* Using @Component, we tell Spring Framework - Hey there, this is a bean that you need to manage.
* Using @Autowired, we tell Spring Framework - Hey find the correct match for this specific type and autowire it in.

# Q :-> [What are the differences between Model, ModelMap, and ModelAndView?](https://stackoverflow.com/questions/18486660/what-are-the-differences-between-model-modelmap-and-modelandview)

**Model:** It is an Interface. It defines a holder for model attributes and primarily designed for adding attributes to the model.

Example:

@RequestMapping(method = RequestMethod.GET)

public String printHello(Model model) {

model.addAttribute("message", "Hello World!!");

return "hello";

}

**ModelMap:** Implementation of Map for use when building model data for use with UI tools.Supports chained calls and generation of model attribute names.

Example:

@RequestMapping("/helloworld")

public String hello(ModelMap map) {

String helloWorldMessage = "Hello world!";

String welcomeMessage = "Welcome!";

map.addAttribute("helloMessage", helloWorldMessage);

map.addAttribute("welcomeMessage", welcomeMessage);

return "hello";

}

**ModelAndView:** This class merely holds both to make it possible for a controller to return both model and view in a single return value.

Example:

@RequestMapping("/welcome")

public ModelAndView helloWorld() {

String message = "Hello World!";

return new ModelAndView("welcome", "message", message);

}

# ViewResolver :

# In Spring MVC, view resolvers enable you to render models in a browser without tying you to a specific view technology like JSP, Velocity, XML…etc. There are two interfaces that are important to the way Spring handles views are ViewResolver and View

* The *ViewResolver* maps view names to actual views.

And the Spring framework comes with quite a few view resolvers e.g. *InternalResourceViewResolver*, *XmlViewResolver*, *ResourceBundleViewResolver* and a few others.

Below are the important viewresolvers provided by spring framework:

1. **AbstractCachingViewResolver :** Abstract view resolver that caches views. Often views need preparation before they can be used; extending this view resolver provides caching.
2. **XmlViewResolver :**Implementation of ViewResolver that accepts a configuration file written in XML with the same DTD as Spring’s XML bean factories. The default configuration file is /WEB-INF/views.xml.
3. **ResourceBundleViewResolver :**Implementation of ViewResolver that uses bean definitions in a ResourceBundle, specified by the bundle base name. Typically you define the bundle in a properties file, located in the classpath. The default file name is views.properties.
4. **UrlBasedViewResolver :**Simple implementation of the ViewResolver interface that effects the direct resolution of logical view names to URLs, without an explicit mapping definition. This is appropriate if your logical names match the names of your view resources in a straightforward manner, without the need for arbitrary mappings.
5. **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(..).
6. **VelocityViewResolver/FreeMarkerViewResolver :**Convenient subclass of UrlBasedViewResolver that supports VelocityView (in effect, Velocity templates) or FreeMarkerView ,respectively, and custom subclasses of them.
7. **ContentNegotiatingViewResolver :**Implementation of the ViewResolver interface that resolves a view based on the request file name or Accept header.