- Async communication using Feign, @Async usage  
- Java: Singleton, List vs ArrayList, dynamic array, utility methods  
- Java output questions, exceptions (division by 0)  
- Java 8: Skip and sum squares with Streams