If you're building scalable and maintainable applications with Spring Boot, applying the SOLID principles can elevate your code from just working to truly clean, modular, and extensible. Here's how each principle applies in real-world Spring Boot development:  
  
🔹 S – Single Responsibility Principle (SRP): Keep your classes focused. A @Controller handles HTTP requests, a @Service contains business logic, and a @Repository manages data access — don’t mix responsibilities. Cleaner separation = easier testing and refactoring.  
  
🔹 O – Open/Closed Principle (OCP): Your classes should be open for extension but closed for modification. Spring's dependency injection makes it easy to inject new behavior via interfaces without touching existing logic.  
  
🔹 L – Liskov Substitution Principle (LSP): Subtypes should be replaceable for their base types without breaking the app. Design your @Service or strategy beans so any implementation can be injected and used seamlessly.  
  
🔹 I – Interface Segregation Principle (ISP): Don’t force classes to implement methods they don’t need. Break down large interfaces into smaller, specific ones — especially helpful with Spring Data repositories or service contracts.  
  
🔹 D – Dependency Inversion Principle (DIP): High-level modules shouldn’t depend on low-level modules — both should depend on abstractions. Use constructor injection with interfaces and let Spring manage the implementation.  
  
✅ Writing SOLID-aligned code not only enhances maintainability, but also improves testability, scalability, and developer happiness 😄  
  
💬 What’s your favorite SOLID principle to apply in Spring Boot — and how has it helped you write better code?

1. What is SQL, and what are its different types?  
   SQL (Structured Query Language) is used to manage and manipulate relational databases. It has various types: DDL, DML, DCL, and TCL.  
     
   2. Explain the difference between SQL and MySQL.  
   SQL is a query language, while MySQL is a relational database management system (RDBMS) that uses SQL for database operations.  
     
   3. What are the different types of joins in SQL? Provide examples.  
   Joins combine rows from two or more tables. Types include INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN, and CROSS JOIN.  
     
   4. What is normalization, and how does it differ from denormalization?  
   Normalization organizes data to reduce redundancy, while denormalization combines tables to improve read performance.  
     
   5. Explain the different types of normal forms.  
   Normal forms (1NF to 5NF) are rules to structure data, ensuring data integrity and reducing redundancy.  
     
   6. What is the difference between DELETE, TRUNCATE, and DROP commands?  
   DELETE removes rows, TRUNCATE removes all rows but keeps the structure, and DROP removes the entire table.  
     
   7. What are primary keys and foreign keys?  
   A primary key uniquely identifies a row, while a foreign key establishes a relationship between two tables.  
     
   8. How do INNER JOIN and OUTER JOIN differ?  
   INNER JOIN returns matching rows, while OUTER JOIN includes unmatched rows (LEFT, RIGHT, FULL).  
     
   9. What is a subquery? Explain with an example.  
   A subquery is a query nested within another query, often used for filtering or calculations.  
     
   10. What are indexes in SQL, and why are they important?  
   Indexes improve query performance by speeding up data retrieval.  
     
   11. What are aggregate functions in SQL? Provide some examples.  
   Aggregate functions perform calculations on a set of values (e.g., SUM, AVG, COUNT, MAX, MIN).  
     
   12. What is the difference between GROUP BY and HAVING clauses?  
   GROUP BY groups rows, while HAVING filters grouped data.  
     
   13. What is the purpose of the WHERE clause?  
   WHERE filters rows based on a condition before grouping or aggregation.  
     
   14. What are the different types of constraints in SQL?  
   Constraints include PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK, and DEFAULT.  
     
   15. What is a stored procedure, and how does it differ from a function?  
   A stored procedure executes a set of SQL statements, while a function returns a value and is used in expressions.  
   1. @SpringBootApplication  
   Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan to set up the Spring Boot application.  
     
   2. @RestController  
   Marks a class as a controller where each method returns a response body (REST APIs).  
     
   3. @RequestMapping  
   Maps HTTP requests to handler methods in a controller.  
     
   4. @GetMapping / @PostMapping / @PutMapping / @DeleteMapping  
   Shortcut annotations for mapping HTTP GET, POST, PUT, and DELETE requests.  
     
   5. @PathVariable  
   Used to extract values from the URI template.  
     
   6. @RequestParam  
   Extracts query parameters from the request URL.  
     
   7. @Autowired  
   Injects dependencies automatically where required.  
     
   8. @Component  
   Marks a class as a Spring-managed component for dependency injection.  
     
   9. @Service  
   Specialized annotation for service layer components.  
     
   10. @Repository  
   Indicates a data access layer component, often used with JPA.  
     
   11. @Entity  
   Marks a class as a JPA entity for database mapping.  
     
   12. @Table  
   Specifies the database table name for a JPA entity.  
     
   13. @Column  
   Defines the column name and properties for a field in a JPA entity.  
     
   14. @Transactional  
   Manages transactions automatically for a method or class.  
     
   15. @Configuration  
   Indicates a class contains Spring configuration.  
     
   16. @Bean  
   Defines a Spring-managed bean in a configuration class.  
     
   17. @EnableScheduling  
   Enables scheduling for tasks with @Scheduled annotation.  
     
   18. @Scheduled  
   Schedules a task to run at fixed intervals or cron expressions.  
     
   19. @EnableAsync  
   Enables asynchronous processing in the application.  
     
   20. @ExceptionHandler  
   Handles exceptions at the controller level.

1.  @Required: Ensures a bean property must be set.  
2.  @Autowired: Automatically injects dependencies.  
3. @Configuration: Declares @Bean methods in a class.  
4. @ComponentScan: Configures component scanning.  
5. @Bean: Produces a managed Spring bean.  
6. @Qualifier: Specifies bean injection options.  
7. @Lazy: Delays bean initialization.  
8. @Value: Injects a property value.  
9. @Component: Marks a Spring component.  
10. @Controller: Handles MVC views.  
11. @Service: Marks service layer components.  
12. @Repository: Marks DAOs with exception translation.  
13. @EnableAutoConfiguration: Enables auto-configuration.  
14. @SpringBootApplication: Configures Spring Boot app.  
15. @RequestMapping: Maps HTTP methods.  
16. @GetMapping: Handles HTTP GET requests.  
17. @PostMapping: Maps HTTP POST requests.  
18. @PutMapping: Maps HTTP PUT requests.  
19. @DeleteMapping: Maps HTTP DELETE requests.  
20. @PatchMapping: Maps HTTP PATCH requests.  
21. @RequestBody: Binds HTTP request body.  
22. @ResponseBody: Binds HTTP response body.  
23. @PathVariable: Extracts URI values.  
24. @RequestParam: Extracts query parameters.  
25. @RequestHeader: Extracts header values.  
26. @RestController: Combines @Controller and @ResponseBody.  
27. @RequestAttribute: Binds request attributes.  
28. @CookieValue: Binds HTTP cookie values.  
29. @CrossOrigin: Enables CORS.  
30. @Profile: Specifies bean profiles.  
31. @Scope: Defines bean scopes.  
32. @Conditional: Registers beans conditionally.  
33. @Primary: Sets primary autowired beans.  
34. @PropertySource: Adds PropertySource to Environment.  
35. @EnableAsync: Enables asynchronous methods.  
36. @EnableScheduling: Enables scheduled tasks.  
37. @EnableCaching: Enables caching.  
38. @RestControllerAdvice: Specializes @ControllerAdvice for REST.  
39. @JsonIgnoreProperties: Ignores JSON properties.  
40. @JsonProperty: Maps JSON properties to Java fields.