Spring Boot Interview Questions & Answers

Chapter 9: Spring Boot – Logging

Q: What is logging in Spring Boot and why is it important?

Logging is the process of recording application events and messages to help developers understand what's happening inside the application. In Spring Boot, logging is important for debugging, monitoring, and troubleshooting both during development and in production.

Use-Case / Example: When a payment fails in an e-commerce app, logs can show whether it was due to a database error, network timeout, or incorrect user input.

Q: What logging framework does Spring Boot use by default?

Spring Boot uses Spring Boot Starter Logging, which internally uses Logback as the default logging framework. It also supports other frameworks like Log4j2 and Java Util Logging if needed.

Use-Case / Example: By default, when you run a Spring Boot app, you'll see startup logs in the console generated by Logback.

Q: What are the common logging levels in Spring Boot?

The common levels are:
- TRACE → very detailed information
- DEBUG → useful for developers
- INFO → general application flow
- WARN → warnings about potential problems
- ERROR → serious issues

Use-Case / Example: In development, you might set logging to DEBUG to see detailed messages, but in production, you use INFO or WARN to reduce noise.

Q: How do you configure logging in Spring Boot?

You can configure logging in application.properties or application.yml by setting levels for packages or classes.

Use-Case / Example:
logging.level.org.springframework=INFO
logging.level.com.myapp.service=DEBUG
This shows only INFO logs for Spring classes but DEBUG logs for your own service classes.

Q: How do you change the logging file output?

By default, logs appear in the console. You can write them to a file by adding:
logging.file.name=app.log
This creates a log file in the project directory.

Use-Case / Example: In a production system, you might write logs to /var/logs/myapp.log so DevOps teams can analyze them later.

Q: How do you use different log levels for different packages?

You can specify log levels per package in application.properties.

Use-Case / Example:
logging.level.com.myapp.controller=DEBUG
logging.level.com.myapp.repository=ERROR
This way, you see detailed logs for controllers but only error logs for repository classes.

Q: Can you use other logging frameworks with Spring Boot?

Yes, you can replace Logback with Log4j2 or even Java Util Logging by excluding the default logging starter and adding the desired logging dependency.

Use-Case / Example: If a company already uses Log4j2 across all applications, they can configure Spring Boot to use Log4j2 for consistency.

Q: How do you use @Slf4j in Spring Boot?

@Slf4j is a Lombok annotation that automatically creates a logger instance for your class. You can then use log.info(), log.error(), etc., without manually creating a logger.

Use-Case / Example:

```
@Slf4j
@RestController
public class UserController {
@GetMapping("/user")
public String getUser() {
log.info("Fetching user details");
return "User";
}
}
```

Q: What is the difference between system.out.println and logging?

System.out.println is just printing to the console, while logging frameworks provide levels, formatting, and output destinations (console, file, monitoring tools). Logging is structured and much more powerful.

Use-Case / Example: Instead of using System.out.println("Error occurred"), use log.error("Error occurred while saving order") which can be filtered, stored, and analyzed.

Q: How is logging used in real-world Spring Boot projects?

Logging is used for debugging issues, monitoring system performance, auditing user actions, and integrating with monitoring tools like ELK (Elasticsearch, Logstash, Kibana) or Splunk.

Use-Case / Example: In an online banking system, all login attempts are logged at INFO level, and failed transactions are logged at ERROR level. These logs are then sent to a central ELK system for analysis.