1. **Tightly coupled java code (Iteration 1)**

GameRunner class

Game classes: Mario,SuperContra, Pacman

**AppGamingBasicJava.java**

import com.supSpring.learnSpring.game.GameRunner;

import com.supSpring.learnSpring.game.MarioGame;

public class AppGamingBasicJava

{

public static void main(String[] args) {

var game =new MarioGame();

var gameRunner= new GameRunner(game);

gameRunner.run();

}

}

**GameRunner.java**

package com.supSpring.learnSpring.game;

public class GameRunner {

private MarioGame game;

public GameRunner(MarioGame game) {

this.game=game;

}

public void run() {

System.*out*.println("Running game");

game.up(); game.down(); game.left(); game.right();

}

}

**MarioGame.java**

package com.supSpring.learnSpring.game;

public class MarioGame {

public void up() {

System.out.println("up");

}

public void down() {

System.out.println("down");

}

public void left() {

System.out.println("left");

}

public void right() {

System.out.println("right");

} }

**Coupling:**  How much work is involved in changing something?

**Iteration 2: Loose Coupling – Interfaces**

GameRunner class

GamingConsole Interface

Game classes: Mario,SuperContra, Pacman

**AppGamingBasicJava.java**

package com.supSpring.learnSpring;

import com.supSpring.learnSpring.game.GameRunner;

import com.supSpring.learnSpring.game.MarioGame;

public class AppGamingBasicJava {

public static void main(String[] args) {

//var game= new SuperContra();

var game =new MarioGame();

var gameRunner= new GameRunner(game);

gameRunner.run();

}

}

**GameRunner.java**

**package** com.supSpring.learnSpring.game;

**public** **class** GameRunner {

**private** GamingConsole game;

**public** GameRunner(GamingConsole game) {

**this**.game=game;

}

**public** **void** run() {

System.***out***.println("Running game");

game.up();

game.down();

game.left();

game.right();

}

}

**GamingConsole.java**

**package** com.supSpring.learnSpring.game;

**public** **interface** GamingConsole {

**void** up();

**void** down();

**void** right();

**void** left();

}

**MarioGame.java**

**package** com.supSpring.learnSpring.game;

**public** **class** MarioGame **implements** GamingConsole {

**public** **void** up() {

System.***out***.println("up");

}

**public** **void** down() {

System.***out***.println("down");

}

**public** **void** left() {

System.***out***.println("left");

}

**public** **void** right() {

System.***out***.println("right");

}

}

**SuperContraGame.java**

**package** com.supSpring.learnSpring.game;

**public** **class** MarioGame **implements** GamingConsole {

**public** **void** up() {

System.***out***.println("up");

}

**public** **void** down() {

System.***out***.println("down");

}

**public** **void** left() {

System.***out***.println("left");

}

**public** **void** right() {

System.***out***.println("right");

}

}

**NOTE:LAUNCHING SPRING CONTEXT**

**HelloWorldConfiguration.java**

**package** com.supSpring.learnSpring;

**import** org.springframework.context.annotation.Configuration;

@Configuration

**public** **class** HelloWorldConfiguration {

@Bean

**public** String name() {

**return** "Supreet";

}

}

**App02HelloWorldSpring.java**

package com.supSpring.learnSpring;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.supSpring.learnSpring.game.GameRunner;

import com.supSpring.learnSpring.game.MarioGame;

public class App02HelloWorldSpring {

public static void main(String[] args) {

System.out.println("hello");

//1. launch a spring context

var context = new AnnotationConfigApplicationContext(HelloWorldConfiguration.class);

//2. configure the things that we want spring framework to manage

//HelloWorldConfiguration - @Configuration

//name - @Bean

//3. Retrieving beans managed by spring

System.out.println(context.getBean("name"));//name of the method

}

}

**Note:** @Configuration on a particular class tells that contains the beans which we want to manage using spring.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|JVM

| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-

| |Spring

| | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
| | |name

| | |

**CREATING MORE JAVA SPRING BEANS**

package com.supSpring.learnSpring;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.supSpring.learnSpring.game.GameRunner;

import com.supSpring.learnSpring.game.MarioGame;

public class App02HelloWorldSpring {

public static void main(String[] args) {

System.out.println("hello");

//1. launch a spring context

var context = new AnnotationConfigApplicationContext(HelloWorldConfiguration.class);

//2. configure the things that we want spring framework to manage

//HelloWorldConfiguration - @Configuration

//name - @Bean

//3. Retrieving beans managed by spring

System.***out***.println(context.getBean("name"));//name of the method

System.***out***.println(context.getBean("age"));

//System.out.println(context.getBean("person"));

System.***out***.println(context.getBean("akh"));

System.***out***.println(context.getBean("person2MethodCall"));

System.***out***.println(context.getBean("person3Parameters"));

System.***out***.println(context.getBean(Address.**class**));//using the type //it will throw an error as two beans have the same class

//to resolve the above issue,we use the annotation @Primary

//list all beans

//System.out.println(context.getBean("personQualifier"));

//Arrays.stream(context.getBeanDefinitionNames()).forEach(System.out::println);

}

}

////////////////////////////////////

**package** com.supSpring.learnSpring;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**record** Person (String name, **int** age) { };//automatically create constructor, getter, setter

@Configuration

**public** **class** HelloWorldConfiguration {

@Bean

**public** String name() {

**return** "Supreet";

}

@Bean

**public** **int** age() {

**return** 10;

}

@Bean

**public** Address address() {

**return** **new** Address("pqr", "sty");

}

@Bean(name="address3")

@Primary

**public** Address addres() {

**return** **new** Address("3", "3");

}

@Bean (name = "akh")

**public** Person person() {

**return** **new** Person("Akhil",27, **new** Address("abc","def"));

}

@Bean

**public** Person person2MethodCall() {

**return** **new** Person(name(),age(), address());

}

@Bean

**public** Person person3Parameters(String name,**int** age, Address address3) {

**return** **new** Person(name, age, address3);

}

}

**NOTE: Spring Container:** Manages spring beans and their lifecycle.

**Bean Factory:** Basic Spring container

**Spring bean:** Anything that is managed by spring framework. Spring uses IOC Container to manage these objects

**NOW SOME STRUCTURAL CHANGES ARE THERE …..AND WE SURROUNDED THE CONTEXT WITH TRY AND CATCH.**

**NOW APPLY THE ABOVE APPROACH FOR OUR GAME.**

**App3GamingSpringBeans.java**

package com.supSpring.learnSpring;

import org.springframework.beans.BeansException;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.supSpring.learnSpring.game.GameRunner;

import com.supSpring.learnSpring.game.GamingConsole;

import com.supSpring.learnSpring.game.MarioGame;

import com.supSpring.learnSpring.helloworld.HelloWorldConfiguration;

public class App3GamingSpringBeans {

public static void main(String[] args) {

try (var context = new AnnotationConfigApplicationContext(GamingConfiguration.class)) {

context.getBean(GamingConsole.class).up();

context.getBean(GameRunner.class).run();

}}}

**GamingConfiguration.java**

package com.supSpring.learnSpring;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import com.supSpring.learnSpring.game.GameRunner;

import com.supSpring.learnSpring.game.GamingConsole;

import com.supSpring.learnSpring.game.PacManGame;

@Configuration

public class GamingConfiguration {

@Bean

public GamingConsole game() {

var game = new PacManGame();

return game;

}

@Bean

public GameRunner gameRunner(GamingConsole game) { //cretaing pacman game and wiring it to the gamerunner

var gameRunner = new GameRunner(game);

return gameRunner;

}}

**GameRunner.java**

**package** com.supSpring.learnSpring.game;

**public** **class** GameRunner {

**private** GamingConsole game;

**public** GameRunner(GamingConsole game) {

**this**.game=game;

}

**public** **void** run() {

System.***out***.println("Running game");

game.up();

game.down();

game.left();

game.right();

}

}

**GamingConsole.java**

**package** com.supSpring.learnSpring.game;

**public** **interface** GamingConsole {

**void** up();

**void** down();

**void** right();

**void** left();

}

**PacManGame.java**

package com.supSpring.learnSpring.game;

public class PacManGame implements GamingConsole {

public void up() {

System.*out*.println("upp");

}

public void down() {

System.*out*.println("downp");

}

public void left() {

System.*out*.println("leftp");

}

public void right() {

System.*out*.println("rightp");

}

}

**VIDEO-37 (CREATED NEW PACKAGE learnSpring-2)**

**In App3GamingSpringBeans.java, we have merged the configuration class**

**package** com.supSpring.learnSpring;

**import** org.springframework.beans.BeansException;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** com.supSpring.learnSpring.game.GameRunner;

**import** com.supSpring.learnSpring.game.GamingConsole;

**import** com.supSpring.learnSpring.game.MarioGame;

**import** com.supSpring.learnSpring.game.PacManGame;

@Configuration

**public** **class** App3GamingSpringBeans {

@Bean

**public** GamingConsole game() {

**var** game = **new** PacManGame();

**return** game;

}

@Bean

**public** GameRunner gameRunner(GamingConsole game) { //cretaing pacman game and wiring it to the gamerunner

**var** gameRunner = **new** GameRunner(game);

**return** gameRunner;

}

**public** **static** **void** main(String[] args) {

**try** (**var** context = **new** AnnotationConfigApplicationContext( App3GamingSpringBeans.**class**)) {

context.getBean(GamingConsole.**class**).up();

context.getBean(GameRunner.**class**).run();

}

}

}

NOTE: Rest all the classes(Gamerunner,gamingConsole,MarioGame,PacmanGme,SuperContra) in com.supSpring.learnSpring.game remains same.

->In the above code, we have created the bean, now lets know how spring framework will create the bean for us.Lets say spring to create the pacman game for us(to create the instance of the specific class) .we have to remove the following code below:

@Bean

**public** GamingConsole game() {

**var** game = **new** PacManGame();

**return** game;

}

We can say the spring by writing the @Component above the PacMan game class.

We have to tell the spring where to search a pacman game by using **@ComponentScan().** In bracket, we have to give the name of the package where the pacman game is present.

**App3GamingSpringBeans.java**

**package** com.supSpring.learnSpring;

**import** org.springframework.beans.BeansException;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.ComponentScan;

**import** org.springframework.context.annotation.Configuration;

**import** com.supSpring.learnSpring.game.GameRunner;

**import** com.supSpring.learnSpring.game.GamingConsole;

**import** com.supSpring.learnSpring.game.MarioGame;

**import** com.supSpring.learnSpring.game.PacManGame;

@Configuration

@ComponentScan("com.supSpring.learnSpring.game")

**public** **class** App3GamingSpringBeans {

@Bean

**public** GameRunner gameRunner(GamingConsole game) { //creating pacman game and wiring it to the gamerunner

**var** gameRunner = **new** GameRunner(game);

**return** gameRunner;

}

**public** **static** **void** main(String[] args) {

**try** (**var** context = **new** AnnotationConfigApplicationContext( App3GamingSpringBeans.**class**)) {

context.getBean(GamingConsole.**class**).up();

context.getBean(GameRunner.**class**).run();

}

}}

**PacMangame.java**

**package** com.supSpring.learnSpring.game;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** PacManGame **implements** GamingConsole {

**public** **void** up() {System.***out***.println("upp");}

**public** **void** down() {System.***out***.println("downp");}

**public** **void** left() {

System.***out***.println("leftp");

}

**public** **void** right() {

System.***out***.println("rightp");

}}

**NOTE: now we want spring to create the following code of gamerunner for us:**

@Bean

**public** GameRunner gameRunner(GamingConsole game) { //creating pacman game and wiring it to the gamerunner

**var** gameRunner = **new** GameRunner(game);

**return** gameRunner;}

**Solution: We can remove the above code and write the @Component above the GameRunner class.**

Now the gameRunner class is like as follows:

**package** com.supSpring.learnSpring;

**import** org.springframework.beans.BeansException;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.ComponentScan;

@Configuration

@ComponentScan("com.supSpring.learnSpring.game")

**public** **class** App3GamingSpringBeans {

**public** **static** **void** main(String[] args) {

**try** (**var** context = **new** AnnotationConfigApplicationContext( App3GamingSpringBeans.**class**)) {

context.getBean(GamingConsole.**class**).up();

context.getBean(GameRunner.**class**).run();

}

}

}

Note:Changing the name of App3GamingSpringBeans to GamingAppLauncherAplication

Code review: (40)

->Now, if we add the @Component above the mario game class, it will throw the error as for the gamerunner,we want an implementation of gaming console. spring finds 2 implementation of it(pacman and mario ). So ,to distinguish between these two, we can use @Primary or @Qualifier

->inspite of @primary is there in MarioGame,we want to run the supercontra game. To achieve this, we can use the @Qualifier in supercontra. We have to change in gamerunner and superContraGame class.

**SuperContraGame.java**

**package** com.supSpring.learnSpring.game;

**import** org.springframework.beans.factory.annotation.Qualifier;

**import** org.springframework.stereotype.Component;

@Component

@Qualifier("SuperContraGameQualifier")

**public** **class** SuperContraGame **implements** GamingConsole {

**public** **void** up() {

System.***out***.println("ups");

}

**public** **void** down() {

System.***out***.println("downs");

}

**public** **void** left() {

System.***out***.println("lefts");

}

**public** **void** right() {

System.***out***.println("rights");

}

}

**GameRunner.java**

package com.supSpring.learnSpring.game;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Component;

@Component

public class GameRunner {

private GamingConsole game;

public GameRunner(@Qualifier("SuperContraGameQualifier") GamingConsole game) {

this.game=game;

}

public void run() {

System.*out*.println("Running game");

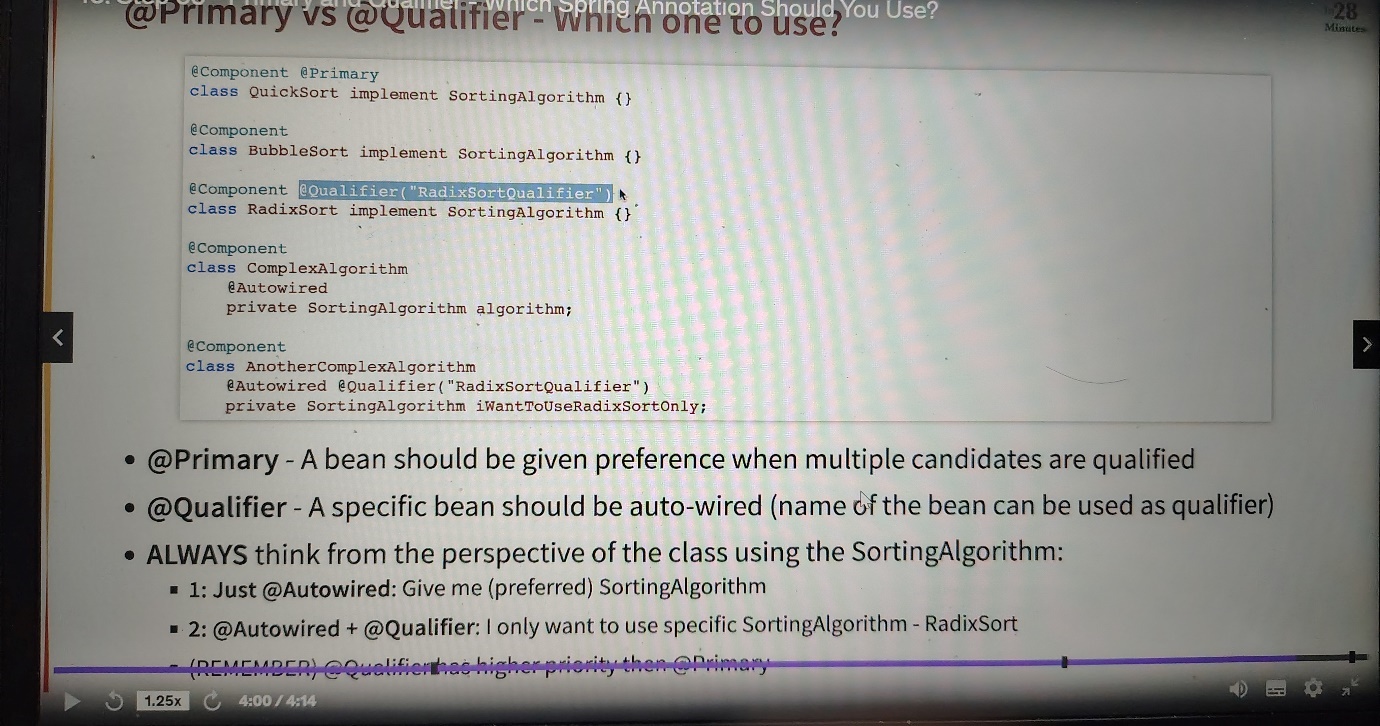
game.up();

game.down();

game.left();

game.right();

}}



**DEPENDENCY INJECTION**

**1.CONSTRUCTER-BASED :**Dependencies are set by creating the bean using its constructor

**2. SETTER-BASED:** Dependencies are set by calling setter methods on your beans

**3. FIELD:**  no setter or constructor. Dependency is injected using reflection

NOTE:CONFIGURATION CHANGE .We have put GamingAppLauncherApplication.java to The com.supSpring.learnSpring.game

And created new packages as shown below for learning dependency injection



SimpleSpringContextLauncherApplication,java

package com.supSpring.learnSpring.examples.a0;

import java.util.Arrays;

import org.springframework.beans.BeansException;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan("com.supSpring.learnSpring.examples.a1")

public class SimpleSpringContextLauncherApplication {

public static void main(String[] args) {

System.out.println("hello");

try (var context = new AnnotationConfigApplicationContext( SimpleSpringContextLauncherApplication.class)) {

Arrays.stream(context.getBeanDefinitionNames()).forEach(System.out::println);

}

}

}

**DepInjectionLauncherApplication.java**

package com.supSpring.learnSpring.examples.a1;

import java.util.Arrays;

import org.springframework.beans.BeansException;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan("com.supSpring.learnSpring.examples.a1")

public class DepInjectionLauncherApplication {

public static void main(String[] args) {

System.out.println("hello");

try (var context = new AnnotationConfigApplicationContext( DepInjectionLauncherApplication.class)) {

Arrays.stream(context.getBeanDefinitionNames()).forEach(System.out::println);

}

}

}

Now , we can explore different types of dependency ‘injection

**1.Field Injection**

package com.supSpring.learnSpring.examples.a1;

import java.util.Arrays;

import org.springframework.beans.BeansException;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.stereotype.Component;

@Component

class YourBusinessClass{

**@Autowired (when we write this ,spring will do a filed injection in this field)**

Dependency1 dependency1;

@Autowired

Dependency2 dependency2;

public String toString() {

return "Using " + dependency1 + " and " +dependency2;

}

public void z(){

System.out.println("g");

}

}

@Component

class Dependency1{

}

@Component

class Dependency2{

}

@Configuration

@ComponentScan("com.supSpring.learnSpring.examples.a1")

public class DepInjectionLauncherApplication {

public static void main(String[] args) {

System.out.println("hello");

try (var context = new AnnotationConfigApplicationContext( DepInjectionLauncherApplication.class)) {

//Arrays.stream(context.getBeanDefinitionNames()).forEach(System.out::println);

System.out.println(context.getBean(YourBusinessClass.class));

}

}

}

1. **Setter Injection**

package com.supSpring.learnSpring.examples.a1;

import java.util.Arrays;

import org.springframework.beans.BeansException;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.stereotype.Component;

@Component

class YourBusinessClass{

Dependency1 dependency1;

Dependency2 dependency2;

@Autowired

public void setDependency1(Dependency1 dependency1) {

System.out.println("Setter injection - setDependency1 ");

this.dependency1 = dependency1;

}

@Autowired

public void setDependency2(Dependency2 dependency2) {

System.out.println("Setter injection - setDependency2 ");

this.dependency2 = dependency2;

}

public String toString() {

return "Using " + dependency1 + " and " +dependency2;

}

}

@Component

class Dependency1{

}

@Component

class Dependency2{

}

@Configuration

@ComponentScan("com.supSpring.learnSpring.examples.a1")

public class DepInjectionLauncherApplication {

public static void main(String[] args) {

System.out.println("hello");

try (var context = new AnnotationConfigApplicationContext( DepInjectionLauncherApplication.class)) {

//Arrays.stream(context.getBeanDefinitionNames()).forEach(System.out::println);

System.out.println(context.getBean(YourBusinessClass.class));

}

}

}

**3.Constructor Injection (@Autowired is not mandatory in constructor injection)**

package com.supSpring.learnSpring.examples.a1;

import java.util.Arrays;

import org.springframework.beans.BeansException;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.stereotype.Component;

@Component

class YourBusinessClass{

Dependency1 dependency1;

Dependency2 dependency2;

@Autowired

public YourBusinessClass(Dependency1 dependency1, Dependency2 dependency2) {

super();

System.out.println("constructor injection");

this.dependency1 = dependency1;

this.dependency2 = dependency2;

}

public String toString() {

return "Using " + dependency1 + " and " +dependency2;

}

public void z(){

System.out.println("g");

}

}

@Component

class Dependency1{

}

@Component

class Dependency2{

}

@Configuration

@ComponentScan("com.supSpring.learnSpring.examples.a1")

public class DepInjectionLauncherApplication {

public static void main(String[] args) {

System.out.println("hello");

try (var context = new AnnotationConfigApplicationContext( DepInjectionLauncherApplication.class)) {

//Arrays.stream(context.getBeanDefinitionNames()).forEach(System.out::println);

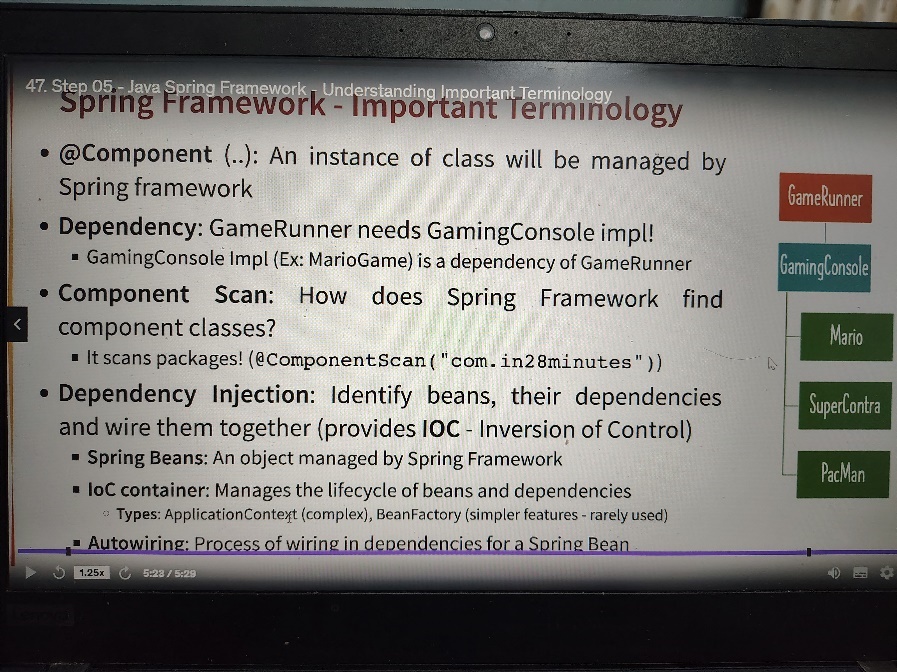
System.out.println(context.getBean(YourBusinessClass.class));

}

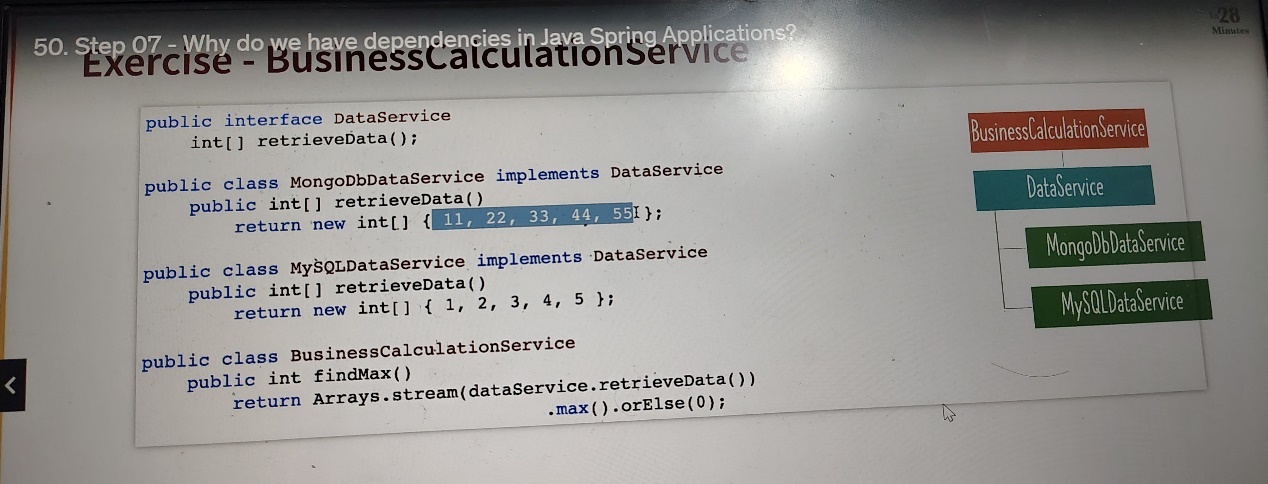
}

}

**NOTE: Spring team recommends Constructor-based injection as dependencies are automatically set when an object is created.**

****

****

**QUESTION:** ****

**Solution:**

**DataService.java**

**package** com.supSpring.learnSpring.examples.question;

**public** **interface** DataService {

**int**[] retrieveData();

}

**MongoDbDataService.java**

package com.supSpring.learnSpring.examples.question;

import org.springframework.context.annotation.Primary;

import org.springframework.stereotype.Component;

@Component

@Primary

public class MongoDbDataService implements DataService {

public int[] retrieveData() {

return new int[] {11,22,33,44,55};

}}

**MySQLDataService.java**

package com.supSpring.learnSpring.examples.question;

import org.springframework.stereotype.Component;

@Component

public class MySQLDataService implements DataService {

@Override

public int[] retrieveData() {

return new int[] {1,2,3,4,5};

}

}

**BusinessCalculationService.java**

package com.supSpring.learnSpring.examples.question;

import java.util.Arrays;

import org.springframework.stereotype.Component;

@Component

public class BusinessCalculationService {

//constructor injection

public BusinessCalculationService(DataService dataService) {

super();

this.dataService = dataService;

}

DataService dataService;

public int findMax() {

return Arrays.*stream*(dataService.retrieveData()).max().orElse(0);

}

}

**LauncherApplication.java**

**package** com.supSpring.learnSpring.examples.question;

**import** java.util.Arrays;

@Configuration

@ComponentScan

**public** **class** LauncherApplication {

**public** **static** **void** main(String[] args) {

**try** (**var** context = **new** AnnotationConfigApplicationContext(LauncherApplication.**class**)) { Arrays.*stream*(context.getBeanDefinitionNames()).forEach(System.***out***::println); System.***out***.println(context.getBean(BusinessCalculationService.**class**).findMax());

}

}

}

**NOTE: LAZYINITIALIZATION [EAGER IS DEFAULT] [Used with @Component, @Bean, @Configuration]**

package com.supSpring.learnSpring.examples.d1;

import java.util.Arrays;

import org.springframework.beans.BeansException;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Lazy;

import org.springframework.stereotype.Component;

@Component

class ClassA{

}

@Component

@Lazy

class ClassB{

private ClassA classA;

public ClassB(ClassA classA) {

//Logic

System.*out*.println("some initialization");

this.classA = classA;

}

public void doSomething() {

System.*out*.println("do something");

}

}

@Configuration

@ComponentScan("com.supSpring.learnSpring.examples.d1")

public class LazyInitializationLauncherApplication {

public static void main(String[] args) {

System.*out*.println("hello");

try (var context = new AnnotationConfigApplicationContext(LazyInitializationLauncherApplication.class)) {

//Arrays.stream(context.getBeanDefinitionNames()).forEach(System.out::println);

System.*out*.println("Initialization is completed");

context.getBean(ClassB.class).doSomething();//whenever classB is called then the bean is loaded otherwise not

}

}}

**NOTE: BEAN SCOPE**

**Singelton(default): one object instance per spring IOC Container. Same bean instance reused**

**Prototype: Possibly many object instances per spring IOC Container :new bean instance created everytime the bean is referred to**

package com.supSpring.learnSpring.examples.e1;

import org.springframework.context.annotation.ComponentScan;

@Component

class NormalClass{

}

@Scope(value=ConfigurableBeanFactory.SCOPE\_PROTOTYPE)

@Component

class PrototypeClass{

}

@Configuration

@ComponentScan("com.supSpring.learnSpring.examples.e1")

public class BeanScopesLauncherApplication {

public static void main(String[] args) {

System.out.println("hello");

try (var context = new AnnotationConfigApplicationContext(BeanScopesLauncherApplication.class)) {

//Arrays.stream(context.getBeanDefinitionNames()).forEach(System.out::println);

System.out.println(context.getBean(NormalClass.class));

System.out.println(context.getBean(NormalClass.class));

System.out.println(context.getBean(PrototypeClass.class));

System.out.println(context.getBean(PrototypeClass.class));

System.out.println(context.getBean(PrototypeClass.class));

//output

// com.supSpring.learnSpring.examples.e1.NormalClass@2462cb01

// com.supSpring.learnSpring.examples.e1.NormalClass@2462cb01

// com.supSpring.learnSpring.examples.e1.PrototypeClass@19b843ba

// com.supSpring.learnSpring.examples.e1.PrototypeClass@64ec96c6

// com.supSpring.learnSpring.examples.e1.PrototypeClass@77659b30

//we can see in normal class hashcode is same

//we get a new bean each time when the prototype class is called. we get different instance each time

}

}}

**NOTE: PreDestroy and PostConstruct**

**package** com.supSpring.learnSpring.examples.f1;

**import** java.util.Arrays;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext;

**import** org.springframework.context.annotation.ComponentScan;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.stereotype.Component;

**import** jakarta.annotation.PostConstruct;

**import** jakarta.annotation.PreDestroy;

@Component

**class** SomeClass {

**private** SomeDependency someDependency;

**public** SomeClass(SomeDependency someDependency) {

**super**();

**this**.someDependency = someDependency;

System.***out***.println("All dependencies are ready!");

}

@PostConstruct

**public** **void** initialize() {

someDependency.getReady();

}

// @PreDestroy

// public void cleanup() {

// System.out.println("Cleanup");

// }

}

@Component

**class** SomeDependency {

**public** **void** getReady() {

System.***out***.println("Some logic using SomeDependency");

}

}

@Configuration

@ComponentScan

**public** **class** PrePostAnnotationsContextLauncherApplication {

**public** **static** **void** main(String[] args) {

**try** (**var** context = **new** AnnotationConfigApplicationContext

(PrePostAnnotationsContextLauncherApplication.**class**)) {

// Arrays.stream(context.getBeanDefinitionNames())

// .forEach(System.out::println);

}}}

**NOTE: Jakarta CDI**

**<dependency>**

**<groupId>jakarta.inject</groupId>**

**<artifactId>jakarta.inject-api</artifactId>**

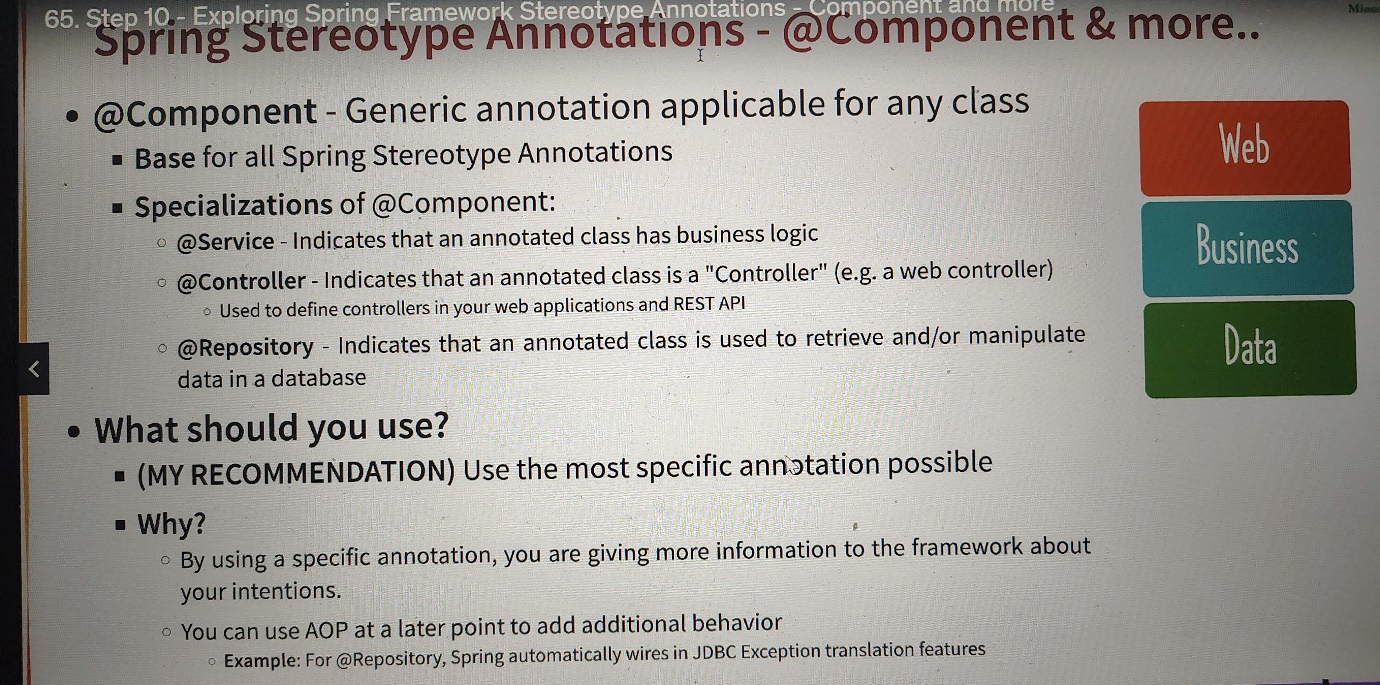
**<version>2.0.1</version>**

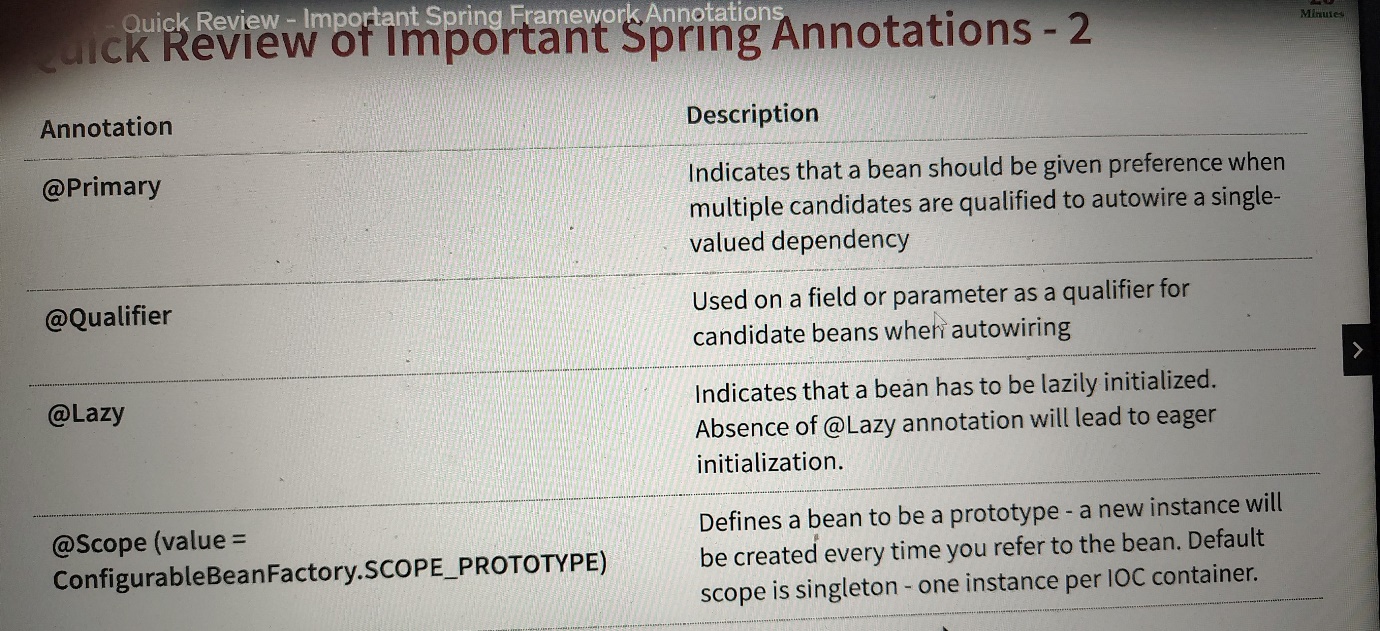
**</dependency>**

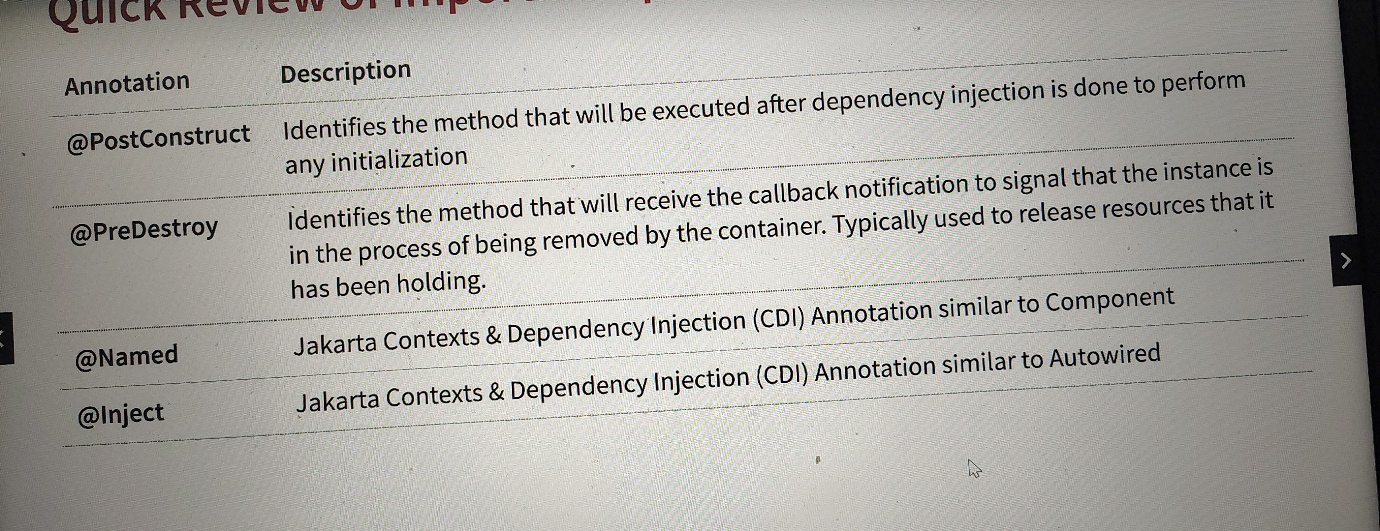
**@Component is replaced by @Named**

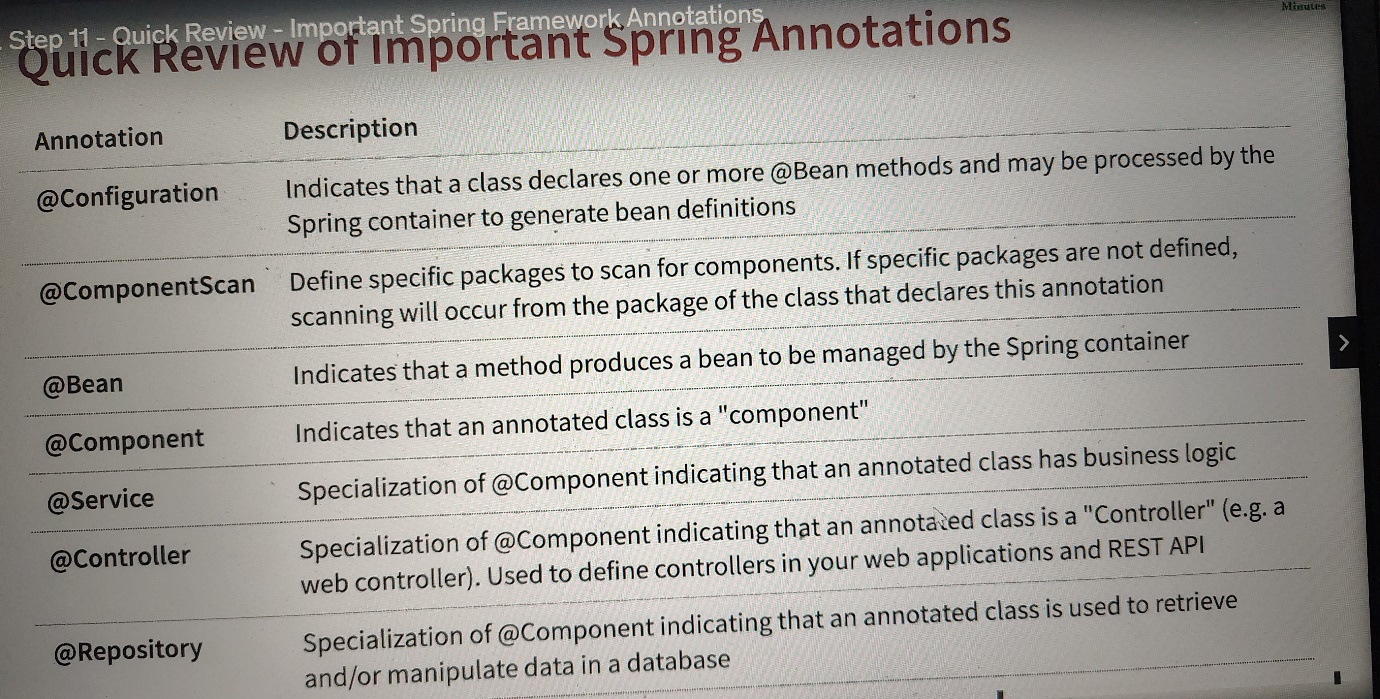
**@Autowired is replaced by @Injext**

**Spring Framework implements CDI. CDI is aa specification (interface)**

****

****

****

****

**SPRING BIG PICTURE**

**Fundamental Features:** Core

**Web**: Spring MVC

**Web Reactive**: Spring WebFlux

**Data Access**: JDBC ,JPA

**Integration**: JMS

**Testing**: Mock Objects, Spring MVC Test

Spring core, Spring Test, Spring MVC, Spring JDBC are part of **Spring Modules**.

**Spring evolves through Spring Projects:**

**First Project**: Spring Framework

**Spring Security**: Secure your web application or REST API or microservice

**Spring Data**: Integrate the same way with dierent types of databases : NoSQL and Relational **Spring Integration**: Address challenges with integration with other applications

**Spring Boot:** Popular framework to build microservices

**Spring Cloud:** Build cloud native applications