**Spring Framework:**   
  
Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable, and reusable code. The core features of Spring can be used in developing any Java application. We can use its extensions for building various web applications on top of the Java EE platform, or we may just use its dependency injection provisions in simple standalone applications.

**Inversion of Control-Dependency injection:**

Dependency Injection, an aspect of Inversion of Control (IoC), is a general concept stating that

you do not create your objects manually but instead describe how they should be created. An IoC container will instantiate required classes if needed. In simple Dependency Injection is a

principle where one object should not depend on other. We can implement this by setter

injection, Constructor Injection, Interface Injection.

When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while unit testing. Dependency Injection helps in gluing these classes together and at the same time keeping them independent.

For example, class A is dependent of class B. Now, let's look at the second part, injection. All this means is, class B will get injected into class A by the IoC.

Class Emp{

Address ad = new Address();

}

Instead:

Emp(Add ad) {}  
  
**Spring MVC:**

Following is the sequence of events corresponding to an incoming HTTP request (Comes from UI – front end) to DispatcherServlet −

After receiving an HTTP request, DispatcherServlet consults the HandlerMapping to call the

appropriate Controller (Controller is the layer in MVC where the code is controlled via Services i.e API’s , where it can get data through database etc)

The Controller takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and

returns view name to the DispatcherServlet. (contains Model data and View name)

The DispatcherServlet will take help from ***ViewResolver*** to pickup the defined view for the

request.

Once view is finalized, The ***DispatcherServlet*** passes the model data to the view which is finally

rendered on the browser.  
more understanding :   
  
<https://www.tutorialspoint.com/spring/spring_environment_setup.htm>