Download spring initializr as setting the Group Name com.techproed, Spring Web & Spring Boot DevTools, all the other options are default. open src.pom.xml. Biz techproed isimli projeyi indirdik ancak kendisi otomatik olarak springframework.boot isimli parent projeyi de indirmiş dedi. Inheritence kavramından bahsetti sonrasında.- parent class handles the basic features. You can work on those basic features and improve them-Spring Boot için inherit edilecek parent features- When you work with springboot projects, all projects

1. should connect to the database.
2. Create User Interface: all projects need to have user interface(eg. amazon.com is a user interface)

3) Security

This parent project has those. When you put your own springboot project, Java put those features automatically.Springboot framework is like getting franchise.(Burda çiz bişeyler) You need to choose and decide basic things such as logo, brand or color of your store unless you use franchise. If you use franchise, you don't need to think basic things. You can start to create value immediately. When you use springboot, all dependencies are created automatically. You dont need to create dependencies by opening the maven & working on sth else. Application’a tıkla In springboot, we'll use annotations.(@SpringBootApplication gibi) By using them, thing will become very easy. In springboot, **dependency injection** is very important concept and in your interviews,most probaby, they will ask you as what is the dependency injenction? or what is the version of controll IOS.

Slaytlara geçti.. **Tight Coupling**: Parts that you cannot seperate from the main(e.g. laptop) **Loose Coupling:** Parts that you can seperate from the main(e.g. laptop) Which one is better? Loose Coupling. With Loose Coupling, every company can do their own task-- laptop makers can make laptops and battery companies can make batteries. If battery is broken, you don't need to change your laptop, you can just change your battery. In maintanence and testing, loose coupling is better as well. Slayttan devam ediyor So far, when we create objects, we use “new” keyword. If you use new keyword, it means tight coupling. -- Laptop class is tightly coupled to BatteryA class. If you want to create BatteryB class, you need to either create new laptop class or new BatteryB object within this class. Thus, it is not possible to use BatteryA and BatteryB on the same class. But the thing is, springboot found a solution for this problem as using the loose coupling in the class and objects.

**TIGHT COUPLING**

Create a laptop and battery class under src/main/java/com.example.com\_techproed

12) SpringFramework send your application to your localhost automatically by TomCat. TomCat i de springboot\_demo/pom.xml de ilk dependency olarak bulabiliriz

Lets create Battery(b) and price inside the Laptop class. create getters and setters:

Let’s create the turnOnLaptop() Method(bs.use() olmadan). Come to Application. Create l1 object(Laptop l1 = new Laptop();) When you create an object by using new keywords, you create it by **tight coupling.**  Under Laptop l1, let’s call setter(l1.setB();). To call setter, I need Battery object, so let’s create the Battery Object(l1.setB(new Battery()); ) See, one more tight coupling for a tight coupling guys. Look how mess it is.. It is like matruşka. Come to Battery Class. Inside the Battery Class, let’s create constructor. (Battery() { System.out.println(“Battery is created…”) ; } Just to make it visible, I created constructor. Otherwise, when I run the code, you’ll see nothing. Application a geri dön ve şunu ekle(l1.setPrice(1200); ) Bu setter’ın ikinci kısmıydı. Then add l1.turnOnLaptop(); Everybody, o make l1 creation visible, let’s create a constructor inside the laptop and put a message in it. Come to Laptop Class a dön ve private int price’ın constructor ekle (Laptop() {System.out.println(“Laptop is created…”); }) Inside the Laptop Class, we created a constructor to make things visible on console. Run the main method(Application) and see the result from the console. If I make mistake inside the Battery Class, everything will be done since there is tight coupling over here. If you have any problem in your code inside the Battery Class, Laptop Class will close also.

**LOOSE COUPLING**

Bundan sonrası için GitHub daki ApplicationLoose a bak. Laptop l1 = deki new Laptop’u, l1.setB(new Battery()); yi, l1.setPrice(1200); ü ve l1.turnOnLaptop(); u sil. Laptop l1 ın üzerindeki run method’u güncelle(ConfigurableApplicationContext con = SpringApplication.run(Application.class, args); ) Laptop l1 ı şuna eşitle(con.getBean(Laptop.class)) In SpringBoot everybody, objects are called **Bean.** getBean= getObject demek oluyor bu durumda. In our case, I told SpringBoot that Hey SpringBoot, get an object from Laptop Class. When you typed the code like that, it’ll be loose coupling. Let’s use l1.turnOnLaptop(); method. Laptop Class a gel ve public class Laptop’un üzerine @Component’I ekle. Run the Application(yani main method). Eğer Port 8080 is already in use hatası alırsan şu kodu cmd de kullan:

netstat -ano | findstr :8080(8080 benim port numaram)

taskkill /PID 3696 /F(3696 benim listenining değerim)

veya durdur tuşuna bas. Loose Coupling example is worked without “**new”** method; why ? Because SpringBoot used the constructor on my behalf even if I haven’t use it manually. Everybody, when you create a SpringBoot Object, SpringBoot creates an object container. But the objects which are put inside the container(l1), the objects in object container are called “Bean”. Let’s create Laptop l2 Object(Laptop l2 = con.getBean(Laptop.class); ) Call l2.turnOnLaptop(); Run the main method. Can you see something weird over here ? (deyip kod çıktısının en sonundaki 6 satırı işaret ettik). When I create 2 objects(l1,l2) constructor(Çıktısı Laptop is created... olan) should run twice, but it used once. If there is an ongoing laptop object(l1), springboot goes only with existent one, doesn't create a new one -- bu iki de olsa 3 te olsa fark etmiyor. This feature is very beneficial in the manner of memory and speed(**singleton concept**: from a class, you can only have a one object). Let’s do the same with “new” keyword. l1 ve l2 için con.getBean() yerine new Laptop’u koyup dene. Run the main method See, there are 2 “Laptop is created…” messages on the console. Where is 3rd one coming from?(İkiliden önceki “Laptop is created…” mesajı) Everybody, SpringBoot is very smart. When you put Component at the top of the Laptop Class, SpringBoot thinks that you’ll need an object from. When I run the code, SpringBoot puts the object inside the container automatically even if you don’t use it. Her şeyi comment out yap. Run the main code. See, it runs the container automatically. If you put @Scope(“prototype”), Singleton Prototype will be cancelled. Everybody, using @Scope with @Component and not using @Component is the same. That’s why we don’t use @Scope with @Component. **Şimdiye kadar “@Component ve @Scope annotation” larını, “loose ve tight coupling” i ve “singleton concept”'i öğrendik.**

Let’s talk about how to make Singleton feature flexible. BatteryS isimli başka bir sınıf daha oluşturduk.Inside the BatteryS, create name and duration variables.(private String name; private int duration;) Since they are private, let’s create getter & setter for them. Create use method(public void use() {System.out.println(“BatteryS is used…”); }) Laptop'u tekrar manipüle ediyoruz. Orjinali bozmamak için Laptop3 adlı yeni bir sınıf oluşturdum ve onunla devam ediyorum. Laptop3 ün içinde price ın altına BatteryS isimli yeni bir variable oluşturduk. Bu variable için de getters & setters oluşturmamız gerekiyor ki oluştururken Insertion point'ini After 'setPrice(int)' olarak set ettik. Let’s create a constructor—duration un altına-- inside BatteryS to make it visible on Console(BatteryS(){System.out.println(“BatteryS is created”…});) Inside BatteryS, I have “use()” method. Let’s turn on Laptop by using BatteryS. Laptop/turnOnLaptop’un içine bs.use(); u ekle Everytime, if you want to craete form an any class, you need to put @Component(BatteryS in başına da @Component ekle). Laptop Class ının içindeki private BatteryS bs; in üstüne @Autowired I ekle. Run the code and see what happens. See, everything is good?How does SpringBoo understands that BatteryS and BS are the same thing? By **autowired;** autowire don't look at the names, it looks at the datatypes.

If you want to select object **by their names**, you can do it by using @Qualifier.1st ,Change the Component to Component(“bS”) from **BatteryS Class.** 2nd, add Qualifier(“bS”) above the private BatteryS bS from **Laptop Class**. In this case, first, SpringBoot changes BatteryS as bS, then goes with it. Run the main method. See, it worked. Ofc, you can see nothing on the console in the manner of name. But it worked by its name. If wouldn’t, we would see an error message.

Diagram

Description automatically generated

* Our ObjectContainer and SpringBootDemoApplication are seperate, not fixed. But they work perfectly fine. This is called **loose coupling**.
* If you do object creation in another process, this is called **inversion of control(IOC)**. As you see,in SpringBoot Object Creation is done in Object Container and the application is created in another part. This is called "Inversion of Control".
* **Dependency Injection** : When you run the application, objects come from Object Container. Object Container is outside the application, it means our application is getting sth from outside to inside, this is "Injection".Because of that, this process is called "Dependency Injection". Dependency Injection made SpringBoot pretty famous.

So far, we made dependency injection by using getters and setters.(BatteryS teki getters & setters ı göster) We can also make dependency injection by using constructors. In order to do that, let's create com.techproed.springboot\_demo2 package and lets rename com.techproed.springboot\_demo as com.techproed.springboot\_demo1. Now, lets create one more Laptop class. See, there is no runner class in the “springboot\_demo2”. Copy the “Application Class” and paste it to “springboot\_demo2” Lets create an interface name it as BatteryI. As you know, interfaces have abstract methods. Let’s put abstract method inside the Abstract Battery(public void use(); Hata verirse sağ klik context actionslardan çöz). Lets create BatteryA and BatteryB Classes. Inside the BatteryA, add implements BatteryI. BatteryA is concrate class, BatteryI is interface. As you know, when a concrate class implements interface, it should override unimplemented method which is “use”. Inside the “use” method, I’ll put the following message(“BatteryA is used…”)Create public int price; inside the BatteryA and BatteryB. We’ll do the same for BatteryB as well. Aynı şeyleri BatteryB için de yap. I’ll inject “BatteryA” and “BatteryB” into my application under the Laptop. But as you see, over here, I used interface. Come to Laptop Class. Create BatteryI b; variable To make the constructor visible, I ‘ll put constructor inside Laptop,BatteryA and BatteryB. Come to Laptop Class. After BatteryI b; type Laptop(){System.out.println(“Laptop is created…”); } Aynı kodu Class isimlerini değiştirerek(Laptop yerine BatteryA & BatteryB olacak şekilde) BatteryA ve BatteryB için de uygula. If you don’t put, your code will works but you won’t see whether the Laptop Object works or not. Let's create public void use(){b.use();} and public void turnOnLaptop() {System.out.println(“Laptop is turned on…”); } methods. After completing Laptop Class, please come to BatteryA and BatteryB, put the "price" variable(public int price;) Let's create l1 object for Application Class(con.getBean(Laptop.class); ) Dont forget to put the @Component annotation in Laptop class, on the top of the public class Laptop .Run the code(varolan kodu durdurmayı unutma)

If I create 2 more objects, how many messages that I'll see on the console? Run the code. Just 1 due to Singleton. If I create any object, will I see any message? l1, l2 and l3 yi comment out yap ve Run the code- When I use Component annotation, SpringBoot thinks that I'll use the object so it makes ready for me.

Let's create a use method inside the Laptop class. // Which use method is going to be used ? The one within the BatteryA or the one within the BatteryB? Let's run the code and see from the console. It gave me an error, why ? Because I have to declare which use method that I am going to use. Laptop class'ını açtık ve private BatteryA bA yı tanımladık. bA.use() u da çalıştırdık. Don't forget to put @Component for BatteryA. Don't forget to use @Autowired annotation inside the Laptop Class, under the private BatteryI bA; That’s it for today guys, see you tw.