Spring MVC:

A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

A Spring MVC provides an elegant solution to use MVC in spring framework by the help of DispatcherServlet. Here, DispatcherServlet is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.

Spring MVC Application Flow:

Step 1: First request will be received by DispatcherServlet

Step 2: DispatcherServlet will take the help of HandlerMapping and get to know the Controller class name associated with the given request

Step 3: So request transfer to the Controller, and then controller will process the request by executing appropriate methods and returns ModeAndView object (contains Model data and View name) back to the DispatcherServlet

Step 4: Now DispatcherServlet send the model object to the ViewResolver to get the actual view page

Step 5: Finally DispatcherServlet will pass the Model object to the View page to display the result

Spring Web Application Without Maven:

web.xml

```
<web-app>
<servlet>
      <servlet-name>dispatcher</servlet-name>
      <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>dispatcher</servlet-name>
      <url-pattern>/</url-pattern>
</servlet-mapping>
<context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/dispatcher-servlet.xml</param-value>
</context-param>
</web-app>
```

dispatcher-servlet.xml

```
<context:component-scan
      base-package="com.spring.mvc" />
<mvc:annotation-driven />
<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix">
            <value>/WEB-INF/jsp/</value>
      </property>
      <property name="suffix">
            <value>.jsp</value>
      </property>
</bean>
```

DispatcherServlet

DispatcherServlet is essentially a Servlet (it extends HttpServlet) whose primary purpose is to handle incoming web requests matching the configured URL pattern. It take an incoming URI and find the right combination of controller and view. So it is the front controller.

When you define a DispatcherServlet in spring configuration, you provide an XML file with entries of controller classes, views mappings etc. using contextConfigLocation attribute.

web.xml

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/dispatcher-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

ContextLoaderListener

ContextLoaderListener creates the root application context and will be shared with child contexts created by all DispatcherServlet contexts. You can have only one entry of this in web.xml.

```
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
```

The context of ContextLoaderListener contains beans that globally visible, like services, repositories, infrastructure beans, etc.

InternalResourceViewResolver

Convenient subclass of UrlBasedViewResolver that supports InternalResourceView (in effect, Servlets and JSPs) and subclasses such as JstlView and TilesView. You can specify the view class for all views generated by this resolver by using setViewClass(..).

Java Configuration

WebAppInitializer.java:

```
public class WebAppInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

	@Override
	protected Class<?>[] getRootConfigClasses() {

		return new Class[] {MyDispatcher.class};
	}

	@Override
	protected Class<?>[] getServletConfigClasses() {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	protected String[] getServletMappings() {

		return new String[] {"/"};
	}

}
```

MyDispatcher.java

Java Configuration:

```
	@Configuration
	@EnableWebMvc
	@ComponentScan(basePackages ={"com.spring.shopping"})
	public class MyDispatcher {
```

```
            @Bean
            public ViewResolver viewResolver(){
                  InternalResourceViewResolver viewResolver = new
InternalResourceViewResolver();
                  viewResolver.setPrefix("/WEB-INF/jsp/");
                  viewResolver.setSuffix(".jsp");
                  return viewResolver;
            }
      }
```

@Controller : annotation marks this class as spring bean which may handle different HTTP requests based on mapping specified on class or individual controller methods.

@RequestMapping: annotation is used for mapping web requests onto specific handler classes and/or handler methods.

ModelMap/Model : is a Map implementation, which saves you from old request.getAttribute/ request.setAttribute. It provides a way to set/get attributes from/to request or session.

<mvc:annotation-driven /> says that we can define spring beans dependencies without actually having to specify a bunch of beans in xml or implement an interface or extend a base class or anything

For example, just by annotating a class with @Controller, spring will know that the specified class contains methods that will handle HTTP requests, no need to define that as a bean in xml.

<context:component-scan base-package="" />

Tells spring to search project classpath for all classes under the package specified with base-package and look at each class to see if it is annotated with specific Spring annotations **[@Controller, @Service, @Repository, @Component, etc..]** and if it does then Spring will register the class with the bean factory as if you had typed in the xml configuration files.