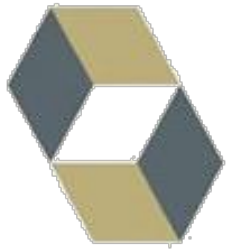


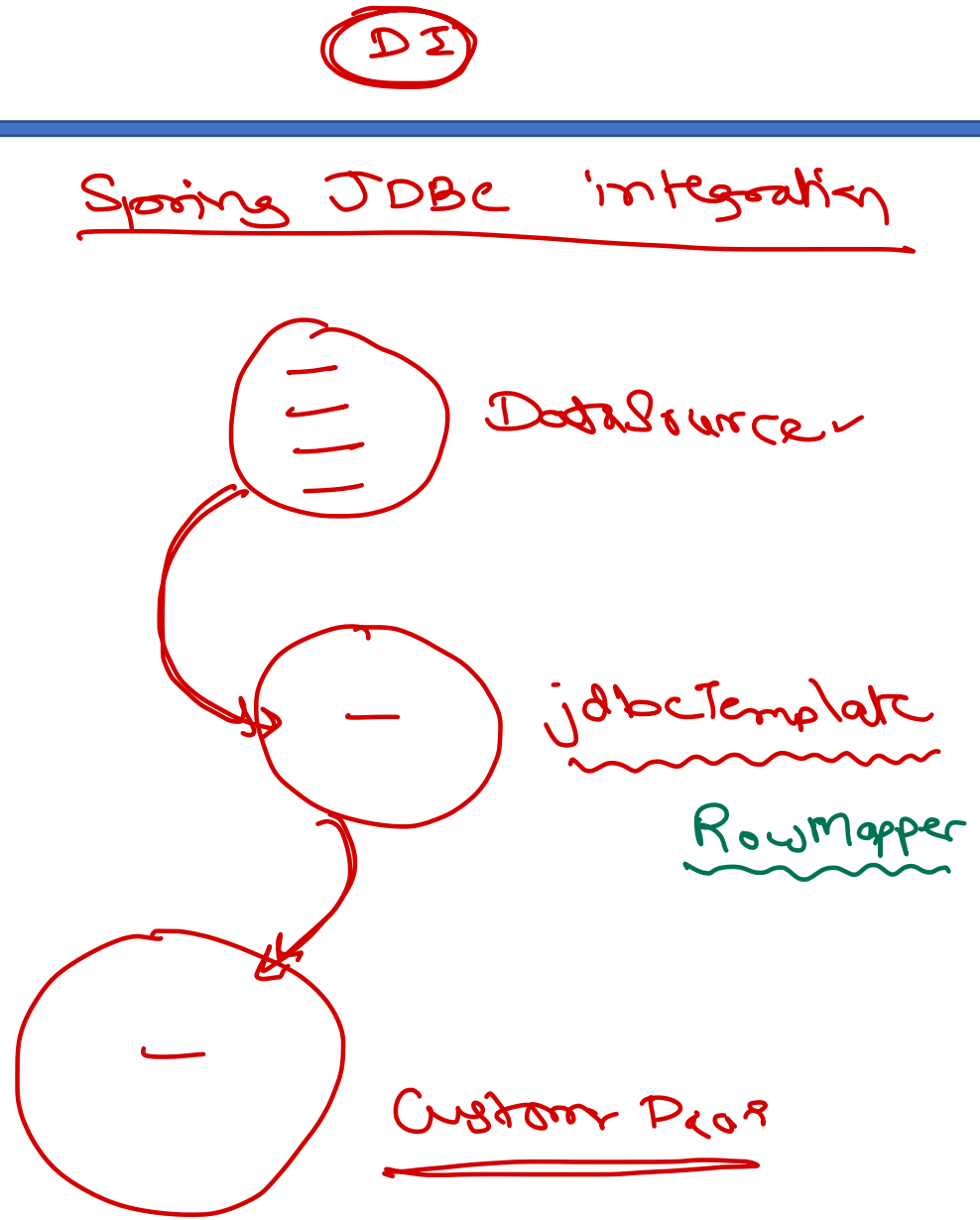
HIBERNATE

Trainer: Nilesh Ghule

Wake up from Hibernate, Spring up!!!

Agenda

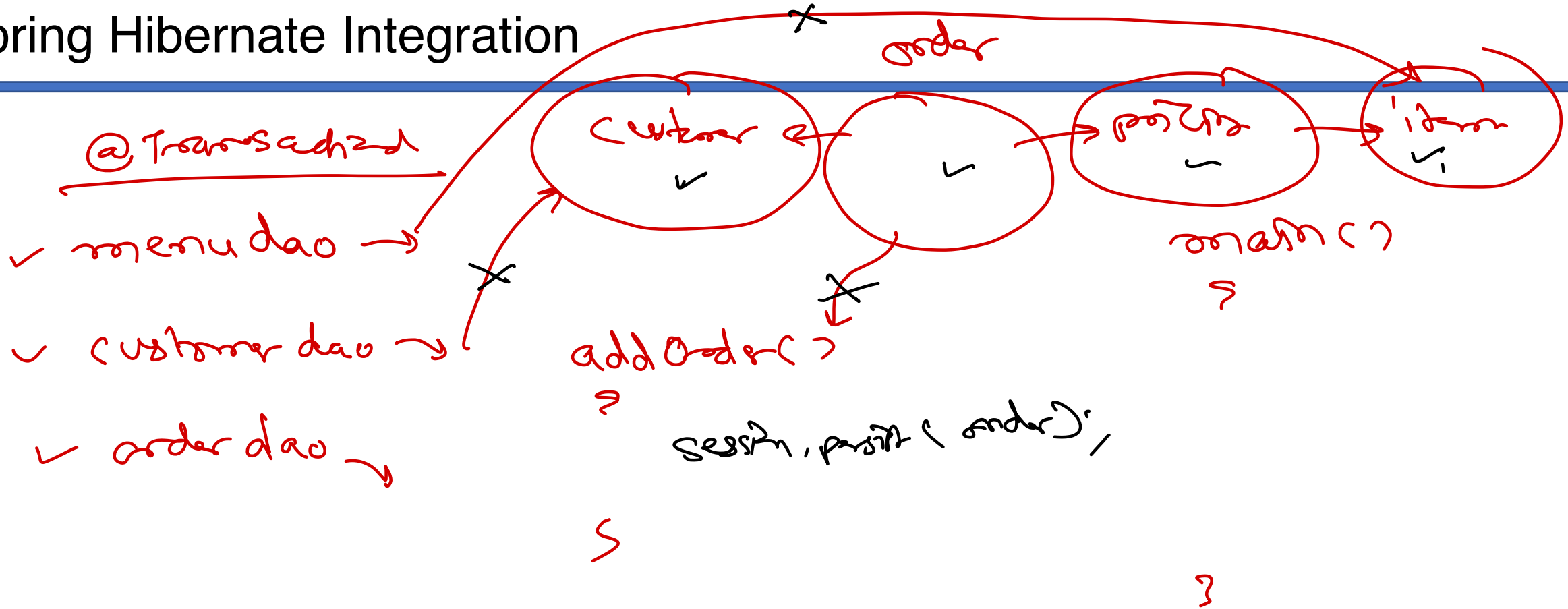
- Spring Hibernate integration
- Implementing @Service layer
- @Transactional annotation
- Spring JPA integration
- Spring Web MVC architecture
- @Controller and Request handler methods
- Using Spring tags in JSP pages



Spring Hibernate Integration

- Spring DI simplifies Hibernate ORM.
- LocalSessionFactory bean provides session factory, while transaction automation is done by Hibernate5TransactionManager bean.
- Steps:
  - In pom.xml, add spring-orm, mysql-connector-java and hibernate-core.
  - Create dataSource, sessionFactory (with hibernate config), transactionManager beans. Also set default transactionManager.
  - Implement entity classes. Ensure that spring session factory config scan them.
  - Implement @Repository class and auto-wire session factory. Use factory.getCurrentSession() to obtain hibernate session and perform operations.
  - Implement @Service layer and mark business logic methods as @Transactional.
    - Note that single business operation (from service layer) may deal with multiple operations on different repositories. @Transactional put all ops under same tx.

Spring Hibernate Integration



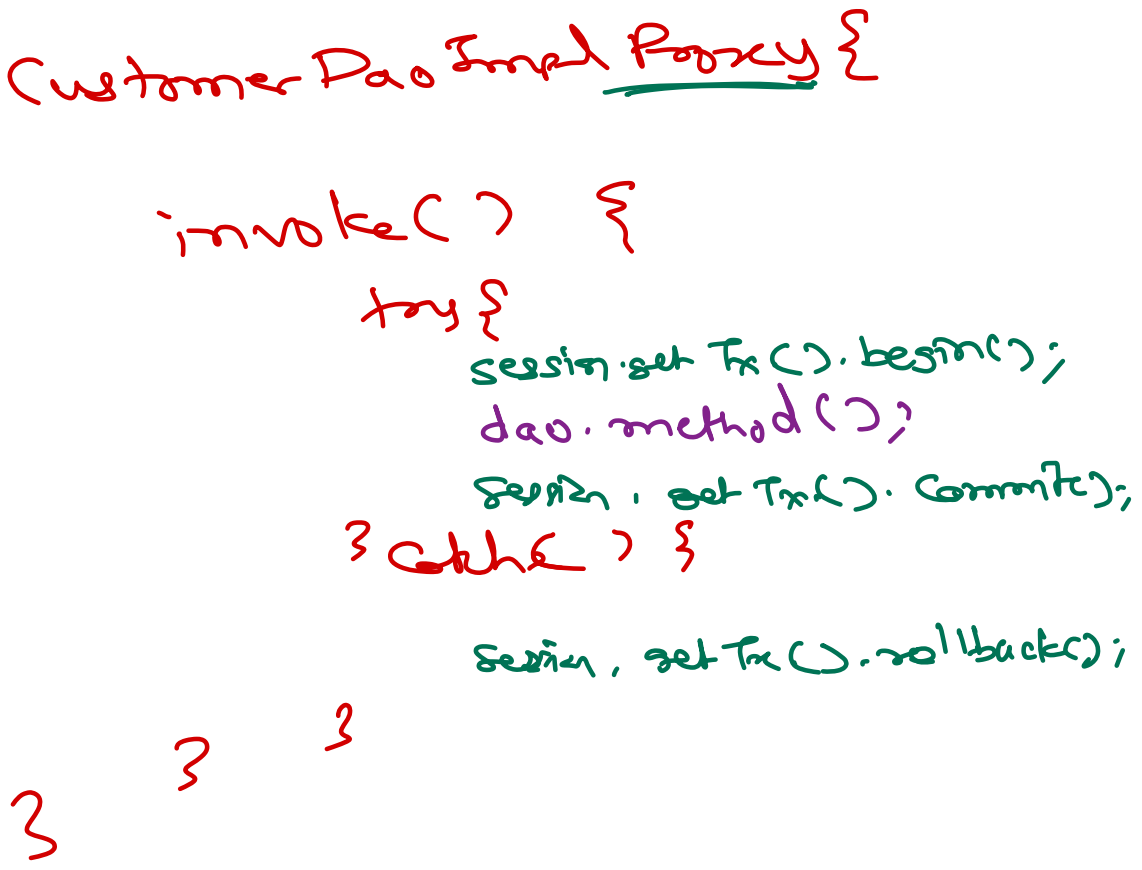
@Service layer

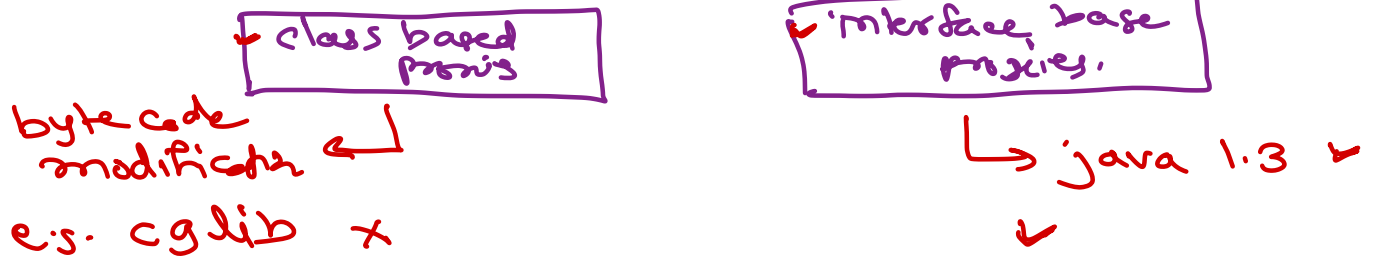
- @Repository layer contains database operations (i.e. CRUD operations, …).
- @Service layer contains business logic. It is implemented as per business operations.
- One @Service component may have one or more DAO component dependencies.
- It is common practice to handle transactions in service layer.

→ ensures that all db ops involved in one buslines op carried out under single transaction.

@Transactional

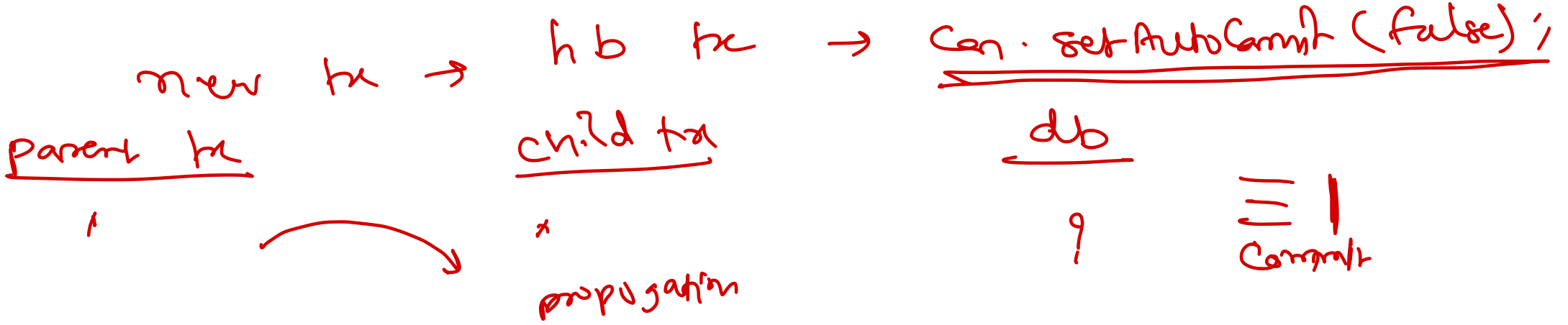
- @Transactional is declarative transaction management of Spring.
- It can be used method level or class level. If used on class level, it applies to all methods in the class.
- Spring internally use JDBC transaction in AOP fashion.
  - start transaction (before method).
  - commit transaction (if method is successful).
  - rollback transaction (if method throw exception).
- Transaction management is done by platform transaction manager e.g. datasource, hibernate or jpa transaction manager.




class based proxies
bytecode modification
e.g. cglib
interface base proxies.
java 1.3

@Transactional

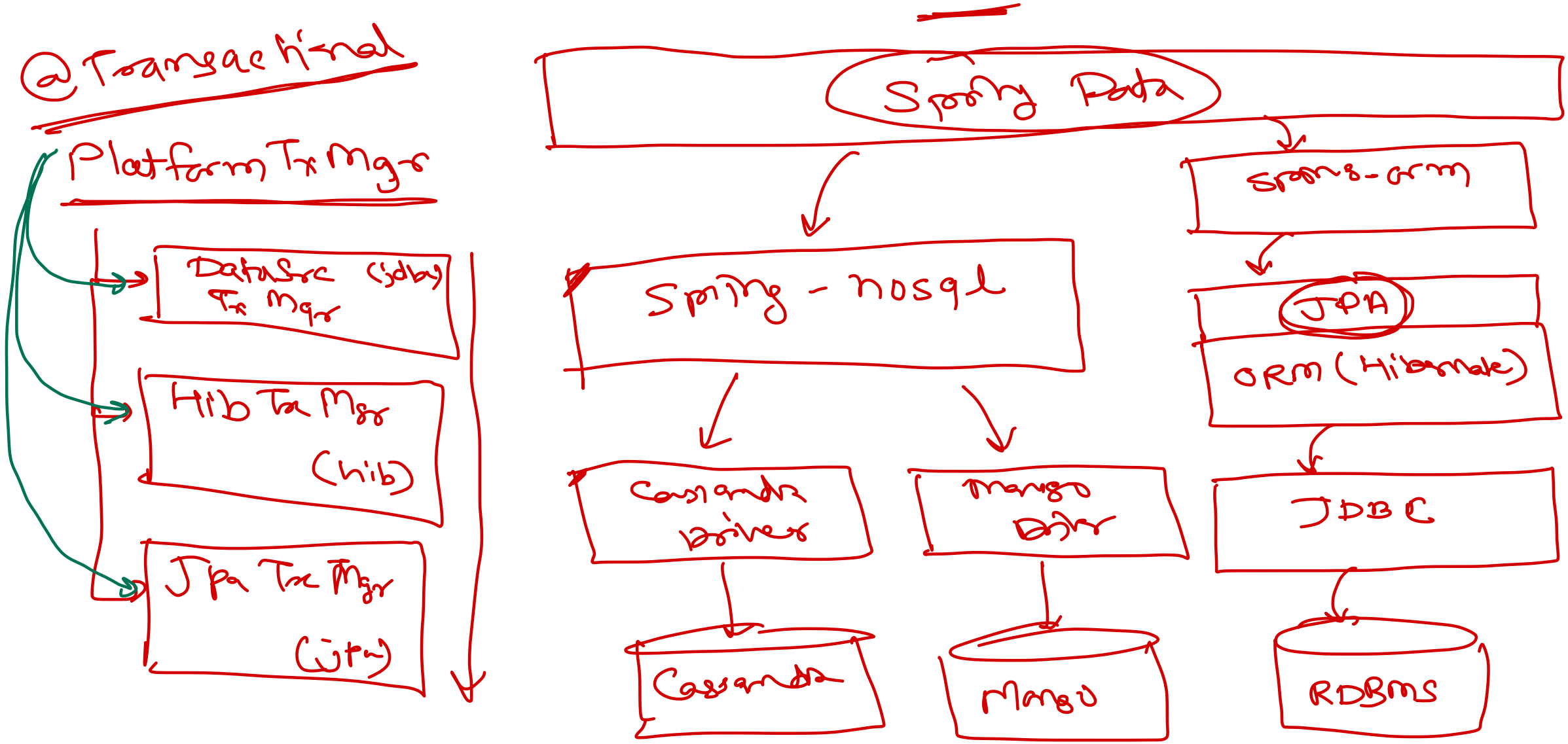
- If one transactional method invokes another transactional method, transaction behaviour is defined by propagation attribute.
  - REQUIRED: Support a current transaction, create a new one if none exists.
  - REQUIRES_NEW: Create a new transaction, and suspend the current transaction if one exists.
  - SUPPORTS: Support a current transaction, execute non-transactionally if none exists.
  - MANDATORY: Support a current transaction, throw an exception if none exists.
  - NEVER: Execute non-transactionally, throw an exception if a transaction exists.
  - NOT_SUPPORTED: Execute non-transactionally, suspend the current transaction if one exists.
  - NESTED: Execute within a nested transaction (save points) if a current transaction exists, behave like REQUIRED otherwise.



Spring JPA Integration

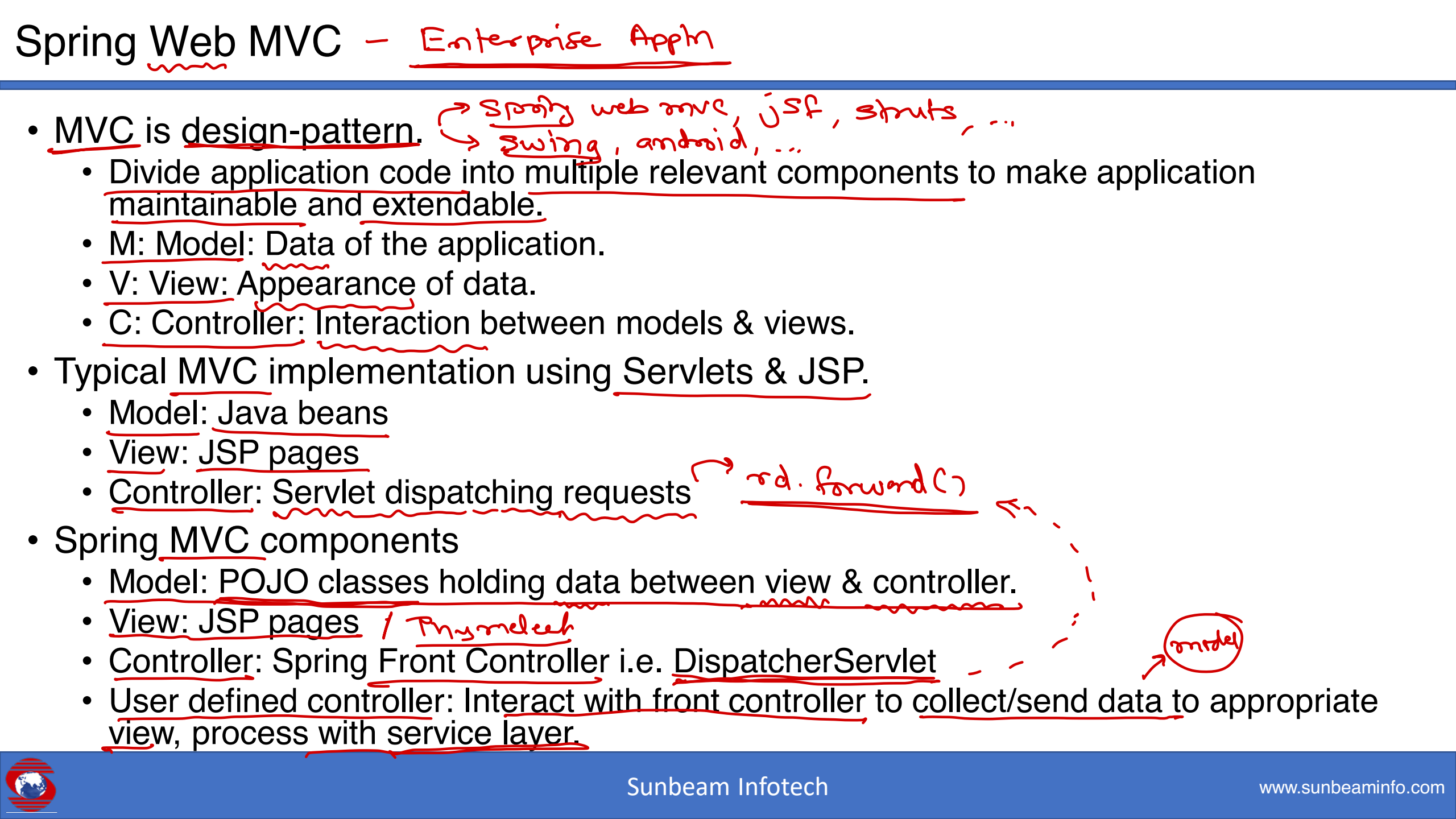
- Spring DI simplifies JPA.
- LocalEntityManagerFactoryBean bean provides entity manager factory, while transaction automation is done by JpaTransactionManager bean.
- Steps:
  - In pom.xml, add spring-orm, mysql-connector-java, hibernate-core.
  - Configure META-INF/persistence.xml.
  - Create dataSource, entityManagerFactory (with JPA PersistenceUnitName configured), transactionManager beans. Also set default transactionManager.
  - Implement entity classes. Ensure that spring session factory config scan them.
  - Implement @Repository class and auto-wire session factory. Use factory.getCurrentSession() to obtain hibernate session and perform operations.
  - Implement @Service layer and mark business logic methods as @Transactional.
    - Note that single business operation (from service layer) may deal with multiple operations on different repositories. @Transactional put all ops under same tx.

Spring JPA Integration

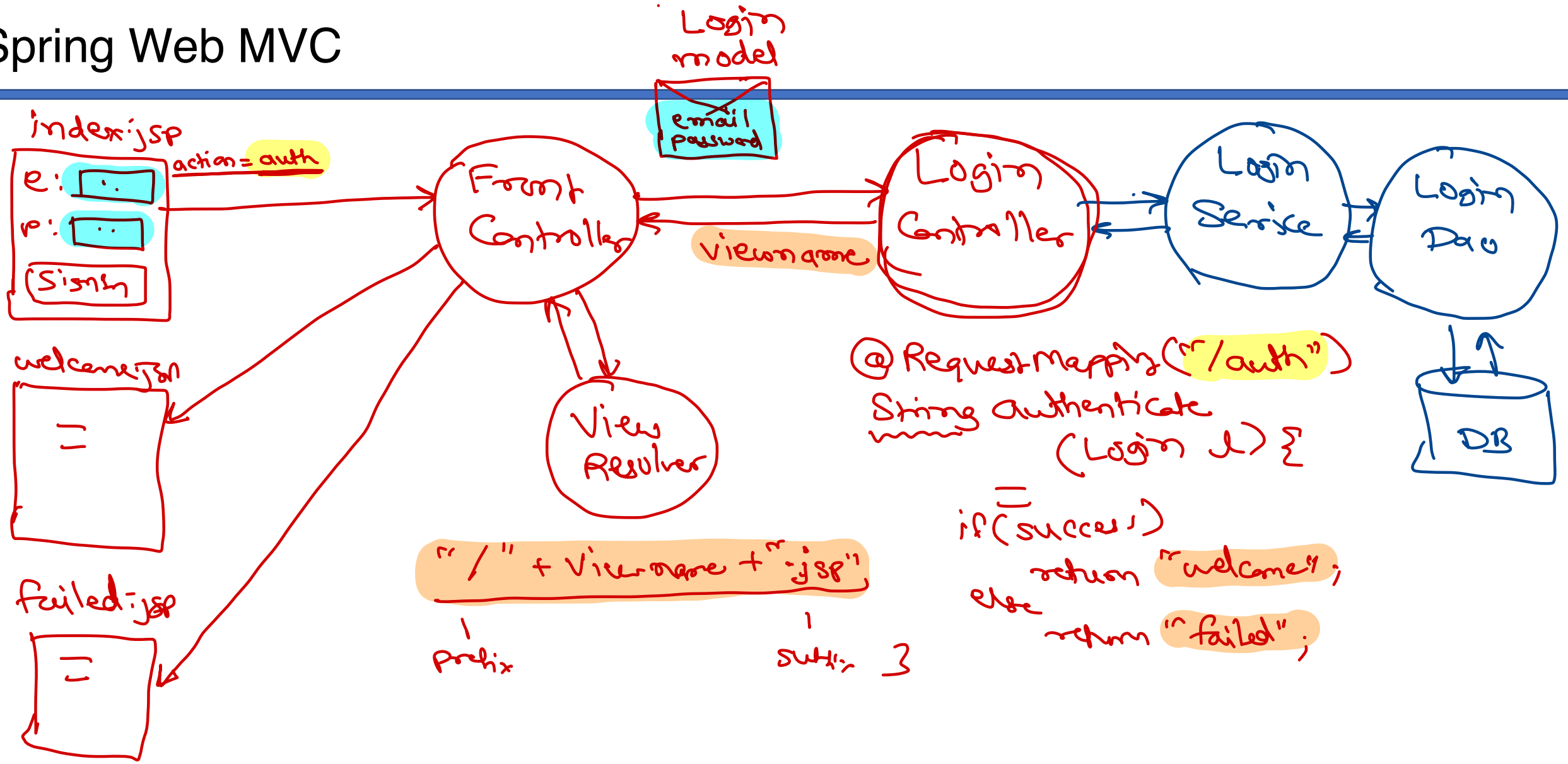


Spring Web MVC – Enterprise Appln

- MVC is design-pattern.
  - Divide application code into multiple relevant components to make application maintainable and extendable.
  - M: Model: Data of the application.
  - V: View: Appearance of data.
  - C: Controller: Interaction between models & views.
- Typical MVC implementation using Servlets & JSP.
  - Model: Java beans
  - View: JSP pages
  - Controller: Servlet dispatching requests
- Spring MVC components
  - Model: POJO classes holding data between view & controller.
  - View: JSP pages / Thymeleaf
  - Controller: Spring Front Controller i.e. DispatcherServlet
  - User defined controller: Interact with front controller to collect/send data to appropriate view, process with service layer.



Spring Web MVC



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>