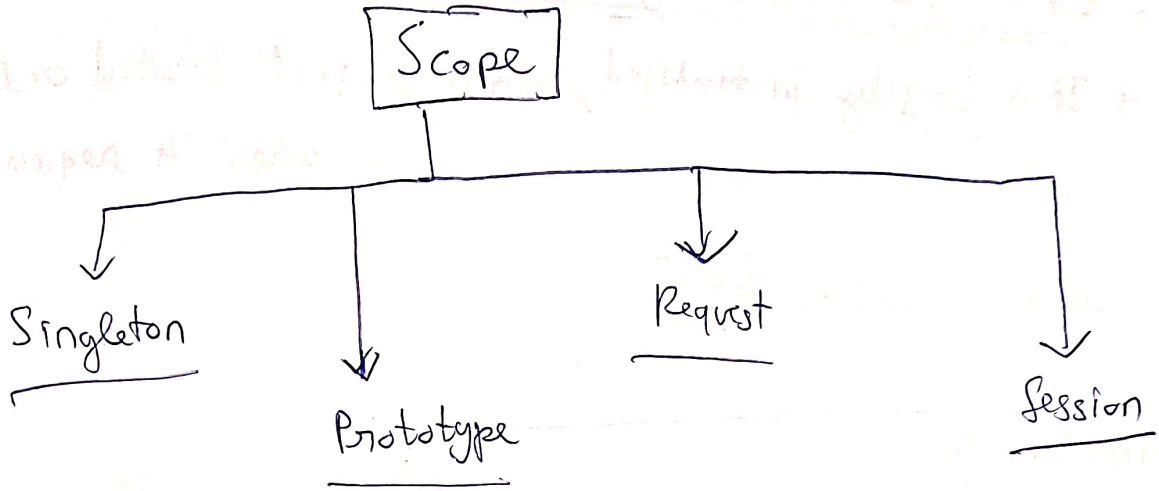
# Video 07. Spring Boot : Bean Scopes | Singleton, Prototype, Request, Session Scopes




## Singleton (1 Object created per IOC)

- Default Scope
- Only 1 instance created per IOC
- Eagerly initialised by IOC [means gets initialised at the time of Application startup].

```
@RestController
@RequestMapping("/api")
@Scope(value = ConfigurableBeanFactory.SCOPE_SINGLETON)
public class TestController2{
}
```

[Only 1 Object is created per IOC]

```
@Component
@Scope("singleton")
public class User{
```

[Both ways can be used to a class can be mentioned to be singleton]

3

# Prototype

→ Each time a new Object is created.

→ It is Lazily initialised, means object created only when its required.

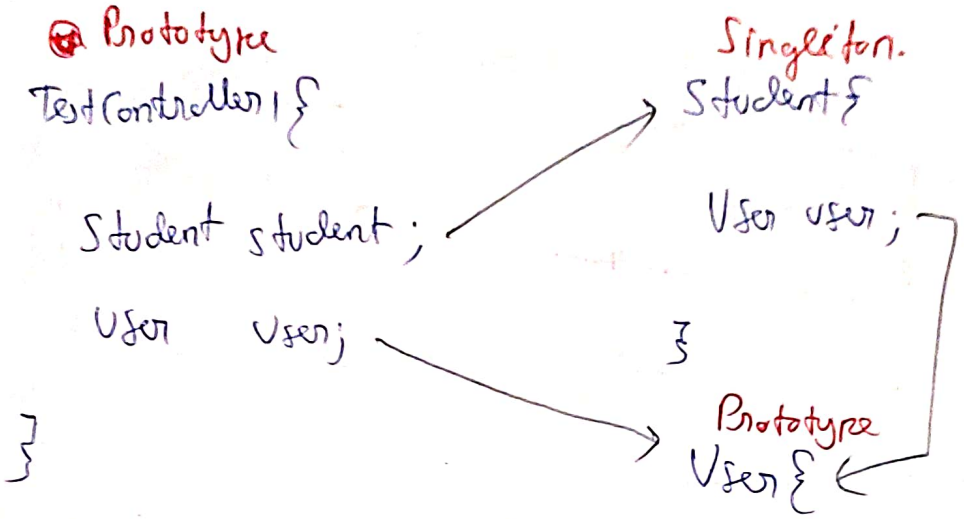
[Example to understand Better]

```
@RestController
@Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
@RequestMapping("/api")
public class TestController1{

    @Autowired
    User user;

    @Autowired
    Student student;

    public TestController1(){
        Sout("TestController1 init");
    }

    @PostConstruct
    public void init(){
        Sout("TestController1 hashcode:" + this.hashcode +
             "User object hashcode:" + user.hashcode +
             "student object hashcode:" + student.hashcode);
    }

    @GetMapping("/fetchUser")
```

```
@Component
public class Student{

    @Autowired
    User user;

    @PostConstruct
    public void init(){
        Sout("student object hashcode" +
             this.hashcode + "user hashcode
             + user.hashcode)
    }
```

```
@Component
@Scope("prototype")
public class User{

    @PostConstruct
    public void init(){
        Sout("user
             object
             hashcode:"
             + this.hashcode
        );
    }
```

From the class definitions you can understand that




Let's now discuss the Object instantiation and Object Creation.

During Startup : IOC checks all the bean Scopes. It sees.

TestController1 → Prototype (Lazily initialised so bean not created now)

Student → Singleton. [Bean created and kept in IOC]

User → Prototype [Lazily initialised so bean not created]

Output → "student initialised".

But Student bean has a dependency on User bean.

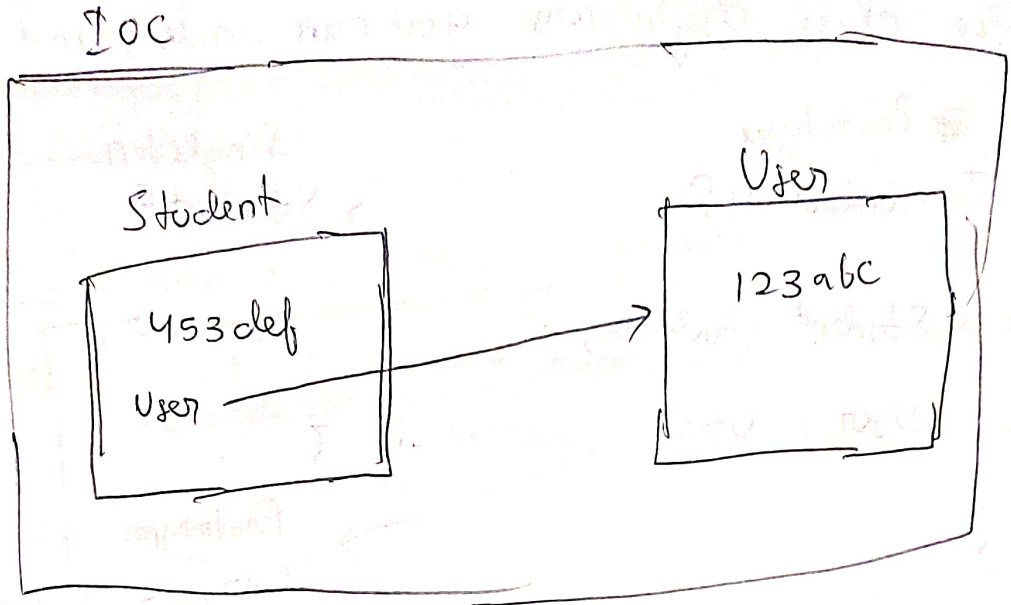
Now User bean has to be injected.

("user initialised")

("user hashcode → 123abc")

Now Student bean dependency injection is done.

@PostConstruct → (student hashcode → 453def
user hashcode → 123abc)



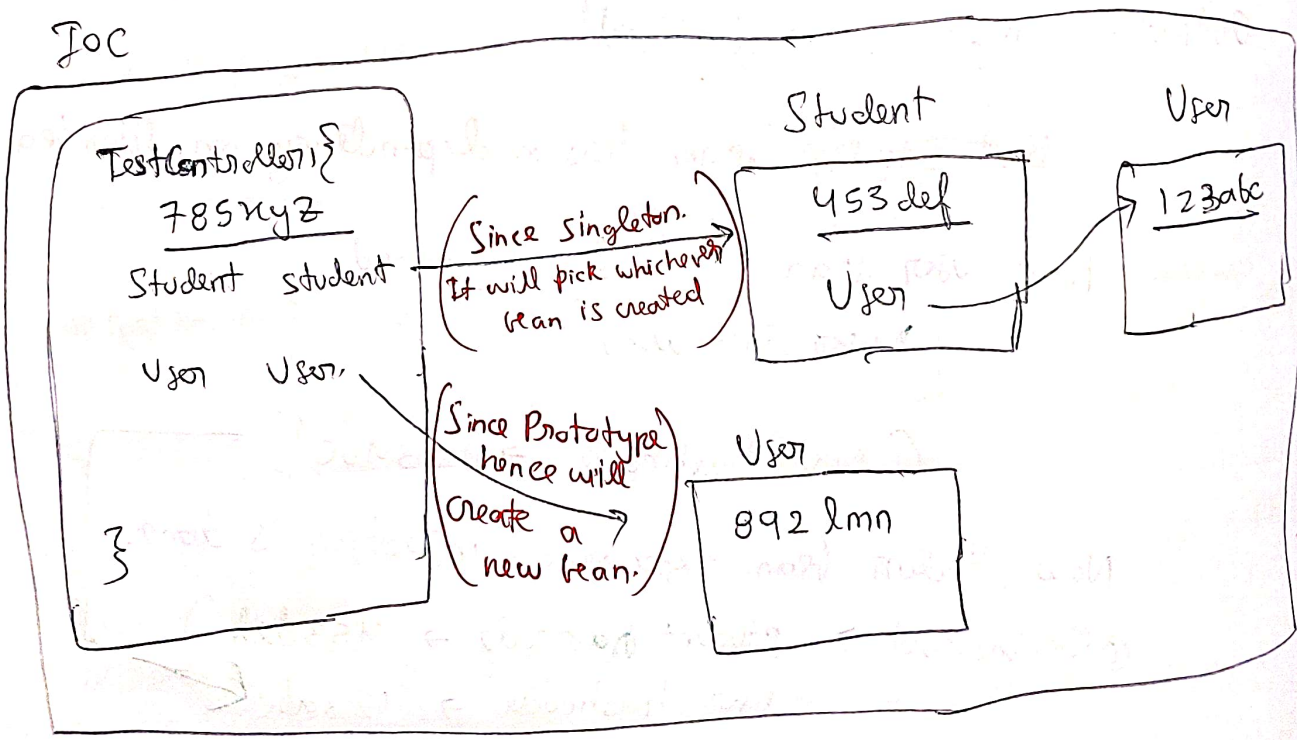

As of now IOC looks like this.

Now Lets say we invoke [/api/fetchUser]

Now TestController1 bean has to be created.

"TestController1 initialised"

Now dependencies will be inject



# Request Scope

→ New Object created for each HTTP request

→ Lazily Initialised

(Example)

```
@RestController
@Scope("request")
@RequestMapping("api")
public class TestController1{

    @Autowired
    User user;

    @Autowired
    Student student;

    public TestController1(){
        Sout("TestController1 initialised");
    }

    @PostConstruct
    public void init(){
        Sout("TestController1 →" + this.hashcode +
             "User →" → user.hashcode +
             "Student →" → student.hashcode);
    }

    @GetMapping("/fetchUser")

}
```

```
@Scope("prototype")
@Component
public class Student{

    @Autowired
    User user;

    @PostConstruct
    public void init(){
        Sout("User →" + user.hashcode
             "Student →" + this.hashcode)
    }
}
```

```
@Component
@Scope("request")
public class User{

    @PostConstruct
    public void init(){
        sout("User" + this.
             hashcode);
    }
```

Now Lets check the output.

During Application Startup → None of the beans will get created as they are all Lazily initialised

Now {/api/fetchUser → this API is invoked.}

TestController1 Object is created now

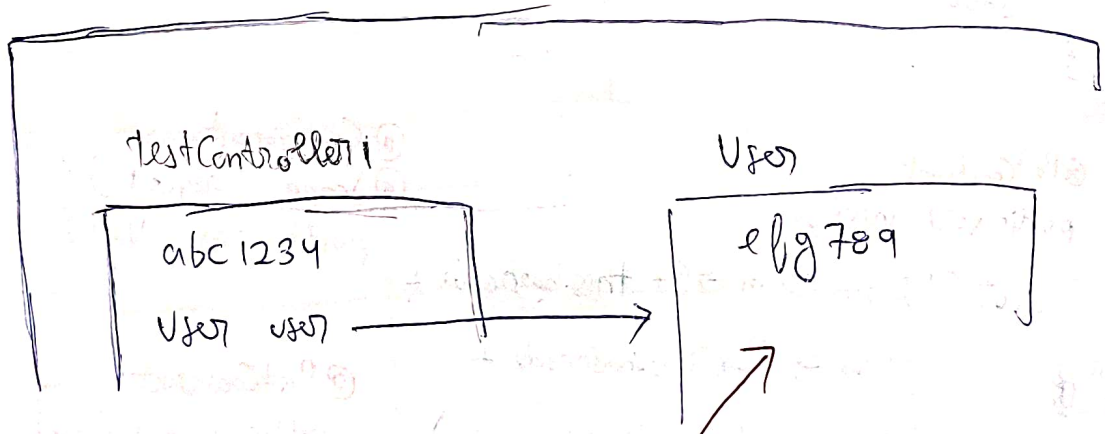
"TestController1 initialised"

Now dependencies to be injected.

1. User Bean will get initialised "User initialised"

{"User hashcode → efg789"}

IOC

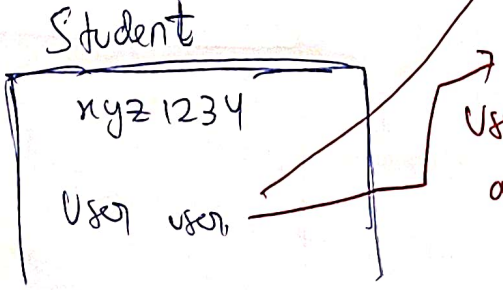



2. Student Bean will get initialised.

IOC

↳ marked as Prototype always new bean will get created.

"student initialised".




User is marked as ~~Prototype~~ "Request"

(This is not a new request so new Bean is not created)

@PostConstruct of Student.

"Student hashcode → xyz1234."

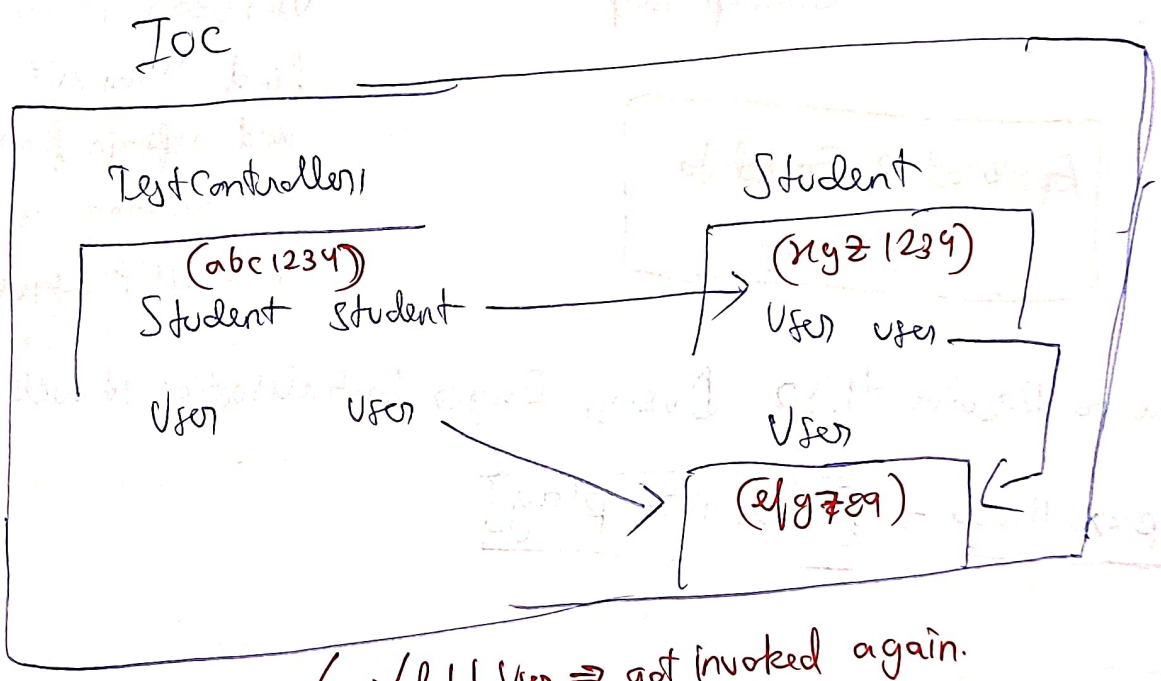
"User hashcode → efg789"

@PostConstruct of TestController1.

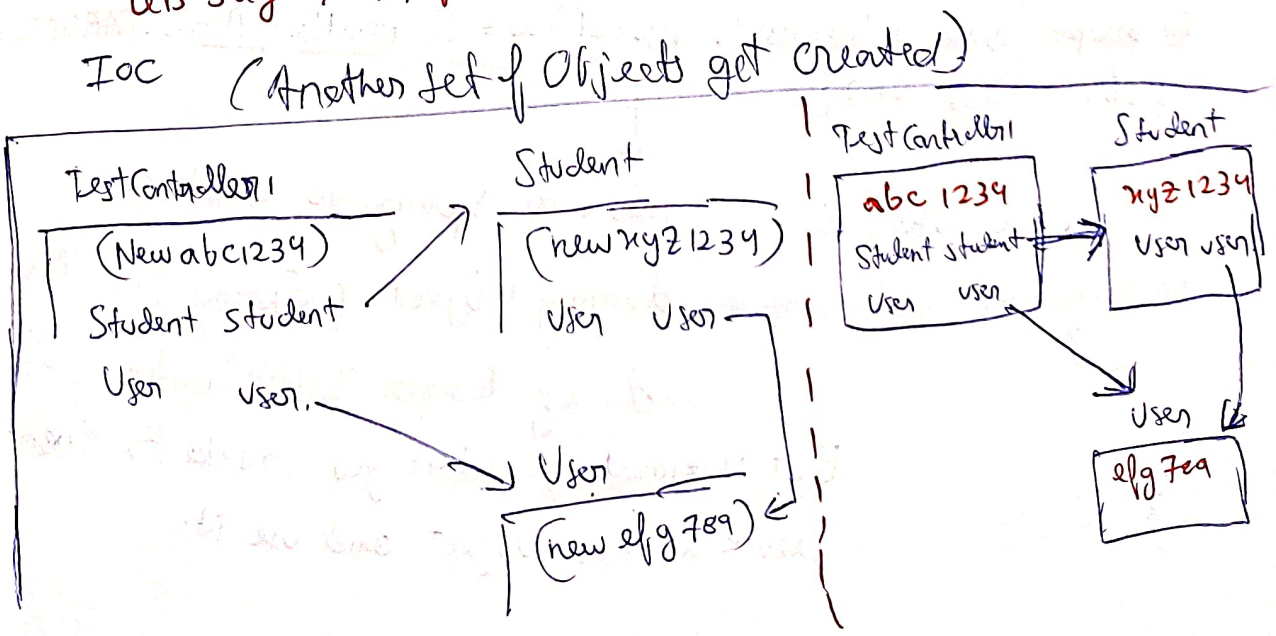
"TestController1 hashcode → abc1234"

"User hashcode → efg789"

"Student hashcode → xyz1234"




Lets say /api/fetchUser ⇒ got invoked again.



(Consider the below Example)

```
@RestController
@Scope("singleton")
@RequestMapping("/api")
public class TestController{

    @Autowired
    User user;
}
```

Since Singleton will create a bean during startup itself.

```
@Component
@Scope("request")
public class User{
```

} while trying to inject User bean it would find "request" and refrain from creating as there is no HTTP request.

Application Failed to Startup

How to Resolve this? During Eager Initialisation it will fail

proxyMode → {Comes into play}

```
@Component
@Scope(value = "request", proxyMode = ScopedProxyMode.TARGET_CLASS)
public class User{

    public User(){

    }
}
```

⇓

this tells Spring to create a dummy Object of ~~Spring~~ the Class during Eager Initialisation of Objects.

But Ultimately when you invoke it, then create a new Object and use it.

# Session Scope

→ New Object is created for each HTTP session.

→ Lazily initialised

→ When user acesses any endpoint session is created.

→ Remains active, till it dos not expire.

(Example)

```
@RestController
@Scope("session")
@RequestMapping("/api")
public class TestController1 {

    @Autowired
    User user;

    @PostConstruct
}
```

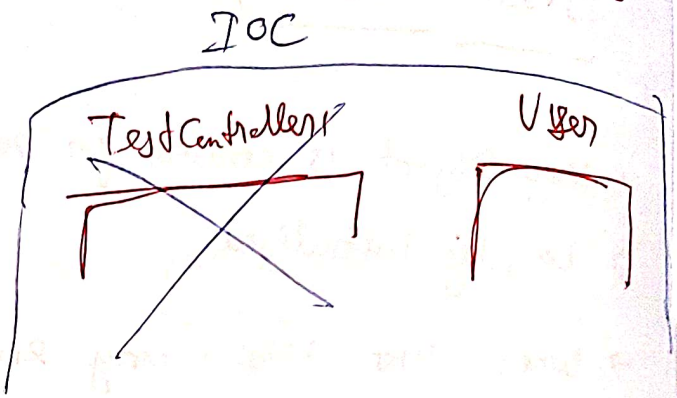
```
@Component
public class User {

}
```

During Application Startup → Only User Object is created.

When HTTP request comes it creates [TestController1] and uses the previously created User Object.

New HTTP request comes, it dos not create a new TestController1, because HTTP session is still going on.

Now we are ending the HTTP session. Lets see what happens




```
@RestController
@Scope("session")
@RequestMapping("/api")
public class TestController1 {

    @Autowired
    User user;

    @PostConstruct
    public void init(){

    }

    @GetMapping("/fetchUser")
    public ResponseEntity<String> getUserDetails(){

    }

    @GetMapping("/logout")
    public ResponseEntity<String> getUserDetails(HttpServletRequest request)
    {
        Sout("end the session")
        HTTP session session = request.getSession();
        session.invalidate();
        return ResponseEntity.status(HTTPStatus.OK).body("");
    }
}
```

When we call the /logout API the HTTP session ends and only the TestController1 class ends.. {User still remains}