Spring Boot provides a wide range of annotations that simplify the development process by automating configuration, wiring dependencies, and enabling features. Here are 25 of the most important annotations in Spring Boot, along with detailed explanations:

1. @SpringBootApplication

- Description: This is a convenience annotation that combines `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`. It marks the main class of a Spring Boot application.
- Usage: Placed on the main application class.
- Example:

```
java
Copy code
@SpringBootApplication
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

2. @Component

- Description: Indicates that a class is a Spring-managed component. The Spring framework automatically detects classes annotated with `@Component` and registers them in the application context.
- Usage: Placed on classes that should be managed by Spring (e.g., services, DAOs).
- Example:

```
java
Copy code
@Component
public class MyService {
    // Service logic
}
```

3. @Service

- Description: A specialization of `@Component`, it indicates that a class performs service tasks. It's typically used on service layer classes.
- Usage: Placed on service classes.
- Example:

```
java
Copy code
```

```
@Service
public class UserService {
    // Business logic
}
```

4. @Repository

- Description: A specialization of `@Component`, it indicates that a class provides the mechanism for storage, retrieval, update, and delete operations on objects. It's typically used on the persistence layer.
- Usage: Placed on DAO (Data Access Object) classes.
- Example:

```
java
Copy code
@Repository
public class UserRepository {
    // Data access logic
}
```

5. @Controller

- Description: A specialization of `@Component`, it indicates that a class is a web controller in a Spring MVC application.
- Usage: Placed on controller classes that handle web requests.
- Example:

```
java
Copy code
@Controller
public class UserController {
    @GetMapping("/users")
    public String getUsers(Model model) {
        // Handle the request
        return "users";
    }
}
```

6. @RestController

- Description: A combination of `@Controller` and `@ResponseBody`. It indicates that a class serves RESTful web services and automatically serializes responses to JSON or XML.
- Usage: Placed on REST controller classes.
- Example:

```
java
Copy code
@RestController
public class UserRestController {
    @GetMapping("/api/users")
    public List<User> getUsers() {
```

```
        // Return a list of users
        return userService.getAllUsers();
    }
}
```

7. @RequestMapping

- Description: Maps HTTP requests to handler methods of MVC and REST controllers. It can be used at both the class and method levels.
- Usage: Placed on classes or methods to define request paths.
- Example:

```
java
Copy code
@RequestMapping("/users")
public class UserController {
    @RequestMapping("/list")
    public String listUsers() {
        return "userList";
    }
}
```

8. @GetMapping

- Description: A specialized version of `@RequestMapping` for handling HTTP GET requests.
- Usage: Placed on methods to handle GET requests.
- Example:

```
java
Copy code
@GetMapping("/users/{id}")
public User getUser(@PathVariable Long id) {
    return userService.getUserById(id);
}
```

9. @PostMapping

- Description: A specialized version of `@RequestMapping` for handling HTTP POST requests.
- Usage: Placed on methods to handle POST requests.
- Example:

```
java
Copy code
@PostMapping("/users")
public User createUser(@RequestBody User user) {
    return userService.saveUser(user);
}
```

10. @PutMapping

- Description: A specialized version of `@RequestMapping` for handling HTTP PUT requests.
- Usage: Placed on methods to handle PUT requests.
- Example:

```
java
Copy code
@PutMapping("/users/{id}")
public User updateUser(@PathVariable Long id, @RequestBody User user) {
    return userService.updateUser(id, user);
}
```

11. @DeleteMapping

- Description: A specialized version of `@RequestMapping` for handling HTTP DELETE requests.
- Usage: Placed on methods to handle DELETE requests.
- Example:

```
java
Copy code
@DeleteMapping("/users/{id}")
public void deleteUser(@PathVariable Long id) {
    userService.deleteUser(id);
}
```

12. @RequestParam

- Description: Binds a web request parameter to a method parameter in a controller.
- Usage: Placed on method parameters to extract query parameters.
- Example:

```
java
Copy code
@GetMapping("/search")
public String searchUsers(@RequestParam String query) {
    return userService.searchUsers(query);
}
```

13. @PathVariable

- Description: Binds a URI template variable to a method parameter in a controller.
- Usage: Placed on method parameters to extract path variables.
- Example:

```
java
Copy code
@GetMapping("/users/{id}")
```

```
public User getUser(@PathVariable Long id) {
    return userService.getUserById(id);
}
```

14. @RequestBody

- Description: Binds the body of a web request to a method parameter.
- Usage: Placed on method parameters to deserialize the body of an HTTP request.
- Example:

```
java
Copy code
@PostMapping("/users")
public User createUser(@RequestBody User user) {
    return userService.saveUser(user);
}
```

15. @ResponseBody

- Description: Indicates that a method's return value should be bound to the web response body. Used mainly in REST controllers.
- Usage: Placed on methods in controller classes.
- Example:

```
java
Copy code
@GetMapping("/users")
@ResponseBody
public List<User> getUsers() {
    return userService.getAllUsers();
}
```

16. @Autowired

- Description: Marks a constructor, field, setter method, or config method as to be autowired by Spring’s dependency injection facilities.
- Usage: Placed on dependencies that should be injected by Spring.
- Example:

```
java
Copy code
@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;
}
```

17. @Qualifier

- Description: Used along with `@Autowired` to specify which bean should be injected when there are multiple beans of the same type.
- Usage: Placed on constructor parameters, fields, or setter methods.
- Example:

```
java
Copy code
@Autowired
@Qualifier("specificBean")
private UserService userService;
```

18. @Configuration

- Description: Indicates that a class declares one or more `@Bean` methods and may be processed by the Spring container to generate bean definitions.
- Usage: Placed on configuration classes.
- Example:

```
java
Copy code
@Configuration
public class AppConfig {
    @Bean
    public UserService userService() {
        return new UserServiceImpl();
    }
}
```

19. @Bean

- Description: Indicates that a method produces a bean to be managed by the Spring container.
- Usage: Placed on methods in configuration classes.
- Example:

```
java
Copy code
@Bean
public UserService userService() {
    return new UserServiceImpl();
}
```

20. @EnableAutoConfiguration

- Description: Enables Spring Boot's auto-configuration feature, which automatically configures your application based on the dependencies on the classpath.
- Usage: Placed on the main application class.
- Example:

```
java
```

```
Copy code
@SpringBootApplication
@EnableAutoConfiguration
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

21. @ComponentScan

- Description: Configures component scanning, which allows Spring to automatically detect and register beans in the application context.
- Usage: Placed on configuration classes or the main application class.
- Example:

```
java
Copy code
@SpringBootApplication
@ComponentScan(basePackages = "com.example")
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

22. @Profile

- Description: Indicates that a component is eligible for registration

4o