`@Configuration`

- This annotation is used to **tell Spring** that the class contains **bean definitions**.
- It **replaces the XML configuration files** used in older Spring versions.
- When Spring sees a class annotated with `@Configuration`, it knows to **look inside** the class for methods that define beans using `@Bean`.
- It's part of the **Java-based configuration** approach in Spring.
- "This class contains instructions for how to set up and configure objects (beans) that Spring should manage."

`@ComponentScan`

- This tells Spring **where to look** in the project for **components to register as beans**.
- It **scans the specified package(s)** and automatically finds classes annotated with `@Component`, `@Service`, `@Repository`, or `@Controller`.
- This scanning helps Spring **automatically discover and create objects** without manual registration.
- Look in these folders and automatically find any components you need to manage

`@Component`

- Marks a class as a **Spring-managed bean**.
- Any class annotated with `@Component` will be **automatically detected and instantiated** by Spring during the component scan.
- It is a **generic stereotype annotation** used for any Spring-managed class (logic, helper, etc.).

`@Autowired`

- This tells Spring to **automatically inject (provide) dependencies** into a class.
- You don't need to manually write code to create objects; Spring will **figure out what to inject and where**.
- It works on fields, constructors, and setter methods.
- Spring will search for the correct bean and **inject it where needed**.

`@Value`

- Used to **inject values into variables**, especially values from `application.properties` or environment variables.
- You can use it to inject **simple values like strings, numbers, or expressions**.
- Commonly used to **configure values** like app name, URLs, or limits that may vary between environments.

`@Bean`

- Used inside a class marked with `@Configuration`.
- It **manually defines a bean**, which means you're telling Spring how to create a particular object.
- You use `@Bean` when you need **more control** over object creation, or when the object is from a **third-party library** that doesn't use `@Component`

`@Repository`

- A **specialized type of `@Component`**, meant to indicate that the class is responsible for **database access or operations**.
- It is used in the **data access layer** of your application.
- Also provides **exception translation**, converting database-specific exceptions into **Spring's unified exception hierarchy**.

`@Qualifier`

- When multiple beans of the same type exist, Spring gets **confused** about which one to inject.
- `@Qualifier` helps Spring choose the **exact bean** by **name**.
- It works **along with `@Autowired`** to clarify which bean you want.

`@Primary`

- This annotation is also used when **multiple beans of the same type** are present.
- It tells Spring:

  "If no `@Qualifier` is given, use **this one** by default."

- It's a way to **set a default preference** when multiple options are available.

`@Scope`

- By default, Spring beans are **singleton**, meaning only **one instance** is created and reused.
- `@Scope` allows you to change the **lifecycle and visibility** of a bean.

Scope	Meaning
`singleton`	One shared instance (default)
`prototype`	A new instance every time it's needed
`request`	One per HTTP request (web apps)
`session`	One per HTTP session (web apps)

`@PostConstruct`

- This marks a **method that should run automatically after the bean is created and dependencies are injected**.
- It's often used for **initialization logic**, like opening a file, database connection, or preloading data.

`@PreDestroy`

- This marks a method that runs **before the bean is destroyed**.
- It's used to **release resources** like closing connections, saving data, etc.
- Works for **singleton beans**, since Spring manages their full lifecycle.