**Design Patterns Used in Java Spring Framework**

Dependency injection/ or IoC (inversion of control) – Is the main principle behind decoupling process that Spring does

Factory – Spring uses factory pattern to create objects of beans using Application Context reference

// Spring uses factory pattern to create instances of the objects  
BeanFactory factory = new XmlBeanFactory(new FileSystemResource("spring.xml"));  
Triangle triangle = (Triangle) factory.getBean("triangle");  
triangle.draw();

Proxy – used heavily in AOP, and remoting.

Singleton – by default, beans defined in spring config file (xml) are only created once. No matter how many calls were made using getBean() method, it will always have only one bean. This is because, by default all beans in spring are singletons.  
This can be overridden by using Prototype bean scope.Then spring will create a new bean object for every request.

Model View Controller – The advantage with Spring MVC is that your controllers are POJOs as opposed to being servlets. This makes for easier testing of controllers. One thing to note is that the controller is only required to return a logical view name, and the view selection is left to a separate ViewResolver. This makes it easier to reuse controllers for different view technologies.

Front Controller – Spring provides DispatcherServlet to ensure an incoming request gets dispatched to your controllers.

View Helper – Spring has a number of custom JSP tags, and velocity macros, to assist in separating code from presentation in views.

Template method – used extensively to deal with boilerplate repeated code (such as closing connections cleanly, etc..). For example JdbcTemplate, JmsTemplate, JpaTemplate.

**What is Aspect, Advice, Pointcut, JointPoint in AOP?**

Aspect: Aspect is a class that implements cross-cutting concerns, such as logger, encryption.

Aspects can be a normal class configured and then configured in Spring Bean configuration file or we can use Spring AspectJ support to declare a class as Aspect using @Aspect annotation.

Advice: Advice is the action taken for a particular join point. In terms of programming, they are methods that gets executed when a specific join point with matching pointcut is reached in the application. You can think of Advices as Spring interceptors or Servlet Filters.

Pointcut: Pointcut are regular expressions that is matched with join points to determine whether advice needs to be executed or not. Pointcut uses different kinds of expressions that are matched with the join points. Spring framework uses the AspectJ pointcut expression language for determining the join points where advice methods will be applied.

Join Point: A join point is the specific point in the application such as method execution, exception handling, changing object variable values etc. In Spring AOP a join points is always the execution of a method.

What’s the difference between @Component, @Controller, @Repository & @Service annotations in Spring?

@Component is used to indicate that a class is a component. These classes are used for auto detection and configured as bean, when annotation based configurations are used.

@Controller is a specific type of component, used in MVC applications and mostly used with RequestMapping annotation. Handles mainly servlet request and response

@Repository annotation is used to indicate that a component is used as repository and a mechanism to store/retrieve/search data. We can apply this annotation with DAO pattern implementation classes. Handles DB Connection and

@Service is used to indicate that a class is a Service. Usually the business facade classes that provide some services are annotated with this.

We can use any of the above annotations for a class for auto-detection but different types are provided so that we can avoid conversion of objects as per layers.

**How to achieve localization/Multilanguage in Spring MVC applications?**

* Spring provides excellent support for localization or i18n through resource bundles. Basis steps needed to make our application localized are:
* Creating message resource bundles for different locales, such as messages\_en.properties, messages\_fr.properties etc.
* Defining messageSource bean in the spring bean configuration file of typeResourceBundleMessageSource or ReloadableResourceBundleMessageSource.
* For change of locale support, define localeResolver bean of type CookieLocaleResolver and configure LocaleChangeInterceptor interceptor. Example configuration can be like below:

|  |  |
| --- | --- |
|  | <beans:bean id="messageSource"      class="org.springframework.context.support.ReloadableResourceBundleMessageSource">      <beans:property name="basename" value="classpath:messages" />      <beans:property name="defaultEncoding" value="UTF-8" />  </beans:bean>    <beans:bean id="localeResolver"      class="org.springframework.web.servlet.i18n.CookieLocaleResolver">      <beans:property name="defaultLocale" value="en" />      <beans:property name="cookieName" value="myAppLocaleCookie"></beans:property>      <beans:property name="cookieMaxAge" value="3600"></beans:property>  </beans:bean>    <interceptors>      <beans:bean          class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">          <beans:property name="paramName" value="locale" />      </beans:bean>  </interceptors> |

* Use spring:message element in the view pages with key names, DispatcherServlet picks the corresponding value and renders the page in corresponding locale and return as response.

**What are some of the important Spring annotations you have used?**

Some of the Spring annotations that I have used in my project are:

@Controller – for controller classes in Spring MVC project.

@RequestMapping – for configuring URI mapping in controller handler methods. This is a very important annotation

@ResponseBody – for sending Object as response, usually for sending XML or JSON data as response.

@PathVariable – for mapping dynamic values from the URI to handler method arguments.

@Autowired – for autowiring dependencies in spring beans.

@Qualifier – with @Autowired annotation to avoid confusion when multiple instances of bean type is present.

@Service – for service classes.

@Scope – for configuring scope of the spring bean.

@Configuration, @ComponentScan and @Bean – for java based configurations.

AspectJ annotations for configuring aspects and advices, @Aspect, @Before, @After, @Around,@Pointcut etc.

**What are the advantages of JdbcTemplate in spring?**

Less code: By using the JdbcTemplate class, you don't need to create connection,statement,start transaction,commit transaction and close connection to execute different queries. You can execute the query directly.

**Explain RowMapper Interface?**

RowMapper interface allows to map a row of the relations with the instance of user-defined class. It iterates the ResultSet internally and adds it into the collection. So we don't need to write a lot of code to fetch the records as ResultSetExtractor.

Advantage of RowMapper :

RowMapper saves a lot of code becuase it internally adds the data of ResultSet into the collection.

Method of RowMapper interface

It defines only one method mapRow that accepts ResultSet instance and int as the parameter list. Syntax of the method is given below:

public T mapRow(ResultSet rs, int rowNumber)throws SQLException

Example:

public class EmployeeDao {

private JdbcTemplate template;

public void setTemplate(JdbcTemplate template) {

    this.template = template;

}

public List<Employee> getAllEmployeesRowMapper(){

 return template.query("select \* from employee",new RowMapper<Employee>(){

    @Override

    public Employee mapRow(ResultSet rs, int rownumber) throws SQLException {

        Employee e=new Employee();

        e.setId(rs.getInt(1));

        e.setName(rs.getString(2));

        e.setSalary(rs.getInt(3));

        return e;

    }

    });

}

}