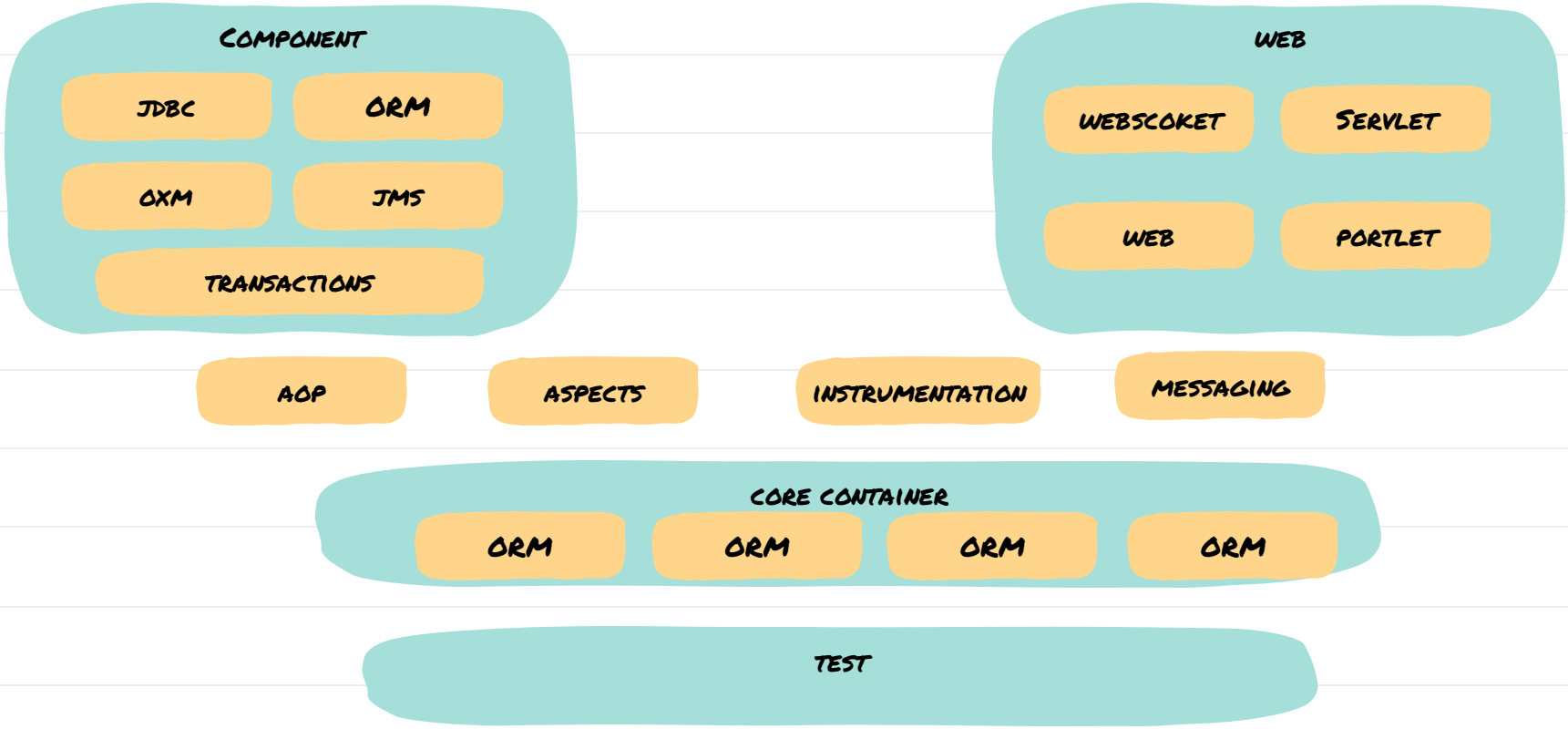
Spring overview

Teaching Faculty: Umur INAN

Prepared by Umur INAN

Spring framework



Main Features

- **Core Technologies:** dependency injection, events, resources, i18n, validation, data binding, type conversion, SpEL, AOP
- **Testing:** mock objects, TestContext framework, Spring MVC Test
- **Data Access:** transactions, DAO support, JDBC, ORM, Marshalling XML.
- **Integration:** remoting, JMS, JCA, JMX, email, tasks, scheduling, cache

Inversion of Control

- Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a **container** or framework.

- **Inversion of Control** (IoC or IOC) describes a system that follows the Hollywood Principle.

- **Hollywood Principle** states, "Don't Call Us, We'll Call You."

Inversion of Control

- Promotes loose coupling between classes and subsystems.
- Adds potential flexibility to a codebase for future changes.
- Classes are easier to unit test in isolation.
- Enable better code reuse.

Inversion of Control

Objects do not create other objects that they depend on.

IoC is implemented using **Dependency Injection**(DI).

Dependency injection

- Dependency injection is a pattern that is used to implement IoC, where the control being inverted is setting an object's dependencies.

Dependency injection

- DI exists in three major variants
- Dependencies defined through
  - Property-based dependency injection.
  - Setter-based dependency injection.
  - Constructor-based dependency injection

Container injects dependencies when it creates the bean.

Dependency injection examples

- Property based[by Type]

```
@Autowired
ProductService productService;
```

Dependency injection examples

- Setter based [by Name]

```
ProductService productService;
  public void setProductService(ProductService productService){
     this.productService = productService;
```

Dependency injection examples

- Constructor based:

```
ProductService productService;

@Autowired
public ProductController(ProductService productService) {
  this.productService = productService;
}
```

Spring framework

- Infrastructure support for developing Java applications.
- Configure disparate components into a fully working application ready for use.
- Build applications from “plain old Java objects” (POJOs)
- Non-intrusive - domain logic has little or no dependencies on framework.
- Lightweight application model is that of a layered [N-tier] architecture.

JavaBean vs POJO vs Spring Bean

- JavaBean
  - Adhere to Sun's JavaBeans specification.
  - Implements Serializable interface.
  - Must have default constructor, setters & getters.
  - Reusable Java classes for visual application composition.

JavaBean vs POJO vs Spring Bean

- POJO
  - ‘Fancy’ way to describe ordinary Java Objects
  - Doesn’t require a framework
  - Doesn’t require an application server environment
  - Simpler, lightweight compared to ‘heavyweight’ EJBs

JavaBean vs POJO vs Spring Bean

- Spring Bean
  - Spring managed - configured, instantiated and injected.

Java object can be a JavaBean, a POJO and a Spring bean all at the same time.

Bean

- The objects that form the backbone of your application and that are managed by the Spring IoC container are called beans.
- A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.
- A bean is simply one of many objects in your application

@Configuration

- It is a class-level annotation indicating that an object is a source of bean definitions.

- @Configuration classes declare beans via public @Bean annotated methods.

Create an Object of AClass

```
public class AClass {
 private BClass b;
 public AClass(BClass b)   {
    this.b= b;
  }
}
```

```
public class BClass {
    private CClass c;
    public BClass(CClass
     c){
     this.c=c;
    }
}
```

```
public class CClass {
    private DClass d;

    public CClass(DClass d){
      this.d= d;
    }
}
```

```
public class DClass {
    private EClass e;
    public DClass (EClass e){
      this.e=e;
    }
}
```

```
public class EClass {

}
```

Better Method?

```
public class Main {
    EClass e = new EClass();
    DClass d = new DClass(e);
    CClass c = new CClass(d);
    BClass b = new BClass(c);

    AClass a = new AClass(b);
}
```

Dependency injection

- Ask to the container to give an object of the class by injecting all dependencies.

```
@Autowired
    private AClass a;
```

How does container find dependencies ???

We need to put our dependencies (classes) to the container.

The container should be aware of our classes !!!

@Component

- is an annotation that allows Spring to automatically detect our custom beans.

```
@Component
public class BClass
```

The container is **aware** of BClass and object of it can be asked.

Advantages

- Decoupling the execution of a task from its implementation.
- Making it easier to switch between different implementations.
- Greater modularity of a program.
- Greater ease in testing a program by isolating a component or mocking its dependencies, and allowing components to communicate through contracts.

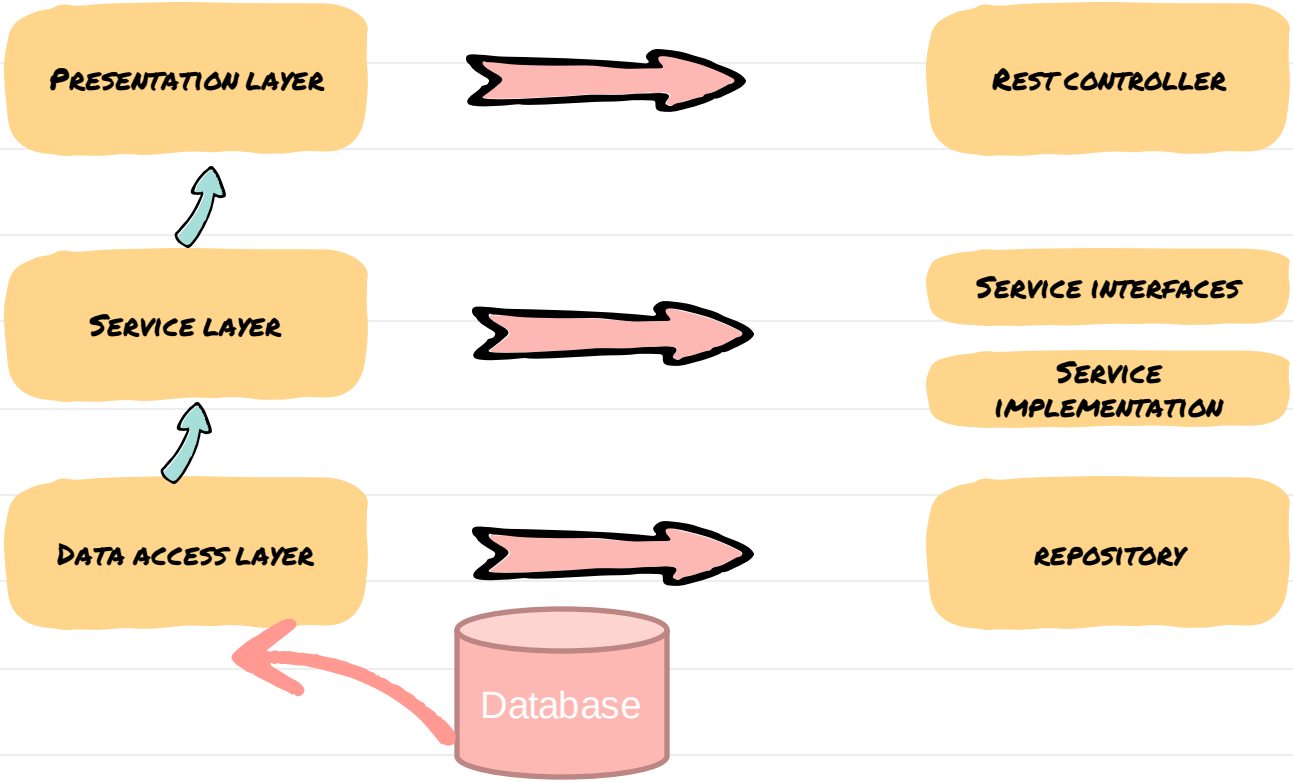
@Bean

- annotation indicates that the annotated method produces a bean to be managed by the Spring container.

- Mostly used for 3rd party dependencies.

Configuration classes can contain bean definition methods annotated with @Bean

N-TIER ARCHITECTURE



N-TIER ARCHITECTURE

- It divides an application into logical layers and physical tiers.
- Layers are a way to separate responsibilities and manage dependencies.
- Each layer has a **specific** responsibility.
- A higher layer can use services in a lower layer, but not the other way around

N-Tier Architecture Advantages

- Separation of concerns
- Design to interfaces
- ???

Stereotype annotations

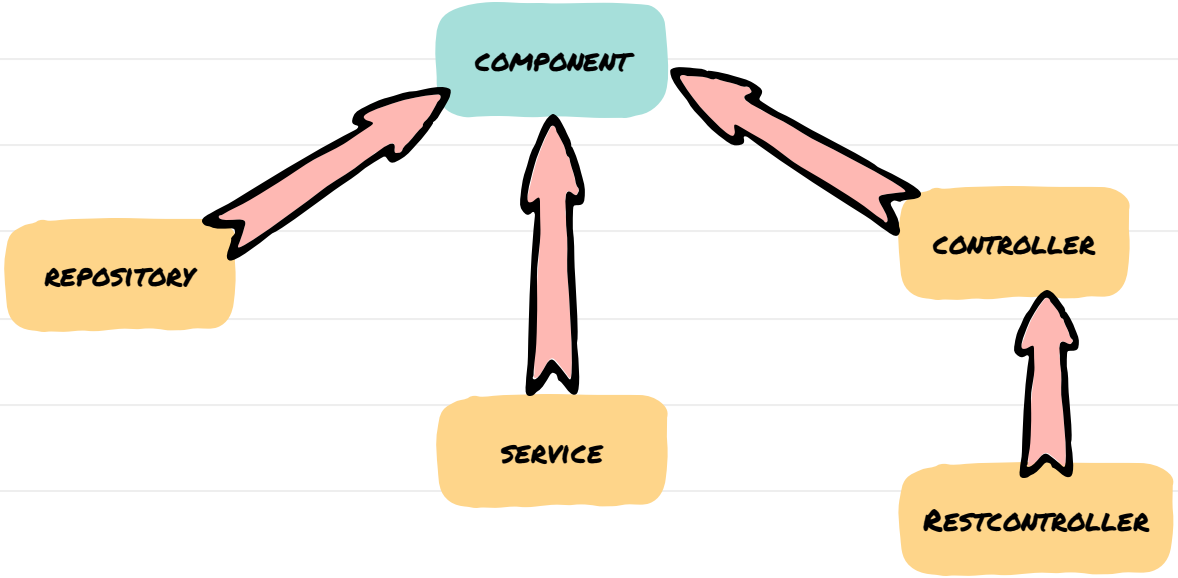
- There are some Stereotype meta-annotations which is derived from **@Component** those are:

- **@Controller**: Which is used to create Spring beans at the controller layer.

- **@Service**: Used to create Spring beans at the Service layer.

- **@Repository**: Which is used to create Spring beans for the repositories at the DAO layer.

Stereotype annotations



@Repository

- This annotation is used on Java classes that directly access the database.
- Its job is to catch persistence specific exceptions and rethrow them as one of Spring's unified unchecked exception.

@Service

- It is a stereotype for the service layer.
- The @Service marks a Java class that performs some service, such as execute business logic, perform calculations and call external APIs

@controller

- is used to indicate the class is a Spring controller.

@RestController

- is a specialized version of the controller.

- It includes the @Controller and @ResponseBody annotations, and as a result, simplifies the controller implementation

Main points

- Frameworks make development easier and more effective by providing a secure and reliable foundation on which to build upon.
- The simplest form of awareness, Transcendental Consciousness, provides a strong foundation for a rewarding and successful life.
- An N Tier Architecture separates an application into layers thereby supporting a separation of concerns making any application more efficient, modular and scalable.
- Life is structured in layers. It is a structure that is both stable and flexible, consistent yet variable and it encompasses an infinite range of possibilities.