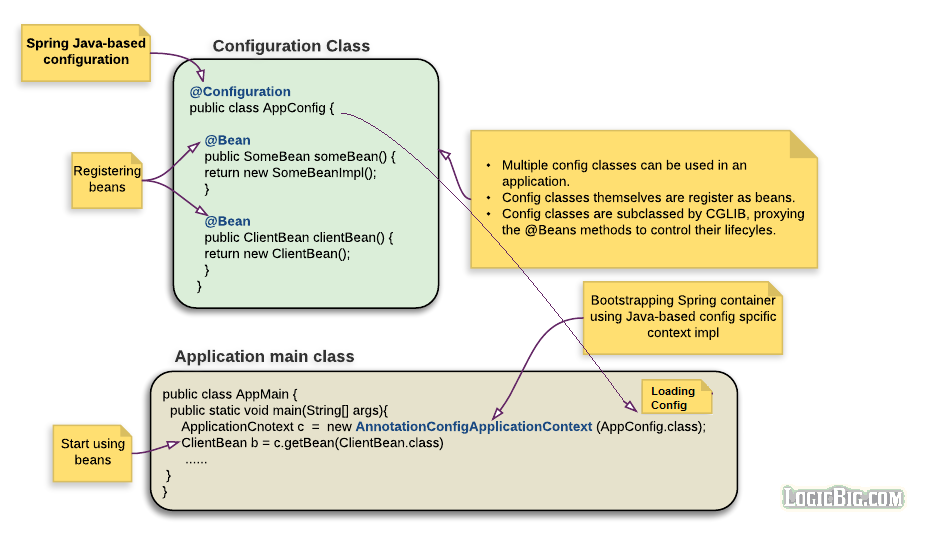
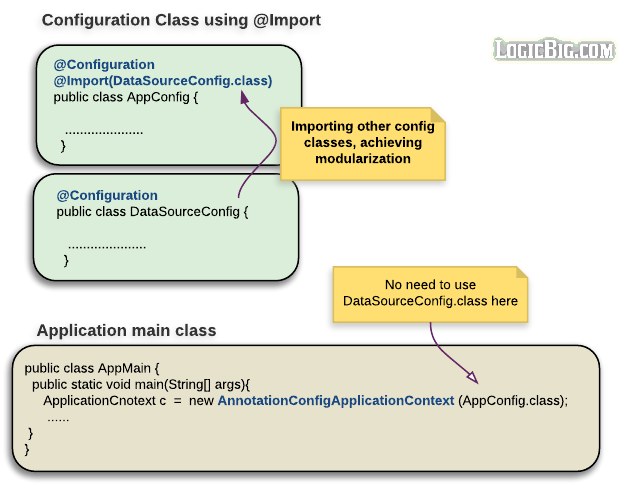
**Spring - Configuration Metadata**

Spring configuration metadata is to tell Spring container how to initiate, configure, wire and assemble the application specific objects

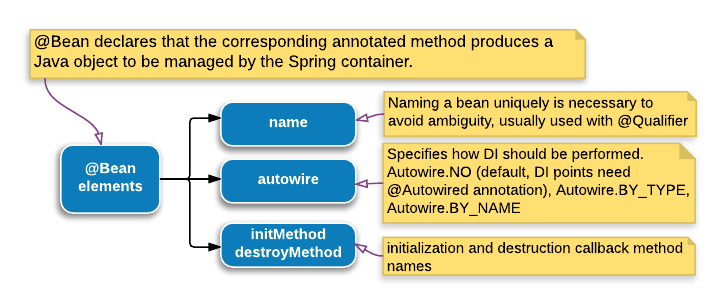
1. **XML-based Configuration :** All configurations are in one or multiple XML files. This is the most verbose way of configuration. Huge projects require tedious amount of XML which is difficult to manage.
2. **Annotation-based configuration :** Spring 2.5 introduces annotation-based configuration. We still have to write XML files but just to indicate "component-scan" on the packages of annotated classes.
3. **Java-based configuration (JavaConfig):**Starting with Spring 3.0, a pure-Java means of configuring container was provided. We don't need any XML with this method of configuration.



# Spring - Using @Import



# Spring - Using @Bean



The @Bean annotation is used in @configuration annotated class

**name :**

The optional bean name.

@Configuration

public class AppConfig {

@Bean(name = "myBean")

public MyBean createBean() {

......

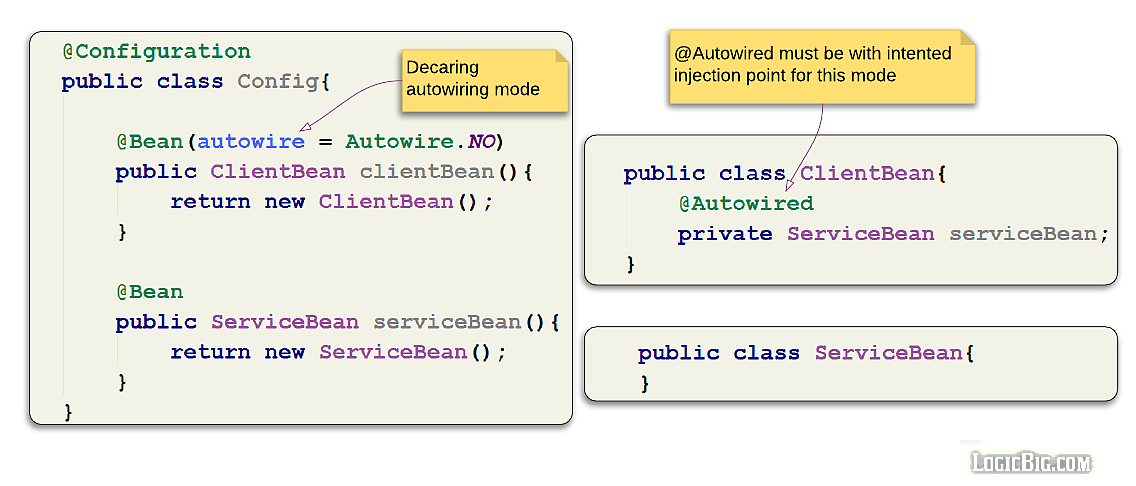
}

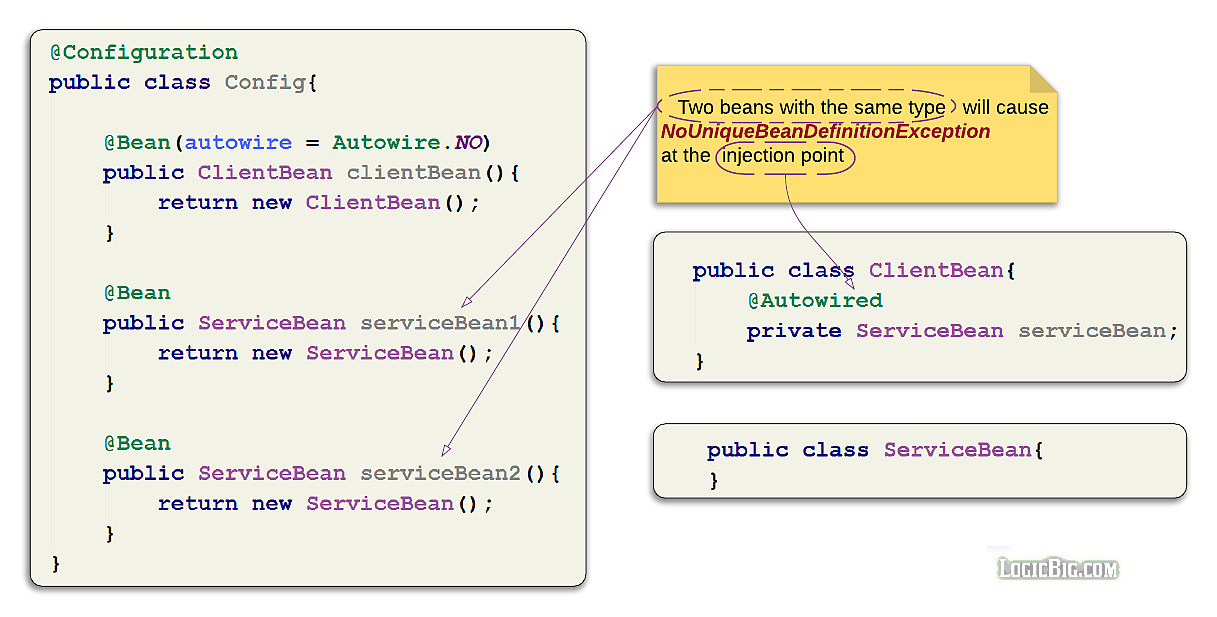
}

**autowire :**

The autowiring mode.

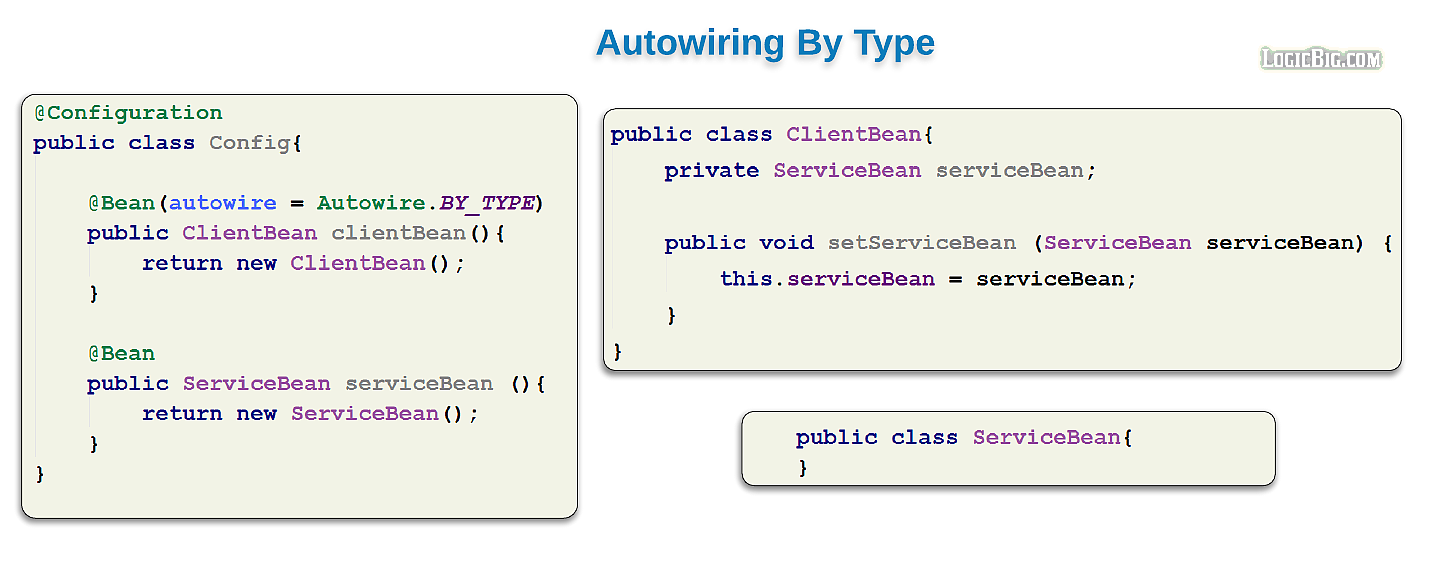
Autowire.NO : This is the default. In this case, we have to explicitly use @Autowired at injection point.



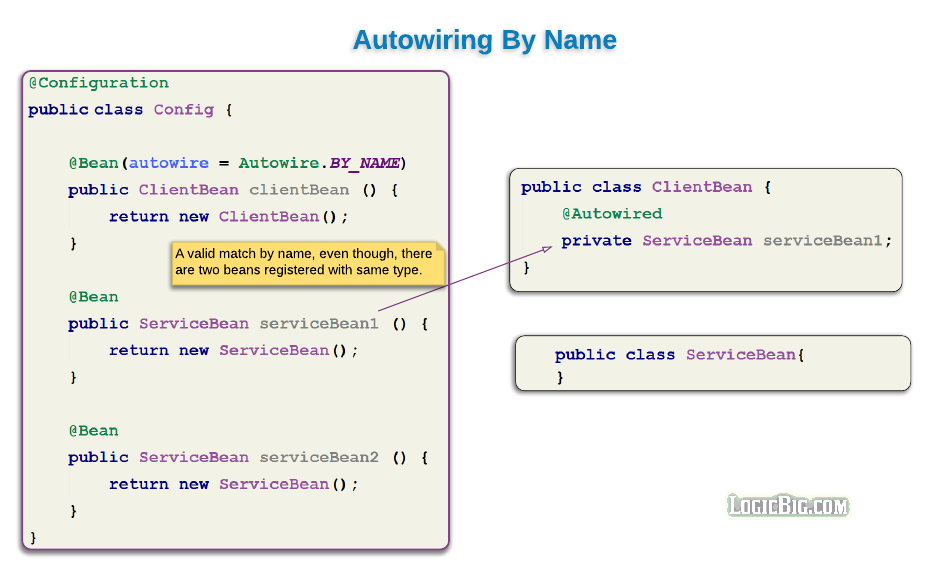


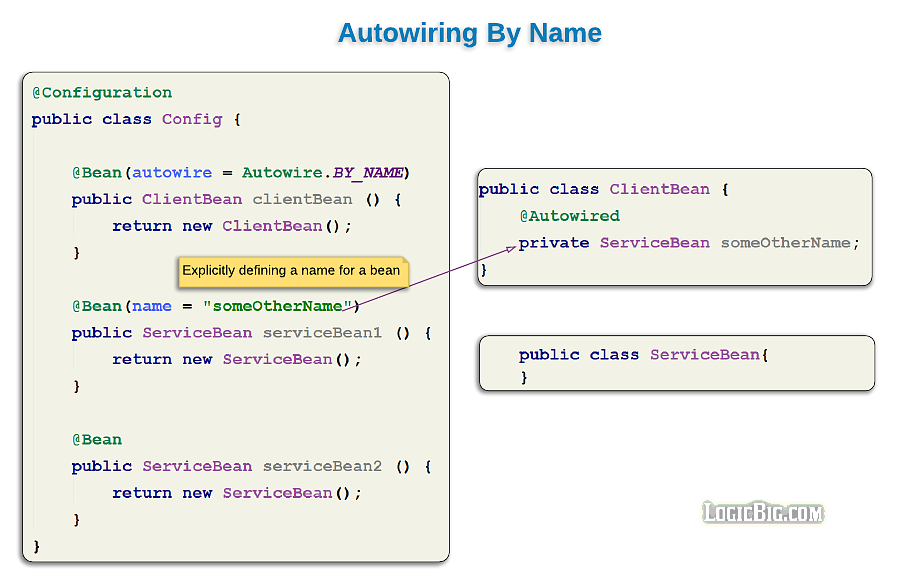
To avoid exception, we have to [use @Qualifier](https://www.logicbig.com/tutorials/spring-framework/spring-core/inject-bean-by-name.html).

Autowire.BY\_TYPE : we don't need @Autowired at the injection point, given that there is only one bean available for the injection. In this mode of autowiring, the field injection doesn't work. There must be a setter.

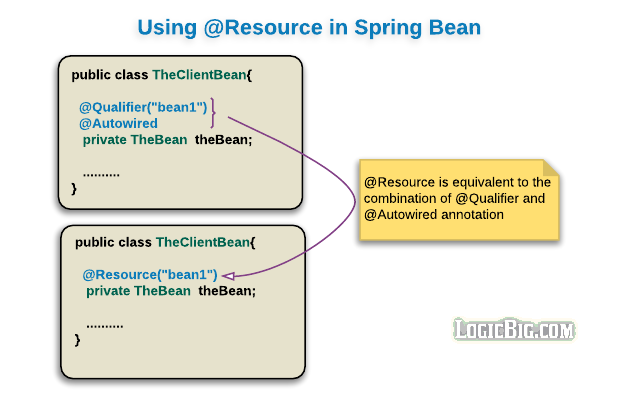


Autowire.BY\_NAME : If this mode of autowiring is specified and injection provider bean has specified name element with the some value in it's @Bean annotation, we have to use @Qualifier along with @Autowiredat injection point.

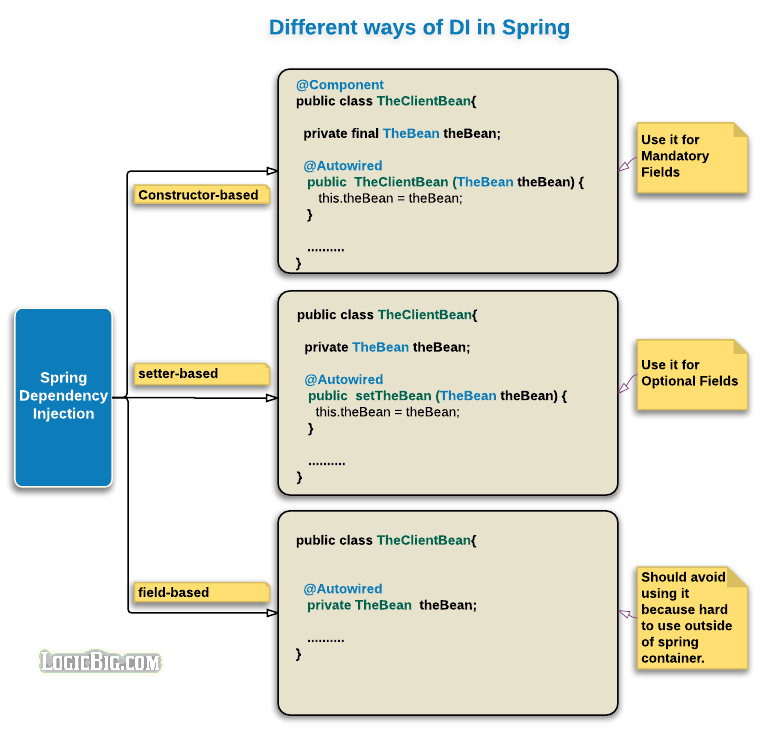




# Spring - Inject Bean By Name using @Resource



# Spring - Different ways of injecting dependencies



Type of auto-wiring means how to inject other beans in this class (no,Bytype or Byname)

Types of dependency injection means where to inject beans (on constructor, on field or on setter method)

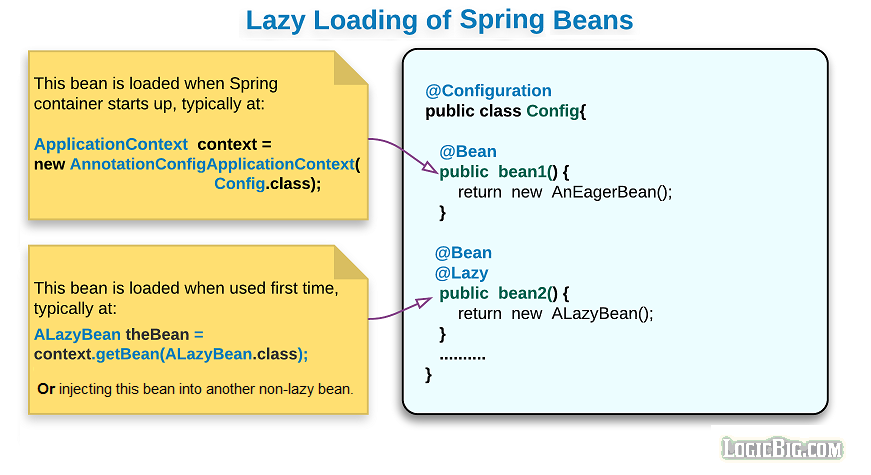
# Spring - Initialization and destruction lifecycle callbacks

There are many ways to handle lifecycle of bean

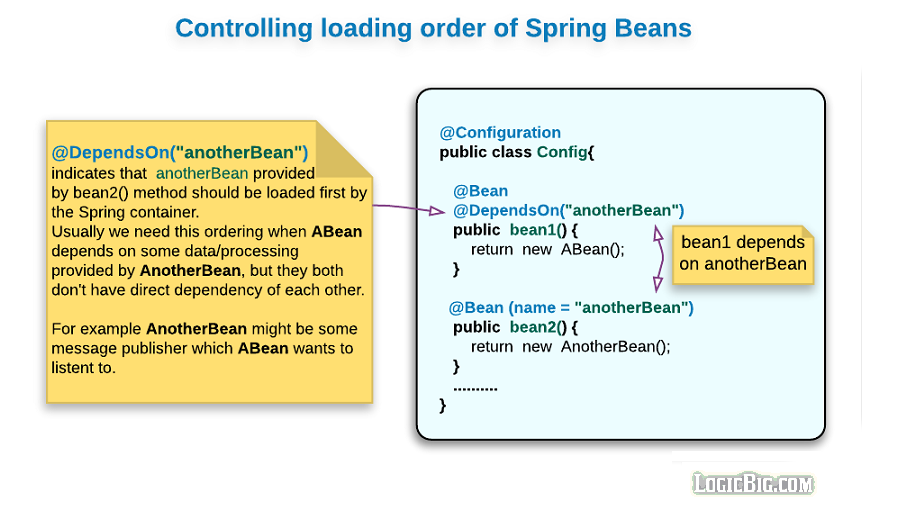
1. @Bean(initMethod = "init", destroyMethod = "destroy")
2. Using @PostConstruct and @PreDestroy
3. Implementing InitializingBean and DisposableBean

# Spring - Lazy Initialization, using @Lazy

By default Spring container instantiates all configured beans at startup (eager loading). In some situations, however, beans might rarely be used during application life cycle. Loading them at startup will, specially, be not good if they are going to use considerable memory to get initialized. In those sort of situations we may decide to initialize such beans only when they are first accessed by application code (i.e. initialize on demand). We can achieve that by using @Lazy on bean configuration method.



# Spring - Controlling Beans Loading Order, using @DependsOn



**Spring - Bean Scopes**

**singleton :** Only one instance of bean per Spring container (here container mean per org.springframework.context.ApplicationContext). That means regardless of how many times, we access/inject the bean there will be only one instance provided by the container. This is the default one.

1. **prototype :** A new instance of a bean is created, each time it's injected to some other bean or accessed via the container (springContext.getBean(...)). In what situations should we use this scope? The answer is, whenever calling code wants to set some personalized session information among multiple method calls on that bean. On the other hand a singleton is not and should not be aware of a caller specific session. In that sense we can say singletons are stateless (suitable for a service, DAO or controller etc) , whereas prototypes are stateful with respect to a particular calling session (for example shopping cart, wizard like steps etc). Why don't we create such beans ourselves using **new** operator rather than registering it to the Spring container? Yes we should if we can, but what if we want to conveniently have Spring to do some DI for us. Well, then of course we should use this scope.

[**Using on bean factory methods of @Configuration class**](https://www.logicbig.com/tutorials/spring-framework/spring-core/java-config.html)**:**

@Scope is used in @Configuration annotated class's method. These methods should primarily [be annotated with @Bean](https://www.logicbig.com/tutorials/spring-framework/spring-core/using-bean-annotation.html).

[**Using on classes annotated with @Component**](https://www.logicbig.com/tutorials/spring-framework/spring-core/javaconfig-with-componnet-scan.html)**:**

@Scope is used on component classes. This classes should be scanned by Spring at startup if @ComponentScan along with packages to scan is defined on @Configuration class.

## No Pre Destroy callback for Prototype

Spring does not manage the complete lifecycle of a prototype bean. The container instantiates, configures, a prototype bean instance, and hands it to the client, with no further record of the instance. That's the reason, the prototype bean's method annotated with PreDestroy will never be called. The initialization lifecycle callback methods (@PostConstruct) are always called on all objects regardless of scope.

Following are the three scopes available, if spring application loaded using WebApplicationContext

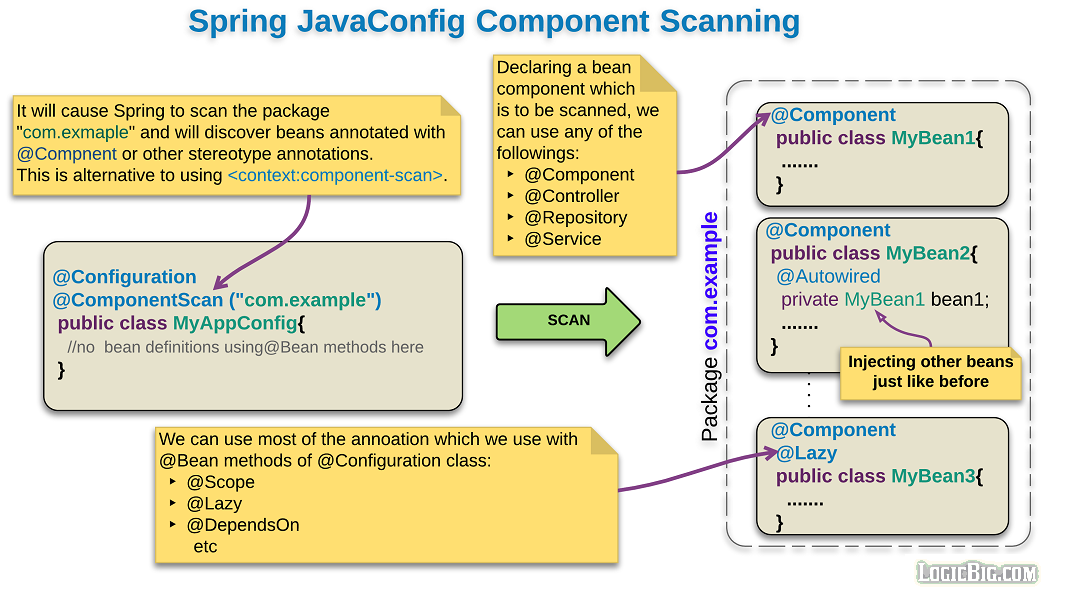
1. **request:**One instance per HTTP request, i.e. every new HTTP request will have its own instance of the bean.
2. **session:**One instance per HTTP session.
3. **globalSession:**One instance per global HTTP session. Typically only valid when used in a portlet context.

**Spring - JavaConfig with Component Scan**

For component scanning to work we must annotate our beans with one of the stereotype annotations

1. Component
2. Controller
3. Repository
4. Service

Classes annotated with one of the above are candidate for spring container registration when using scanning.



**Spring - Implicit constructor Injection**

Starting Spring 4.3, it is no longer necessary to specify the @Autowired annotation if the target bean only defines one constructor.

public class FooService {

private final FooRepository repository;

public FooService(FooRepository repository) {

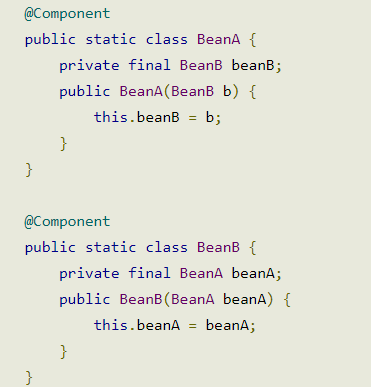
this.repository = repository

}

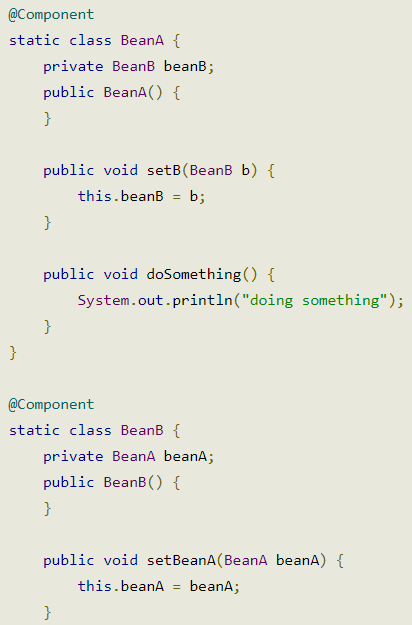
}

**Spring - Circular Dependencies**

Circular dependencies is the scenario when two or more beans try to inject each other via constructor.



## Fixing circular dependencies by using setter injection



## Fixing circular dependencies by using @Lazy at constructor injection point

