IOC&DI

Inversion of Control (IoC): design principle where the control over certain parts of the code is inverted, meaning that instead of the code controlling everything, the control is handed over to an external framework or library. This principle allows for greater flexibility and decoupling in software design.

بدل ما كلاس واحد او جزء واحد من الكود مسؤول عن كل حاجة , بيخلي الكود اقل اعتمادية حيث في جزء خارجي هو اللي بيتحكم في أجزاء معينة من الكود (بستقبل ميثود معينة بدل ما أنشئها ) بيكون عن طريق استخدام

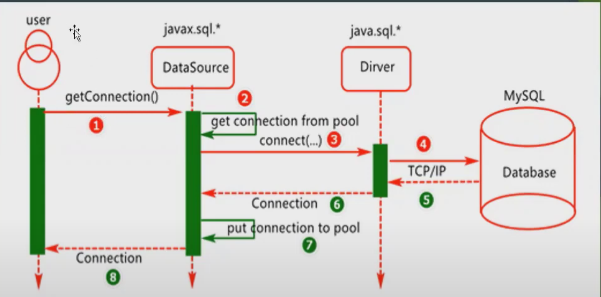
Strategy Pattern /DI (most popular) / Service Locator Pattern/ Template Method.

**Dependency Injection (DI):** a specific technique to implement IoC. DI involves injecting dependencies into a class from the outside, rather than the class creating these dependencies itself. This makes the code easier to maintain, test, and extend.

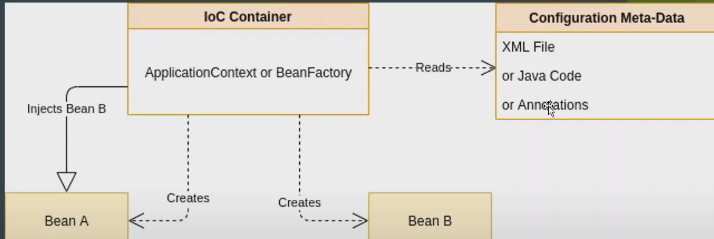
* Constructor injection: Dependencies are provided through a class constructor.
* Setter Injection: Dependencies are set using setter methods.
* Field Injection: Dependencies are directly injected into class fields, often using annotations Spring.
* **Spring JDBC**: framework that simplifies working with databases using JDBC. includes:
  + **DataSource**: Manages connections to the database.
  + **JdbcTemplate**: Provides an easier interface for executing queries.
  + **NamedParameterJdbcTemplate**: Supports named parameters in queries.

C**onfiguration in “application.properties”:**

* Contains settings for database connection such as URL, username, and password.
* **Connection pooling: mechanism** can be enabled to improve performance by reusing connections, allocating new connections, managing available connections, and closing connections.



* Configuration types:
* XML
* Java based
* Annotations



Annotations: metadata provides data (supplemental information) about the program

1. @Autowired: dependency injection 🡪used on setter methods, variables, and constructors. Spring manages & creates the necessary objects.
2. @component: class-level annotation indicate a class is a Spring class(as a Bean)) to be discovered & managed by Spring during the classpath.
3. @componentScan: specify the packages that Spring should scan for components to be managed as Beans.
4. @Configuration: indicate that a class contains one or more definitions to be managed by Spring Controller.
5. @Bean: used inside a class with @configration to indicate. that a method returns an object that should be managed as a Bean by spring.
6. @Service: define a class as a service managed by Spring.
7. @Repository: define a class as a Repository managed by Spring.