---

1. @SpringBootApplication

English:
This annotation is used at the main class of a Spring Boot application. It combines **@Configuration, @EnableAutoConfiguration, and @ComponentScan**.

Hindi:
Ye Spring Boot ke main class me use hota hai. Iske andar 3 cheezein automatically hoti hain:

1. Beans ka configuration (`@Configuration`)
2. Auto-configuration (`@EnableAutoConfiguration`)
3. Component scanning (`@ComponentScan`)

```
// Main entry point of Spring Boot App
@SpringBootApplication
public class MySpringBootApplication {

    public static void main(String[] args) {
        // SpringApplication.run() app ko start karta hai
        SpringApplication.run(MySpringBootApplication.class, args);
    }
}
```

---

2. @Configuration

English:
Marks a class as a configuration class where beans are defined.

Hindi:
Isse ek class ko configuration ke liye use karte hain jaha beans banaye jaate hain.

```
@Configuration
public class MyConfig {

    // @Bean ek object ko Spring container me register karta hai
    @Bean
    public String getMessage() {
        return "Hello from Bean!";
    }
}
```

---

3. @ComponentScan

English:
Tells Spring where to search for components (@Component, @Service, @Repository, etc.)

Hindi:
Ye Spring ko batata hai ki konse package me components dhoondhne hain.

```
@SpringBootApplication
@ComponentScan(basePackages = "com.example.demo")
// ab com.example.demo package aur uske sub-packages scan honge
public class App {
    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }
}
```

---

4. @Bean

English:
Used inside `@Configuration` class to define Spring-managed beans.

Hindi:
Ye method ke return object ko bean bana deta hai jo Spring container ke control me hota hai.

```
@Configuration
public class BeanConfig {

    @Bean
    public UserService userService() {
        // yaha hum manually ek bean create kar rahe hain
        return new UserService();
    }
}
```

---

5. @Component

English:
Marks a class as a Spring-managed component.

Hindi:
Isse Spring automatically ek class ka object banata hai aur manage karta hai.

```
@Component
public class ProductService {
    // Spring khud is class ka bean create karega
    public String getProductName() {
        return "Laptop";
    }
}
```

---

6. @Service

English:
Marks a class as a service provider. Used for **business logic layer**.

Hindi:
Isse ek class ko **service layer** (business logic handle karne wali) batate hain.

```
@Service
public class OrderService {
    public String placeOrder() {
        return "Order placed successfully!";
    }
}
```

👉 Matlab: Business logic ke liye class define karte ho to uspe `@Service` lagate ho.

---

7. @Repository

English:
Marks a class as a repository (DAO layer). Used for database operations.

Hindi:
Database operations ke liye use hone wali class ko batata hai.

```
@Repository
public class UserRepository {
    public String saveUser(String name) {
        // yaha DB me data save karne ka code hoga
        return "User saved: " + name;
    }
}
```

👉 Matlab: Database se baat karne wali classes ke liye use hota hai.

---

8. @Autowired

English:
Automatically injects required dependency (by type).

Hindi:
Automatically ek bean (object) ko doosre bean me inject kar deta hai.

```
@Service

public class PaymentService {


    @Autowired   // PaymentGateway ka object yaha inject hoga

    private PaymentGateway gateway;


    public void pay() {

        gateway.process();

    }

}


@Component

class PaymentGateway {

    public void process() {

        System.out.println("Payment processed!");

    }
```

}

👉 Matlab: `new` karne ki zarurat nahi, Spring khud inject karega.

---

9. @Qualifier

English:
Used with `@Autowired` when multiple beans of the same type exist, to specify which one to inject.

Hindi:
Jab ek type ke multiple beans hote hain to ye decide karta hai ki kaunsa inject karna hai.

```
@Service
public class NotificationService {

    @Autowired
    @Qualifier("emailNotification")   // specify bean name
    private Notification notification;

    public void send() {
        notification.send("Hello User!");
    }
}

@Component("emailNotification")
```

```
class EmailNotification implements Notification {
    public void send(String msg) {
        System.out.println("Email sent: " + msg);
    }
}

@Component("smsNotification")
class SmsNotification implements Notification {
    public void send(String msg) {
        System.out.println("SMS sent: " + msg);
    }
}
```

👉 Matlab: `@Qualifier` decide karta hai kaunsa bean inject karna hai.

---

10. @Controller

English:
Marks a class as a web controller that handles HTTP requests and returns views (for web applications).

Hindi:
Ek class ko **web controller** banata hai jo HTTP requests handle karta hai aur **view (JSP/HTML)** return karta hai.

```
@Controller
public class HomeController {
```

```
    @GetMapping("/home")
    public String homePage() {
        return "home"; // yaha "home.jsp" view return hoga
    }
}
```

👉 Matlab: Normal MVC applications me use hota hai jo **view** return karti hain.

---

11. @ResponseBody

English:
Indicates that the method return value will be written directly to the HTTP response body (JSON/text).

Hindi:
Method ka return value **direct HTTP response body me bhej diya jata hai** (JSON/text ke form me).

```
@Controller
public class MyController {

    @GetMapping("/hello")
    @ResponseBody   // view return nahi karega, seedha data bhej dega
    public String hello() {
        return "Hello World!";
    }
```

}

👉 Matlab: Direct data bhejna hai, page (view) render nahi karna.

---

12. @RestController

English:
Combination of `@Controller` + `@ResponseBody`. Used to create REST APIs that return JSON/XML.

Hindi:
`@Controller` aur `@ResponseBody` ka shortcut hai. REST API banane ke liye use hota hai jo JSON/XML return karti hai.

```
@RestController
public class MyRestController {

    @GetMapping("/api/hello")
    public String helloApi() {
        return "Hello from REST API!";
    }
}
```

👉 Matlab: REST API banate ho to `@RestController` use karo.

---

13. @RequestMapping

English:
Maps an HTTP request (GET/POST/PUT/DELETE) to a method/class.

Hindi:
Kisi HTTP request ko method ya class ke sath map karta hai.

```
@Controller
@RequestMapping("/users") // class level mapping
public class UserController {

    @RequestMapping("/all") // method level mapping
    @ResponseBody
    public String getAllUsers() {
        return "Returning all users!";
    }
}
```

👉 Matlab: Class aur method dono level pe request mapping karne ke liye.

---

14. @GetMapping

English:
Shortcut for `@RequestMapping(method = RequestMethod.GET)`. Used to fetch data.

Hindi:
`@RequestMapping(GET)` ka shortcut hai. Data fetch karne ke liye use hota hai.

```
@RestController
```

```
public class ProductController {

    @GetMapping("/products")
    public String getProducts() {
        return "All products list";
    }
}
```

👉 Matlab: Jab GET request se data chahiye ho.

---

15. @PostMapping

English:
Shortcut for `@RequestMapping(method = RequestMethod.POST)`. Used to create/add new data.

Hindi:
`@RequestMapping(POST)` ka shortcut hai. Naya data create/add karne ke liye use hota hai.

```
@RestController
public class ProductController {

    @PostMapping("/products")
    public String addProduct() {
        return "New product added!";
    }
```

}

👉 Matlab: Jab POST request se data bhejna ya create karna ho.

---

16. @PutMapping

English:
Shortcut for `@RequestMapping(method = RequestMethod.PUT)`. Used to update existing data.

Hindi:
`@RequestMapping(PUT)` ka shortcut hai. Existing data ko update karne ke liye use hota hai.

```
@RestController

public class ProductController {


    @PutMapping("/products/{id}")

    public String updateProduct(@PathVariable int id) {

        return "Product with ID " + id + " updated!";

    }

}
```

👉 Matlab: Jab data ko update karna ho.

---

17. @DeleteMapping

English:
Shortcut for `@RequestMapping(method = RequestMethod.DELETE)`. Used to delete existing data.

Hindi:
`@RequestMapping(DELETE)` ka shortcut hai. Data ko delete karne ke liye use hota hai.

```
@RestController

public class ProductController {


    @DeleteMapping("/products/{id}")

    public String deleteProduct(@PathVariable int id) {

        return "Product with ID " + id + " deleted!";

    }

}
```

👉 Matlab: Jab data ko delete karna ho.

---

18. @RequestBody

English:
Indicates that a method parameter should be bound to the body of the HTTP request (usually JSON).

Hindi:
Iska use tab hota hai jab HTTP request ke **body me bheja gaya JSON data** method ke parameter me bind karna ho.

```
@RestController

public class ProductController {
```

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

👉 Matlab: Request body ka data seedha object me convert ho jata hai.

---

19. @Service

English:
Marks a class as a **service layer component**. Used to write business logic.

Hindi:
Ek class ko **service layer (business logic)** ke liye mark karta hai.

```
@Service
public class ProductService {
    public String processProduct(String name) {
        return "Processing product: " + name;
    }
}
```

👉 Matlab: Business logic handle karne ke liye.

---

20. @Repository

English:
Marks a class as a **data access layer (DAO)** component. It interacts with the database.

Hindi:
Ek class ko **data access layer (database interaction)** ke liye mark karta hai.

```
@Repository

public interface ProductRepository extends JpaRepository<Product, Integer> {

}
```

👉 Matlab: Database operations ke liye.

---

21. @EnableAutoConfiguration

English:
Tells Spring Boot to automatically configure beans based on dependencies in the classpath.

Hindi:
Spring Boot ko bolta hai ki classpath me jo dependencies hain unke basis pe beans ka **automatic configuration** kare.

```
@SpringBootApplication

@EnableAutoConfiguration

public class MyApp {

    public static void main(String[] args) {

        SpringApplication.run(MyApp.class, args);
```

```
    }
}
```

👉 Matlab: Configurations manually likhne ki zarurat nahi, Spring Boot khud kar lega.

---

22. @EnableWebMvc

English:
Used to enable Spring MVC in a Spring Boot application. Allows full control over MVC configuration.

Hindi:
Spring Boot app me **Spring MVC enable** karta hai. Jyada customization karna ho to use hota hai.

```
@Configuration
@EnableWebMvc
public class WebConfig {
    // Custom MVC config
}
```

👉 Matlab: Agar default config ke alawa khud apna MVC setup karna ho.

---

23. @PropertySource

English:
Used to load external `.properties` or `.yml` files into Spring's Environment.

Hindi:
External **properties file** (config file) ko load karne ke liye use hota hai.

```
@Configuration
@PropertySource("classpath:application.properties")
public class MyConfig {
}
```

👉 Matlab: Alag-alag config file ko project me include karne ke liye.

---

24. @Value

English:
Used to inject values from properties files into fields.

Hindi:
Properties file ke value ko **direct variable me inject** karne ke liye use hota hai.

```
@RestController
public class MyController {

    @Value("${spring.application.name}")
    private String appName;

    @GetMapping("/app")
    public String getAppName() {
        return appName;
    }
```

```
}
```

👉 Matlab: Properties file ke values ko direct access karna.

---

25. @ConfigurationProperties

English:
Binds all properties with a given prefix into a POJO class.

Hindi:
Ek prefix ke saath multiple properties ko ek **POJO class me bind** karne ke liye use hota hai.

```
@Component

@ConfigurationProperties(prefix = "app")

public class AppProperties {

    private String name;

    private String version;

    // getters and setters

}
```

👉 Matlab: Ek sath multiple config values ko ek class me map karna.

---

26. @ConditionalOnProperty

English:
Enables a bean only if a specific property is set in `application.properties`.

Hindi:
Bean tabhi create hoga jab ek specific property `application.properties` file me set ho.

```
@Configuration
public class MyConfig {

    @Bean
    @ConditionalOnProperty(name = "feature.enabled", havingValue = "true")
    public MyBean myBean() {
        return new MyBean();
    }
}
```

👉 Matlab: Agar `feature.enabled=true` hai to bean banega, warna ignore.

---

27. @Entity

English:
Marks a Java class as a JPA entity (database table).

Hindi:
Java class ko ek **database table** banane ke liye use hota hai.

```
@Entity
public class User {
    @Id
    private Long id;
```

```
    private String name;
}
```

👉 Matlab: Ye class database me table banegi.

---

28. @Table

English:
Specifies the table name for the entity.

Hindi:
Entity ko kis naam se database me table create karna hai, wo define karta hai.

```
@Entity
@Table(name = "users")
public class User {
    @Id
    private Long id;
    private String name;
}
```

👉 Matlab: Class ka naam kuch bhi ho, table ka naam `users` hoga.

---

29. @Id

English:
Marks a field as the **primary key** of the entity.

Hindi:
Database table ka **primary key column** batata hai.

```
@Entity

public class User {

    @Id

    private Long id;

    private String name;

}
```

👉 Matlab: Har row ko uniquely identify karne ke liye.

---

30. @GeneratedValue

English:
Specifies that the primary key should be generated automatically.

Hindi:
Primary key ki value **auto-generate** karne ke liye use hota hai (jaise auto-increment).

```
@Entity

public class User {

    @Id

    @GeneratedValue

    private Long id;

    private String name;

}
```

👉 Matlab: Id manually dene ki zarurat nahi, DB khud generate karega.

---

31. @Column

English:
Used to specify details of a column in the database table.

Hindi:
Database table ke **column ka naam, size, nullable, unique** jaise details set karne ke liye use hota hai.

```
@Entity

public class User {

    @Id

    @GeneratedValue

    private Long id;


    @Column(name = "full_name", nullable = false, length = 100)

    private String name;

}
```

👉 Matlab: `name` field DB me `full_name` column banega, null allowed nahi hoga aur max 100 characters hoga.

---

32. @Transient

English:
Marks a field that should **not be stored in the database**.

Hindi:
Jo field database me save nahi karni hai, usko mark karne ke liye use hota hai.

```
@Entity

public class User {

    @Id

    private Long id;



    private String name;



    @Transient

    private int age; // This will not be stored in DB

}
```

👉 Matlab: `age` DB me save nahi hoga.

---

33. @EntityListeners

English:
Defines callback listeners for entity lifecycle events.

Hindi:
Entity ke **life-cycle events** (insert, update, delete) ke liye listener set karne ke liye use hota hai.

```
@Entity
```

```
@EntityListeners(AuditListener.class)
public class User {
    @Id
    private Long id;
    private String name;
}
```

👉 Matlab: Jab User entity save/update/remove hogi, `AuditListener` class ka method chalega.

---

34. @PrePersist

English:
Method runs **before an entity is inserted (saved)** into the database.

Hindi:
Entity ko database me **save karne se pehle** ye method chalega.

```
@Entity
public class User {
    @Id
    private Long id;
    private String name;

    @PrePersist
    public void beforeSave() {
        System.out.println("Before saving User!");
```

```
    }
}
```

👉 Matlab: DB me insert se pehle kuch logic chalana.

---

35. @PostPersist

English:
Method runs **after an entity is inserted (saved)** in the database.

Hindi:
Entity database me **save hone ke baad** ye method chalega.

```
@Entity
public class User {
    @Id
    private Long id;
    private String name;

    @PostPersist
    public void afterSave() {
        System.out.println("User saved successfully!");
    }
}
```

👉 Matlab: Save hone ke baad success message ya extra logic.

---

36. @PreUpdate

English:
Method runs **before an entity is updated** in the database.

Hindi:
Entity ko database me **update karne se pehle** ye method chalega.

```
@Entity

public class User {

    @Id

    private Long id;

    private String name;


    @PreUpdate

    public void beforeUpdate() {

        System.out.println("Before updating User!");

    }

}
```

👉 Matlab: Update ke time pehle kuch check ya modify karna.

---

37. @PostUpdate

English:
Runs **after an entity is updated** in the database.

Hindi:
Entity ko **update karne ke baad** ye method chalega.

```
@Entity
public class User {
    @Id
    private Long id;
    private String name;

    @PostUpdate
    public void afterUpdate() {
        System.out.println("User updated successfully!");
    }
}
```

👉 Matlab: Update hone ke baad message ya logic run karna.

---

38. @PreRemove

English:
Runs **before an entity is deleted** from the database.

Hindi:
Entity ko **delete karne se pehle** ye method chalega.

```
@Entity
public class User {
    @Id
```

```
    private Long id;


    @PreRemove
    public void beforeDelete() {
        System.out.println("Before deleting User!");
    }
}
```

👉 Matlab: Delete hone se pehle check karna ya log banana.

---

39. @PostRemove

English:
Runs **after an entity is deleted** from the database.

Hindi:
Entity ko **delete karne ke baad** ye method chalega.

```
@Entity
public class User {
    @Id
    private Long id;


    @PostRemove
    public void afterDelete() {
        System.out.println("User deleted successfully!");
```

```
    }
}
```

👉 Matlab: Delete hone ke baad action karna.

---

40. @OneToOne

English:
Defines a **one-to-one relationship** between two entities.

Hindi:
Do entities ke beech **1:1 relation** set karta hai.

```
@Entity
public class User {
    @Id
    private Long id;

    @OneToOne
    private Profile profile;
}
```

👉 Matlab: Ek `User` ka ek `Profile` hoga.

---

41. @OneToMany

English:
Defines a **one-to-many relationship**.

Hindi:
Ek entity ke paas **multiple related entities** ho sakti hain.

```
@Entity

public class User {

    @Id

    private Long id;



    @OneToMany

    private List<Order> orders;

}
```

👉 Matlab: Ek `User` ke multiple `Orders` ho sakte hain.

---

42. @ManyToMany

English:
Defines a **many-to-many relationship**.

Hindi:
Do entities ke beech **many-to-many relation** hota hai.

```
@Entity

public class Student {

    @Id

    private Long id;
```

```
    @ManyToMany
    private List<Course> courses;
}
```

👉 Matlab: Ek `Student` ke multiple `Courses` ho sakte hain aur ek `Course` ke multiple `Students`.

---

43. @Query

English:
Used to **write custom queries** in a repository method.
Hindi:
Custom SQL/JPQL queries likhne ke liye use hota hai.

```
public interface UserRepository extends JpaRepository<User, Long> {
    @Query("SELECT u FROM User u WHERE u.name = :name")
    User findByName(@Param("name") String name);
}
```

👉 Matlab: Repository ke andar apni query likh sakte ho.

---

44. @Param

English:
Used to **bind method parameters to query parameters**.

Hindi:
Method ke parameter ko query ke parameter se bind karta hai.

```
@Query("SELECT u FROM User u WHERE u.age = :age")

User findByAge(@Param("age") int age);
```

👉 Matlab: `age` method ka parameter query ke `:age` se link ho gaya.

---

45. @Transactional

English:
Marks a method or class to **run inside a database transaction** (all queries succeed or all fail).
Hindi:
Ye annotation ensure karta hai ki method ke andar saare DB operations **ya to ek saath ho jayein ya rollback ho jayein**.

```
@Service

public class UserService {

    @Transactional

    public void updateUser(User user) {

        // Multiple DB operations

    }

}
```

👉 Matlab: Ek hi transaction me multiple queries execute hongi.
