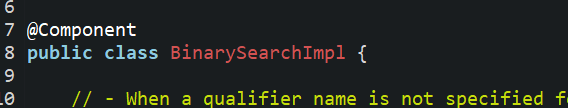
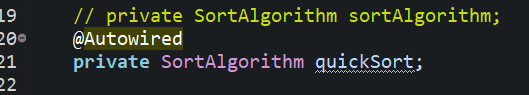
--- Missing introductory notes ---

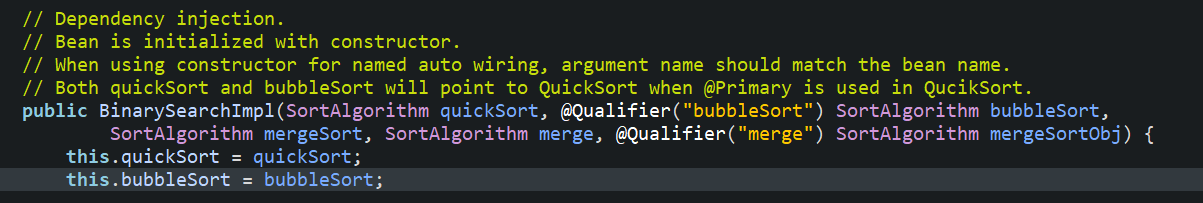
1. Auto wiring
   1. Components of an application are marked or highlighted as components using @Component annotation.



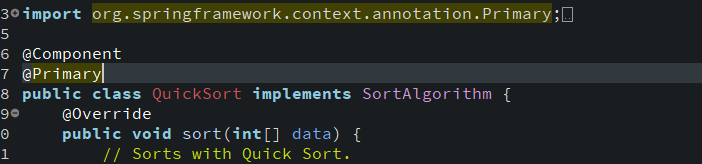
* 1. Components can be mapped to members of other components using @AutoWiring annotation.



* 1. Auto wiring can be performed in two ways:
     1. Directly to a member as show in the above screenshot.
     2. Using a constructor. This takes precedence over member auto wiring.

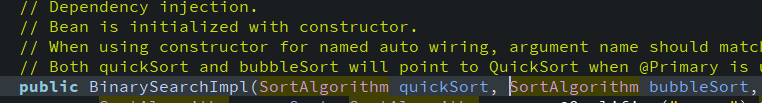


* 1. When there are more than one implementation for a bean; the implementation to be used can be mandated by add @Primary annotation.



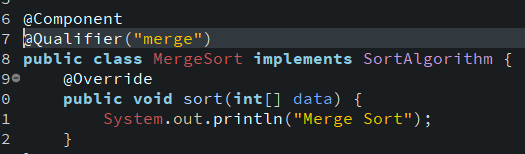
* 1. Conflict with multiple bean implementation can also be resolved by using name based auto wiring. This is achieved by naming either the member or the constructor argument same as the implementing bean name.

In the below example, SortAlgorithm as QucikSort and BubbleSort implementation. The corresponding beans are auto wired by naming the constructor argument quickSort and bubbleSort.

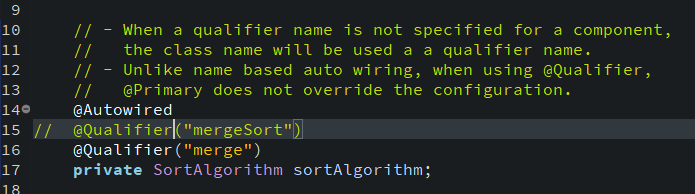


Note: When a primary bean is defined, name based auto wiring will not work.

* 1. Component names can be customized using @Qualifier annotation.

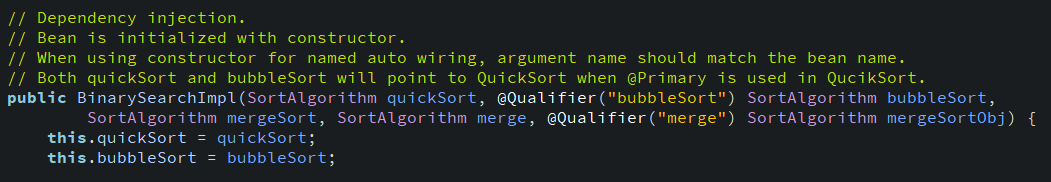


Bean is auto wired to component by using same annotation.

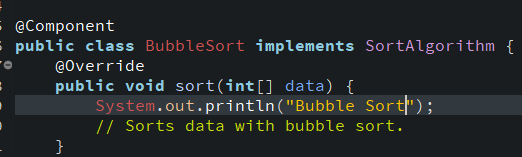


It is not mandatory to name a component using @Qualifier. During auto wiring we can use the class name to similar to named auto wiring.

In the below example, QuickSort is set as primary bean and MergeSort has qualifier bean name “merge”.



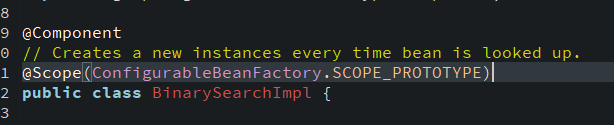
* quickSort is auto wired to instance of QuickSort – Since it is primary.
* bubbleSort is auto wired to instance of BubbleSort – Because of @Qualifier, even when BubbleSort does not have a @Qualifier annotation.



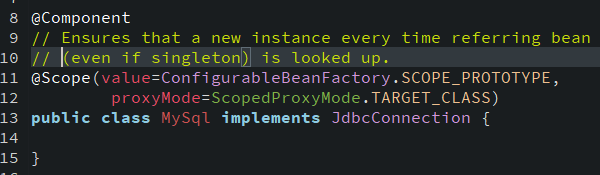
* mergeSort is auto wired to instance of QuickSort – Because, quick sort is primary.
* merge is auto wired to instance of QuickSort – Because, quick sort is primary..
* mergeSortObj is auto wired to instance of MergeSort – BEcaue of @Qualifier.

1. **Scope**
   1. Available bean scope:

* singleton – Default scope. Created beans are singleton in nature.
* prototype – A new object created for each lookup.
* request – A new bean is created for each request.
* session – A new bean is created for each user session.

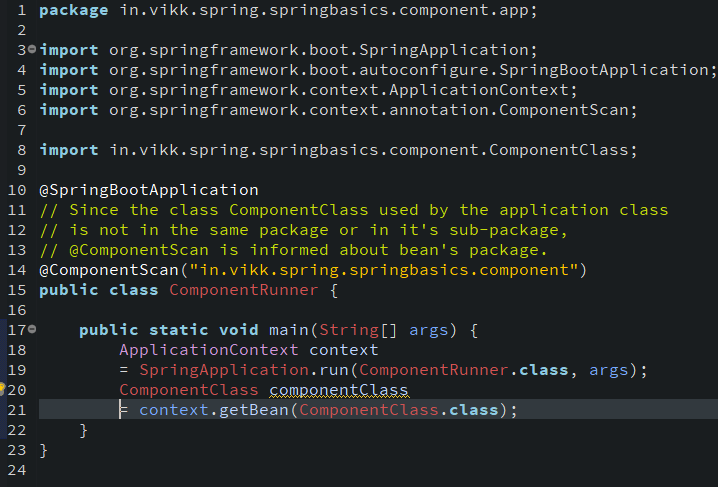


* 1. If a prototype component is referring a singleton bean, all the instances of prototype bean will refer the same instance of singleton bean.
  2. If a singleton bean is referring a prototype bean. Prototype bean can be configured to create a new instance for every new instance of singleton bean setting the **proxy mode** of the **scope** to **target class**.



1. **Component Scan**

If application class uses beans residing in different package then Spring’s component scan should be informed about the package it should scan for potential spring beans using @ComponentScan annotation to the application class.

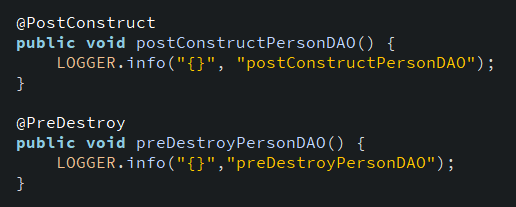


1. **Bean life cycle**

We can hookup methods during creation or destruction of a bean using **@PostConstruct** and **@PreDestroy** annotations.

@PostConstruct is executed soon after the bean constructor and before any other bean methods are executed.

@PreDestroy is executed when bean is unloaded from the memory.

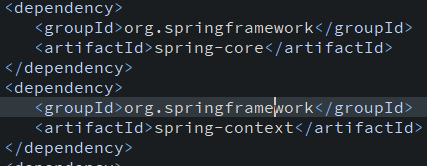


**Observations:**

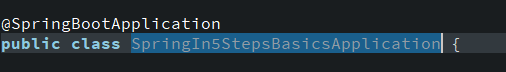
* In inheritance hierarchy, child’s post constructor and pre destroy hooks are executed first.
* If beans scope is set to prototype and proxyMode is set to Target Class then it’s preconstruct bean life cycle methods are not executed.
* If bean’s scope is set to prototype then it’s postDestroy lifecycle methods are not called.
* If multiple pre and post hooks are present then they are executed in the order in which they appear in Class<T>.getMethods().

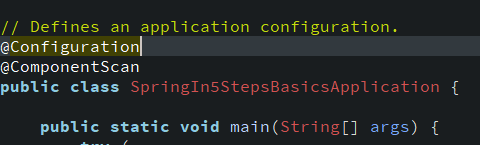
1. Spring Core

Spring core provides bean initialization and bean mapping ability without the heavy weight of spring boot. We can use spring code by using the following dependency in pox.xml.



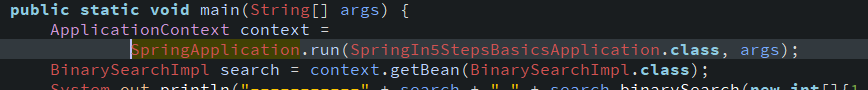
In case of spring boot, configuration bean is set by annotation **@SpringBootApplication**. In case of spring core, it is specified using **@Configuration**.

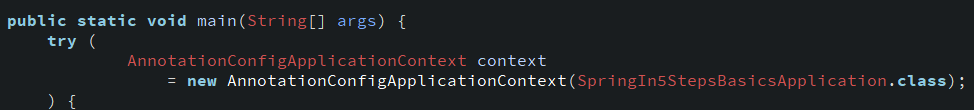




Spring core does not provide an implicit component scan like spring boot. So, component scan should be specified by the configuration bean using **@ComponentScan** annotation.

In spring boot, AnnotationConfigApplicationContext is initialized using SpringApplication.run. But, in case of spring core it is obtained directly instantiating the class.





If application XML is used for bean specification the XmlServletWebServerApplicationContext.

1. **IOC Containers**

Auto wiring or bean lookup ability of spring is called Inversion of Control. Spring provide two ICO container:

* Bean Factory

Bean factory provided the ability to initialize beans and map dependencies.

* Application Context

Along with the functionality of bean factory, application context provides all the ability an enterprise application needs.

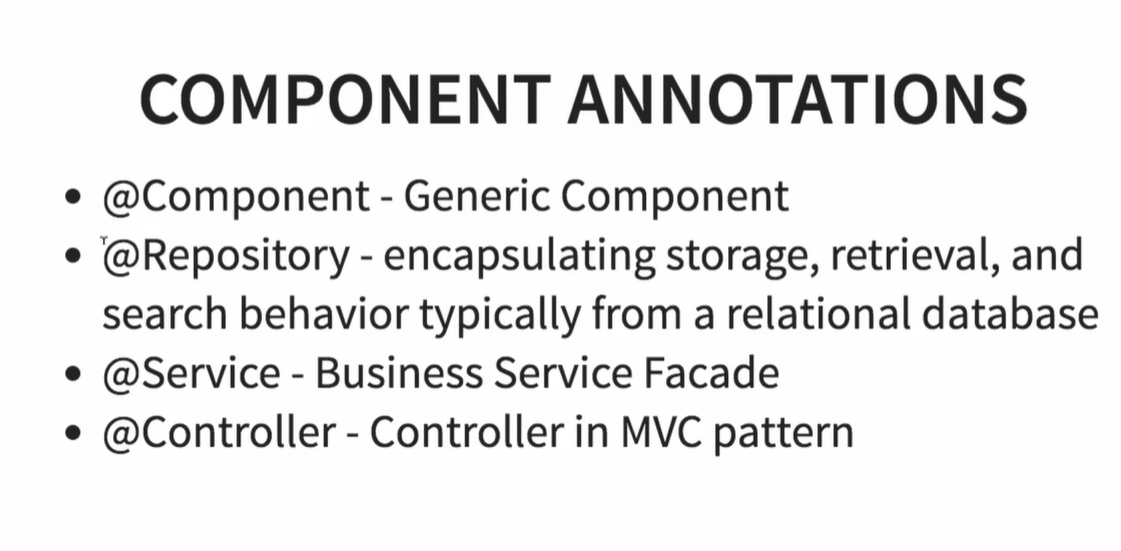




1. **Component Annotations**

Component annotations are of four types:

* **@Component** – Generic component
* **@Controller** – MVC controller in a web application
* **@Repository** – Database component
* **@Service** – Business layer component.



1. **Property files**

External property files can be read using **@PropertyScan** and **@Value** annotations.

