* **@Configuration** indicates that class can be used by the Spring IoC container as a source of bean definitions
* **@Bean** indicates that method will return an *object* that should be registered as a bean in spring application context
* **@Scope("singleton")** sets the scope of the bean.Default value is singleton.
  + **Singleton:** All requests for that bean name will return the same object. Any modifications to the object will be reflected in all references to the bean.
  + **Prototype:** will return a different instance every time it is requested from the container.

There are other web related scopes like: request, session, application, websocket. We can use constant to define scope, instead of string.

@Scope (value = ConfigurableBeanFactory.SCOPE\_PROTOTYPE)

* **@Component("thatSillyCoach")** Indicates that class is considered for auto-detection for IoC container, when using annotation-based configuration and classpath scanning.
* **@PropertySource("classpath:sports.properties")** for adding property source file to the environment.
  + We can also use a placeholder to register property file.

@PropertySource ({“classpath:persistence- ${envTarget:mysql}.properties”})

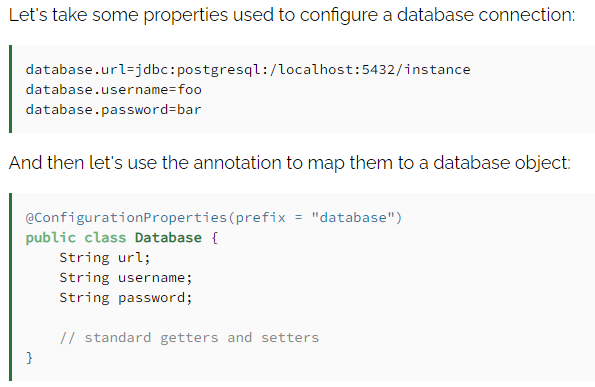
* + We can use multiple property file.

@PropertySource("classpath:foo.properties") @PropertySource("classpath:bar.properties")

**Or**

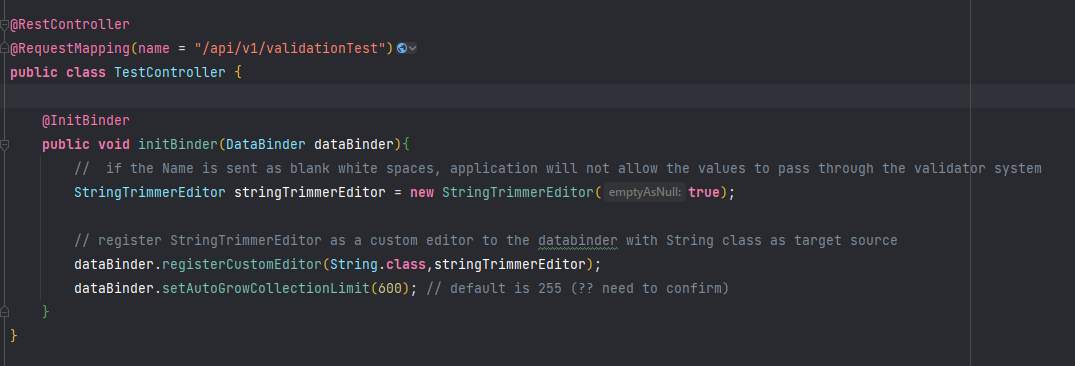
@PropertySources({ @PropertySource("classpath:foo.properties"), @PropertySource("classpath:bar.properties") })

* + in the event of a property name collision during multiple property files, the last source read takes precedence.
* **@Value(**"${jdbc.url:aDefaultUrl}"**)** injects the default value from property file.
  + We can use Environment variable to access different profiles, which we can’t do with @value
* **@ConfigurationProperties** If we have properties that are grouped together, we can make use of the @ConfigurationProperties annotation, which will map these property hierarchies into Java objects graphs.

****

* **@SpringBootApplication** encapsulates **@Configuration, @EnableAutoConfiguration,** and **@ComponentScan** annotations with their default attributes.
* **@Autowired**
* **@ExceptionHandler** It says that this method is an exception handler.
* **@ControllerAdvice**

**@InitBinder:** we can use this annotation in a controller method to initialize **WebDataBinder.**



**@Async:** Simply put, annotating a method of a bean with **@Async** will make it execute in a separate thread. In other words, the caller will not wait for the completion of the called method. Used for asynchronous programming.

**@Scheduled:** This annotation defines when a particular method runs.

