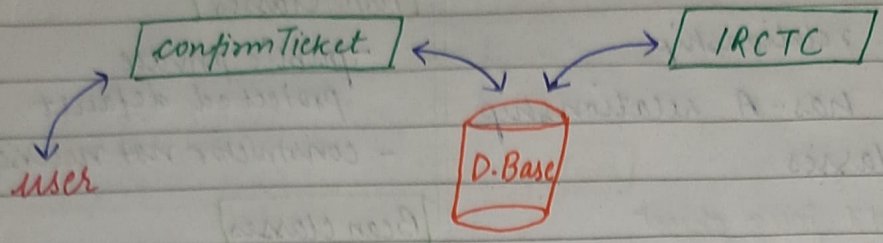
DATE
PAGE

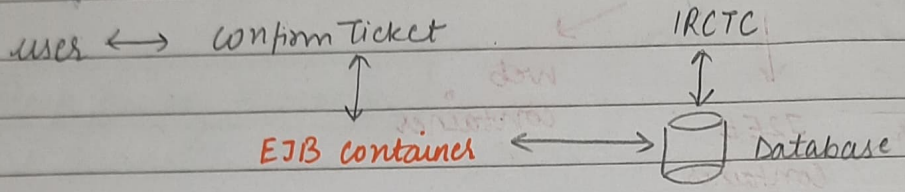
22nd Aug '22
Monday

Spring Framework - day ①



In this situation the user has direct access to database which is NOT SAFE

Introduction to EJB containers



No direct access of user to database
↓
SAFE

* Enterprise Java Bean Containers
* Helps in communication b/w two containers

* Both application must have SAME prog. language
* It is HEAVY weighted

Introduction to Spring

* Founder - Rod Johnson
* Earlier named Interface21

WHY USE SPRING?

① Alternative for EJB containers
② It is open source
③ It is light weighted → Spring runs on top of JRE while EJB runs on top of operating system.
④ It is called frameworks of frameworks

Isha Bansal

DATE
PAGE (2)

Rules of spring framework

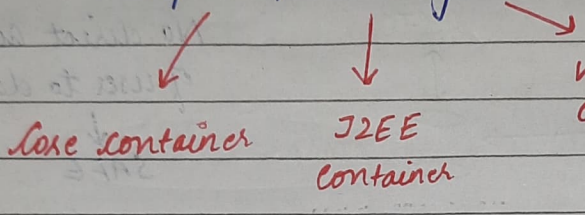
① Avoid IS-A relationship
↳ use has-A relationship

② use (POJO) classes
plain old java object

POJO Classes
- public private protected default
- constructor not necessary

Bean classes
- only private class
- constructor is mandatory

Introduction to containers
↳ Interface b/w two objects



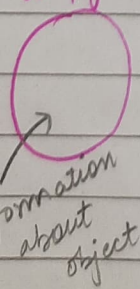
↳ Spring framework
↳ Inversion of controller

Spring (IOC) framework → core + J2EE container
> Core container : Bean Factory interface
> J2EE container : Application context interface

~~IOC~~ * core of spring
* Responsible to instantiate, configure, assemble } Objects

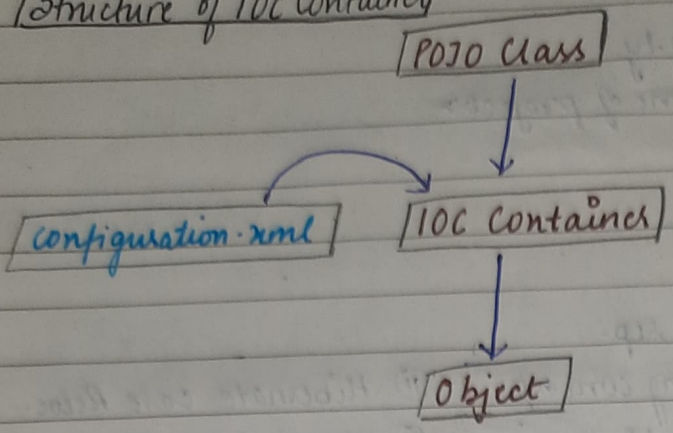
In hibernate developers create the object
↳ But in IOC, the IOC containes creates the objects.

configuration.xml



Then using config.xml file, IOC containes container creates the object.

Structure of IOC Container

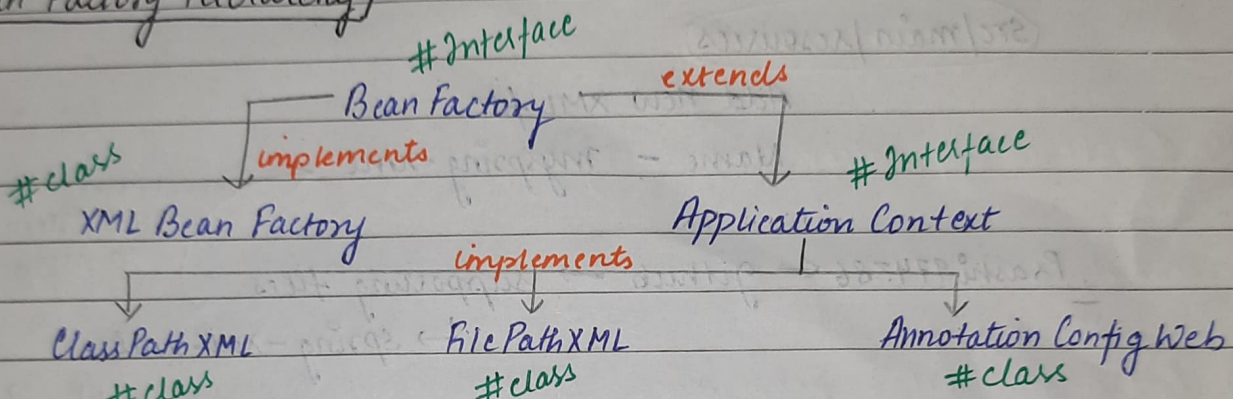


POJO class is passed to IOC container

Passing information about object to IOC container

IOC container creates an object for the user which is ready to use.

Bean Factory Hierarchy



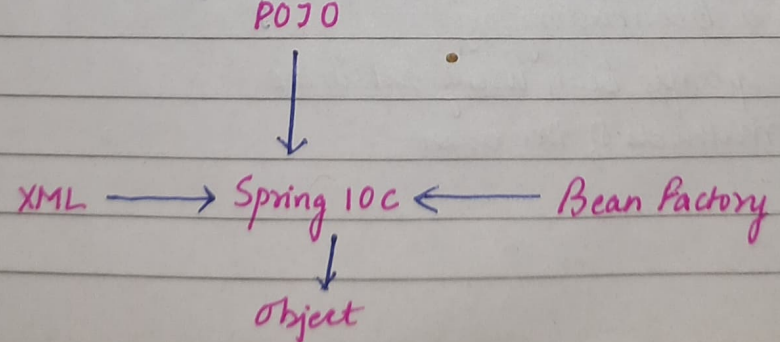
> Bean Factory

① interface

② org.springFramework.beans.Factory package → java object

MAIN TASK – creating the object
config the object
assemble the object

beans are config at RUNTIME by spring IOC container.



Isha Bansal

PRACTICAL IMPLEMENTATION

Maven Project

- Groupid — com.ty
- Artifactid — <name of project>

pom.xml

<dependencies>

(i) MVN rep.
(ii) Spring context
(iii) 5.3.18 version
(ii) Hibernate core Reloc.
(ii) mysql Connector

</dependencies>

src/main/resources

Add new XML file
Name - myspring.xml

Prashi974386 ← github → supporting files
↳ spring-xsd.xml

add content to myspring.xml

<beans>

<bean> ↺ <bean id = "mypen" class = "com.ty.PenClass01">
// add some attributes
</bean>

go to your class → right click class name → copy fully qualified name

TestBeanFactory01.java

main() {
(i) Attach the .xml file
(ii) Create a BeanFactory obj
(iii) Get a specific bean using attribute
(iv) Call methods of the bean
}

PenClass01.java

write() {
Sysout ("write");
}

Isha Bansal

① Attach the .xml file [ClassPathResource]

```
ClassPathResource cpR = new ClassPathResource("myspring.xml");
```
(name of the .xml file)

② Create a BeanFactory object [BeanFactory]

```
BeanFactory bF = new XMLBeanFactory(cpR);
```
(pass the classResource object)

③ Call a specific bean [PenClass01]

```
PenClass01 p1 = (PenClass01) bF.getBean("mypen01");
```
(pass the id of the bean)

↳ explicitly convert to class object.

④ Call methods of bean object

```
p1.write();
```
Output : write

class the write method of PenClass01.java class.

Introduction to Application Context

→ Create J2EE container

- BeanFactory is parent of ApplicationContext → All methods of BeanFactory are inherited by ApplicationContext
- interface
- org.springframework.context package

MAIN TASK: Create, config, assemble objects (beans)

Three children:
- ClassPathXMLA.context → load XML file
- FileSystemXMLA.context → load a file
- Annotationsconfig A.context → when we use annotations

Used to create class for Application context

Isha Bansal

① Create a Pen class
② Create myspring.xml file
③ Create a class "TestApplicationContext".

TestApplicationContext.java

main() {
① Create an application context object and connect .xml file
② Get a particular bean using "id" attribute
↳ downcasting
③ Call methods of the object.
}

① Create an application context object

pass name of .xml file ←

```
ApplicationContext aC = new ClassPathXMLApplicationContext(<file>);
```

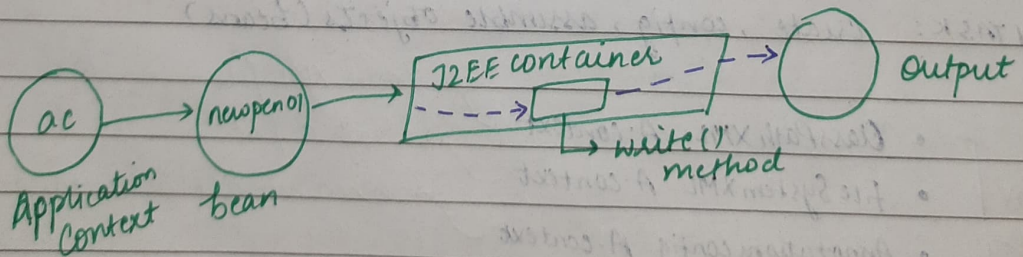
② Get a particular bean using "id" attribute

```
PenClass01 p1 = (PenClass01) aC.getBean("newpen01");
```

③ Call methods of the object

```
p1.write();
```

Output : write



BeanFactory

* parent of App. context
* XMLBF (implementing class)
* Annotations not possible
* Lazy loading

Application Context

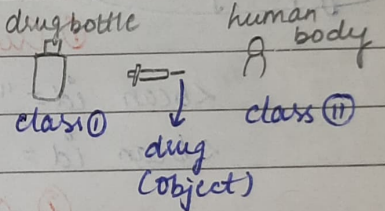
* Child of BeanFactory
* Implementing classes:
  - Classpath ← CPXAC, ACAC (↳ annotation)
  - FSXAC (↳ file)
* Annotations possible cuz of annotation config
* Eager loading

Introduction to Dependency ~~Induction~~ Injection

one of the major reason for using spring IO framework

Dependency Injection → Real life ↳ inject drug into body

↳ initialize a value from one class to another class



① Constructor dependency injection (for variables)

SongClass01.java
- id
- sName
- viewsCnt

TestSongConstDInjection.java

```
main() {
  ① Create ApplicationContext object   ac ← CPXMLAC(<file>)
  ② Get a particular bean              s1 ← getBean(<id>)
  ③ Call the method                    s1.display()
}
```

myspring-xml

```
<bean id="song01"
      class="com.ty.dto.SongClass01">
  <constructor-arg index="0" value="1"></constructor-arg>        id
  <constructor-arg index="1" value="Lovea"></constructor-arg>    sName
  <constructor-arg index="2" value="234"></constructor-arg>      viewsCnt
```

index = "2" → not mandatory if the order is followed.

Isha Bansal

6

Constructor Dependency Injection

CDI for variable (done before)

CDI for object

Constructor dependency injection for object

Car class

```
public
Player p1;
Car (Player p) {
  this.p1 = p; }
```

Player class

```
music() {
  print("playing music");
}
```

myspring.xml

① Create bean for both classes

```
<bean id="m1" class="com.ty.Player"></bean>
<bean id="c1" class="com.ty.Car">
```

② Constructor of car is taking an object

```
<constructor-arg ref="m1"></constructor-arg>
```

'ref' attribute

ref is used to refer an object

pass id of bean object created

Controller.java

```
ApplicationContext ac = new ClassPathXmlAC("myspring.xml");
Car car1 = (Car) ac.getBean("c1");
car1.p1.music();
```

Output = playing music

Isha Bansal

Setter Dependency Injection

Setters are used to set values for private members.
<property> tag is used under <bean> tag.

```
<beans>
    <bean>
        <property> </property>
    </bean>
</beans>
```

<property name = "id" value = "1" > </property>
↑ private data members name
value.

Understanding Annotations

DogClass

```
@Component
class dog {
    pv play() {
        sysout("playing");
    }
}
```

@Component Annotation

→ consider the class as a component and it is used to make the class auto detection possible

mySpring.xml

```
<beans>
    <context:component-scan base-package="com.ty">
    </context:component-scan>
</beans>
```

TestDog Class

```
main() {
    ApplicationContext ac
    dog d1 = (dog) ac.get("dog");
    d1.play();
```

↑ mention class name only

Output : playing

Isha Bansal

Variable Injection ⇒ Injecting values into the class.

@value annotation

Employee class

```
@Component
class Employee {
    @Value(value = "1");
    int id;
    @Value(value = "Kiran")
    String name;
}
```

(#) In @value it is not possible to set multiple values.

@value to setter annotation

Employee Class

```
@component
class Employee {
    int id;
    @Value(value = "1")
    public void setId(int id) {
        this.id = id }
```

setter method

@value to constructor

Employee Class

```
@Component
class employee {
    int id;
    public Employee(@Value(value = "1") int id) {
        this.id = id; }
}
```

Isha Bansal

DATE
PAGE 11

@Autowired Annotation

Car Class

```
@Component
class Car {
    @Autowired
    public MusicPlayer mp;
}
```

automatically connected/ wired to the class.

Music Player Class

```
@Component
class MusicPlayer {
    pubic void music() {
        sysout("playing");
    }
```

Is there a way to avoid the myspring.xml file completely?

package com.ty.config

```
@Configuration
@ComponentScan(basePackages = "com.ty")
class MyConfigClass {
}
```

configuration class
alternative for myspring.xml file

package com.ty.dto

MobileDto

```
SimDto s1
void disp() {
    s1.dispSim();
}
```

SimDto

```
void dispSim() { }
```

When put inside Config class will return a new object

```
@Bean("m1")   // id attribute
public Mobile getM() {
    return new Mobile();
}
```

package com.ty.controller

```
main() {
    AC ac = new AnnotationConfigAppContext(MyConfigClass.class);
    MobileDto m1 = ac.getBeans("mobileDto");
    m1.disp();
```

making use of config. class