Core Concepts of Spring Boot

1. Dependency Injection (DI)

Definition:
Dependency Injection (DI) is a design pattern used to implement **Inversion of Control**. It allows a class to receive its dependencies from external sources rather than creating them itself.

Example (Without DI):

```
UserService userService = new UserService(); // Tightly coupled
```

Example (With DI in Spring Boot):

```
@Autowired
private UserService userService;  // Dependency is injected by Spring
```

Types of Dependency Injection in Spring:

- **Constructor Injection**
- **Setter Injection**
- **Field Injection** (most common with `@Autowired`)

Benefits:

- Reduces tight coupling between components
- Improves testability and maintainability
- Promotes reusability

2. Inversion of Control (IoC)

Definition:
Inversion of Control is a principle where the control of object creation and wiring is given to a framework (like Spring), instead of being handled manually in the application code.

Spring IoC Container:

- Uses configuration metadata (annotations or XML) to manage the lifecycle of application objects.
- Injects beans automatically wherever required.

3. Key Annotations in Spring Boot for Bean Management

Annotation	Purpose	Layer
`@Component`	Generic bean; used when the class doesn't fit other roles	Utility/helper class
`@Service`	Marks a class as a service layer component	Business logic
`@Repository`	Marks a class as a DAO layer component	Data access layer
`@Controller`	Handles HTTP requests for MVC apps	Web controller
`@RestController`	Combines `@Controller` and `@ResponseBody`	REST APIs
`@Autowired`	Injects a Spring-managed bean automatically	Used across all layers

Example:

```
@Service
public class ProductService {
    public List<String> getProducts() {
        return List.of("Laptop", "Phone");
    }
}
```

4. Creating RESTful Web Services in Spring Boot

Key Annotations:

- `@RestController`: Marks the class as a REST controller.
- `@RequestMapping`: Sets the base path for API routes.
- `@GetMapping`: Handles HTTP GET requests.
- `@PostMapping`: Handles HTTP POST requests.
- `@PutMapping`: Handles HTTP PUT requests.
- `@DeleteMapping`: Handles HTTP DELETE requests.
- `@RequestBody`: Maps HTTP request body to a Java object.

Sample Code:

```
@RestController
@RequestMapping("/api")
public class ProductController {

    @GetMapping("/products")
    public List<String> getProducts() {
        return List.of("Laptop", "Mobile");
    }

    @PostMapping("/products")
    public String addProduct(@RequestBody String product) {
        return "Product added: " + product;
    }
}
```

5. Controller-Service-Repository Layering in Spring Boot

Spring Boot follows a clean separation of concerns via **Layered Architecture**.

Layer	Responsibility	Annotation
Controller	Handles incoming HTTP requests	`@RestController`
Service	Implements business logic	`@Service`
Repository	Interacts with the database (DAO)	`@Repository`

Example of Layered Application

1. Model Class:

```
public class User {
    private int id;
    private String name;
    // Getters and Setters
}
```

2. Repository Layer:

```
@Repository
public class UserRepository {
    public List<User> findAllUsers() {
        return List.of(new User(1, "Akshay"), new User(2, "Sneha"));
    }
}
```

3. Service Layer:

```
@Service
public class UserService {
    @Autowired
    private UserRepository repository;

    public List<User> getUsers() {
```

```
        return repository.findAllUsers();
    }
}
```

4. Controller Layer:

```
@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired
    private UserService service;

    @GetMapping
    public List<User> getAllUsers() {
        return service.getUsers();
    }
}
```

Summary:

Concept	Key Idea
DI (Dependency Injection)	Inject objects rather than creating them manually
IoC (Inversion of Control)	Spring takes control of object creation and wiring
Spring Annotations	Annotations like `@Component`, `@Service`, `@Autowired` simplify configuration
REST APIs	Built using `@RestController`, `@GetMapping`, `@PostMapping`
Layered Architecture	Clean separation: Controller → Service → Repository