Spring Boot & Microservice's

09 March 2024 08:53

(Servlet) + JDBC

* Pre-Requisite : Core Java + Adv Java + SQL

done done done

* Teacher : 6+ years

* Course Content :

1) Spring Core
2) Spring Boot
3) Spring Data JPA
4) Spring Web MVC
5) RESTful services
6) Microservices
7) Spring Security
8) Spring Cloud
9) Cache
10) Cloud deployment
11) Dockers
12) Kafka
13) K8
14) unit testing

* Technology ?

S/w → programming

* servlet
* JDBC

* Programming Lang *

→ Java, C++

Application + Tools + S/w + Framework

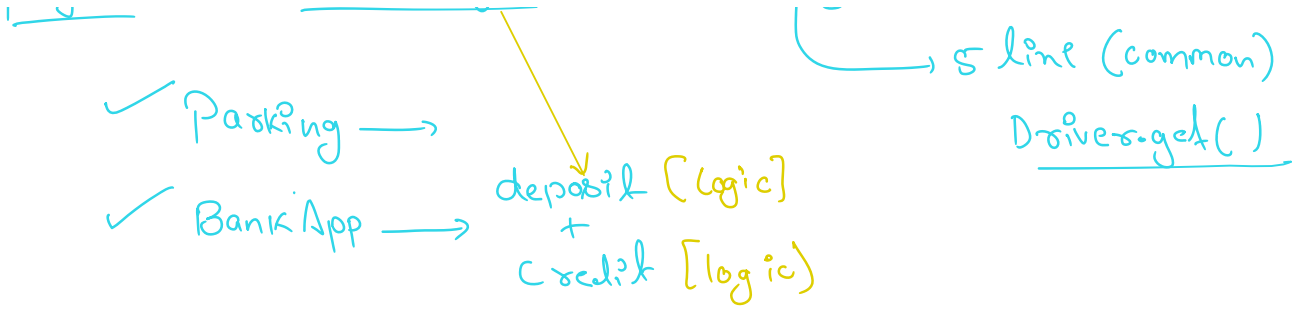
Syntax / Rule

* Servlet

project = Business logic + Common Logic

project => Business Logic + Common Logic

→ 5 line (common)



Q What is a framework?

— To over come loop holes of "Technologies" framework came

→ framework is semi developed s/w

→→ provides Re-usable Components

eg: JDBC code

```
Class.forName("")
DM.getCon()
createStmt()
executQuery()
Result
close
```

multiple time

Boiler plate code

Redundant code

Project = B.L + C.L → productivity ire

* eg: Java : Hibernate, Structs, Spring → framework

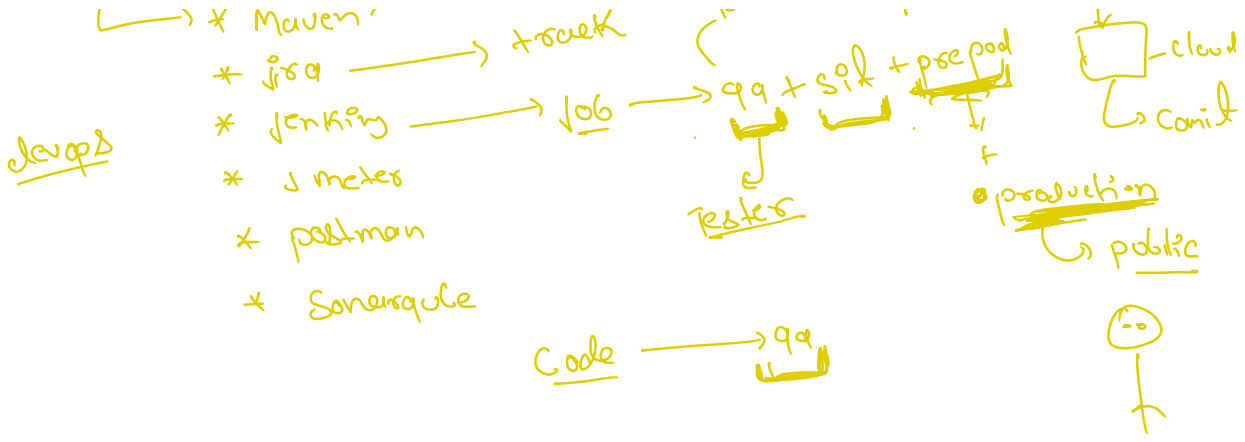
python : Django, flask

Salesforce : Lightning

.Net ⇒ WCF

* Tools *

* Maven → Gradtool
* jira → trucK



* Application Architecture

HTML $CSS
JS
Bootstrap
Angular

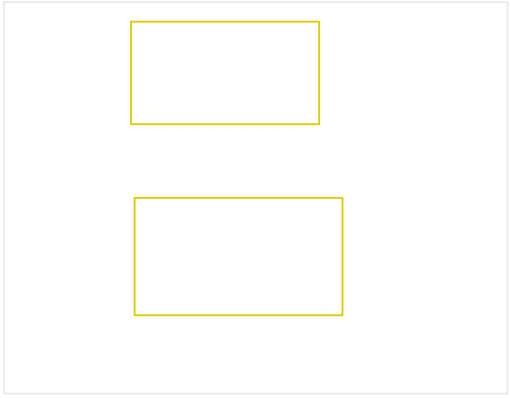
1) Java & J2EE
2) Spring Core
3) Spring Boot
3) JPA
4) REST

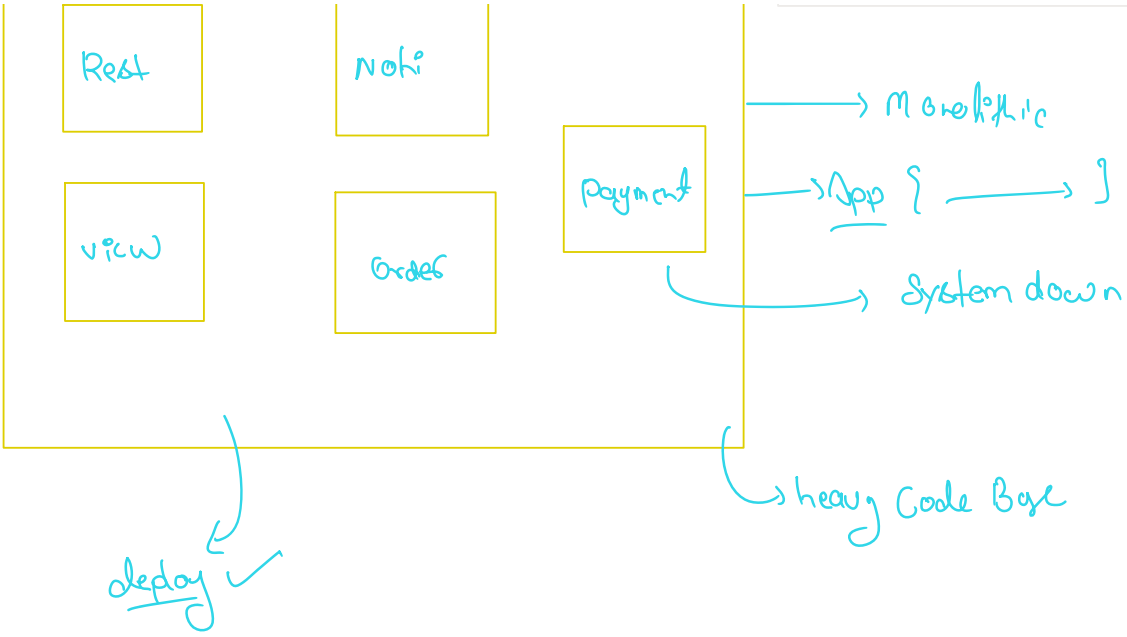
5) Json
6) micro
7) security
8) cache.

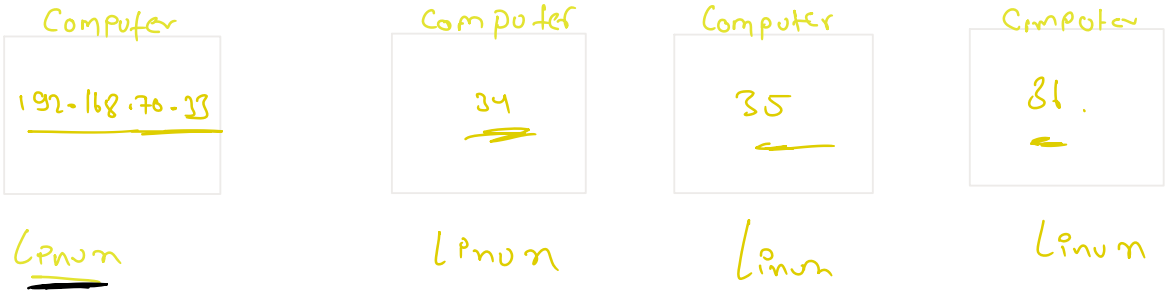
→ oracle
→ mysql
→ Mongo.
→ postgress

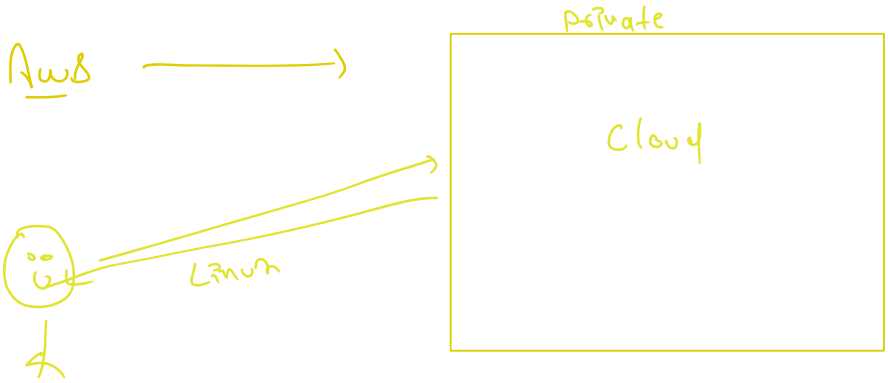
* Architecture Types *

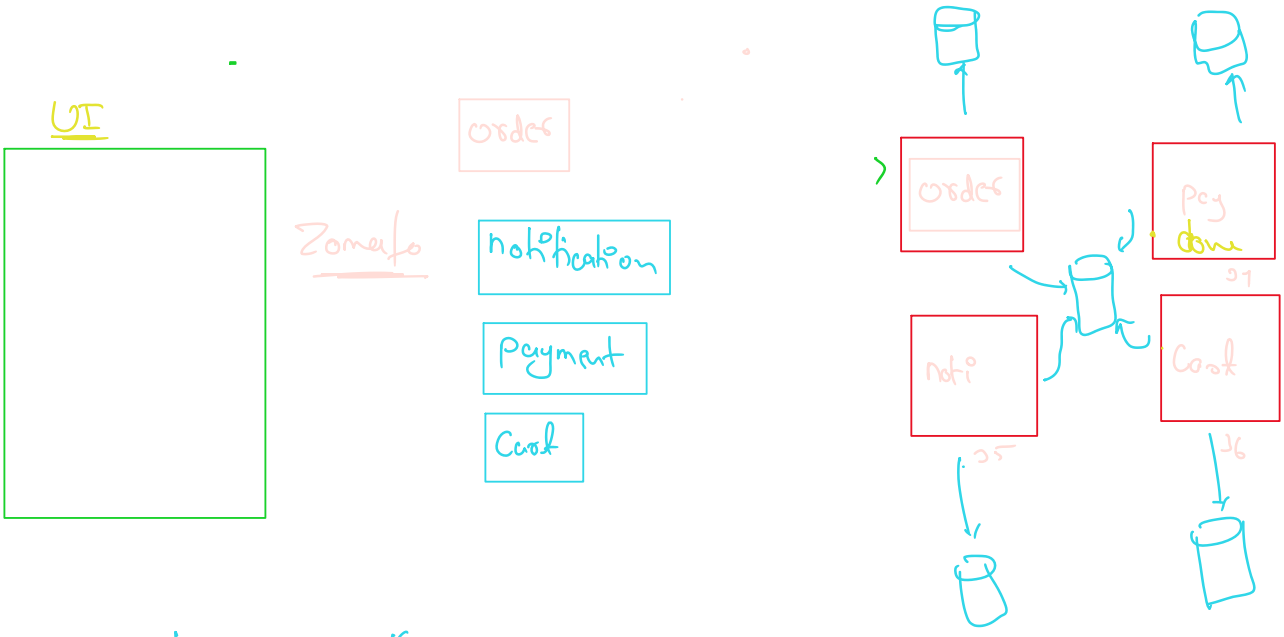
1) Monolith
2) micro




Rest
Noti
view
Order
Payment
Monolithic
App { }
System down
heavy Code Base
deploy


Computer
192.168.70.33
Linux
Computer
34
Linux
Computer
35
Linux
Computer
81
Linux


Aws
private
Cloud
Linux



* Types of Framework

1) frontend : Angular, React(lib) *

2) web Framework : eg structs (outdated) => web layer

3) ORM " : To develop persistance layer (DB)
eg: Hibernate

Note: To overcome struct Framework SpringFramework came into market

Spring → flexible framework. It will not tough to learn every module

* Spring Modules

1) Spring Core *
2) Spring Context
3) Spring JDBC

7) Spring Security *
8) Spring Batch
9) Spring Data JPA *

2) Spring Context
3) Spring JDBC
4) Spring ORM
5) Spring Aop *
6) Spring web Mvc *
8) Spring Data JPA *
10) Spring REST *
11) Spring Social
12) Spring cloud *

1) Spring Core: It is base module (main)

⟹ provides fund Concept of SF

- 1) IOC Container ✓
- 2) Dependency Injection ✓
- 3) Bean life cycle ✓
- 4) Bean Scop ✓
- 5) Autowiring ✓

2) Spring Context: deal with Configurations

3) Aop : → Aspect oriented prog
   - → Secondary method
   - → logging, Auditing, exception

4) JDBC: simplify D-C logic

5) web Mvc: → web App & D's App

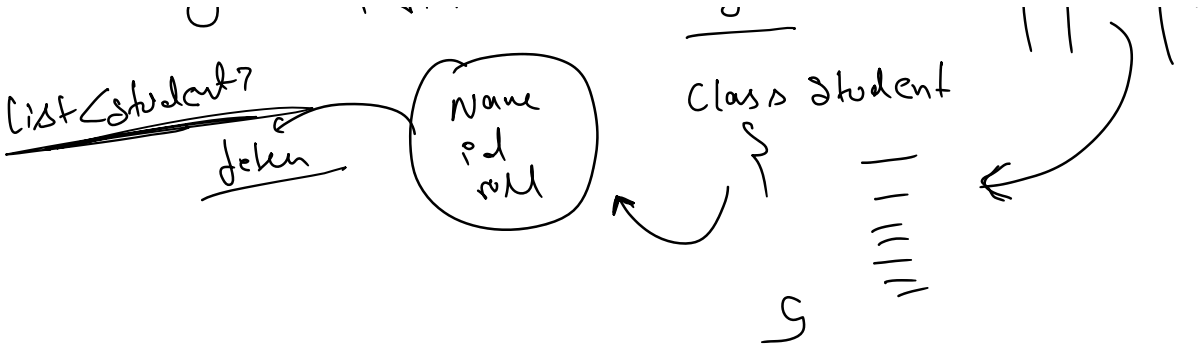
6) ORM (object Relational Mapping) — text

Student

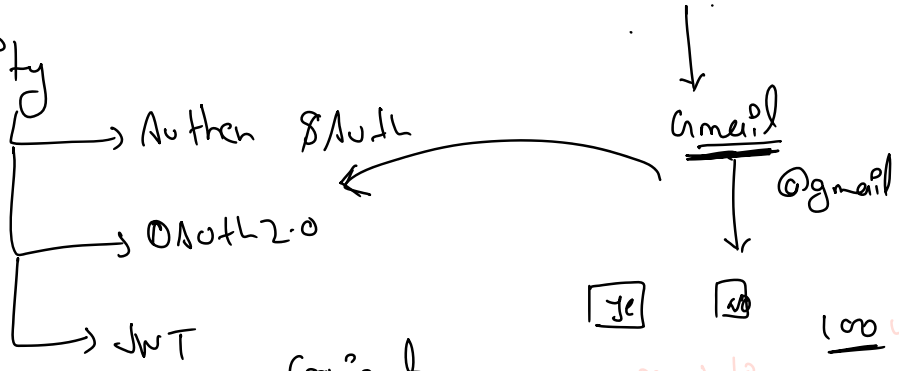
eg: Data JPA → object

name	id	Add	roll

class student

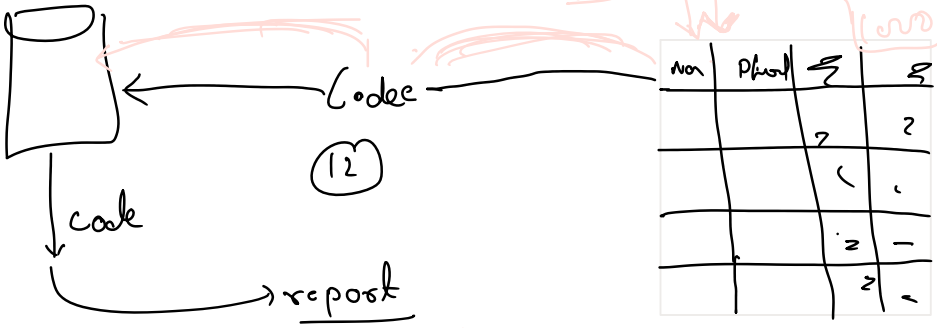


7) Spring Security



Cognizat

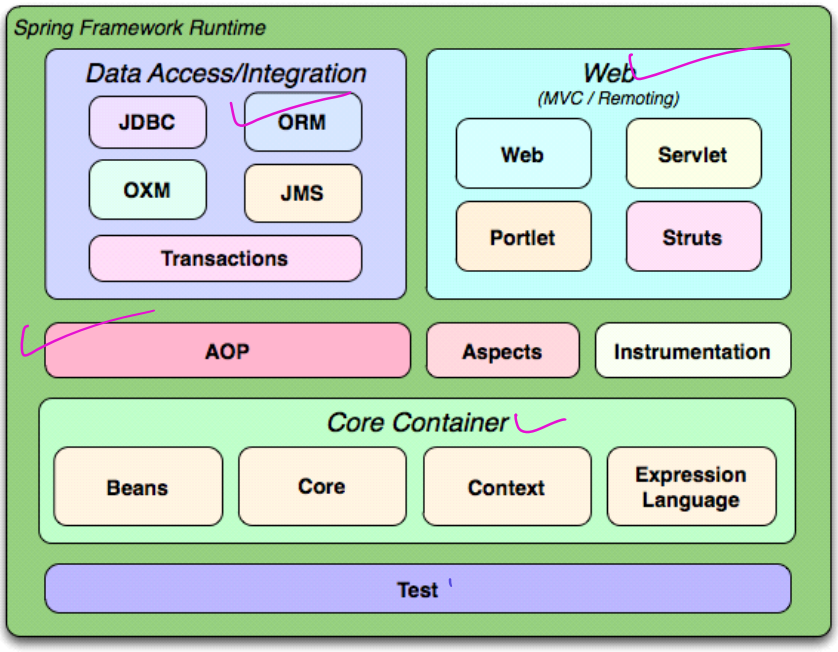
8) Batch

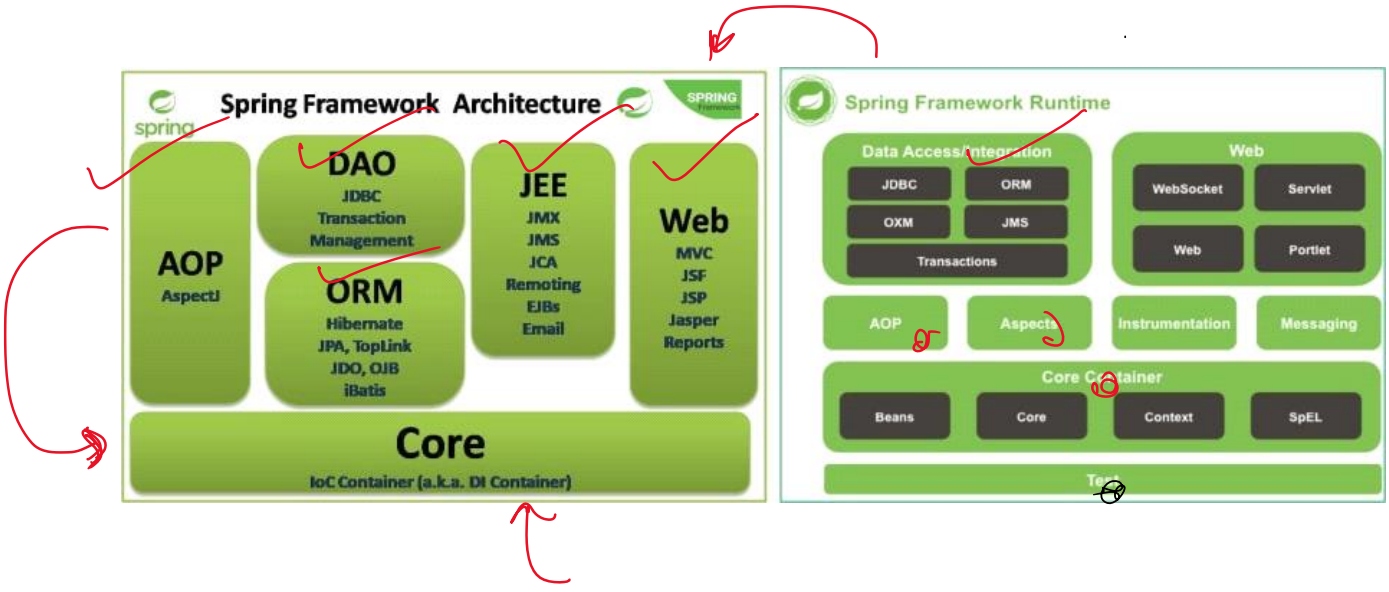


9) Spring Cloud : provide common tool to build distributed system

- discovery Service
- Gateway
- Load Balancer

10) Test f° Unit Testing framework





* Spring Core * ⇒ It is all about managing dependencies among the classes with loosely coupling

classes

· Core : Class Ray extend Simran

{

pen()

}

Class Simran

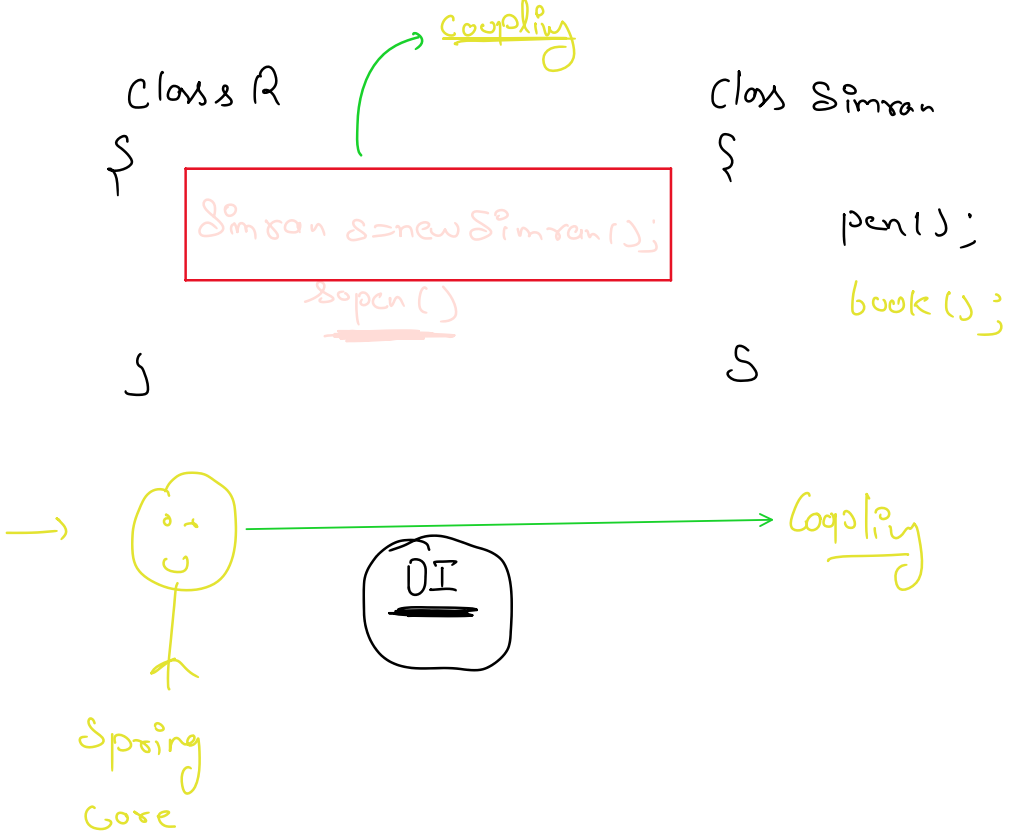
{

pen();

book();

}

→ tightly Coupled



⇒ project → class

↳ 3 types class

↳ 1) Pojo

2) Java Bean

3) Component

* Pojo (Plain old java object)

→ Any java class that can be compiled by using JDK

eg: class A {
int age
String name
}

eg: class Demo extends Thread {
}

Eg: class Test imp. Runnable {

}

Eg: class B extends A {

}

Eg: class Student imp Serulet {

}

* Java Beans

1) Any java class whic follow specific rules
   a) class should imp Serializable interface
   b) " " have private variable
   c) private variable should have public get & set
   d) class should have zero-argu const

* Componet
  └ containg business logic

  Eg: Controller, Service, Dao Classes

  Class → deal with Req & Res

  Business Logic

  DB logic

Dependency Injection

↳ It is the process of injecting one class object into another class is called DI

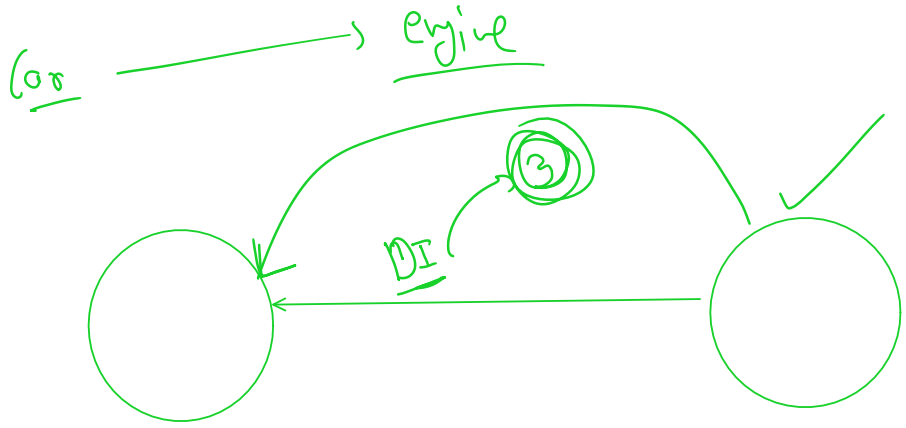
DI
- ⟶ Setter DI
- ⟶ Constructor DI
- ⟶ Field DI

Eg:

```
public class Car {

    private Engine eng;

    public void setEng(Engine eng)
    {
        this.eng=eng;
    }

    public void drive()
    {
        int start = eng.start();
```

engine

```
        if(start>1)
        {
            System.out.println("Car Started....")
        }
        else{
            System.out.println("Engine not started...")
        }
    }

}
```



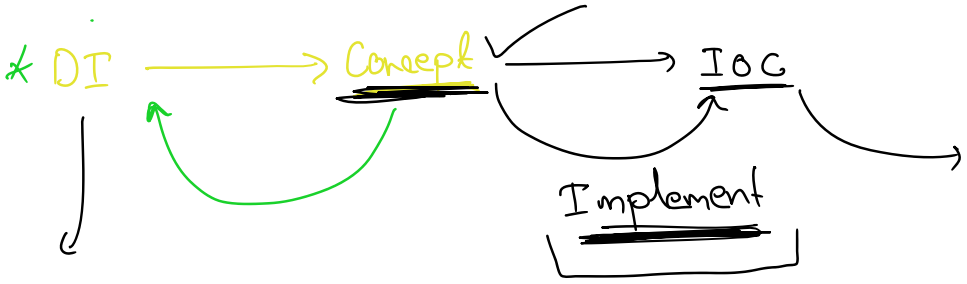
※ Ioc Containers

- Ioc → Inversion of Control.
- why → responsible for DI in spring Application

DI → Create object and inject into the target bean class

Note: Ioc will manage lifecycle of spring Bean

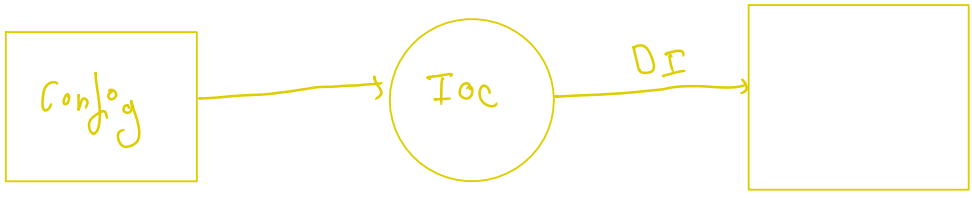
Note: we need to provide java classes + Bean Config as i/p for IOC then IOC perform DI and provide Spring Bean which are ready use



Type of IOC

1) BeanFactory (outdated)
2) ApplicationContext (recommended)

Beans.xml → read → parse → id



How to create Spring Core App

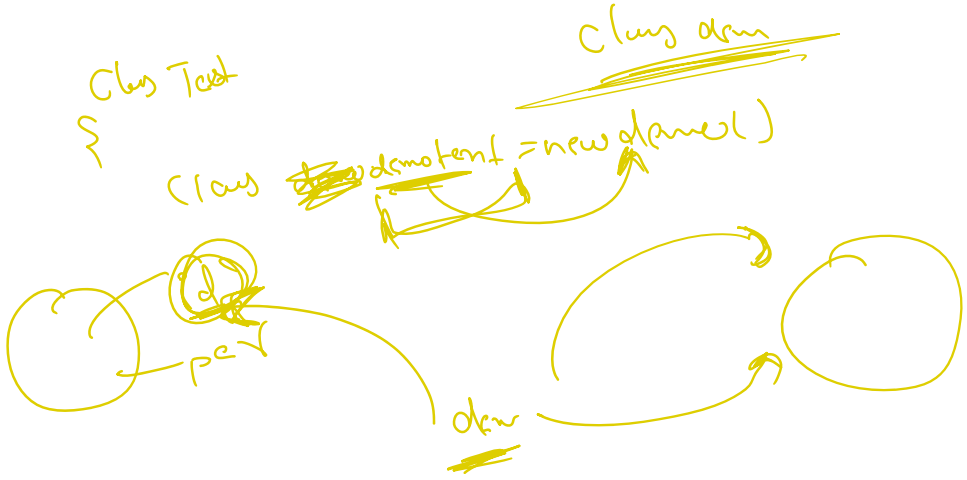
1) Create maven project

2) Add Spring Content dependency

3) Create Required class

4) Creat Beans.xml

5) Create Main (driver) to Start IOC Container



reference → object

Engine

Config → rule syntax

IOC → pahchey

80% → Rent

Agra Noida 40-50

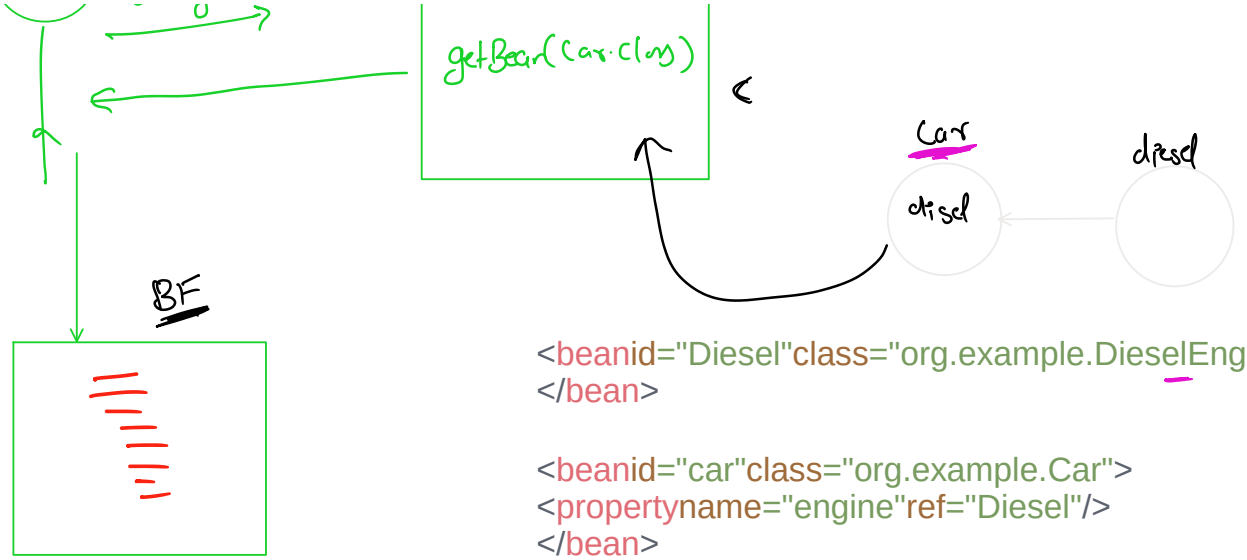
* ApplicationContext

↳ IOC Container [ ]

* BeanFactory :- follow Lazy loading Concept that means when we request then only it will create Bean object

* ApplicationContext :-

↳ follow Eager Loading




```
<beanid="Diesel"class="org.example.DieselEngine">
</bean>

<beanid="car"class="org.example.Car">
<propertyname="engine"ref="Diesel"/>
</bean>
```

Q) Spring Bean?

Q) Ioc containers?

Q) DI — type

Q) How ~~Ex~~ to Start Ioc?

Q) Beanconfig file?

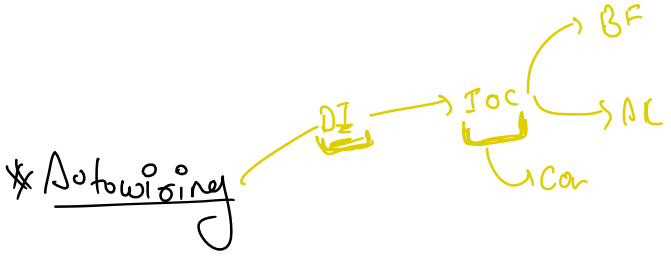
* SI

* Con DI

when we use DI & CI both at sametime then SI will override CI

Bean Scope

1) Singleton (default) ✓ → getBean( )
* prototype ✓
3) request → request →
4) session → new object session



1) inject dependent bean into target in 2 ways
* Manual wiring <property> <constructor-arg> ref.
* Autowiring.

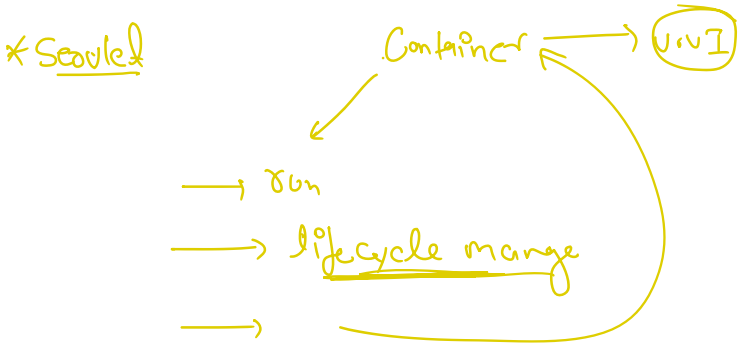
→ Autowiring means IoC container will identify dependent bean and it will inject into target bean
(we no need to use ref attribute in bean cog file)

→ Autowiring will work based on:
1) byName
2) byType
3) Constructor
4) no

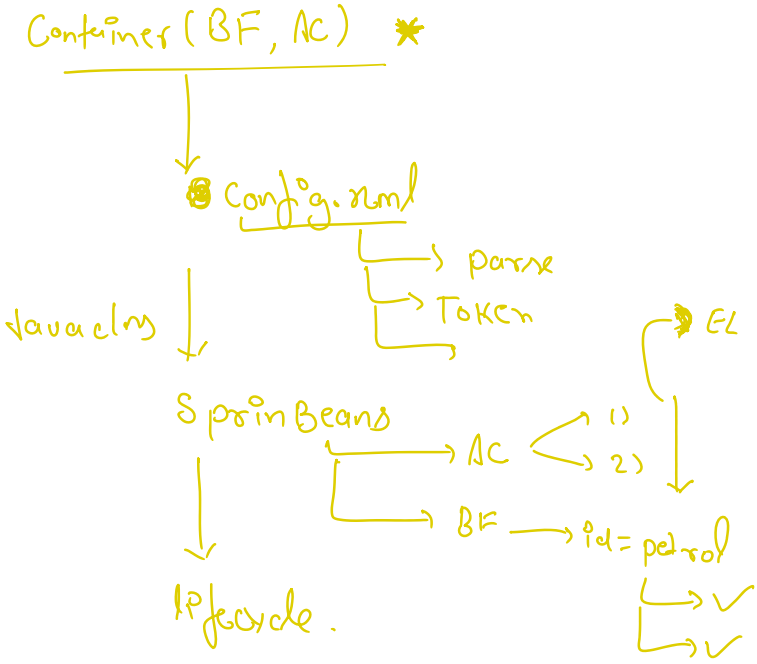
* → Container → run (Code)

* spring → Container → BF
→ AC

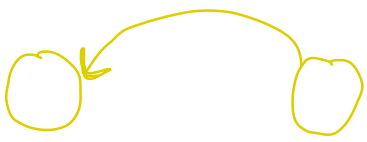
* Servlet



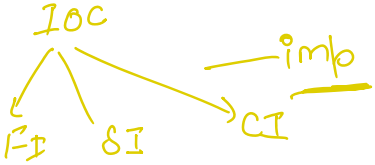
* Spring



DI



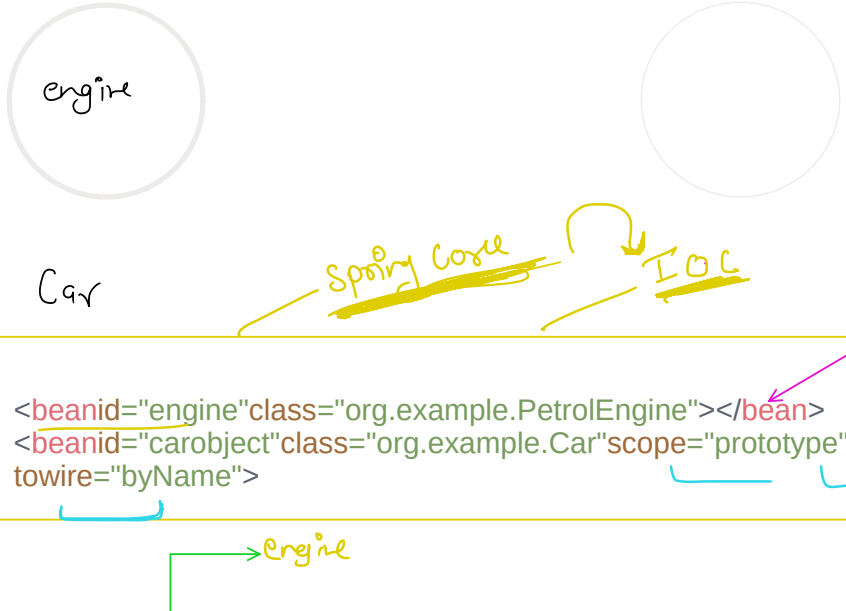
IOC

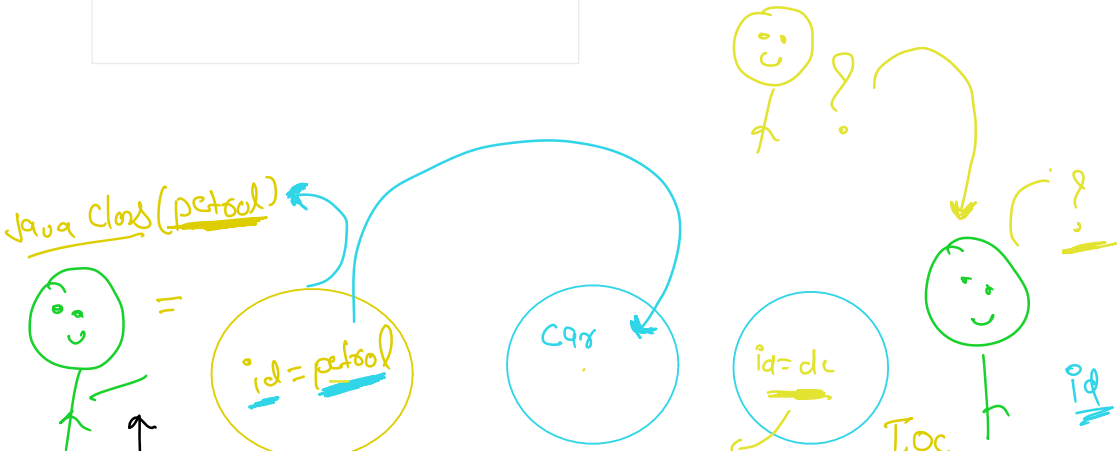


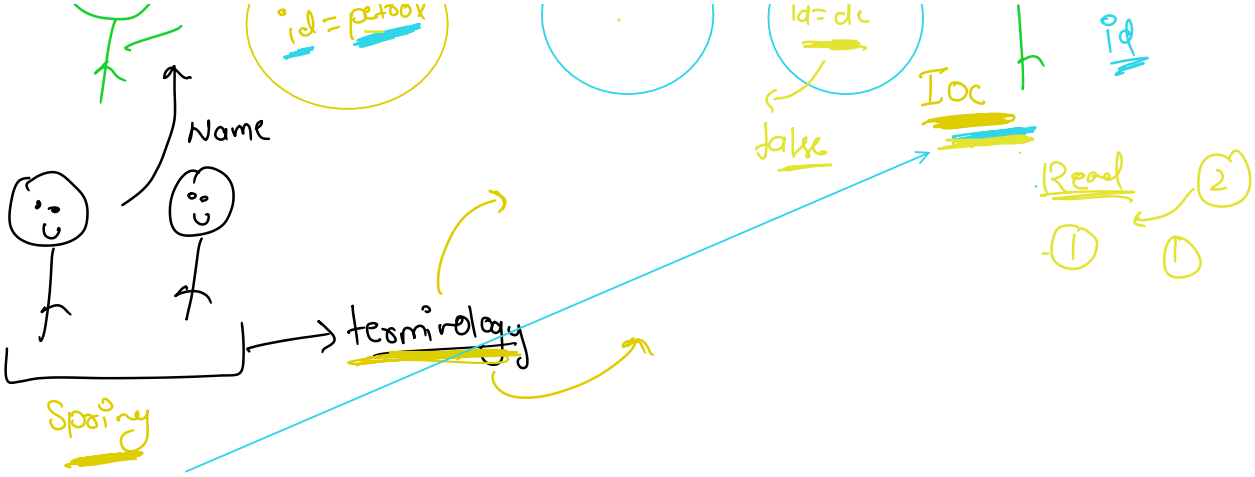
* Manually

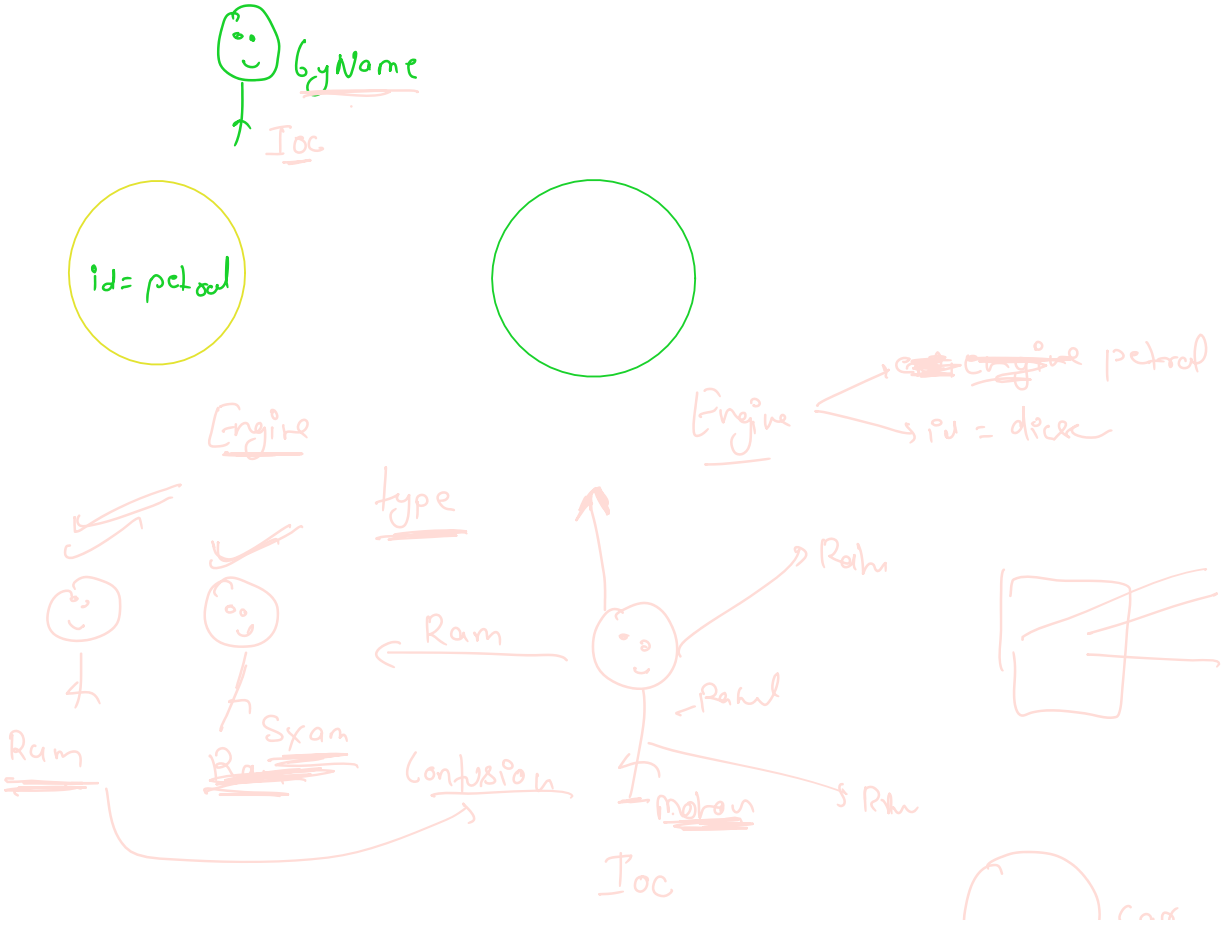
* Automatically

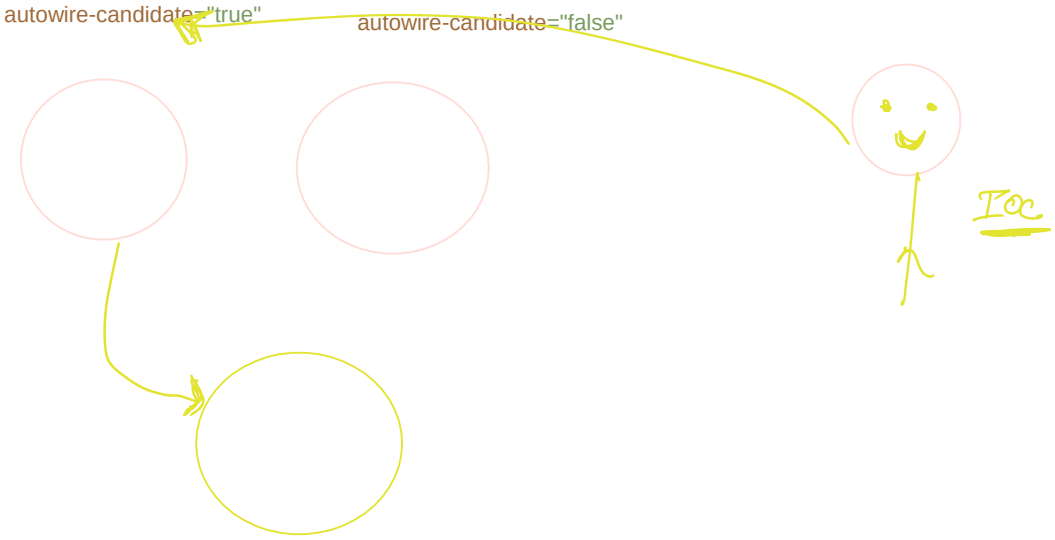
* Autowiring
  - byName
  - byType
  - Con
  - no






id=petrol
id=dc
Name
terminology
Spring
Ioc
Read
id
false


byName
Ioc
id=petrol
Engine
type
Ram
Ram
Ram
Ram
Ram
Confusion
Ioc



* Spring Bean Life cycle

→ IoC

→ init() → Load

→ destroy() →

1) XML Approach

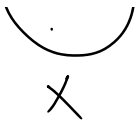
2) Programatic

3) Annotation

postral

X

3) Annotation



Init-method →

destroy-method →