1. Different Logging Levels in Log4j and SLF4J:

- TRACE : Most detailed level, used for fine-grained debugging.
- DEBUG : Detailed information for debugging
- INFO : General operational message to track the application flow.
- WARN : Indications of potential issues, but not necessarily errors.
- ERROR : Errors that affect functionality but do not crash the application.
- FATAL : Critical errors causing application shutdown (Log4j).

SLF4J vs Log4j : SLF4j is a facade that allows switching between logging frameworks while Log4j is an actual Logging implementation.

---

2. Configuring Log4j to Log to Console and File Simultaneously:

In Log4j.properties

```
Log4j.rootLogger = DEBUG, console file
Log4j.appender.console = org.apache.log4j.Console Appender
Log4j.appender.console.layout = org.apache.Log4j.pattern
                                                    Layout

log4j.appender.console.layout.ConversionPattern = %d
 {yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L-%m%n

Log4j.appender.file = org.apache.Log4j.File Appender
Log4j.appender.file.File = application.log
Log4j.appender.file.Layout = org.apache.log4j.Pattern Layout
```

log4j.appender.file.layout.conversionPattern=%d{yyyy-MM-dd HH:mm:ss}%-5p%c{1}:%L - %m%n

3. Performance Issues with DEBUG/TRACE in Production and Mitigation

Issues:-

- Increased CPU & I/O usage
- Large log files consuming disk space

Mitigation:-

- Set the logging level to INFO or WARN in production
- Use asynchronous logging (Logback AsyncAppender)
- Limit log file size with Rolling File Appender.

4) Appender vs. Logger vs. Layout

- Logger : The main logging entity that logs messages based on level.
- Appender : Specifies where logs should be written (console, file, database)
- Layout : Defines the format of the log message. (Eg: JSON, XML, pattern)

5) Prevent sensitive Information from Being logged

- Use logging filters to mask sensitive data
- Implement MDC (Mapped Diagnostic Context) to track sensitive logs.
- Avoid logging raw user input directly.
- Use custom log sanitization utilities before logging

6) Rolling File Appender in Log4j (Keeping Only Last 10 Log Files)

```
log4j.appender.rolling = org.apache.log4j.RollingFileAppender
log4j.appender.rolling.File = application.log
log4j.appender.rolling.MaxFileSize = 10 MB
log4j.appender.rolling.layout = org.apache.log4j.PatternLayout
log4j.appender.rolling.layout.ConversionPattern = %d
        {yyyy-MM-dd HH:MM:ss} %-5p %c{1}:%L - %m%n
```

7. Enable Asynchronous Logging in Logback / Log4j:

Logback :-

```
<appender name = "ASYNC" class = "ch.qos.logback.classic.
                                         AsyncAppender">
        <queue size>5000 </queuesize>
        <appender-ref ref = "FILE" />
</appender>
```

Log4j2 :-

```
<AsyncLogger name = "com.example" level = "INFO" />
```

8) SLF4J vs. Log4j

- SLF4J : Acts as a facade; supports multiple logging frameworks
- Log4j : A specific logging framework with more built-in features.

Why choose SLF4J? If you need flexibility and want to switch frameworks easily.

9) Best Practices for logging in Spring Boot

- Use application.properties or application.:

```
logging.level.root= INFO
logging.level.com.example = DEBUG
logging.file.name= app.log
```

- Use structured logging with JSON for better readability
- Avoid excessive logging in production.

---

11) HQL vs. SQL & Why HQL is preferred

- HQL : Object-oriented, works with Hibernate entities
- SQL : Works directly with database tables.
- Why HQL ? Database-independent, supports lazy loading and uses entity relationships.

---

12) HQL Query to Fetch Employees with Salary > x

```
String hql = "FROM employee e WHERE e.salary > :salary";
Query query = session.createQuery(hql);
query.setParameter("salary", 50000);
List <Employee> employees = query.list();
```

---

13) Pagination in HQL

```
Query query = session.createQuery("FROM Employee");
query.setFirstResult(10); //skip first 10 records
query.setMaxResults(20); //Fetch next 20 records
List <Employee> employees = query.list();
```

---

14) JOIN in HQL vs SQL

- HQL

```
String hql = "SELECT e FROM employee e JOIN e.department
        d WHERE d.name = :deptName";
```

• SQL

```
SELECT * FROM Employee e INNER JOIN Department d ON e.dept...
WHERE d.name = "Sales";
```

• key Difference : HQL uses entity relationships instead of table joins.

15) Bulk UPDATE & DELETE in HQL

HQL Update:

```
Query query = session.createQuery("UPDATE Employee e SET e.
salary = :newsalary WHERE e.id = :empId");
query.setParameter("newSalary", 70000);
query.setParameter("empId", 101);
query.executeUpdate();
```

HQL Delete:

```
Query query = session.createQuery("DELETE FROM Employee e
                          WHERE e.id = :empId");
query.setParameter("empId", 101);
query.executeUpdate();
```