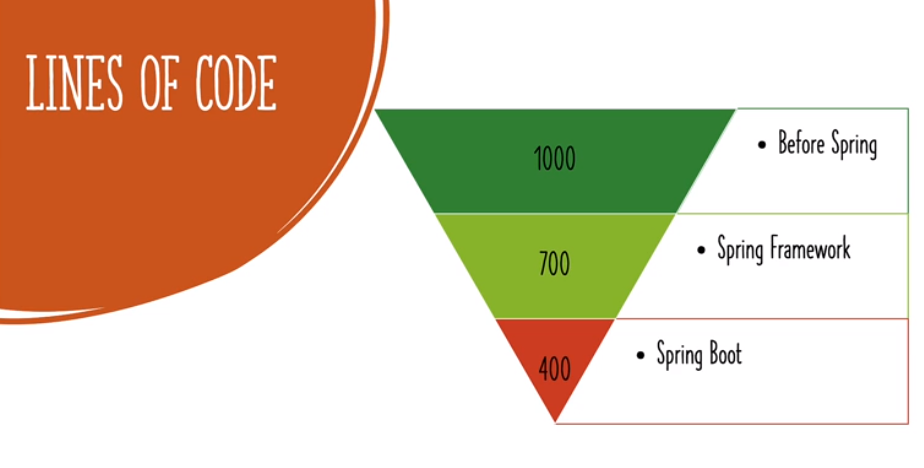
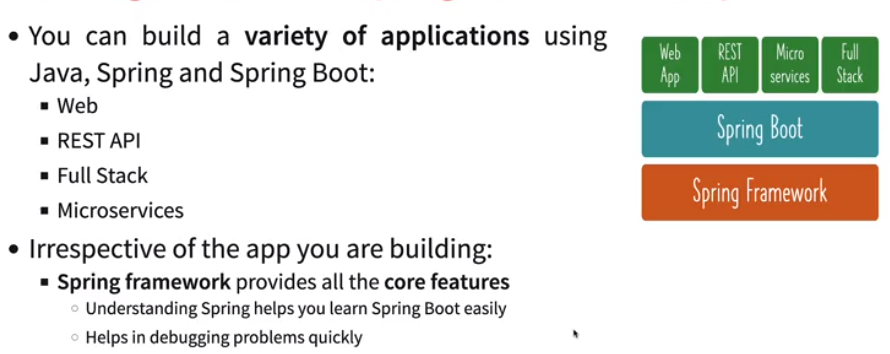
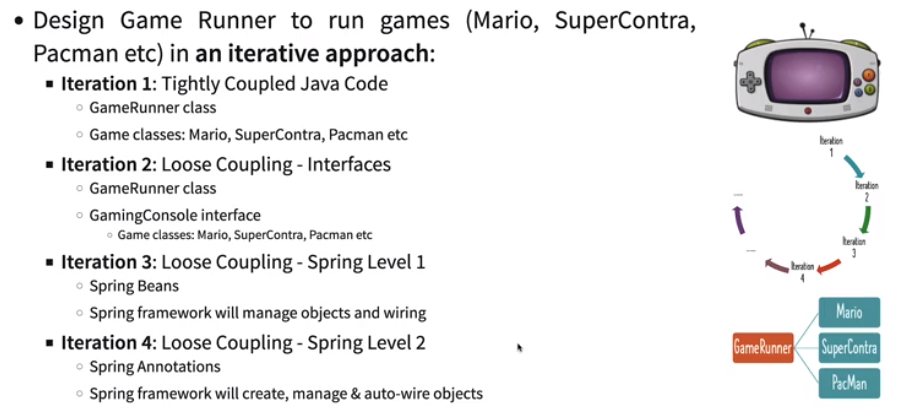
**Spring Framework**

Before spring we needed to write too much lengthy line of codes for developing production ready applications. With introduction of Spring framework and Spring Boot this has greatly reduced for production ready application.

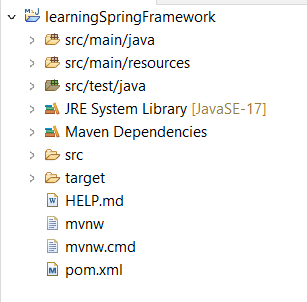




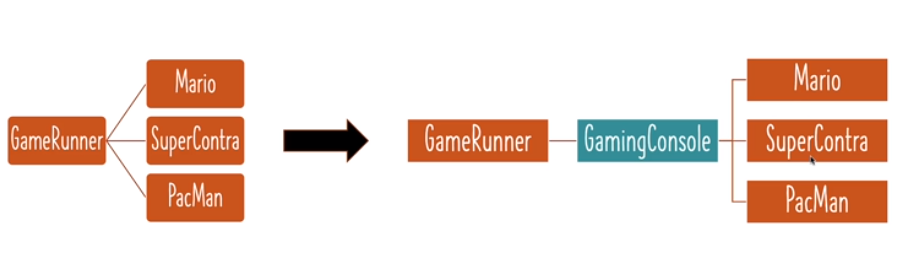
* **What we will do in basic?**



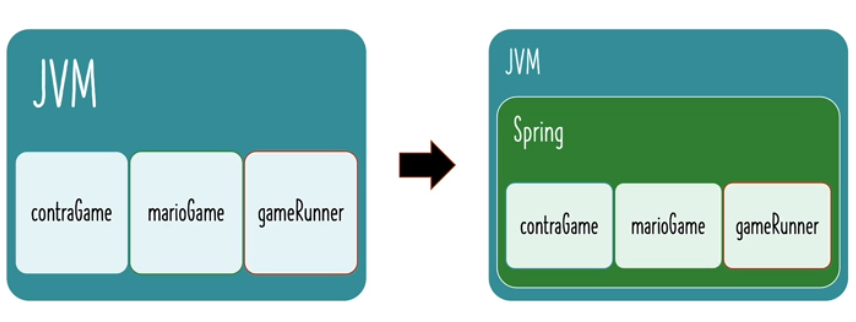
* **Creating a spring project:** 
  + *Go to* [*https://start.spring.io/*](https://start.spring.io/)*.*
  + *Fill out artifact Id, group Id, project name, language, choose maven or gradle and Spring version (Do not use snapshot version as they are still in development from Spring team).*
  + *Click on generate, download zip and extract.*
  + *Import as Maven in Eclipse and let build of project import complete.*

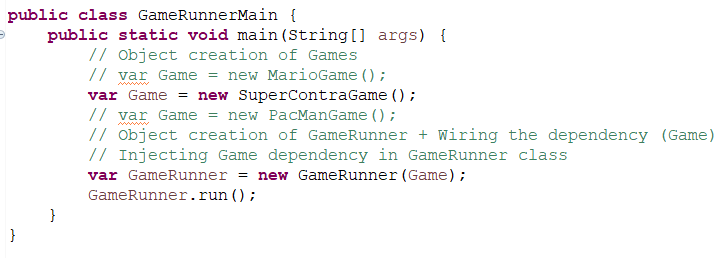


* ***Tight coupling and Loosely coupling:*** *Coupling measures how much of code changes are required to be worked on making some changes. In order to do loosely coupling we can use interface, whose methods will be overridden by other classes having same implementation.*



* Generally when tightly couple we’re asking JVM to create objects, using Spring we can do it automatically.



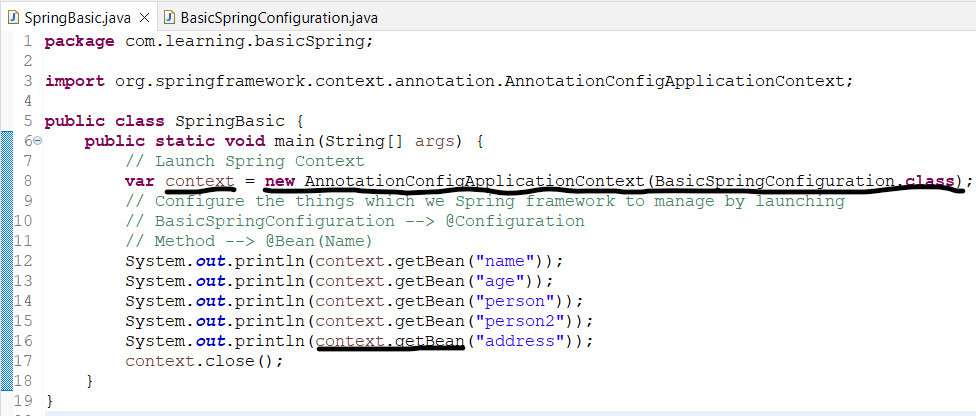


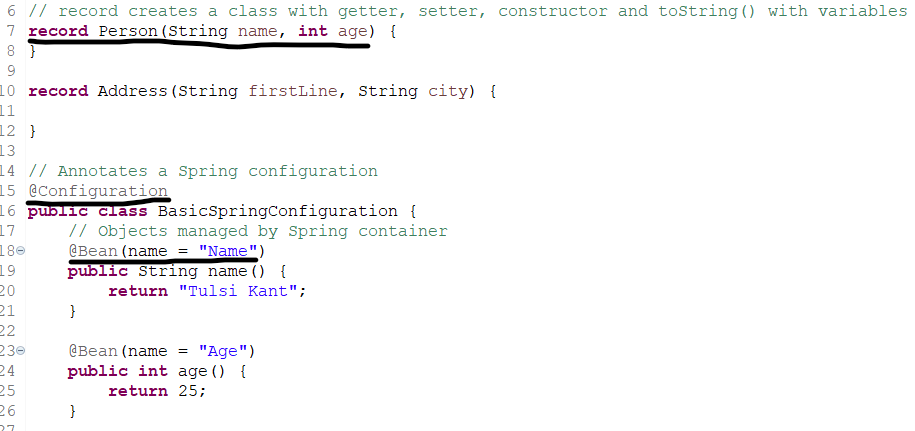
* ***Beans:*** *Elements which are managed by Spring container is called beans.*
* ***Configuring an application by Spring:***
  + Create a main class where we will launch Spring Context 🡪 *Create object of AnnotationConfigApplicationContext(<Configuration>.class) 🡪 Imports org.springframework.context.annotation.AnnotationConfigApplicationContext*
  + Configure the things which we want Spring framework to manage by creating SpringConfiguration class. 🡪 *Annotate class as @Configuration 🡪 Imports org.springframework.context.annotation.Configuration*
  + Create a SpringConfiguration will have beans to be managed via Spring. 🡪 *Annotate* *with @Bean (Method, Class, Variables) to be managed. 🡪 Imports*

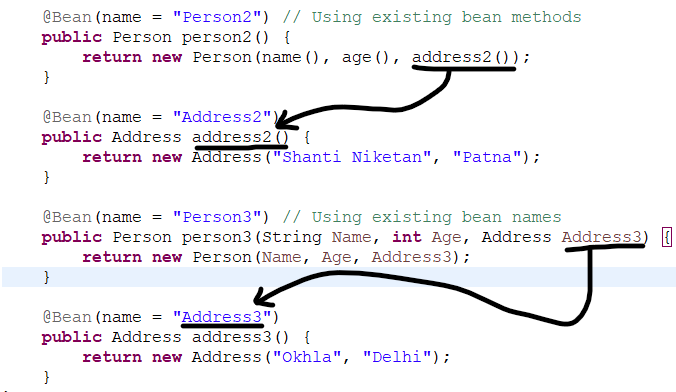
*org.springframework.context.annotation.Bean*

* + *record keyword is used in JDK 16 to create a class with getter, setter, constructors, toString() automatically.*
  + In main class we can access beans 🡪 *<ContextObj>.getBean(<Bean\_Name>); or either by using type or className. 🡪 @Bean(name = “<CustomName>”)*





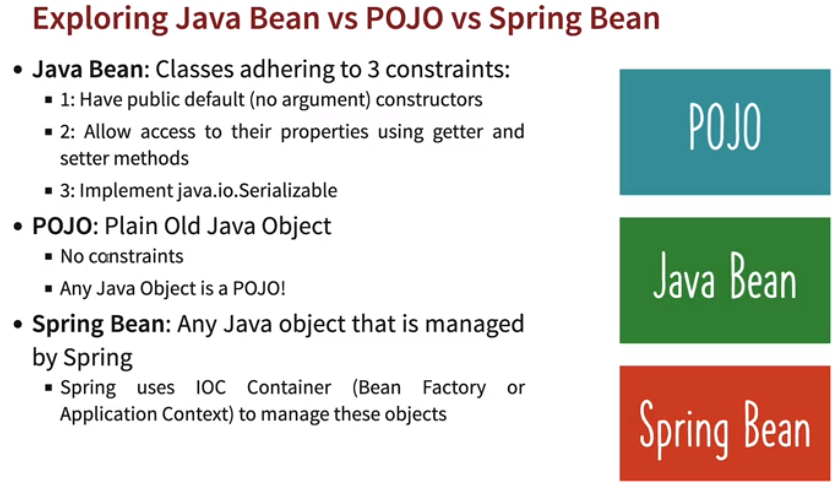




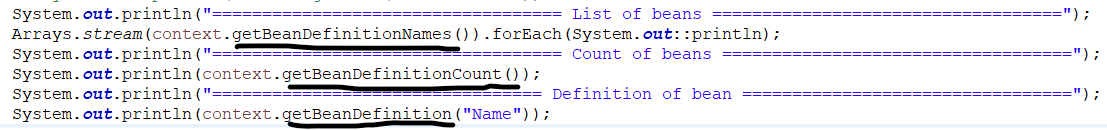
* ***Spring Container:*** *Takes input like class, beans etc and provides a runtime environment for them.* *The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction.*



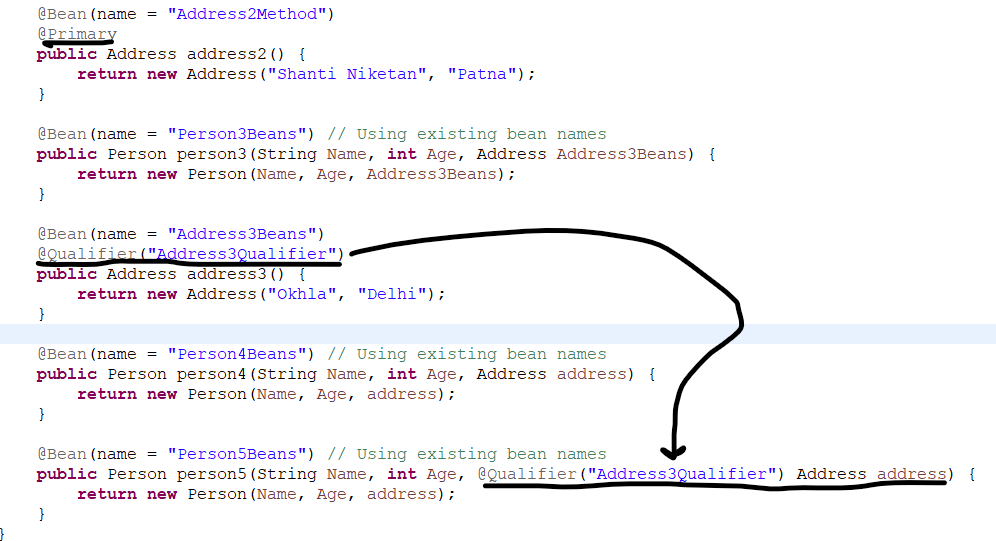
* ***Pojo:*** *Plain Old Java Object: Any object in Java is POJO.*
* ***EJB:*** *Enterprise Java bean: It consists of no-arg constructor,* *getter, setters and implements serializable (serializable has no methods in it, converts to stream).*
* ***Spring bean:*** *The objects that form the backbone of your application and that are managed by the Spring container.*



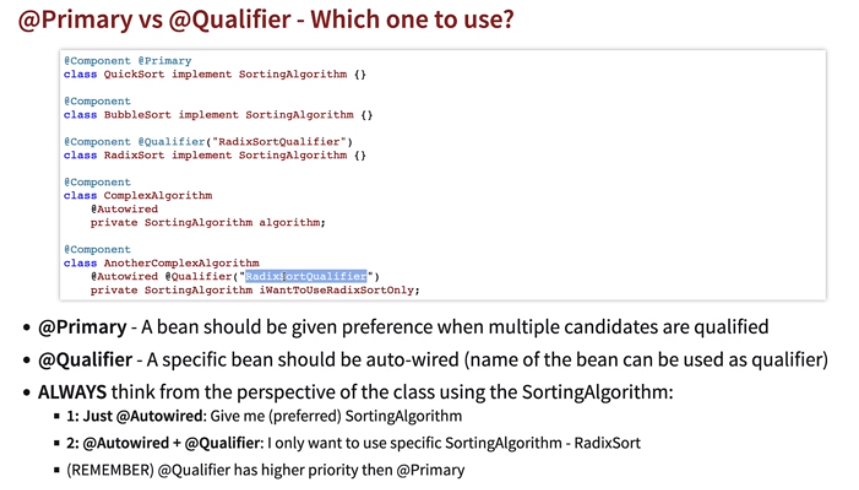
* ***How to get list of Beans in a spring application?***
  + *context.getBeanDefinitionNames 🡪 List all bean names.*
  + *context.getBeanDefinitionCount 🡪 Get total count of beans.*
  + *context.getBeanDefinition 🡪 Gives Bean definition.*



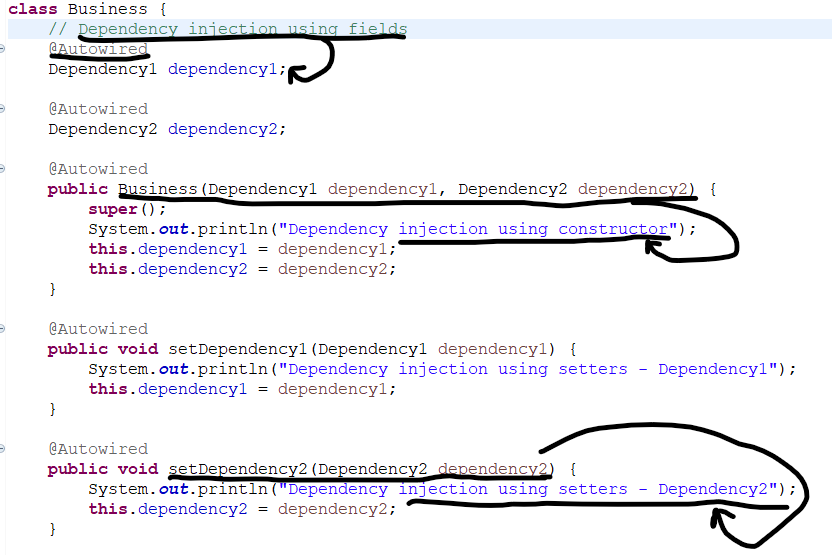
* ***How to prioritize any specific bean when there’s matching beans present?***
  + *By adding Qualifier (@Qualifier(<Name>)): Autowired to bean with Qualifier name.*
  + *By annotating beans as Primary (@Primary): Autowired to 1 bean which is primary.*



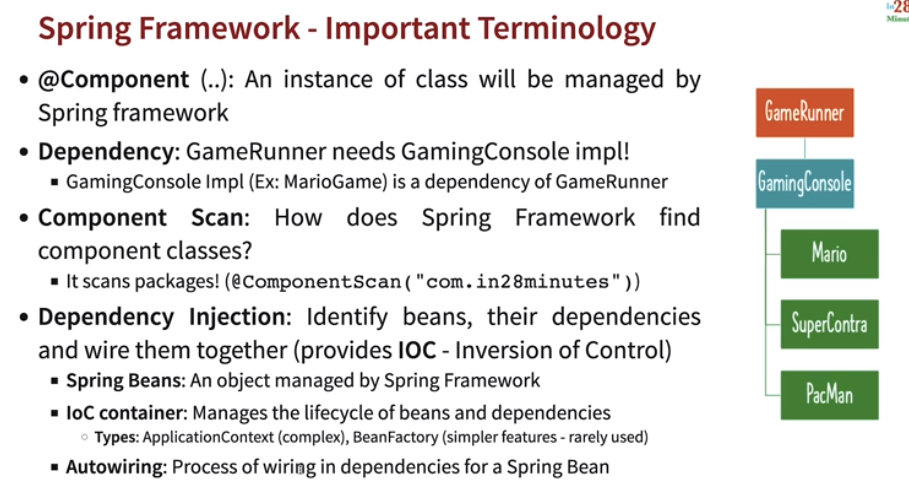
* ***Component and Component Scan:*** 
  + *@Component 🡪 It creates an instance/beans of the class via spring.*
  + *@ComponentScan(<Pckg\_name>) 🡪 It scans the components present in a package. If package name removed, it will scan current package.*
* ***Primary:*** *@Primary 🡪 It gives preference to beans when there are multiple candidates.*
* ***Qualifier:*** @*Qualifier 🡪 It is used to differentiate a bean among the same type of bean objects. Qualifier has higher priority over primary if added.*



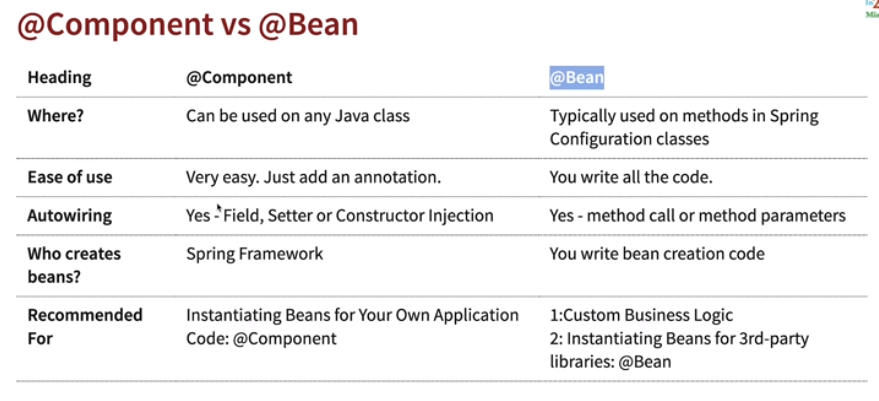
* <https://github.com/in28minutes/java-a-course-for-beginners/tree/master>
* ***Autowired:*** *Process of wiring dependencies for a Spring bean.*
  + *@Autowired 🡪 Creates dependency injection using field injection implicitly.*
* ***Dependency Injection:*** *Identifies beans and their dependencies and auto wire them is called dependency injection.*
  + ***Constructor based:*** *Dependencies are set by creating beans using constructors. Need not to specify @Autowired to inject dependency as it will auto inject, since constructor will be called for the configuration class.*
  + ***Setter based:*** *Dependencies are set by calling setter methods in beans.*
  + ***Field:*** *No constructor, setters. Dependencies injected using fields.*



* ***IOC:*** *Inversion of control, as in previous java programs programmers needed to create objects, now it is autowired using Spring and control is shifting from programmer to Spring. This is IOC.*



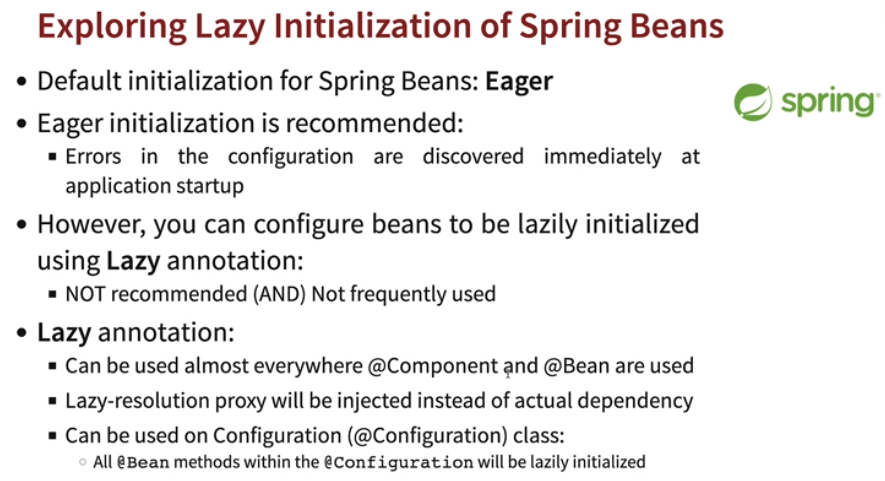
* ***@Component vs @Bean:***

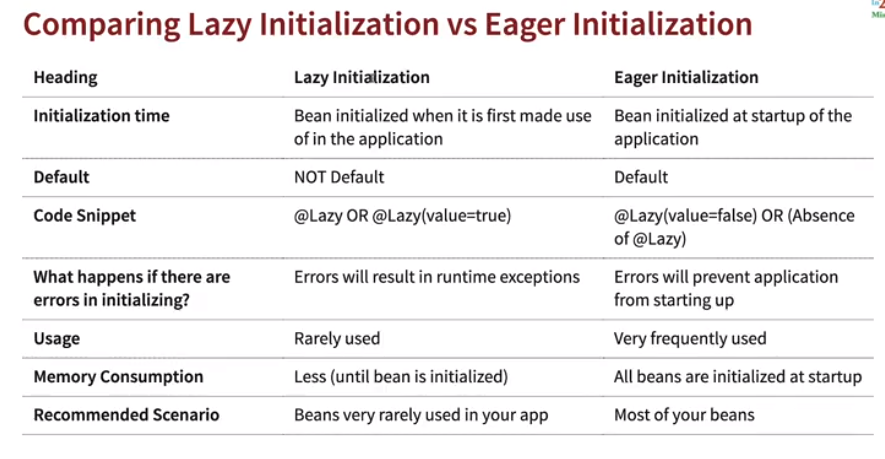


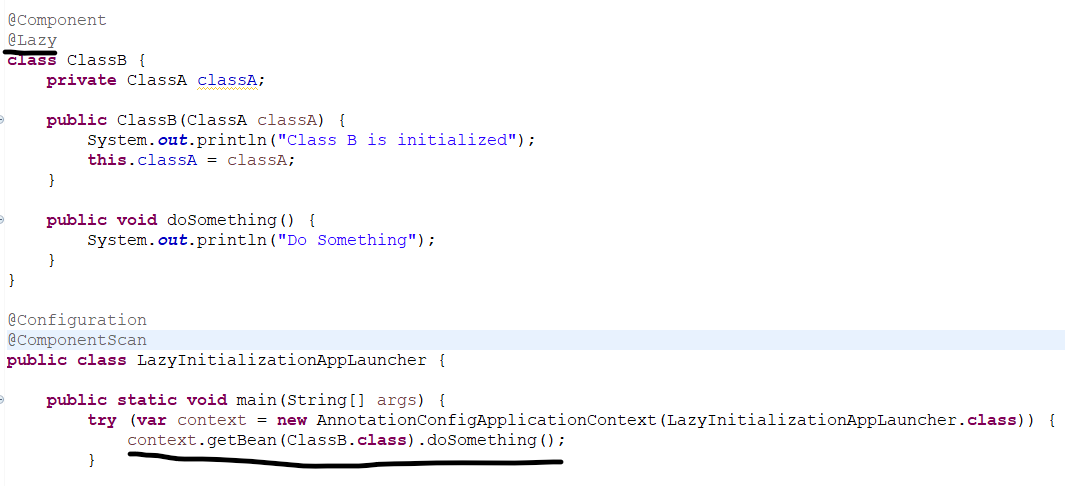
* ***Why do we need dependencies in real world?***



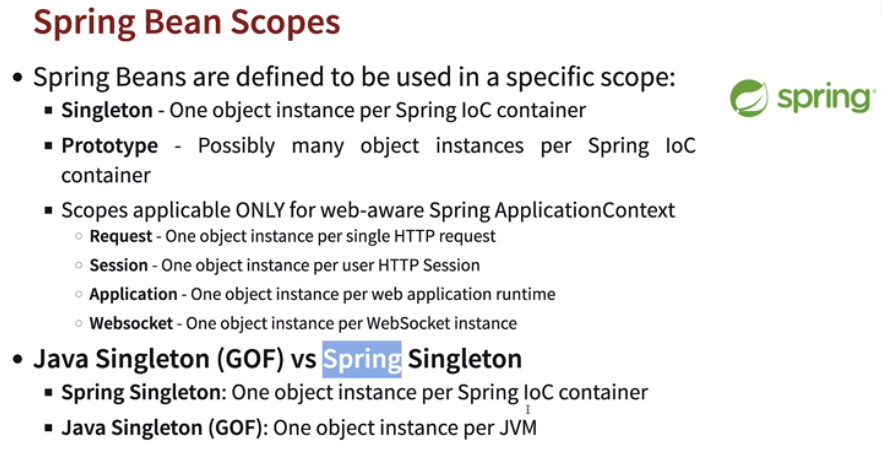
* *Process of creating simple Spring application:*
  + *First create an interface having common methods and implement the methods in classes which implements the interface.*
  + *Create a service class where we instantiate the interface and create a constructor of interface and dependency inject and implement business logic.*
  + *Create a configuration class and object of AnnotationConfigApplicationContext(). Run service class method context.getBean(<Implclass>).<method\_call> to run the application.*
* ***Lazy and Eager:*** *Spring initializes all classes by default (eagerly). In case we need to initialize a class only when used we can annotate that class as @Lazy. Lazy improves speed, but is not recommended since we would not be able to identify if there are issue in program at start. Can be used on @Component and @Bean.*

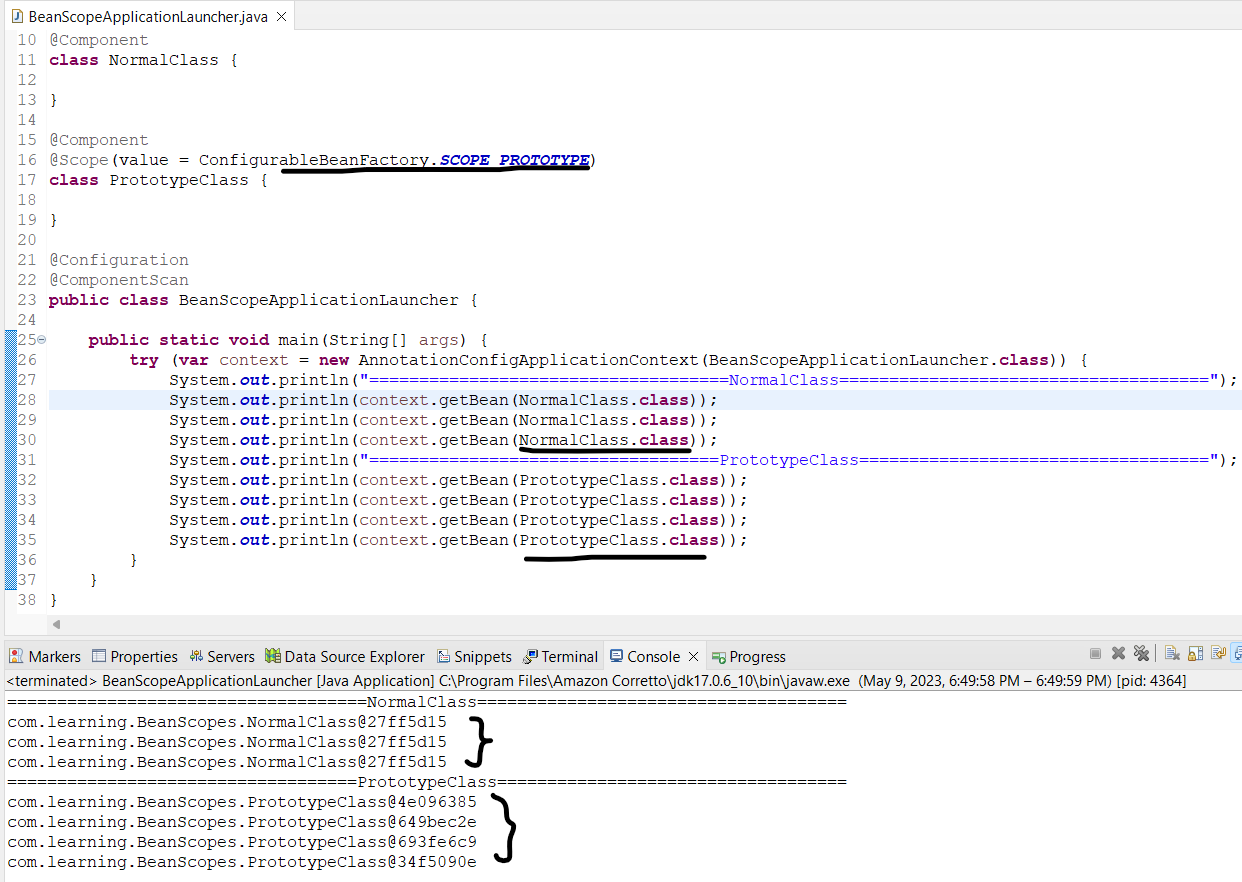




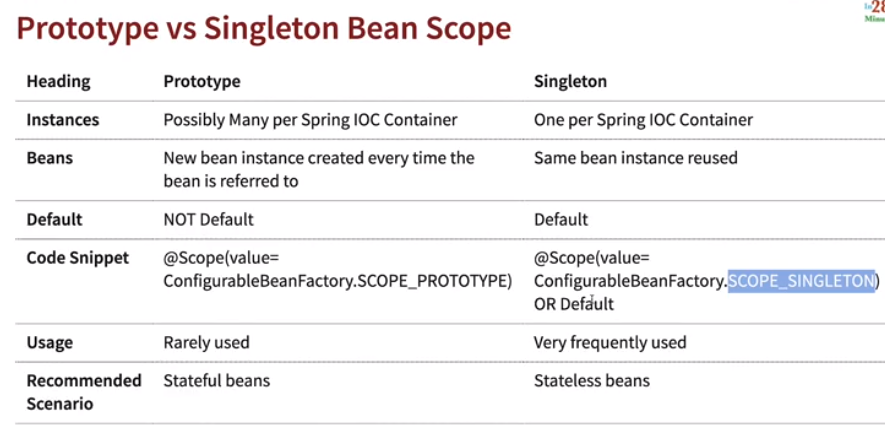


* ***Spring Bean Scopes:*** *Defines for a component how many instances will be created.* 
  + *@Scope(value=ConfigurableBeanFactory.****SCOPE\_SINGLETON****)*
  + *@Scope(value=ConfigurableBeanFactory.****SCOPE\_PROTOTYPE****)*
  + ***Singleton:*** *By default it is singleton, which means even if we call a bean multiple times, it will create only one instance of it and refer to that or reused.*
  + ***Prototype:*** *Prototype which means which means even if we call a bean multiple times, it will create new instances of it and refer to that.*
* ***Java Singleton vs Spring Singleton:*** *Java singleton will create one instance in a JVM, Spring singleton will create one instance in Spring container, which means in a single JVM we can have multiple Spring containers, so Spring singleton will creates instances in each Spring containers.*

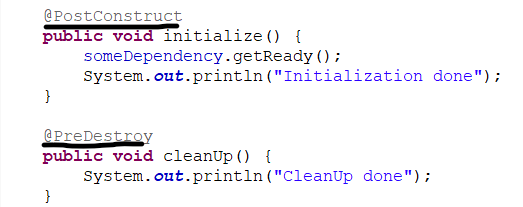


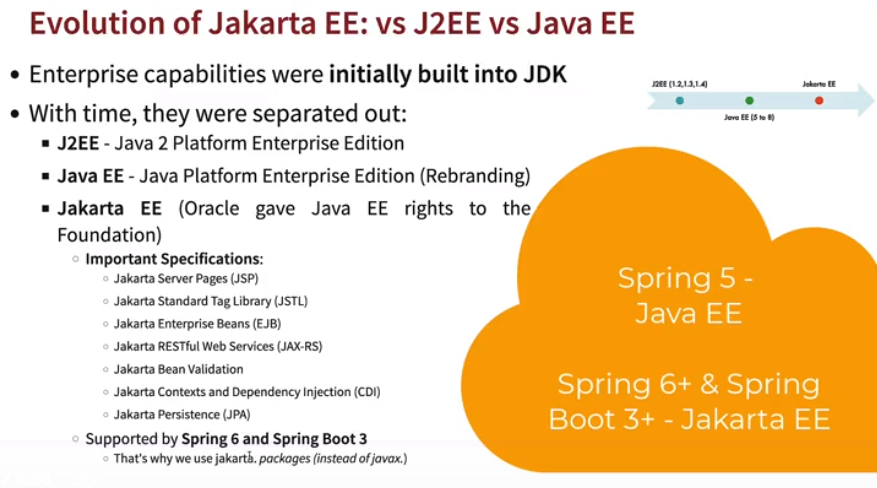


* ***When to use Singleton and Prototype and differences between them?***
  + In stateful beans we use prototype for example storing user information.
  + In stateless beans we use singleton where we do not need to store any information.

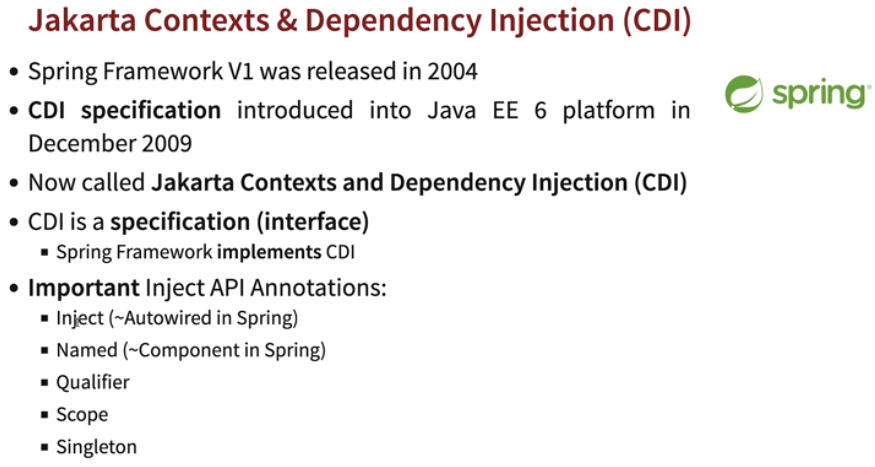


* ***@PostConstruct:*** *Spring calls the methods annotated with @PostConstruct only once, just after the initialization of bean properties and wiring dependencies. Ex: Initializing connections for DB.*
* ***@PreDestroy:*** *A method annotated with @PreDestroy runs only once, just before Spring removes bean from the application context. Ex: Closing connections from DB.*

****

****

* ***Jakarta:*** *The Jakarta EE is an open-source framework for developing web applications in Java or, as currently stated, cloud-native Java. Previously known as Java EE (Java Enterprise Edition), developed and maintained by Oracle through JCP (Java Community Process).**It is similar to Spring CDI, instead in place of @Component we use @Named and for @Autowired we use @Inject. It also supports CDI annotations.*

****

****

* ***ClassPathXmlApplicationContext:*** *ClassPathXmlApplicationContext is used to launch configurations from XML configuration file located in resources folder.*
* ***Launching XML Configuration:***
  + Create a XML configuration file in /src/main/resources folder.
  + In XML file created add basic Spring framework beans. Paste below lines in the XML file created.

*<?xml version="1.0" encoding="UTF-8"?>*

*<beans xmlns="http://www.springframework.org/schema/beans"*

*xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"*

*xmlns:context="http://www.springframework.org/schema/context"*

*xsi:schemaLocation="http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.0.xsd">*

*<!-- bean definitions here -->*

*</beans>*

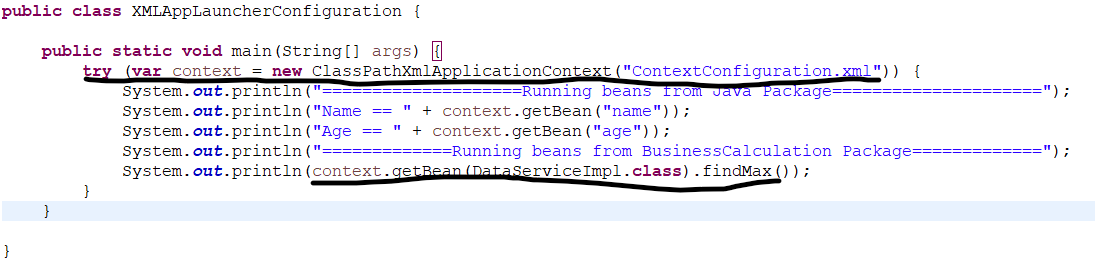
* *Adding bean definitions, you can add beans using <bean> tags. If we need to pass any value or initialize construct we can use <constructor-arg (value for variables/ref for class as an arguments)>.*

**

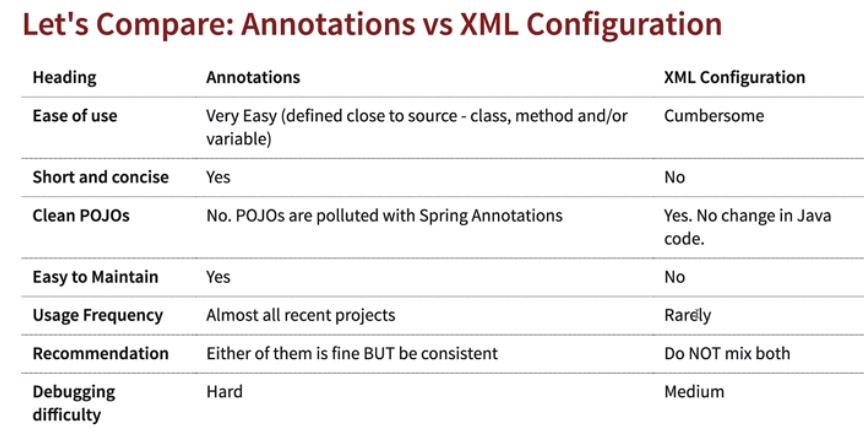
* *Go to Main Application launching class and add*

*var ctx = new ClassPathXmlApplicationContext("<XML\_File>")*

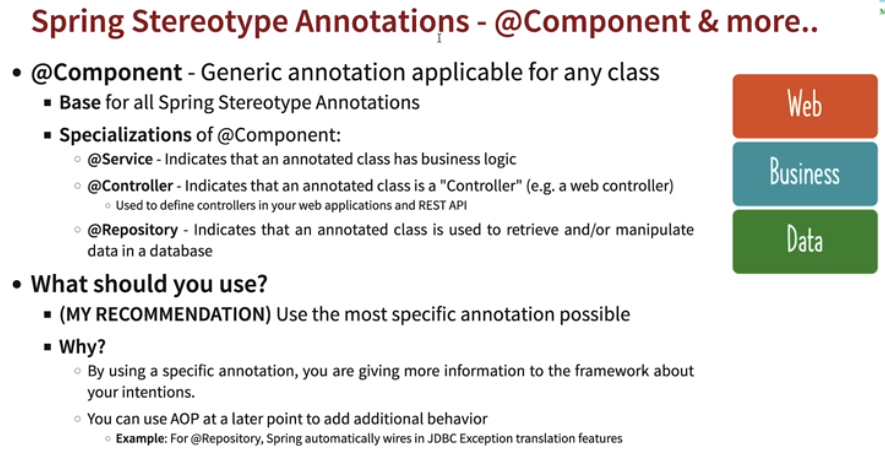
* *Run beans using ctx.getBeans(“<bean\_name>”);.*

**

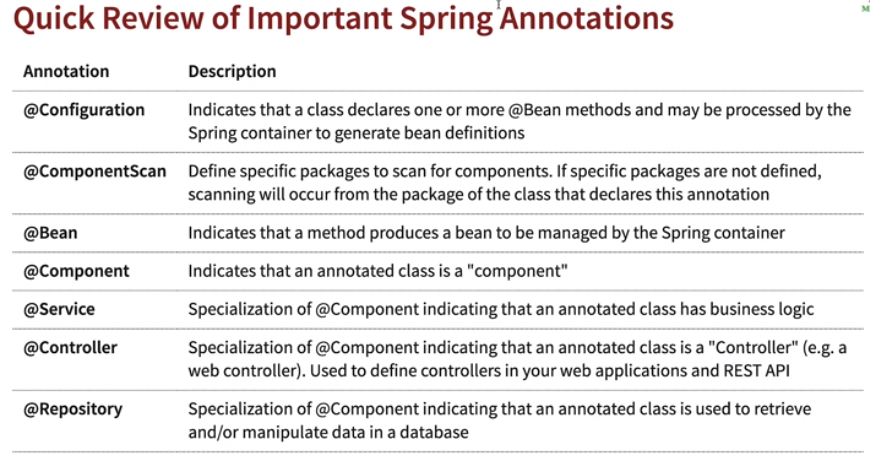
* ***Annotations vs XML configuration:***

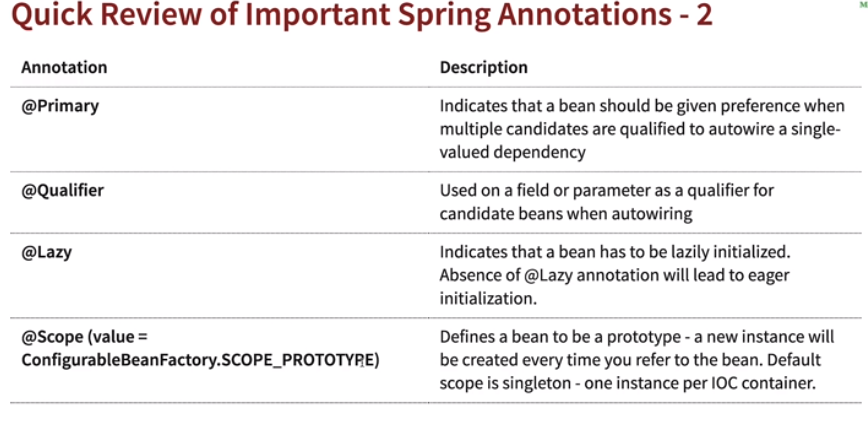
**

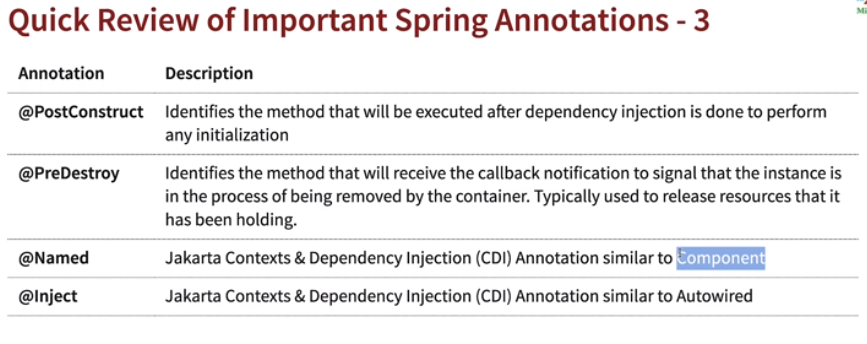
* ***@Component:*** *Generic annotation for any class.*
  + ***Specializations of @Component (Stereotype annotations)***
    - ***@Service:*** *Indicates that annotated class has business logic.*
    - ***@Controller:*** *Indicates that annotated class has Web controller.*
    - ***@Repository:*** *Indicates that annotated class is used to retrieve/manipulate data in database.*

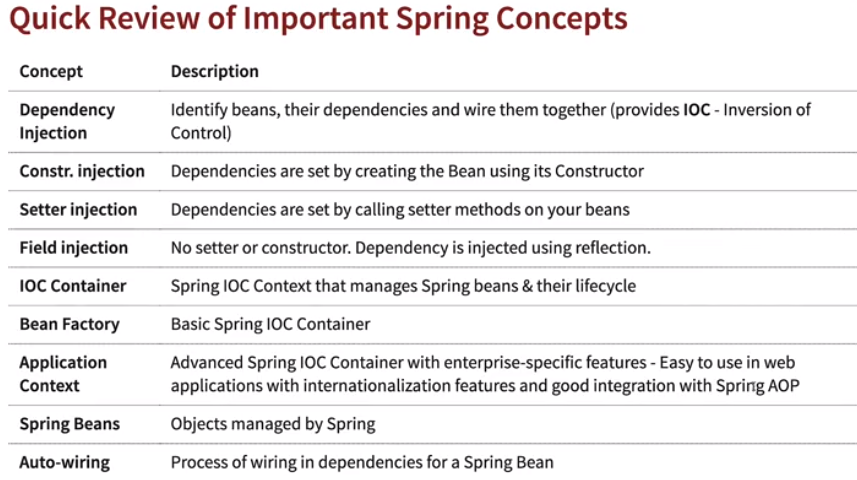
**

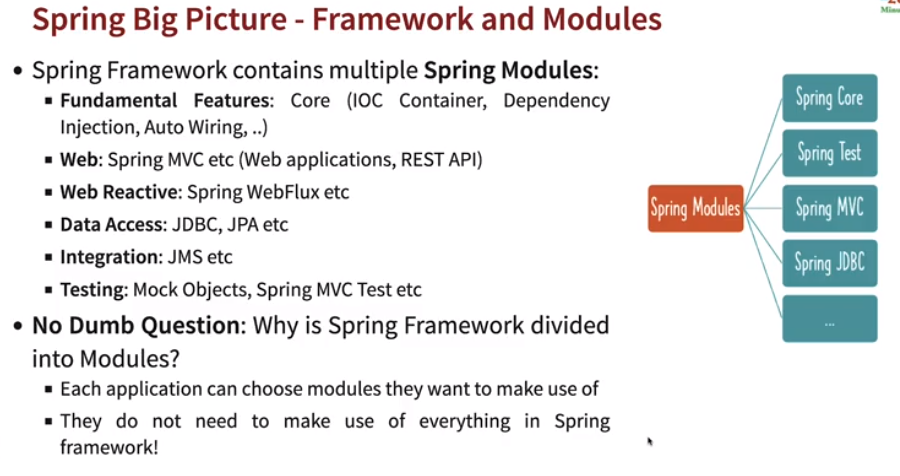
* ***Important Annotations:***

**

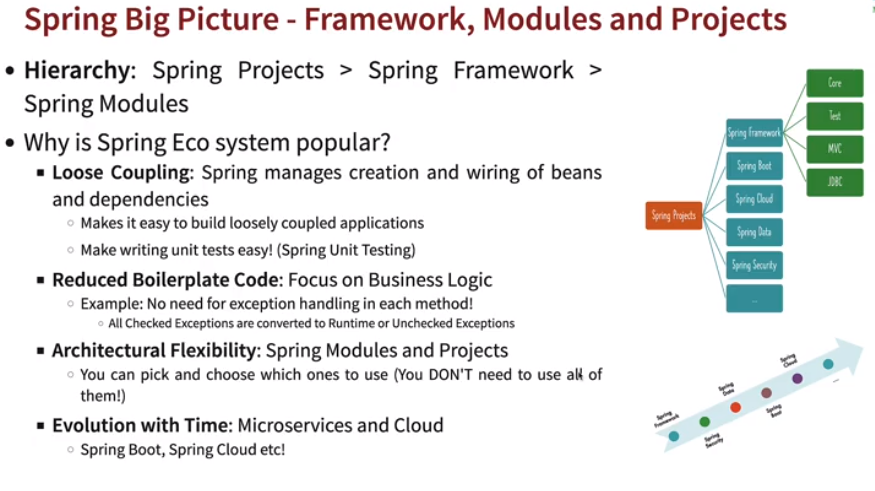
**

**

**

**

**

**

**SPRING BOOT**