

Spring 1 class

- **Framework:** It is workplace which provide pre-identified classes and interfaces to develop an application.
- Framework is a bunch of classes and interfaces which provides boiler plat logic.
- Framework makes developers work easy by providing pre-developed classes which reduces the code.
- Frameworks contains different tools, docs,etc.

- **Spring Framework:** It contains pre-identified classes and interfaces which provides boiler plat logic by Spring developers.
- There are several way to explain about SF.
- **Why Spring Framework?**
 - There are a lot to talk about why SF.
 - Before SF Struts is the highest popular framework in the market. If you see more before SERVLET and JSP were more popular in the market to develop web application.
 - SunMicroSystem has provided Core Java to develop Desktop application which is very famous in market . Letter Servlet and Jsp came in feature to develop web application.
- It seem to be good but there are so many demerit available (**API Drawbacks**)
 - To develop an application we have to know all the classes and interfaces in Java API.
 - We can not develop an application by knowing one or two classes.
 - B'z all classes are internally depends on all other classes and interfaces.
 - Ex. To write an JDBC program we have to use different classes and interfaces to establish a connection. Only Establishing the connection we have to learn all respective classes and interfaces.
 - Java API does not provided any boiler plat logic. If 100 developer want to establish a connection then 100 developer has to write same code in the respective application.

- Using servlet and jsp we have to write more number of lines of code. More code more bugs, more bugs more time , more time more budget, more developers, more testing , more maintenance.....there are so many problems we have to face.
- We cant develop an application using one or two APIs.
- Again a lot more problems are available.....

Spring 2 class

- **Struts**: Struts is the Framework which specially developed for web Application development.
- Struts is the combination of Servlet and Jsp.
- It provides bunch of classes and interfaces to develop an web based application.
- Struts also provided boiler plat logic for web development.
- To develop an enterprise and distributed app struts may not sufficient, B'z there are no other part available like business logic, persistency logic to develop an distributed app.
- Struts will not fulfill the requirements of the market .
- **To Overcome the above problems Spring Frameworks has been invented by Spring people.**

Spring 3 class

- **Spring Framework:**
- It is the Framework which provide end-to-end application development.
- SF is the very big Frameworks which can support multiple kinds of application.
- We can develop core app, web app, RMI app, distributed app,..etc.
- SF is the Super Framework it is also called as Frameworks of Framework.
- It provides boiler plat logic.

- **Advantages:**

- SF provide boiler plat logic which help developer in many way.
- Developer need not write same code multiple time in the app.
- It reduces all the drawbacks of the Java API.
- By help of SF we can reduces the line of code, maintenance of the code, less code means less bug , less testing time, less budget, less developer.....etc.
- B'z of pre-identified classes our code will be more efficient compared with java API.
- SF people removed total dependency between classes. Means one model is completely independent on other model.
- We need not to learn all the packages or classes to develop an app.

Spring 4 Class

- Architecture of Spring

MVC

ORM

Jdbc

**Transacti
on
Mngt
Model**

aop

Spring Core model

- Why Spring called loosely coupled ?
- As we seem in the above architecture diagram, according to arch. SF is loosely coupled Framework.
- B'z SF divided into different models and these model are totally independent to each other.
- If we wanna develop an app using Spring code and respective model we can develop an app.
- We need not to learn whole models to built an app.
- Loosely coupled means there is no complete dependencies between the classes and interfaces while building an app. Almost it's zero dependency.
- Arch. Itself broken into different part which lead to loosely coupled design.

- Why spring Framework so much popular in the market?
- There are so many reason but the most popular and most strong and unique features are
 - 1. versatile application development.
 - 2. Non-invasiveness app development.
- Versatile:
 - The word versatile make you to understand.
 - Versatile means Flexible.
 - Let's take an ex.
 - If a org. developing a app using struts there are so many drawbacks available in the struts. its not a complete Framework to develop an app. It doesn't have business layer and persistency layer to develop a complete project.
 - SF provide a feature called versatile which easily integrate any application without rewriting a code in it.
 - We can easily integrate an application at any part of an application without any change in the existing project, that is the greatness of SF.
 - We can integrate any technology with the SF. It's too flexible

- We can add SF at any part of the project no problems at all.
- It is easy to integrate and built the application

Non-Invasiveness

- Non-Invasiveness means it does not affect our code even though some don't want to use the Spring, it remain same .
- At any point of your project you can remove the Spring package you need not do any changes in the code it will automatically managed by spring.

Spring 5 Class

- **Spring core model:**

- It is the basic model in Spring Arch. If we want to develop any app without core model its not possible.
- To develop an app its contains different types of classes and interfaces.
- 1.POJO class
- 2.java been classes
- 3.component class or been classes
- 1)POJO: (pain old java object)
- =>A class compile and execute underlying jdk without help of any third party classes or jar called as **POJO class**.
- 2) Java been classes:
- => A class contains attributes and accusers methods called as java been classes. (getter and setter methods).
- 3)Component class or been classes
- A class contains attributes and methods with business logic to perform some processing called as Component class or been class.

- While developing any kind of app we have to use multiple classes and interfaces.
- Every time we can't use inheritance to reuse the code.
- There are sort of drawback available in inheritance.
- While discussing of strategy Design pattern we will learn.

Spring 6 Class

- Strategy Design pattern
- In strategy Design pattern 3 principles are there:
 - 1. favor composition over inheritance.
 - 2. design interfaces do not create concrete classes
 - 3. open for extension and closed for modification.
- do not use abstract classes with inhering the features.
- There are sort of drawbacks available in inheritance.
- There are two ways to access the features of one class to another class .
- --1.inheritance
- --2.composition

- 1.inheritance
- Most of the time we recommended to use composition over inheritance. B'z in the project it is not possible a class can use only one class feature, it may possible to use multiple classes. So most of the Object prog. Lang. not supporting multiple inheritance.
- 1.)Inheritance means IS-A relationship. IS-A means a child as similar as parent. We can easily replace the parent class to child class easily.
- Its means a class may want to use some feature of the class but when we inherit the class it forces you to use all the features of the particular class.
- Lets see in below diagram...

A

B

C

D

M1()

m2()

m3()

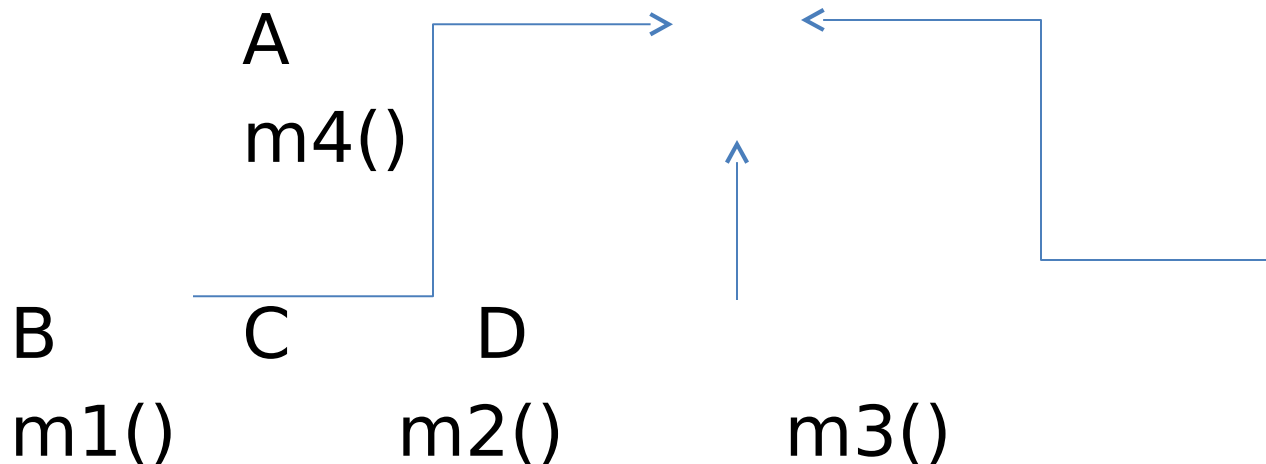
m4()

m4()

m4()

In the above structure m4() method duplicate in all the classes. Better to keep the method in one class and inherit that class to all other classes. Let see below structure how to avoid the duplicate method in the classes.

Spring 7 Class



Here All the classes B,C,D inherit the features of class A using extends keyword. But the problems is if class B and class D want to use method `m5()` which is in class A but the class C don't want to use even though class C forcibly inherit that method in it. It is the drawback of inheritance.

•Ex.

```
Class A{  
    public int m4(){}  
    public int m5(){}
```

```
}
```

```
Class B extends A{  
    public int m1(){  
        A a = new A();  
        a.m4();  
        a.m5();  
    }
```

```
}
```

```
Class C extends A{  
    public int m2(){  
        A a = new A(); // class C don't want to use method m5() B'z of extends C class forcibly  
        a.m4(); //inherit the method m5() in it.  
    }
```

```
}
```

```
Class D extends A{  
    public int m3(){  
        A a = new A();  
        a.m4();  
        a.m5();
```

```
}
```

```
}
```

Spring 8 Class

- **2. Another drawback of inheritance is fragile of classes.**

- Fragile means change in super class may break the all subsequent class those are using this super class.

```
Class A{  
    public int m1(){  
        return 10;  
    }  
}
```

```
Class B extends A{  
    public int m1(){  
        int l =0 ;  
        l = super.m1();  
        return l + 10;  
    }  
}
```

Here class B overriding the method which is in super class and extending the features. And there are other classes also available those are depends on class B. here the problem is that, if super class suddenly change the return type of the method int to float then this change may break class B and B's of B class all respected to B all classes will be break.

- If return type of super class changed then it is not possible to compile and execute the class B'z it's look like two methods present in one class. For Example

```
Class A{  
    public int m1(){}  
    public float m1(){}  
}
```

It not possible to have two method with same name but different return type.

It throws compile tile error.

Spring 9 Class

- 3) Testability with inheritance
- In testability also we have to face the problems if we use inheritance.
- In an org. it may not possible to develop a app by one developer. There are number of developers available to construct the app. A app contains no. of classes if A developer developing a app on car there are many classes are available in the app. So there are two classes Car class and Engine class Car class depends on Engine class. If Car class has completed the implementation of the Car class and he has to wait for the Engine class to complete, but in company we cant wait for any other class we have to create a mock class which is act as an Engine class. By help of MockEngine class we can perform the testability of the Car class.
- Lets see the below program to understand

```

Class Car extends MockEngine {
    public int drive(Engine mg){

        int i = mg.start();
        if(i==0){
            sop("oops! Car failed to start, plz retry");
        }else if(i==1){
            sop("car started on automatic mode");
        }else if(i==2){
            sop("car started on manual mode");
        }
    }
}
Class FailuerMockEngine extends Engine{
    public int start(){
        return 0;
    }

}
Class AutomaticMockEngine extends Engine{
    public int start(){
        return 1;
    }

}

Class ManualMockEngine extends Engine{
    public int start(){
        return 2;
    }

}
Class maindemo{
    Public static void main(Stirng[] args)
    {
        Engine e = new FailurMockEngine();

    }
}

```

- **Composition:** Composition means HAS-A relationship. Composition is the collection of multipart of the class or object called as composition.
- Composition means combination of different part of the entity.
- To overcome the all above problems we use composition. The first principle of the Strategy design pattern is favor composition over inheritance.
- It will make your class free from all the above problems.
- **Advantages of composition**
- Using composition we can avoid the fragility , make efficient testability, access only required features from the parent class.
- Using composition we can use multiple classes in our class.
- Composition more efficient then the inheritance.
- But there are also have some drawbacks in composition.

- Composition make our class tightly coupled.
- Means one class is completely depends on other class. Take an ex.

```
Class A {  
    public int m1(){  
        return 10;  
    }  
}
```

```
Class B  
{  
    private A a;  
    public int m1(){  
        a= new A();  
        int l =0;  
        i=a.m1();  
        return l + 100;  
    }  
}
```

In the above class B completely depends on class A. we are creating an composition relation between two classes but we are tightly coupling these two classes.

If class B is not there means there is no use of class A . Here class A directly talking class B.

B'z of composition we make our classes as tightly coupled. Is it not a good programming practice.

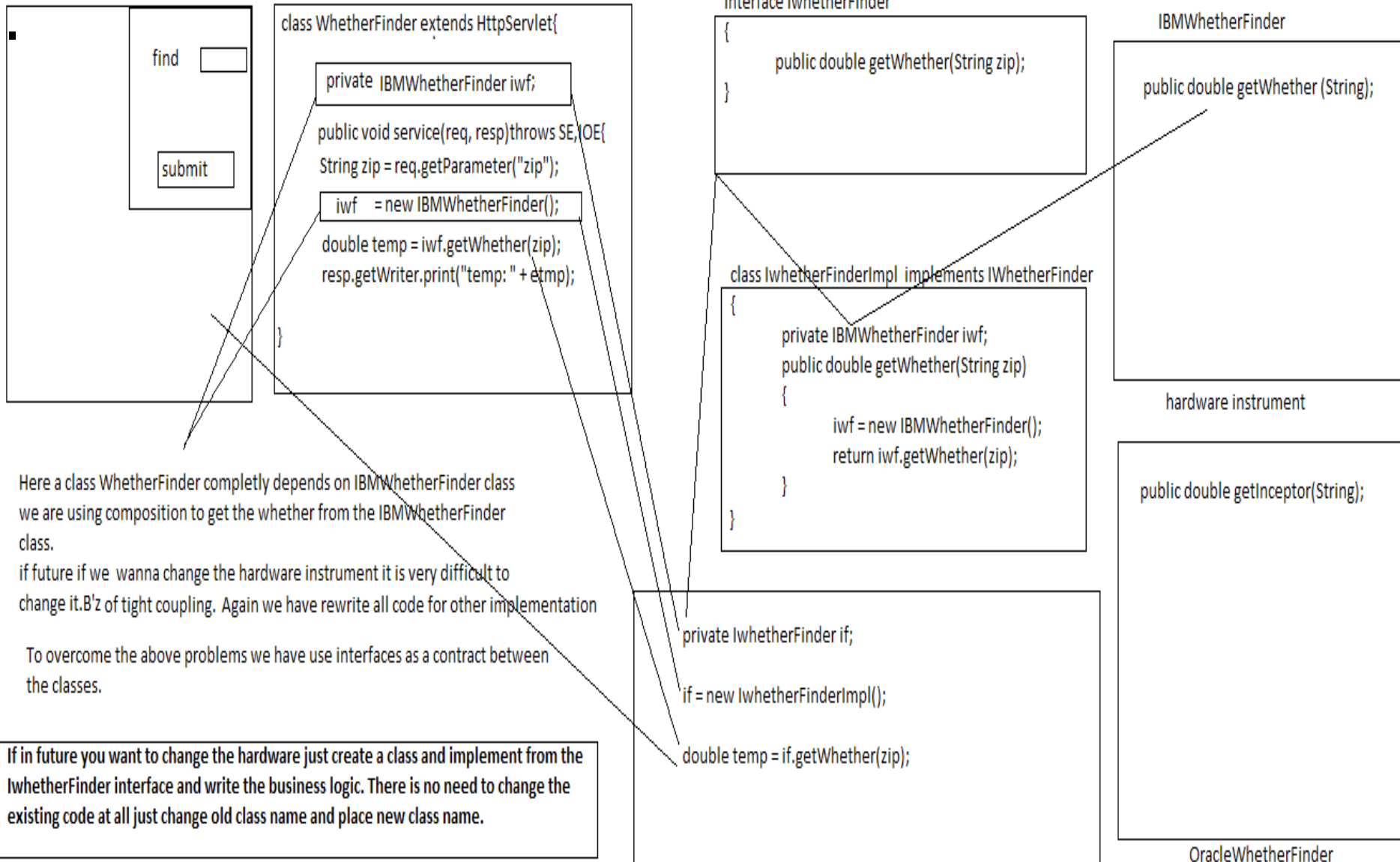
- Means two concrete classes cant completely depends on each other it lead to high maintenance and high changes in the app.
- B'z of the Design interfaces do not use concrete classes. This is the second principal of Strategy design pattern.

By Mr.Sachin Gaikwad

Spring 10 Class

- Read the editplus file which contains information about
- Why interface only used to remove the tight coupling between classes? And what are the drawback if we use concrete class or abstract class?

Spring 11 class



Realtime design to make our classes loosely coupled and it is called as Adapter Interface Design pattern

By Mr. Sachin Gaikwad

- To develop a app if we use composition as part of our application then it must and should use interface as contract or service provider.
- without service provider if we develop a app then it lead to many problems in the future.
- We cant manage our application B'z our app. Is tightly coupled.
- It may lead to huge loss to the client.
- As per the above figure we can guess how much it is important.
- B'z of interface we easily change our code and migrate with other classes.
- There are so many advantages are available as we discussed in our previous classes.

Spring 12 class

- **The third principal of strategy design pattern is 3.=>open to extension of the code and close for modification.**
- In the above example we implemented a new class which implements the IwhetherFinder interface and talking to the OracleWhetherFinder class.
- If in future again we wanna shift from OracleWhetherFinder class to IBMWhetherFinder class then there is no need to change in the existing class just create one more class and implements the IwhetherFinder interface and write the logic to talk with IBMWhetherFinder.
- And make that class non-modified by using final keywords.
- Lets see the ex. below

```

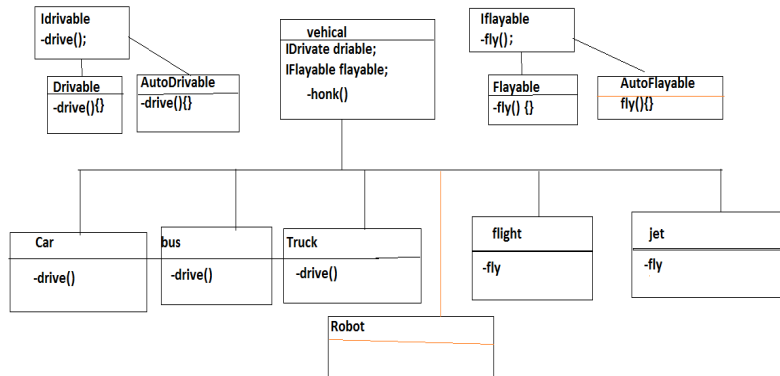
Final Class IBMWhetherFinderImpl implements IwhetherFinder{
    private IBMWhetherFinder iwf;
    double getWhether(String zip){
        iwf = new IBMWhetherFinder();
        Stirng zipNo = zip;
        String temp=iwf.getWhether(zipNo);
        resp.getWriter.print("temp :"+temp);
    }
}

```

- To switch from OracleWhetherFinder to IBMWhetherFinder just change the instantiation object from OracleWF to IBMWF.
- The above class now final means no one can extends this class and override the IBMWhetherFinderImpl class for replacements.

That whats the principal we are using here **open for extension but closed for modification.**

Spring 13 class



- Just zoom and see the diagram.
- How we distributed the classes how we used the strategy design pattern.

Spring 14 class

- In this session we discussed about the above diagram and how we used strategy design pattern.
- What is the use of strategy design pattern?
- What are the problems present in Inheritance?
- What are the problems present in Composition?
- What are the principles present in SDP?
- Why do we use SDP?
- Just reviews the all topics prior to the above questions.

- Just see the another example which use the Spring code and Strategy design pattern to manage the dependency.

There is class MessageWriter which is used to take input and print the output for the uses. It contains a method writeMessage(string message).

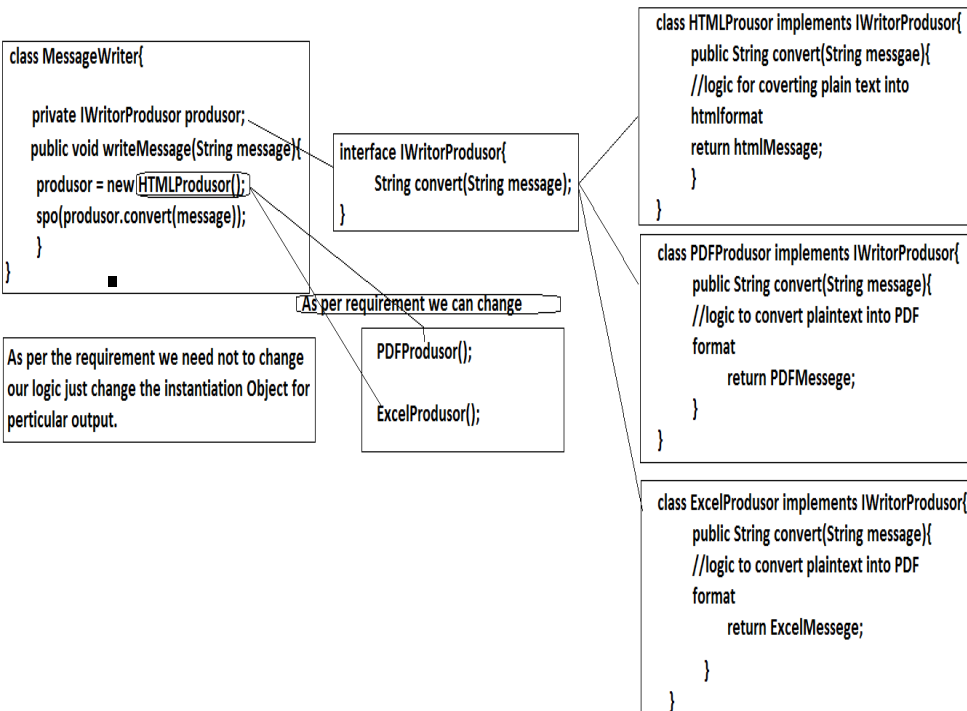
WriteMessage() method take input and produce the output in different format.

```
Class MessageWriter{  
    void writeMessage(String message){  
        String htmlMessage = "<html>" + message + "</html> ";  
        sop(htmlMessage);  
    }  
}
```

But the problems is user never want one format data he can approach for html format , excel format, PDF format...etc.

So we can't develop one single class for HTML, PDF, excel, ..etc.

- If we take one class and written code for HTML after period of time user can approach for PDF or Excel.
- So better to use a Strategy Design Pattern to develop such kind of application.
- Take an interface with one `convert(string message)` methods which can implements all the classes and provide implementation for that interface by writing respective logic in it.
- For better understanding Lets see



Spring 15

- As per the strategy design pattern we developed above example. But using SDP we can't develop our application completely loosely coupled.
- **There are two problems are generated.**
- **1.** while instantiating an object we are using concrete class which can affect our application, if we change an object from one concrete class to other concrete class we have to change all code which is concern to that concrete class.
- Example if HTMLMessageProduser is the concrete class and there may be other classes which are depends on the HTMLMessageProduser.
- If we change that instantiation object from HTMLMessageProduser to PDFMessagePraduser then it may impact on our application.
- For seeing it is one line code only but it impact several classes.

- 2. while instantiating an object we have to know all the information about the corresponding class.
- For example

```
Class A{  
    A(B a){}  
}  
Class B{  
    B(C c){}  
}  
Class C{  
    C(){}  
}
```

In the above example if I want to create an object of class A then how we'll create lets see

We can not create directly **A a = new A();** B's **A class** constructor want Object of **B class** and if want to create an object of **class B** the we want object of **class C**.

Then the procedure is

```
C c = new C();  
B b = new B(c);  
A a = new A(b);
```

- Lets see the below diagram to

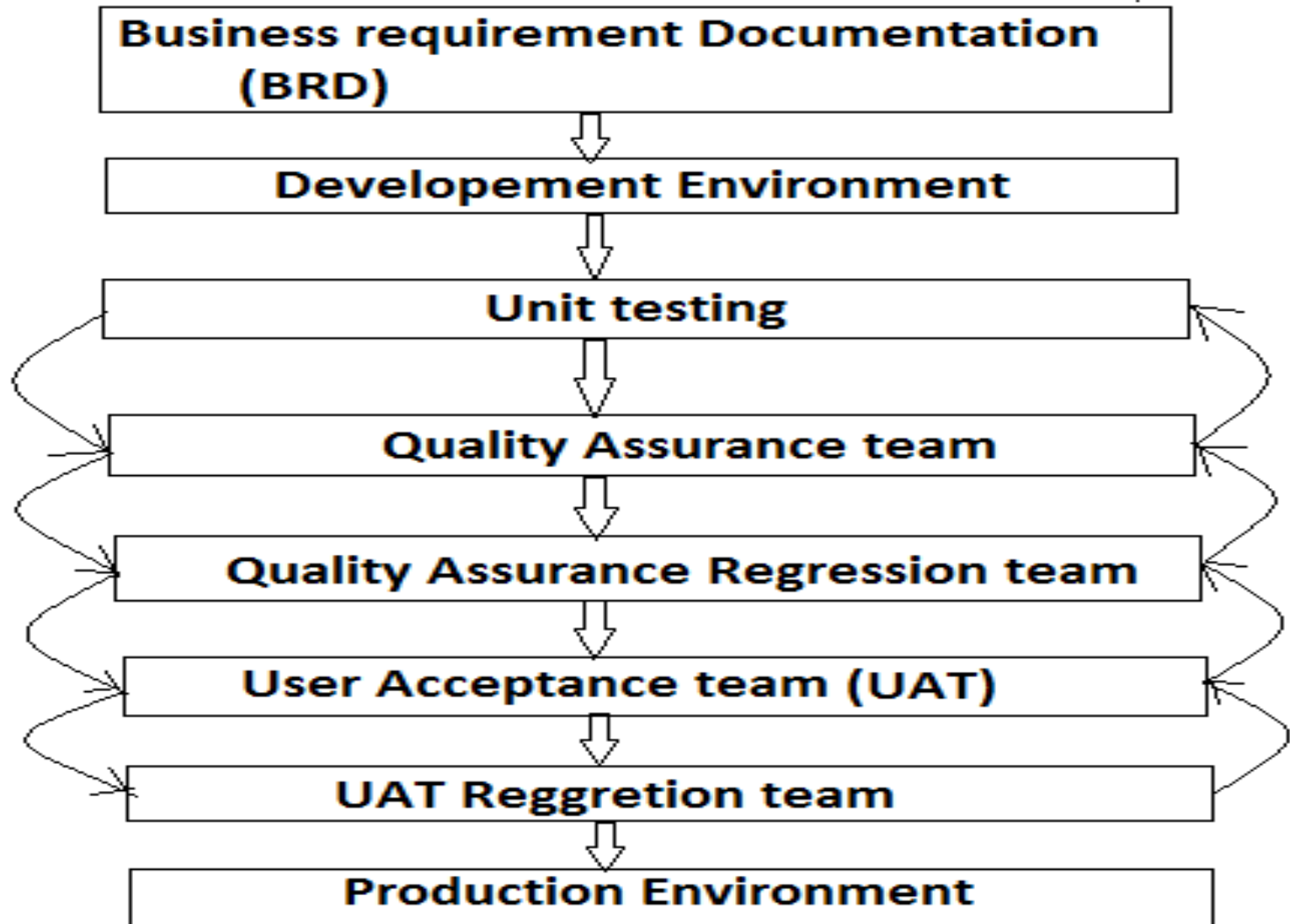


fig. Real time project flow in the Org.

- **BRD(Business Requirement Documentation)**
- It is the first phase where client will give the all requirement.
- Org. people gather all the data in the form of documentation.
- To finalize the document there are different techniques are used.
- Before the development environment there should be a perfect design, perfect architecture, finalize the budget, time, required developer and so on.
- Every one has to follow the BRD only to meet the determination of the application.
- Single mistake may lead big problems in the project, understanding the BRD is very important.
- After the design only project given to the development environment.

- **Development Environment**

- DE is the place where Org. will provide the system to the all the developers to develop an application.
- Developer job is to follow the design all develop an application. After completion of app. Developer has to test it, this procedure called as unit testing.
- Developer has to write some test cases also for understanding to he other people.
- After unit testing application will give to the QAP(Quality Assurance people), QAP people going to check whether any bug are available in the project, if bug are found by QAP then they report the DE to fixed the bugs and asking progress report.
- Until and unless they get zero bugs they will repeat the above procedure to remove the bugs.

- **Quality Assurance Regression**
- After the QAP there is other phase will again check whether application will contains any bugs. If bugs are available again the same procedure will done by QAR phase.
- **User Acceptance Team(UAT)**
- It will perform business operation to check whether our application working as per client requirement or not. By help of BRD they can analysis the application.
- After UAT again there is a phase called **UAT Regression** which cross checks all the requirement, and so on.

- There are numbers of shortcut keys are available
- 1) **Ctrl + T** for finding class even from jar
- 2) **Ctrl + R** for finding any resource (file) including config xml files
- 3) **Ctrl + 1** for quick fix
- 4) **Ctrl + Shift + O** for organize imports
- 7) **Ctrl + O** for quick outline going quickly to method
- 9) **Alt + right** and **Alt + left** for going back and forth while editing.
- 12) **Alt + Shift + W** for show in package explorer
- 13) **Ctrl + Shift + Up** and down for navigating from member to member (variables and methods)
- 15) **Ctrl + k** and **Ctrl + Shift + K** for find next/previous
- 24) Go to a type declaration: **F3**, This Eclipse shortcut is very useful to see function definition very quickly.

- **Eclipse Shortcut for Editing Code**

- These Eclipse shortcuts are very helpful for editing code in Eclipse.
- 5) **Ctrl + /** for commenting, un commenting lines and blocks
- 6) **Ctrl + Shift + /** for commenting, un commenting lines with block comment
- 8) Selecting class and pressing **F4** to see its Type hierarchy
- 10) **Ctrl + F4** or **Ctrl + w** for closing current file
- 11) **Ctrl+Shift+W** for closing all files.
- 14) **Ctrl + I** go to line
- 16) Select text and press **Ctrl + Shift + F** for formatting.
- 17) **Ctrl + F** for find, find/replace
- 18) **Ctrl + D** to delete a line
- 19) **Ctrl + Q** for going to last edited place

- **Miscellaneous Eclipse Shortcuts**

- These are different *Eclipse keyboard shortcuts* which doesn't fit on any category but quite helpful and make life very easy while working in Eclipse.
- 20) **Ctrl + T** for toggling between super type and subtype
- 21) Go to other open editors: **Ctrl + E**.
- 22) Move to one problem (i.e.: error, warning) to the next (or previous) in a file: **Ctrl + .** For next, and Ctrl + , for previous problem
- 23) Hop back and forth through the files you have visited: **Alt + ←** and **Alt + →**, respectively.
- 25) **CTRL+Shift+G**, which searches the workspace for references to the selected method or variable
- 26) **Ctrl+Shift+L** to view listing for all Eclipse keyboard shortcuts.
- 27) **Alt + Shift + j** to add javadoc at any place in java source file.
- 28) **CTRL+SHIFT+P** to find closing brace. Place the cursor at opening brace and use this.
- 29) **Alt+Shift+X, Q** to run Ant build file using keyboard shortcuts in Eclipse.
- 30) **Ctrl + Shift + F** for Autoformatting.

Spring 16

- As we discussed in the above theory what are the problems we going to face using SDP.
- To overcome the above problem we have to use **Factory Design Pattern.**

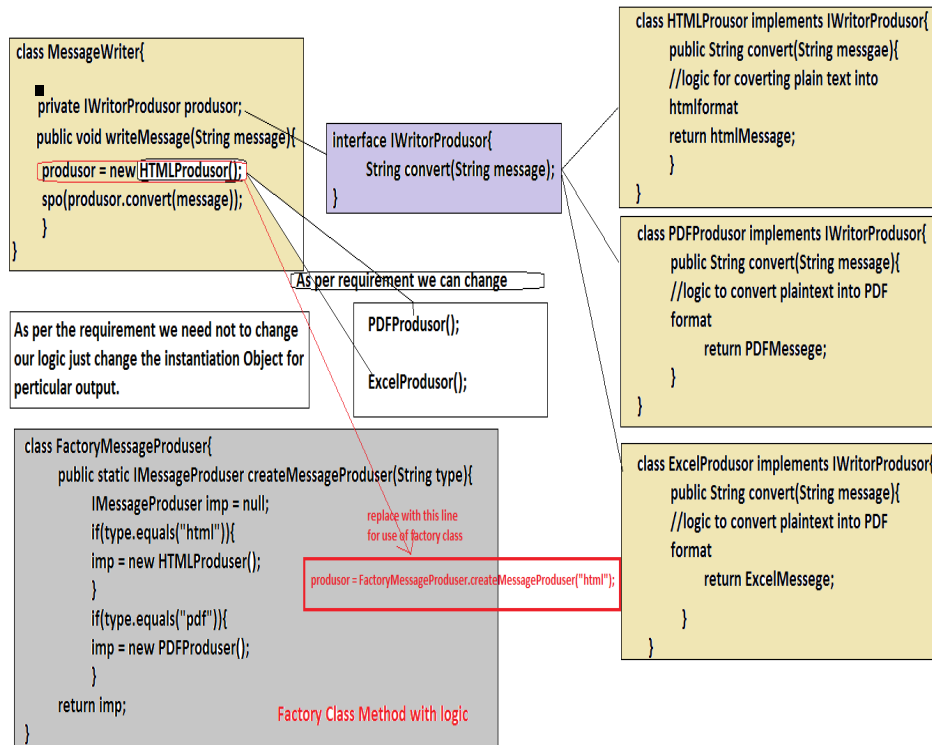
- **Factory Design Pattern:**

FDP use to remove the complexity while creating an Object of the respective class. It hide the complexity and it will allow to create an object without knowing the internal details of the class and respective belongings.

- As per the above examples we used Strategy Design Pattern, now we will use FDP to solve the problem. Just create a class which can able to return the object of particular class.

- As per the above usecase we going to create Factory class.

```
Class FactoryMessageProducer{  
    public static IMessageProducer  
createMessageProducer(String type){  
    private IMessageProducer = null;  
    if(type.equals("html"))  
        IMessageProducer = new  
HTMLMessageProducer();  
    else if(type.equals("pdf"))  
        IMessageProducer = new  
PDFMessageProducer();  
    return IMessageProducer;  
}
```



```

class WhetherFinder extends HttpServlet{
    private IBMWhetherFinder iwff;

    public void service(req, resp) throws SE, OE{
        String zip = req.getParameter("zip");
        iwff = new IBMWhetherFinder();
        double temp = iwff.getWhether(zip);
        resp.getWriter.print("temp: " + temp);
    }
}

```

```

interface IwhetherFinder
{
    public double getWhether(String zip);
}

```

```

IBMWhetherFinder
{
    public double getWhether (String);
}

```

```

class IwhetherFinderImpl implements IWhetherFinder
{
    private IBMWhetherFinder iwff;
    public double getWhether(String zip)
    {
        iwff = new IBMWhetherFinder();
        return iwff.getWhether(zip);
    }
}

```

```

hardware Instrument
{
    public double getInceptor(String);
}

OracleWhetherFinder

```

```

class FactoryWhetherFinder{
    public static IwhetherFinder
    createWhetherFinder(String vendors){
        IwhetherFinder iwff=null;
        if(vendors.equals("IBM")){
            iwff = new IWhetherFinderImpl();
        }
        if(vendors.equals("oracle")){
            iwff = new OracleWhetherFinder();
        }

        return iwff;
    }
}

```

Here a class WhetherFinder completely depends on IBMWhetherFinder class we are using composition to get the whether from the IBMWhetherFinder class.

if future if we wanna change the hardware instrument it is very difficult to change it.B'z of tight coupling. Again we have rewrite all code for other implementation

To overcome the above problems we have use interfaces as a contract between the classes.

If in future you want to change the hardware just create a class and implement from the IwhetherFinder interface and write the business logic. There is no need to change the existing code at all just change old class name and place new class name.

```

private IwhetherFinder if;
if = new IwhetherFinderImpl();

double temp = if.getWhether(zip);

```

Factoryclass logic

Realtime design to make our classes loosely coupled and it is called as Adaptor Interface Design pattern

- Factory class contains static method only B'z there is no need to create an object for Factory classes.
- We never use any attributes in factory class so Object is not required.
- Example

Class A{

// int i=10;

public static void m1(){}

}

A a = new A();

Here object 'a' contains attribute 'i' but not m1() methods B'z m1() methods for all the objects which are created in to the class. So it is not possible to place method into every object.

To access m1() method there is no need to create an object with class name we can access it. Make that method static and access by using class name.

A.m1();

- To get more examples just see the spring folder.

Spring 17

- As we used factory class for making our classes completely loosely coupled but physically we solved it but still logically we have to provide the type of requirement to create an object of the particular class .
- We are using
 - `Messageproducer = FactoryMessageProducer.createMessageProducer("html");`
- If we want PDFMessageProducer then we have to manually change the logic.
- Here we are asking to other person to create an object of concern class but we have to tell him which class object he has to create by passing logical name of the class this concept called as **Dependency pulling**.
- We approaching to factory class to create an object of particular class and return to the main class called as Dependency pulling.
- We can solve this problem, don't create any object, even don't ask any one to create an object, whoever want the object let them to create an object and set to our class . Just provide one proven to get the object from other class.
- Lets see in the below example


```

class MessageWriter{
    ■ IMessageProducer messageProducer= null;
    public void writeMessage(String message){
        String cmessage = messageProducer.convert(message);
        Sop(cmessage);
    }
    public void setMessageProducer(IMessageProducer messageProducer){
        this.messageProducer = messageProducer;
    }
}

```

main class

```

interface IMessageProducer{
    public String convert(String message);
}

```

```

class HTMLMessageProducer implements IMessageProducer{
    public String convert(String message){
        //implementation logic
    }
}

```

```

class PDFMessageProducer implements IMessageProducer{
    public String convert(String message){
        //implementation logic
    }
}

```

```

class XMLMessageProducer implements IMessageProducer{
    public String convert(String message){
        //implementation logic
    }
}

```

```

class SPTest{
    p s v m (String [] args){
        IMessageProducer messageProducer = null;
        MessageWriter messageWriter = null;
        messageProducer =
        FactoryMessageProducer.createMessageProdicer("html");
        messageWriter = new MessageWriter();
        messageWriter.setMessageProducer(messageProducer);
        messageWriter.writeMessage("welcome to S D P");
    }
}

```

SPTest class

Factory class

```

class FactoryMessageProducer{
    public static IMessageProducer createMessageProducer(String type){
        IMessageProducer messageProducer = null;
        if(type.equals("html")){
            messageProducer = new HTMLMessageProducer();
        }else if(type.equals("pdf")){
            messageProducer = new PDFMessageProducer();
        }else if(type.equals("xml")){
            messageProducer = new XMLMessageProducer();
        }
        return messageProducer;
    }
}

```

- SPTTest class is going to take an object from concern MessageProducer and he is passing that object to the MessageWriter, but the problem is if SPTTest want pdf, after that html, xml then SPTTest has to change there logic to get particular object.
- SPTTest has to pass which MessageProducer he want. He has to pass the type of producer. means still some amount of coupling is there.
- We are unable to make our classes completely loosely coupled even by using SPTTest class. Still there is

Spring 18 and 19

- As we discussed above SPTest also having problem. To make our application completely loosely coupled we have to use properties file. If we use properties then there is no need to create an object in classes just read the class name from the properties file and create an object.
- Properties file is the collection class which going to store character based data in the form for key and value.
- Key represent the identity of the value.
- Properties is a Map type class B'z it going to store data in the form of key and values.

For example:

Name=com.sp.sachin_gaikwad

Mob=8125060647

Here Name and Mob represent the key and '=' is the separator which split the key and value.

'com.sp.sachin_gaikwad and 8125060647 are the values.

- There is some sort of procedure to create a properties file and read a file and use it.
- To get value from properties file we should have provide the key as input to the file then only it will return the value by some sort of procedure.
- Assume **AppProp.properties** file.

Lets see the example

```
Properties prop = new Properties();  
FileInputStream fis = new FileInputStream(new  
File("a:\\AppProp.properties")) ;  
Prop.load(fis);  
string value=prop.getProperty(key);  
sop(value);
```

- Using above procedure we can deals with properties file.
- To create an object use below procedure

```
Object obj = Class.forName(className).newInstance();  
Let see the example in eclipse.
```

Spring 20

- While making our application loosely coupled we used properties file but still we are lacking to achieve it. In properties file we have provided absolute path as the property file location. If tomorrow our project location going to change from D: drive to E: drive then our project will not work.
- Once after the compilation we unable to change the code which is in the AppFactory.java class .
- So it is not possible to run our application in other systems.
- To make our application as global executable use relative path.
- After compilation our .class files going to stored in bin directory which is common for storing .class files.
- Eclipse IDE will take care of compilation procedure and placing .class files into bin directory and it also place all extra file which is in the scr folder.
- So as programmer job is to give the relative path in to the project.

- Java has provided a method which is able to copy the relative path and give to the properties to load the data.
- `<class_name.class>.getResourceAsStream("../../../<file_name.extention>");`
- Ex.

`InputStream is=null;`

`is=AppFactory.class.getResourceAsStream("../../../AppProperties.properties");`

Just see the application in eclipse with the name FDAppUsingProperties

- **public class AppFactory {**
- **public static Object createObject(String lclassNm) throws IOException, ClassNotFoundException, InstantiationException, IllegalAccessException{**
- Object obj= **null**;
- String className = **null**;
- Properties properties =**null**;
- //FileInputStream fis =null;
- InputStream is
- =AppFactory.**class.getResourceAsStream("..\\..\\..\\AppProperties.properties");**
- //fis = new FileInputStream(new File("A:\\Spring Data\\Spring WorkPlace\\FDPusingProperties\\src\\AppProperties.properties"));
- properties = **new Properties();**
- //properties.load(fis);
- properties.load(is);
- **if(properties.size()<0){**
- **throw new ClassNotFoundException("Unable to initialized your file");**
- **}else if(properties.containsKey(lclassNm)==false){**
- **throw new ClassNotFoundException("Unable to find key in the file "+lclassNm);**
- **}**
- className = properties.getProperty(lclassNm);
- obj = *Class.forName(className).newInstance();*
- **return obj;**
- **}**
- **}**

Spring 21 & 22

- Using property file we can manage the dependency between the classes but there are multiple classes available as part of application so it is not possible to manage by using the properties file.
- Using AppFactory we will manage dependency but we have to write more number of lines of code .
- So as programmer we will go to write boiler plate logic in every application while developing .
- To manage it effectively we have to give our classes to the spring as ask to the spring to manage the dependency.
- Spring has provided a spring-bean-configuration-file , which is used to configure our classes and to manage the dependency between them.
- Spring-bean-configuration-file root tag is `<beans></beans>`, so we have to configure our classes into the bean-configuration file. To configure use `<bean>` is one more tag which having two properties `id` and `class`.

- Let see the example to understand

```
<beans>
```

```
  <bean id="first"  
class="com.ss.beans.FirstClass"></bean>
```

```
  <bean id="second"  
class="com.ss.beans.SecondClass"></bean>
```

```
</beans>
```

It is the convention to write the spring-bean-configuration-file. We have to follow the syntax and rule of the dtd and xsd.

By help of **id** we going to get particular class object.

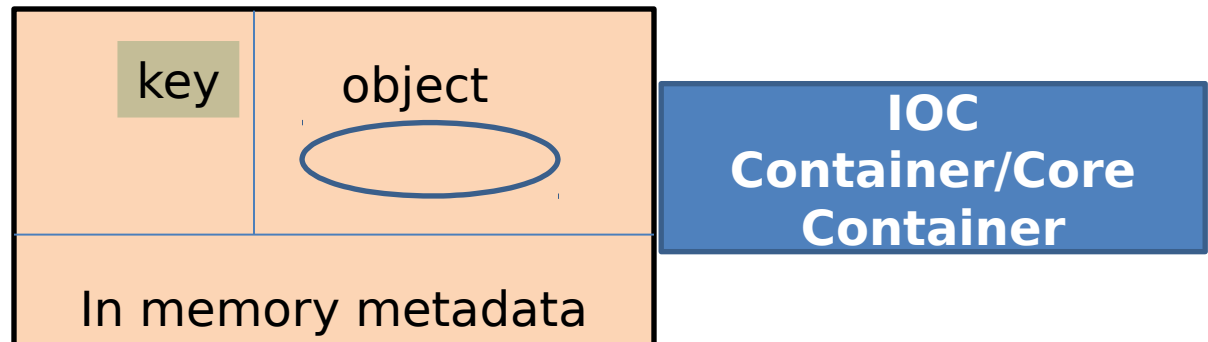
And **class** property represent the actual class path location for creating an object.

- If we want to refer an object of the one class to the other class then another tag has been used i.e. **property** tag.
- It contains name and ref attribute, name is used for

Spring 24

- **IOC principle:**

- It is used to managing the dependency between the objects.
 - It collaborating an object and managing the lifecycle of the object.
- IOC is the logical memory in the JVM memory.
- When we going to execute the XMLBeanFactory() method this memory will be created by XMLBeanFactory to keep an object .
- IOC also called as Core Container.
- IOC container memory having two parts
 - In memory METADATA
 - Empty(for storing the object (key=value format)).
- XMLBeanFactory() method will take an object resource from the ClassPathResource() and create the object and keep it in to the IOC Container.



- IOC principle is not the part of the Spring.
- It is categorized into two ways
 - 1).Dependency pull
 - i).dependency lookup
 - li).contextual dependency
 - 2).Dependency Injection
 - i). Constructor injection
 - li).setter injection

1).Dependency pull: A class asking or taking help from other class to perform a task called as dependency.

As we seem in the above example how we going to pull the object from the factory classes .

i).Dependency lookup:

A class is completely depend on the other class object is called as dependency lookup.

A programmer want to persist data into the database but he can't write logic in his program B'z it is J2EE application. For persisting the data we have to take connection from the JNDI registry and then we can persist the data into database.

What is JNDI registry?

It is global memory where sharable data can be placed. To exposing sharable resources to the user or client we going to use JNDI Registry.

- Before placing connection object into JNDI registry first we have to create an object and place into the connection pool.
- Using DataSource object we can provide the required info to the application server and server going to create the connection with database. And it will place into the connectionPool.
- So any programmer going to get the connection from the connection pool.

```
Class person{  
    public void savePerson(string id ,String Name){  
        InitialContext ic = new InitialContext();  
        DataSource ds =(DataSource)ic.getConnection("/JNDI Registry  
Name");  
        .....  
        .....  
    }  
}
```

In the above example a **person class** want to store data into the database but without connection a person class can't do anything. To get the connection person class has to **lookup** the **JNDI registry** to get the connection.

ii). Contextual dependency lookup

Here also we are taking a object from other class or container or runtime environment to perform an operation.

If we want to get the internal details of the servlets, version, absolute path, port No, ip, so we have to use Servlet Context object. But to get the servlet context object first we have to follow some steps. we should have to implements the Servlet interface and override the `init(ServletConfig config)` method.

Lets see the procedure...

```
Class Xservlet extends HttpServlet implements Servlet{
    ServletConfig cong;
    public init(ServletConfig config){
        this.config=config;
    }
    ServletContext context = config.getServletContext();
    .....//logic to get internal details
}
```

By the above example we come to know the contextual procedure to get the context object.

There are certain rules are available and we should have to follow them.

Spring 25

• 2).Dependency Injection

- In dependency pull a class is totally depends on the other class or other runtime environment. As we learnt in previous classes in factory design pattern how one class is depends on other class.
- To make our classes completely loosely coupled ,we have to use dependency injection concept from IOC principle.
- While performing an dependency injection there are two components are compulsory.
- And one component is depends on other components.
- One is considered as **target** and another one is **dependent**.

i). Setter injection :

In setter injection a dependent object is going to inject to the target component.

Class B is going to inject with class A is called as setter injection.

Setter injection we achieve by **property** tag which is provided by the **spring bean configuration file**.

```
<bean id="" class="">
```

```
<property name="<attribute_name>" ref="<class_name>">
```

```
</bean>
```

- In spring bean configuration file **property** tag act as a **setter** method. Here the bean class itself going to inject the object to the target class.
- B'z of that we called as setter injection.
- **ii). Constructor injection**: It is also same as setter injection only a dependent class is going to inject the object to the target class.
- But here we are using different tag to inject the object.
- In spring bean configuration file under bean tag there is another tag called **constructor-arg** which is used to inject the object of another class to the target class.

```
<beans>
```

```
<bean id=" " class=" ">
```

```
<constructor-arg ref=" <class_name>">
```

```
</bean>
```

```
//..other bean logic
```

```
</beans>
```

- IOC going to support all the four ways of the managing the dependency.
- IOC(Inversion of control)

What do you mean by IOC ?

A

B

dependency pulling

In traditional ways we used to use dependency pull to get the object of other class.

A class going and getting the object of B, means B is the target class for A and A totally depends on the class B. Here A is pulling the object and performing the task.

A

B

dependency Injection

The process of setting the object of one class to another class called as injection.

In dependency injection total reverse procedure is there. Here A and B are two components which sit in the container and B is going and injecting the object to class A and A is dependent on B. Here A is the target for the B class. totally reverse procedure from pulling concept.

Inversion of control : It is totally about the making our application completely loosely coupled by using dependency pull and dependency injection. dependency injection is reverse of dependency pull.

Q. All ready setter injection is there so what is the need for constructor injection?

"The process of setting the object of one class to another class called as inversion of control".

"The process of the injecting object to the target class called as inversion of control".

Spring 26

- As per the previous class we learnt that what is setter injection and what is constructor injection.
- In case of setter injection always a depends class is injected after the target class object created.

```
A{//A is the target
```

```
B a;
```

```
A(){
```

```
b.m2();
```

```
}
```

```
setB(B b){}
```

```
}
```

```
B{//dependent class inject with target class
```

```
}
```

```
A a = new A();
```

```
B b =new B();
```

```
a.setB(b);
```

In the above example A is the target and B is the dependent class, B is going to inject with the class A .

But the until creating the object of target class we can not use B. means first target class object has to created after the dependent class. Here dependent component is depends on the target class .and we can not access dependent class properties in the constructor of the target class.

- But in constructor injection while creating the object of target class we have to inject the dependent class to the target class.
- Means we can use dependent class properties into the target class easily.

```
A{//target class
```

```
A(B b){
```

```
b.m2()
```

```
}
```

```
}
```

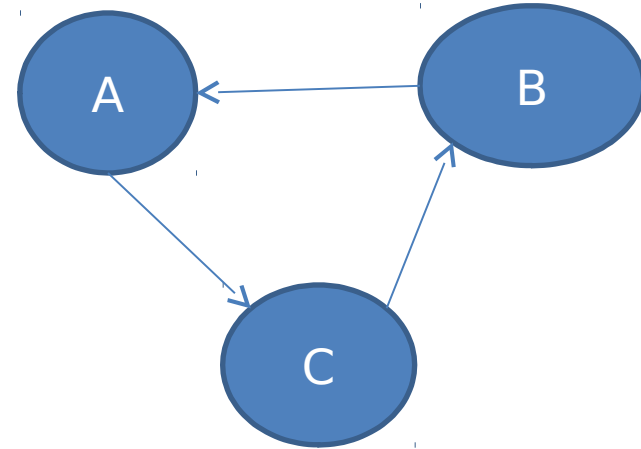
```
B{ }//dependent class
```

```
A a = new A(new B);//inject B while creating A class Object.
```

- Without B class object we unable to create the object of A.
- In constructor injection, we can use the dependent properties into the target class.

- 2.) We can not handle circular dependency using the constructor injection B'z its leads to he deadlock. One is totally depends on other one in circular manner.

```
A{  
A(B a){}  
}  
B{  
B(C c){}  
}  
C{  
C(A a){}  
}
```



In the above example we can't create the object of A class without B class and we can't create the object of B without C and also Without A we can't create the object of C class so it is circular dependency, and we can't handle by using constructor injection.

- But we can handle circular dependency using the setter injection .

```
A{  
  A(){}  
  setB(B a){}  
}  
B{  
  B(){}  
  setC(C c){}  
}  
C{  
  C(){}  
  setA(A a){}  
}  
A a = new A();  
B b = new B();  
C c = new C();  
a.setB(b);  
b.setC(c);  
c.setA(a);
```

- As per the above example we can understand the use of setter injection.

- Spring support four kinds of list collection injection dependency.
 - List
 - Set
 - Map
 - Properties

Spring 28

- Problems with List collection injection dependency and how to use the **<util> tag** and what are the different attributes are available with util tag.

Spring 29

- constructor confusion and different attributes are used to resolve the constructor confusion, i.e
 - Type attribute
 - Index attribute
 - Name attribute

Spring 31

- What is mean by Inner Bean and way we use inner bean concept?
- What is the bean inheritance ?

What is bean aliasing? How to work with bean aliasing?

Spring 2.0 > name is the attribute are used to give the aliases to the beans

Spring 2.0 < alias tag used to give the alias name to the bean.

Way two attributes are there for aliasing the bean?

Spring 34

- How to inject null to the constructor?
- Importance of bean autowiring?

```
class A{  
private B b;  
//setter  
}  
class B{  
}
```

```
<bean id="a" class="A"/>
```

```
<bean id="b" class="B"/>
```

- In this case IOC container can only create the objects ,but it can't manage the dependencies, to manage the dependencies we have to provide **additional configuration details**.

```
<bean id="a" class="A">
```

```
<property name="b" ref="b"/> </bean>
```

```
<bean id="b" class="B"/>
```

- Instead of we provide the additional configuration details to manage the dependencies, IOC container can manage dependencies using autowiring, by default autowire is turned off.

```
<bean id="a" class="A" autowire=MODE>
```

```
<bean id="b" class="B">
```

- We have to use autowire at bean level, and MODE means how the IOC will manage the dependencies, there are 4 MODEs are there, they are

- 1)byName

- 2)byType

- 3)constructor

- 4)autodetect [it is deprecated in Spring 2.5 and not available from Spring 3.0]

- 1)byName:

- When we set autowire="byName", the IOC container will manage the dependencies via **setter**, after creation of the target class, it will check the autowire mode, if it is byName then it holds the target object and then checks the target class attributes which having setter and then it checks the attribute name with the bean id, if both are matched, then it will create the object of dependent bean and pass it as an argument to the setter, then it returns the target class object.

Ex:-

```
class Car{  
private Engine engine;  
//setters  
}
```

```
class Engine{  
}
```

```
<bean id="car" class="Car" autowire="byName"/>
```

```
<bean id="engine" class="Engine"/>
```

- Here, IOC will create the car(Target) object, and then holds it, then it creates the object of Car class attribute name which having setter i.e., engine, and passes as parameter to the Target class.

2)byType:

- In this case it will check the attribute type which is having the setter and matches with the bean class type.

```
<bean id="car" class="Car" autowire="byType"/>
```

```
<bean id="engine" class="Engine"/>
```

```
<bean id="engine1" class="Engine"/>
```

- In this case IOC will be ambiguous, why means both the beans class type is Engine, to resolve this, we use **autowire-candidate="false"**, by default it is **"true"**.

```
<bean id="car" class="Car" autowire="byType"/>
<bean id="engine" class="Engine" autowire-
candidate="false"/>
<bean id="engine1" class="Engine"/>
```

- Here the bean tells the IOC that , when you manage the dependencies automatically don't consider me.

3)constructor:-

- It is also matches attribute type and bean class type, but for those attributes which are having the Constructor.(Constructor Injection).
- If multiple constructors are there, then it gives the highest priority to the maximum parameter constructor, if one of the parameter is not configured bean , then the priority is second maximum parameter constructor. If every constructor is having same parameters then highest priority given to the first configured constructor in the class in top to bottom manner.

```
class Car{
private Engine engine;
private Wheel wheel;

Car(Engine engine){}
Car(Wheel wheel){}
Car(Engine engine,Wheel wheel){}
}
```

```
<bean id="car" class="Car" autowire="constructor"/>
<bean id="engine1" class="Engine"/>
<bean id="wheel1" class="Wheel"/>
```

Here, we configured every bean, so highest priority goes to the 2 parameterized constructor

```
<bean id="car" class="Car" autowire="constructor"/>
<bean id="engine1" class="Engine"/>
```

Here, we haven't configured every bean, so priority goes to Engine Constructor, Why means, it's configured first

Spring 37,38

- **Nested BeanFactory:** Means one beanfactory inside another beanfactory.
- There are some situation we have to use such kind of concept. For example if car is there but without engine car cannot run. Means to drive a car engine must be there.
- Another example if motor is there without chain motor can not work.....
- Actually Nested BeanFactory means we can create one IOC container inside another IOC container.
- A IOC container going to inject into another IOC container called parent IOC container, and another one called as Child IOC Container.
- There parent IOC container going to inject inside the child IOC container.

- First configure parent class into the spring bean configuration file and give Parent.xml as name.
- Now Configure child classes into the spring bean configuration file and give Child.xml as name.
- After creating both configuration file, now create parentBeanFactory, now create childBeanFactory and pass parentBeanFactory reference to the ChildBeanFactory.
- For example..

```
public static void main(String[] args) {  
BeanFactory parentFactory = new XmlBeanFactory(new  
ClassPathResource("com/nbf/common/parent-  
bean.xml"));  
BeanFactory childeFactory = new XmlBeanFactory(new  
ClassPathResource("com/nbf/common/child-  
bean.xml"),parentFactory);  
  
.....  
}
```

- Example

Parent-bean.xml

```
<bean id="chain" class="com.nbf.beans.Chain">
  <property name="id" value="10" />
  <property name="type" value="metal" />
</bean>
```

Child-bean.xml

```
<bean id="motor" class="com.nbf.beans.Motor">
  <property name="chain" ref="chain">
    <!-- <ref parent="chain"/> -->
    <!-- <ref local="chain"></ref> -->
  </property>
</bean>

<bean id="chain" class="com.nbf.beans.Chain">
  <property name="id" value="20"/>
  <property name="type" value="Rubber"/>
</bean>
```

There are three attributes are available to

- **Ref** : it will check first in local context, if required bean available in local then it will inject local, if it is not then it will check into parent-bean.xml.
- **<ref local="chain">**: it will check in local only.
- **<ref parent="chain">**: it will check in parent -bean.xml only.

Spring 39

- **IDREF**
 - To make our classes loosely coupled there are two ways
 - Dependency pull
 - Dependency injection
- We learnt how to use dependency injection make our classes loosely coupled, but there are some limitations are available with the dependency injection.
- So to make our classes loosely coupled how can use dependency pulling concept. One of the concept is **IDREF** which can make classes loosely coupled.
- IDREF word itself describes, it refers to the id of another bean.
 - By using IDREF attribute of spring we can make our classes loosely coupled,

```
public class Car {  
  
    private IEngine engine;  
    public void run()  
    {  
        engine.start();  
        System.out.println("Car is running.....");  
    }  
    public void setEngine(IEngine engine) {  
        this.engine = engine;  
    }  
}
```

- There are two ways available to get the values from other class
- 1). Make our method to get the value from other class

For example

```
public void m1( int i){}
```

But it is specific to the method only.

- 2).Declared Attribute at class level

if a value used by through out the class then declared that variable as class level and initialize into the constructor, for example

```
class A{  
    private int i;  
    A(int i){  
        this.i=i;  
    }  
}
```

we know the above procedure but there are some problems available.

1. `<bean id="car" class="com.idr.beans.Car">
 <property name="engine" ref="suzukiengine"></property>
</bean> -->`
2. `<!-- <bean id="car" class="com.idr.beans.Car">
 <property name="beanId" value="yamahaengine"></property>
</bean> -->`

- By using dependency injection we can inject the other bean in to the target bean but here i dont want to use dependency injection, rather my target class pull corresponding object from the Spring bean configuration file, we can make our target class to pull the correspond object by two ways
- 1. by declared one attribute in target class and inject a perticular bean id to the target attribute
- 2. by using IDREF attribute
- if we use value="yamahaengine" it may not give a correct context about what kind of value we are injecting. other developer may get confuse, but if we use IDREF it will clearly specify Idref attribute using other bean id for injection.

- So we can configure the bean using property attribute of bean and clearly specify the idref to the bean.

```
<bean id="car" class="com.idr.beans.Car">
  <property name="beanId">
    <idref bean="suzukiengine"/>
  </property>
</bean>
<bean id="yamahaengine1" class="com.idr.beans.YamahaEngine"></bean>
<bean id="suzukiengine" class="com.idr.beans.SuzukiEngine"></bean>
</beans>
```

- Let see the example idref.
- **public class Car {**
- **private String beanId;**
- **public void run()**
- {
- IEngine engine = **null**;
- */*System.out.println(beanId);*/*
- BeanFactory factory = **new XmlBeanFactory(new**
- **ClassPathResource("com/idr/common/application-context.xml"))**;
- engine = factory.getBean(beanId, IEngine.**class**);
- engine.start();
- System.***out.println("Car is running.....")***;
- }
- **public void setBeanId(String beanId) {**
- **this.beanId = beanId;**
- }
- }

Spring 40

- **P and C NameSpaces:**
- It is the shortcut procedure to use the property and constructor attributes into the bean.
- Because there are multiple properties are available into the class, so it is too heavy to configure all the properties using property tag and constructor tag.
- To make it is simple we can use P and C nameSpaces.
- For example
- `<bean id="person" class="com.pc.beans.Person" p:ssn="10" p:name="sachin" p:address-ref="address" p:details-ref="personalDetails"></bean>`
- `<bean id="personalDetails" class="com.pc.beans.PersonalDetails" c:mobNo="+91-8125060647" c:qualification="MCA" c:experience="5 years"></bean>`
- *while taking spring bean configuration file add two additional beans information.*
- `xmlns:c="http://www.springframework.org/schema/c"`
- `xmlns:p="http://www.springframework.org/schema/p"`

- Bean Scope:
- Singleton Design pattern:

A class is going to allow you to create only one object of a class called as singleton class.

- There are several reasons available, why we are creating singleton class.
- In some cases a object or a configuration will be common to whole application.
- If every one going to create the object of common thing then is it duplicating among the application, and we are wasting JVM memory.
- If there is common requirement then create a singleton class which going to share same object through out the application.
- Let see the example of how to create singleton class.

Singleton design pattern

Class DateUtil implements Cloneable{

private static DateUtil dateUtil;

private DateUtil(){}

public static DateUtil getInstance(){

if(dateUtil==null){

 DateUtil dateUtil = new DateUtil();

}

return dateUtil;

}

public Object clone()throws CloneNotSupportedException{

throws new CloneNotSupportedException();

}

While we are dealing with normal application we can use such kind of singleton design pattern.

There are lot more thing available to discuss about singleton design pattern.

- Why constructor must be private ?
- Why we should implements from Cloneable interface?
- Is it create only one object into the JVM or not ?
- Again there are several thing available which talks about singleton class.
- First of all **why constructor should be private**, B'z we are restricting other classes to creating the object of a class. To create a object default constructor is mandatory if we made as private then other people unable to create the object of the class.
- Second one is **why should we implements Cloneable interface**, Actually Cloneable is in object class and it's return type is protected, even we can not implements also no one can clone our class, but in singleton class there is specific reason is available to implements the Cloneable interface.
- Actually my class is not using cloneable interface but in feature there is requirement every class has to provide security to there own classes by extending Authentication class, and here if Authentication class is implements from the Cloneable interface then my singleton class became cloneable. To avoid such kinds of situation cloneable interface implements.

- For Example :

```
class Authentication implements Cloneable{  
    //auth logic  
}
```

```
Class DateUtil extends Authentication{  
    //to avoid such situation we have to  
    implements cloneable interface  
}
```

```
Class DateUtil extends Authentication impl  
Cloneable{  
    public Object clone()throws  
CloneNotSupportedException{  
    throw new CloneNotSupportedException()  
}
```

1).

```
public static DateUtil getInstance(){  
    if(dateUtil == null){  
        dateUtil = new DateUtil();  
    }  
    return dateUtil;  
}
```

In the above code if we use thread concept to get the DateUtil object then there may create multiple object. bz thread going to execute simultaneously and it lead to create multiple object , As we see in the current example.

2).

```
public static synchronized DateUtil getInstance(){  
    if(dateUtil == null){  
        dateUtil = new DateUtil();  
    }  
    return dateUtil;  
}
```

In the above code if we make our method as synchronized then our application performance going to down, bz synchronized method allows only one thread or request at a time until and unless other request has to wait to get the object.

3).

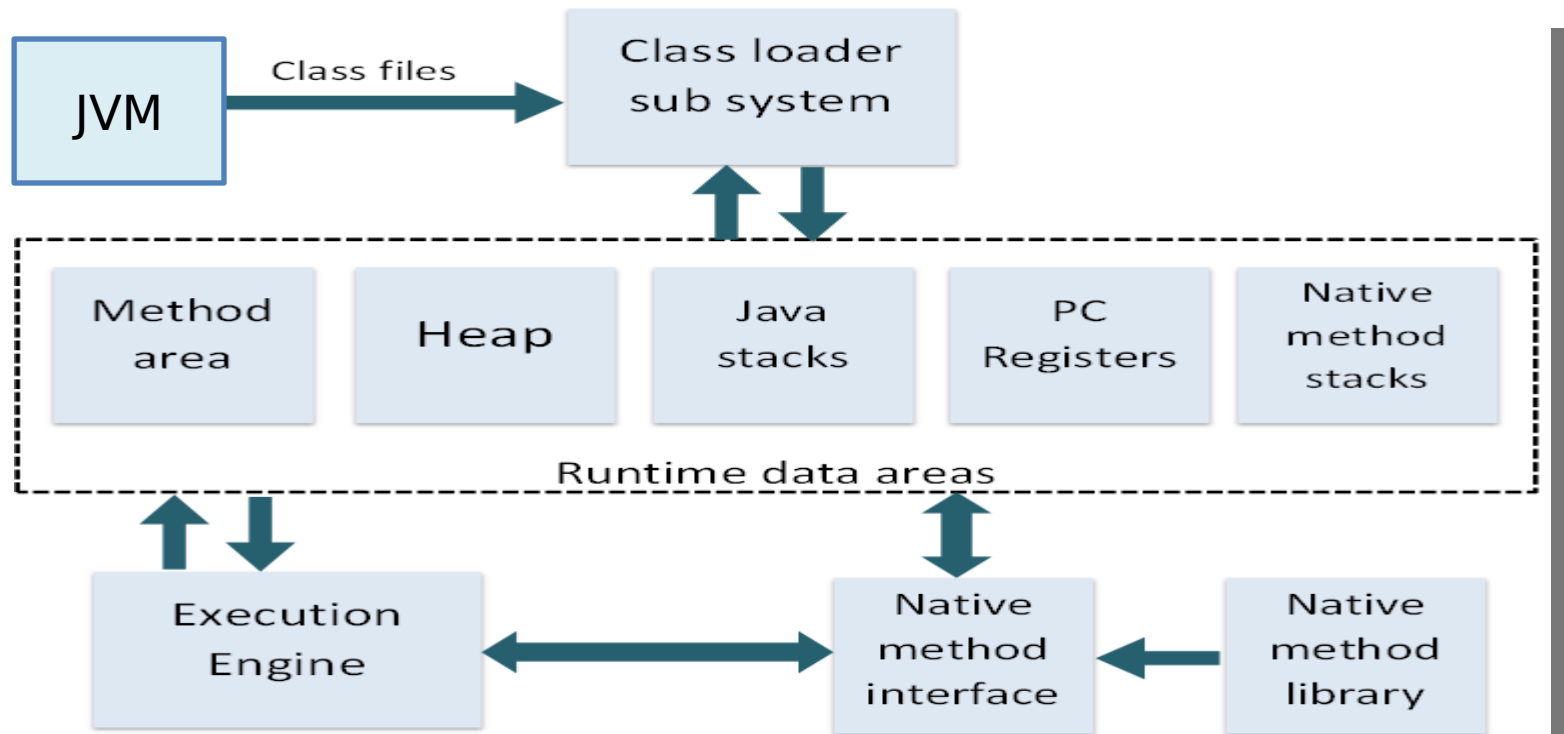
```
public static DateUtil getInstance(){
    int count=0;
    synchronized(DateUtil.class){
        if(dateUtil== null){
            count++;
            dateUtil = new DateUtil();
            System.out.println("first time"+count);
        }
        return dateUtil;
    }
}
```

- In the above code we are using synchronized block which is used to lock and unlock the particular lock, but the problems with synchronized block is it will allow one request to entered into the block and place the lock after getting the object release the lack but as per above example synchronized block westing to much time to locking, checking, unlocking the request. As per the above example it is also time consuming process and it will kill the applicaion performance.

- 4).
static{
dateUtil = new DateUtil();
}
public static DateUtil getInstance(){
return dateUtil;
}
- In this context we going to create the object of the DateUtil class at the class loading time. Some time it is important to create at class loading time. But the disadvantages are at loading it self it consume the jvm memory , and we can't predict we will use that object or not in future. it may useful or not we dont no. it is the drawback while using static block in the singleton design pattern.

Spring 45

- Class Loader: Classloader is a subsystem of JVM that is used to load class files.



- Actually JVM will call the Classloader to load the classes into the JVM memory.

- Class loader will take the .class byte code into the memory but before load into the JVM memory, it will follow certain set of procedure.
- Actually there are three class loaders available into the JVM memory
 - 1. BootStraps ClassLoader
 - 2. Extention ClassLoader
 - 3. Application ClassLoader
- 1) BootStraps ClassLoader:
 - BootStraps classLoader is the root class loader which is come with core jdk, Actually platform to platform this bootStraps class loader will be changes. B'z it will loaded by native machine libraries, means bootStraps class loader will loaded by operating system native libraries.
 - bootStrap class loader will load all the core jdk libraries and all the common environment which help other class loader to load the all functionalities.
 - Actually bootStrap class loader is the parent class loader for remaining two class loader.
 - Bootstrap class loader directory is `java_home/jdk/bin`.

- **2)Extension ClassLoader** : It is the Child classloader of BootStrap classloader.
- Extension classloader loaded by BootStrap classLoader.
- **3)Application ClassLoader** : It is the Child of Extension class Loader.
 - It is loaded by extension class loader.
 - While loading any class into the classloader first class come into feature that is Application class loader.
 - These classloaders follow certain rules to load the byte code in to the jvm memory.
 - Jvm memory is expensive memory without validation it is not possible.
 - To validate the byte code classloader follows three principles.

- Principles of classloaders:
 - 1). Principle of delegation
 - 2). Principle of visibility
 - 3). principle of uniqueness
- Above declared principles plays vital role while loading any byte code into the JVM memory.
- It will restrict duplication of the byte code over the cycle.
- 1).Principle of delegation=>
 - Actually byte code loaded by Application class loader but to check it is available through out the cycle or not principle of delegation will be used.
 - First Application classloader will check into cache of the application classloader , if it is available it will not load same bytecode into the JVM memory.
 - If it not available then it will delete to the extension class loader to check it is available or not, ECL will check into cache, if it is available it will load the same byte code neither it will delegate to the BootStrap classloader.
 - BCL will check into the cache if it available it will load same , if it is not available , it will again delegates to the ECL and ECL will delegates to the ACL. And ACL will load the byte code into the memory.

- 2). Principle to visibility:
 - Here parent-child relation available . Same as parent child relationship in java.
 - Child can access all the properties of the parent but parent can not access child properties.
 - Here parent class properties are visible to the child class loader but child class loader properties can not be visible to the parent.
 - Application Classloader can see the ECL and BCL loaded classes but ECL can not see the ACL loaded byte code and BCL can not see the ECL bytecode.
 - But ECL can see the BCL bytecode.

- 3). Principle of Uniqueness:
 - JVM memory is very expensive we can not use anywhere.
 - Uniqueness principle take care of loading unique classes only. Mean no one class loader load duplicate bytecode into the memory.
 - Here principle to delegation plays vital role, it will always delegate and check bytecode has been loaded or not.
 - It will not allow duplication of bytecode into the JVM memory.

- **Phases of ClassLoading:**

- 1). Loading
- 2). Linking
 - Verifying
 - Preparing
 - Resolving
- 3). Initializing

1)Loading

JVM call Classloader to load the .class file into the JVM memory. While loading the bytecode of the class, it follows principles of classloaders.

Once loading has been completed by one of ClassLoader, it will sent to the linking phase.

2)Linking:

Linking phase has classified into three parts

- **Verifying** : it will verify the bytecode compatibility means generated bytecode can be executed by current JVM or not.
- It will check the structure of the class is valid or not.

- **Preparing:** It is very important part of the linking because it will generate the symbolic link to the referenced class, method, variable and so on.
- Actually it will keep the link, but if we change the corresponding class, method, or variable it will throws an exception.
- **Resolving :** Here classloader going to check the corresponding class is available or not, if it is available it will load the class. But there are two ways to load the classes
 - **1). Implicit classloading:**
 - Ex:

```
class A{ B b = new B();  
}
```
 - **2). Explicit Classloading:**
 - Ex:

```
class A{ Class.forName("B");  
}
```
 - While resolving the class references we will get `ClassNotFoundException` and `NoClassDefFoundError` Exception.

- 3).Initialization:
 - Here all the static contexts going to initialize.
 - Actually static context going to execute at the time of class loading , but we can make some delay or restrict static to be display.
 - If one class want to load other class and other class contains static block, so we can restrict the static block to execute.
 - Ex: class B { static{ sos("static block");}}}
 - Class A
{ main{Class.forName("B",false,A.class.getClassLoader());}
}

- **Class Loader internal:**
 - By default class loading start from the application class loader and it will obey all the principles.
 - But if programmer want to load the class explicitly, he can load the class by using `Class.forName()` method .
 - Actually `forName()` method used for loading the classes only.
 - If we want to know which classloader loading our class then use one method in `Class` class. i.e. `getClassLoader()`;
 - `getClassLoader()`, `getParent()`, `getSystemClassLoader()`, `getClass()`, `getName()`, ...so on.

```
Ex: Class A {  
sop(A.class.getClassLoader());  
}
```

It will print the current classloader name and object reference value.

```
Ex. Sop(A.class.getClassLoader().getParent());
```

We will get parent classloader name and corresponding object reference value.

- Ex:
- public class Test {
- public static void main(String[] args) throws ClassNotFoundException, InstantiationException, IllegalAccessException {
- ClassLoader classLoader = Test.class.getClassLoader();
- System.out.println(classLoader);
- Class aclass = classLoader.loadClass("com.cl.beans.Calculator");
- //Object classobj = Class.forName(aclass.getCanonicalName()).newInstance();
- //System.out.println(classobj.hashCode());
- Object obj = aclass.getClass().getClassLoader().getSystemClassLoader();
- Object obj1 =
aclass.getClass().getClassLoader().getSystemClassLoader().getParent();
- Object obj2 =
aclass.getClass().getClassLoader().getSystemClassLoader().getParent().getParent();
- System.out.println(obj);
- System.out.println(obj1);
- System.out.println(obj2);
- System.out.println("aclass.getName() = " + aclass.getSimpleName());
- }
- }
- Actually above program talks about classloader return types, parent, and so on.

• Setting classpath to the classloaders

1) BootStrap classloader

- By setting into environment variable we can get all the core jdk library.

2) Extension Classloader

- `Java.ext.dirs`
- Set into environment variable
- `Java.ext.dirs` : By this we can set the extension classloader path to the current jar. Extension classloader only loads the .jar files. It can not load .class file.

3)Application classloader

- There are three ways available to set the classpath to the application classloader.
 - `Classpath`
 - `-cp`
 - `Manifest.mf` file

- When to use singleton class?
- There are three usecase where we can use singleton class.
 - 1).When object state is empty.
 - 2).when object state is read-only
 - 3).
- 1)object state is empty:
 - Actually class attributes represents the state of the object.
 - A class contains method and attributes but when we create the instance of the object only instance variable are injected with the object and that attributes represents the state of the object.

– For example

```
Class circle{  
    public double area(int radius){  
        return 3.14*radius*radius;  
    }  
}
```

- In the above example a circle class contains one method i.e. area() with one parameter i.e. radius.
- If someone want to call that method then he has to create the object of that class.

– Circle c1 = new Circle();

– Circle c2 = new Circle();

If we create number of object also there is no difference B'z there is no change in the method, so using one object or 100 object we will get same output.

– There is no use of creating multiple object to call the area() method of the circle class. So make that class singleton.

– If anyone want to call the area() method then using singleton object the can call.

By Mr.Sachin Gaikwad

- 2).when state of an object is read-only:
 - A attributes of the class which defines the state of the object. If a class contains one of the attribute and it is common to all the object creation then go for singleton class.
 - If you want to use a particular value through out the object creation then use this concept.
 - For Example:

```
Class circle{  
private final double PI=3.1412;  
public double area(int radius){  
return PI*radius*radius;  
}  
}
```

- In the above example the object state is read only because the final attribute value no one can change.
- If we create 100 object also the common state of the 100 object is read-only.
- Means all the cases object will not change then make our class as singleton class.

- 3)
 - While developing the project there are certain standards we have to follow.
 - if a project contains jsp page and it contains dropdown list then we should not hardcode the dropdown list items into the project.
 - B'z there are multiple jsp pages are available into the project then we have to write same code all over the jsp pages.
 - So I don't want to hardcode the dropdown list , but we have add the items into the dropdown list.
 - So there are three approach available we add data into the list.
 - Text file
 - Properties file
 - Database
 - Text file:
 - Prepare the text file which contains all the items which we will use throughout the application.
 - But there are some problems we have to face while working with the text file. B'z in text file we cannot differentiate the multiple kind of the data. (ex> city,state country).
 - And also we cannot specify the relationship between the data.
 - Text file used for store the data in sequential manner we can't differ it.

- Properties file:
 - Here we can clearly differ the data using key and value.
 - Actually a dropdown list contains label and value.
 - Ex: property.properties
 - Hyd=hyderabad
 - Chn=chanai
 - Blgr=Benglore
 - Actually property file one to the place, so we can use for dynamic loading data into list.
 - **When to use properties file**
 - If items are fixed and there is no change into the file in future then we can easily use the properties file.

- 3)Database approach:
- Most of the time rendering data should not be same through out of the application.
- Depends upon the requirement it will be change so if we write property to rendered data into the jsp page then it is difficult to justify.
- Because of that we propose to use the database to store data. If we use the DB then we easily define the relationship between the data.
- If in near future data going to add or remove then there is no impact on the coding which we done as part of the application.
- We can easily add and remove the data into DB.
- Actually there are two types of tables present into the database in application prospective.
 - 1) Master Tables:
 - Master tables generated by business people or system design people, they only decide which data should be available into the master tables
 - Master table data shared across the application and most of the time it will not be change.
 - Master table data rendered into the jsp pages before the jsp used by the user.
 - 2) operational Tables:
 - Operational tables are general table which is field by end user .
 - These tables are changeable, its up to the requirement.

• Possible ways of injecting data into the DDL

(1)

- There is a jsp page available and my jsp page have dropdown list, which indicates cities, states ,country.
- How to add the cities, states, and country into the jsp.

– Ex:

```
<select name="cities">
```

```
    <option value="hyd">hyderabad</option>    <option  
    value="chn">Chanai</option>
```

```
</select>
```

.....

- If we write the above procedure do add the data into the dropdown list we end up with hardcode.
- If other pages want the same data then again we have to write the same code into other jsp page.
- If there is change in the data then we have to change the entire code into the jsp page, there are several problems available.
- So we should not hardcode into the jsp page.
- So if data present into the DB and my jsp page want the data rendered into the dropdown list, so how we can inject that data to the DDlist.

- (2)
- Second way is write scriptlet into the jsp page, by writing the scriptlet we can load data into the DDL but there are bunch of problems
 - 1) jsp is the view controller we can not mix business logic with view logic.
 - 2) if there is change into the business logic may impact our view logic also.
 - 3) To managing such kind of code into the jsp page with is fall into management problems, maintenance problems.
 - 4) A jsp page contains HTML code , jsp code which is very hard to differentiate.
 - 5) actually most of the time UI developer going to involved into designing the presentation view . If we written the java code into the jsp pages then UI developer unable to understand the java code and if there is problem with view controller then it may difficult so solve it.
 - 6)if there are number of jsp page want the same data then we have to write the same logic into multiple places wherever it required.
 - 8)so we should not write the java code into the jsp page.

- (3)
 - 1) If we can not write the java code into the jsp page then write the code outside of the jsp and call the class from jsp page and get the data from java class.
 - 2) here also we writing some amount of code into the jsp page which is not understandable by UI developer.
 - 3) there are several classes are want the same data then again we have to write same logic in multiple jsp pages.

I don't want to write single line of java code into the jsp page, but I want to load by data into the DDL.

Because of that Tag library came into feature.

- Actually tag library use to manage the data and rendered the data into the jsp page, but most of the people thinking it is for business logic .
- each tag is the one class which contains java code for managing and displaying the data into the jsp pages.
- When to use the tag library, actually there are certain problems with the tag library b'z of the no one show interest to developing tag libraries.
- If there is a complicated logic , which is changeable repeatedly then go for tag library.
- Tag library not simple in use we have to manage view logic also into the tag library.
- B'z of that people not showing much interest to words the tag library.

- (4)
 - Now we can not use tag library also then what is the next, which help more easily to load the data into the DDL.
 - So java has provided a JSTL library(java standard tag library)
 - Which is used into the jsp page and it is almost look like html code only.
 - But problem is how the JSTL tags use the java code into the jsp page, Actually these JSTL tags developed to work with scope of the java.
 - Means there are three scope available into the java
 - Request scope
 - Session scope
 - Application scope

—

- What is the reason java has developed the such scope concept as the part of the java.
- Actually in normal java programs one class can access the attributes of the other class by creating the object of the class, but every object has new copy of the instance means no one object can access the attribute of the other object.
- Here also one servlet can not cerate the object of another servlet. Means one servlet can't use the attributes of another servlet.
- Because of that **scope came into feature**.
- Scope talks about sharing the data between the servlet classes.
- **Request scope** shared data to the next servlet. The life of data is up next servlet.
- **Session scope** share the data to the all the classes but life of the data is upto the session class live.
- **Application scope** is share the data through out the application , upto the application die the data should be available.
- **Get vs Post**
 - Get**
 - When you are getting the data from somewhere and preparing for some one then use get method.
 - Get method used to send simple data from one servlet to another servlet.
 - Post**
 - post method use when we want to send some sensitive data.
- **setAttribute and getParameter.**

- So JSTL developed work with scope of the object.
- There are several tags are available into JSTL which is used as per requirement.
- IMP
 - Never one jsp calls the servlet but servlet can call the jsp pages.
 - Servlet easily shared the data from servlet to jsp page using the different scopes, and bu JSTL tag jsp use the data.
 - Actually my jsp page want the data means first one of the servlet has to execute and send the data into the jsp page. B'z my jsp don't want to write the java code in it.
 - Before getting the call to the jsp page one servlet has to execute and that servlet has to call my jsp page with rendering all my DDL with data.

```

/viewRegistration
class viewRegistrationServlet extends HttpServlet{
    List<String> cities = null;
    List<String> state = null;
    List<String> country = null;
    public void service(req,resp){
        try{
            //connection logic
            //query the data
            // get the data into resultset
            //iterate it and put into the appropriate attribute

        }catch(SQLException e){
            throw new ServletException(e);
        }
        request.setAttribute("cities", cities);
        request.setAttribute("state", state);
        ...
        request.getRequestDispatcher("/register.jsp");
    }
}

```

```

<html>
    \\write the html logic

    <select name="city">

        <c:forEach var="city" items=${cities}>
            <option value=${city}>${cities}</option>
        </select>
    <select name="state">

        <c:forEach var="state" items=${state}>
            <option value=${state}>${state}</option>
        </select>
    <select name="country">

        <c:forEach var="county" items=${country}>
            <option value=${county}>${country}</option>
        </select>

    .....
</html>

```


- Now UI developer can easily can interact with my jsp code and he easily understand what we have written into the jsp page.
- But still problem is there
 - There are multiple users can access my jsp page each and every request my viewRegistration servlet has to call and pre-populate the data into DDL.
 - Each and every request my viewRegistration servlet will go to the DB and fetch the data and populate into the attribute and bind that attribute to the one of the scope and call the register.jsp page.
 - For each and every request we populate the same data then we unnecessarily going to the DB and fetching the same data and returning it. If data is same then why we are fetching data from DB every time, it is waste of time and we are killing the application performance, scalability problems also be there.
 - So we should go to the DB every time, so we can store that data into the one of the attribute into the servlet and when first request will come then it will go to the DB and fetch the data and stored into attribute, and next consecutives request they get data from the attributes itself.
 - So we will improve the performance of the application.

```
/viewRegistration
class viewRegistrationServlet extends
HttpServlet{

    private List<String> cityData;
    public void service(req,resp) {

        if(cityData==null) {
            try{
                //connect to the DB
                //query the data
                // stored into the ResultSet
                cityData = new ArrayList<String>();
                while(rs.next()) {
                    cityData.add(rs.getString("city"))
                }
            }catch(SQLException e) {
                throw new ServletException(e);
            }
            request.setAttribute("cities" cityData);
            request.getRequestDispatcher("/register.jsp");
        }
    }
}
```

- It seems to be good but there also have a problem, actually session object shared within the JEE classes but apart from that we can not use that data which is available into the one of the scope.
- If multiple classes want the same data then they have to write same code into the every servlet class to check data is available or not, and populate into the scope and rendered into the DDL.
- B'z of the above problems Cache Came into feature.

- Cache:
 - Cache is used for storing the data in it. So we are avoiding the round trips to visiting the DB every time and storing the data into the cache.
 - Cache improve the performance of the application.
 - When to use the cache?
 - There are several places we can use the cache.
 - If data is common for throughout the application then go to cache concept.
 - If we want to shared the data among the application then use cache.
 - Every class with in the application can access the cache class and he can use the data.
 - Most of the time cache must be singleton only.
 - B'z data remain same for every request then there is no need to create an multiple object of the class.
 - Using cache we going to avoid duplication logic as part of the application.
 - Actually in cache data will be stored in the form of key and value.
 - It may help in organizing the data.
 - A cache contains other member methods also which going to shared the state of the object in it.
 - Cache can has multiple method to add, retrieve and to check the data.

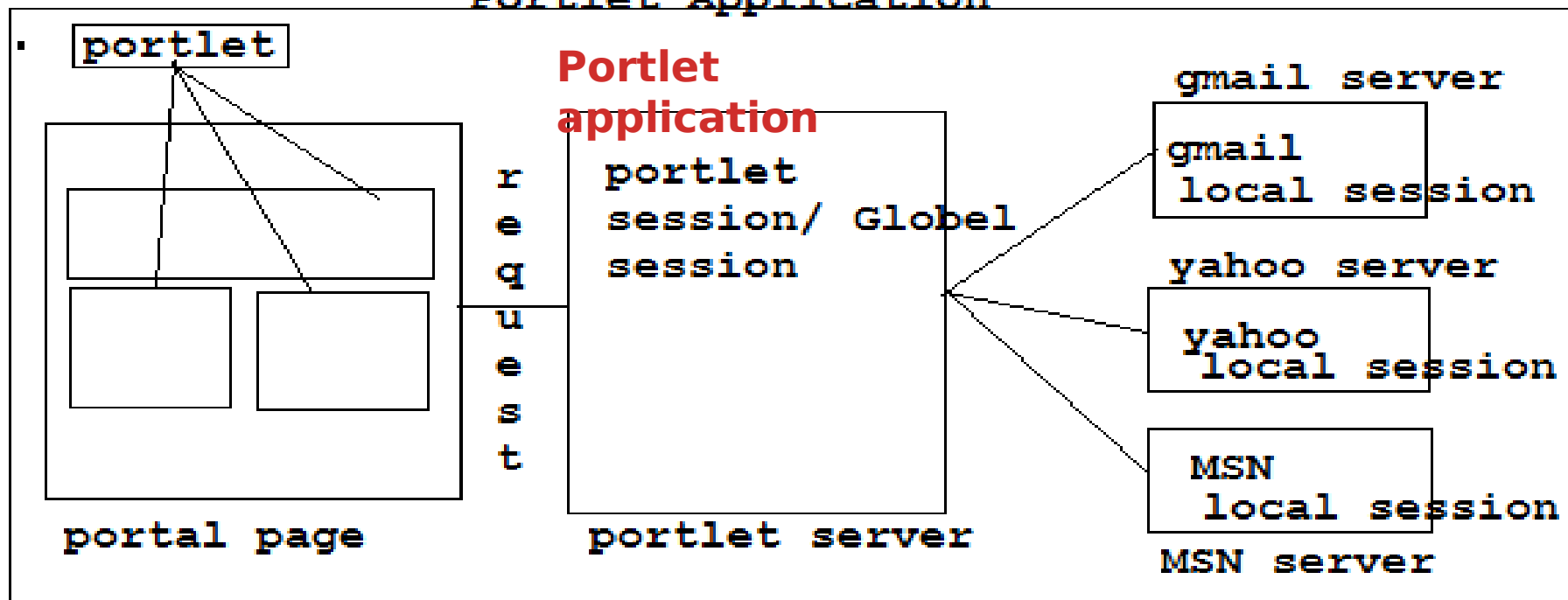
- Why I should not write Map into every servlet to handle the roundtrips?
 - If there are 10 classes want the same data then every class has to take separate Map to maintain the data.
 - Every class has to write the same logic to connect to the DB and add to the Map object.
 - If there is change in underlying structure then we have to manually make the changes into the Maps.
 - To avoid such kind of things we have to use the Cache concept, even cache provide the corresponding methods to work with Map objects.
- There are several classes can access the from the cache and several classes can add the data there is no restriction. But if number of classes started adding the value to the cache then it is very hard to get the data from the cache.
- B'z it may contains duplicate data also or cache may fail to decide which data a other class asking for.
- We can't justify a particular data into the Map, b'z of the duplication keys.
- To solve this kind problems we have to write our cache more intelligence means take Map inside other Map which organize the data in well format.
- As per the table or as per class it will store the data and clearly differentiate the city, state , country.

```
class Cache{
    private Map<string,Map<String,Object>> dataMap;
    private static Cache instance;
    private Cache(){
        dataMap = new HashMap<String,HashMap<String,Object>>();
    }
    public static synchronized Cache getInstance(){
        if(instance==nul)
        {
            instance = new Cache();
        }
        return instance;
    }
    public void put(String key, Object value){
        dataMap.put(key,value);
    }
    public Object get(String key){
        return dataMap.get(key);
    }
    public boolean containsKey(String key){
        return dataMap.containsKey(key);
    }
}
```

- When to use singleton third use case?
- {3}
 - When state of the object is there and it will sharable into the member method.
 - While creating the cache class we have to use the third use case of the singleton class.
- In use case first when the object state is empty then we can make our class as singleton to access the method, but we can make our class static and we can avoid to create single object also.
 - we can make our method static and we can use it by class name itself. But we can't restrict to create other class to create the object. Other classes can create as many number of object for the corresponding class, so to create multiple object for using the same method it waste of JVM memory.
 - And it will kill the performance of the application, b'z of that we have to make our class as singleton only.

- **Bean scope:**
 - **1. singleton scope**
 - Actually in spring every bean have default scope i.e. singleton.
 - If we try to create multiple object also we will get same object as part of the class.
 - **2.prototype scope**
 - Prototype is the another bean scope which is used for creating multiple object as part of the class.
 - IOC container will read the scope and depends on the scope it will create or return the object.
 - **3.request scope**
 - **4.session scope**
 - **5.globel session scope** [portal application]
deprecated and removed form spring 3.0.

Portlet Application



Portlet

=> portlet application application takes the different data from different server and place into portlet session/ global session, whenever request come to the portlet server will send the response to the portlet pages.

=> if same data requested by multiple time then it is very costly to get data every time from different server because of that global session came into feature.

=> global session going to store corresponding data in it and whenever request come for the same data it will returns the same data from the global session.

- As we discussed in the previous classes what are the benefits of singleton and when to use singleton and what are the rule we have to follow while using the singleton.
- **Static factory method instantiation**
 - By default bean scope is singleton only but when there are some classes will not allow to create the object directly, if you want to create the object then these classes provide static factory method to create an object.
 - Ex:
 - `<bean id =“cal” class=“java.util.Calendar”/>`
 - Here we can not create the object of the calendar b'z calendar has private constructor.
 - Actually calendar has one static factory method which going to return the Calendar class object.
 - Here IOC container can not create the object of Calendar class b'z calendar class is the abstract. To get the object we have to write static factory method which going to return the object.
 - We have to use the bellow procedure to get the object.
 - Ex:
`<bean id=“cal” class=“Calendar” factory-method=“getInstance” />`

- When to use static factory method in the application, in some cases we have a partial implementation of the class at the time we usually use abstract class, but generally we can not create the object of abstract class, but we want the object then we can use static factory method instantiation strategy to get the object.

- For example

```
Class Alarm{  
    private Calendar time;  
    public void sayTime()[  
        sop( time.getTime());  
    }  
    public void setTime(Calendar time){  
        this.time = time;  
    }  
}
```

Here calendar is the abstract class, IOC container unable to create object, but Calendar class has one method called "public static Calendar getInstance(){
return new Calendar();
}

Which going to return the object of the calendar class.

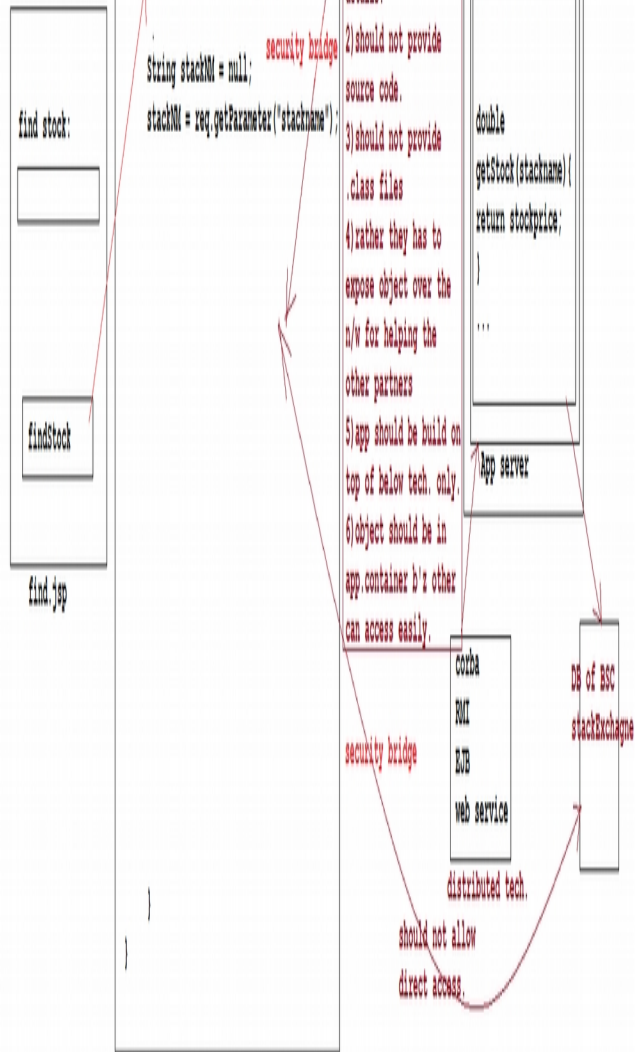
So now we easily can get the time by using calendar object.

– To get calendar class object we have to call

– `<bean id="cal" class="Calendar" factory-method="getInstance" />`

– By this we can get the object.

• Instance factory method instantiation (60,61,62,63)



INSTANCE FACTORY METHOD INSTANTIATION USECASE

By Mr.Sachin Gaikwad

• **Problems while dealing with remote based application**

- As per the above diagram there are so many problems are involved , but still we want to talk the remote application.
- so remote application server has to put the object in the JNDI registry they other partner can lookup and object and they will easily access the feature of the remote application.
- If my servlet writing the logic for getting the object of remote application i.e. BSCStockExchage then it is seem to be good if one class is writing the logic, but icicidirect.com application contains multiple classes they want to talk to the BSC stock Exchange then every class has to write the lookup logic for getting the object.
- **First problem** involved is duplication of logic.
- **Second problem** is if there is change in technology driven then again all the classes has to rewrite the lookup logic for getting the remote object from JNDI registry, b'z every tech. specific lookup jars are different.
- **Third problems** is if there is change in application server then again we have to rewrite whole logic for connecting to the other application server.
 - Ex: if remote application is running on JBOSS server and there is problems with JBOSS server and they moved from JBOSS to the WEB LOGIC SERVER then again I have to rewrite the whole logic for connecting the web logic server.
- **Fourth problem** is specific to the environment platform if there change in environment then again we have to rewrite the logic .
 - Ex: if application running on Joss server in machine because of lack of hardware capacity or performance issue application moved from one machine to another machine then problem will encounter.

- To overcome from these problems we will use one of the design pattern called serviceLocator design pattern.
- Why it is called as serviceLocator?
 - Actually serviceLocator is the class which will avoid all the problems which are available in our application.
 - Actually this class get the reference object from the remote location by performing lookup and it will provide this locator reference service to the current application b'z of that is it called as serviceLocator.
 - serviceLocator provide transference of the remote location. Actually my other classes don't no from where we are accessing the information, they think that the accessing class as part of our application only.
 - serviceLocator also opetimize the performance of the application.
 - Ex: if there are 100 classes want the remote application object, and if serviceLocator going each time and getting the object means it is very heavy job.
 - Bz of that serviceLocator manages one ConnectionPool which help the serviceLocator and optimize the performance.
 - At the loading of the application it will go and get the sufficient amount of object from the remote location and put into the connection pool.
 - Next time when request will come then serviceLocator will use the connection pool object for responding.

- Instance factory method instantiation which is available from spring 2.0 version.
- It is the special kind of factory method instantiation.
- Actually IOC container usually while creating an object, it instantiate the new keyword for creating an object. But there are some cases IOC container will not create the object, so for that we have to use instance and static factory method instantiation to create an object.
- In the above example my serviceLocator will get the object from the remote location and it will allow as to use in our application as normal object.
- But IOC container will create an object for classes which are available in the application, here serviceLocator class is the one who going to locate the object which is available on the remote location.
- So IOC container can not create the object for remote application classes.bz those classes not with the current application.
- Here serviceLocator class have one method which is responsible for getting the object from the target class and method name is getServiceLocator().
- getserviceLocator is the instance method so IOC container can easily perform the instance factory method instantiation to get the object.
- so how IOC container perform the instance factory method instantiation lets see.
 - IOC container will get the object of serviceLocator class and by help of object it will call the instance method of the serviceLocator .

Ex:

```
<bean id="googlemaprender"
      class="com.ifm.beans.GoogleMapRenderer">
  <property name="engine"
    ref="usgooleEngine"></property>
</bean>
instance of serviceLocator      ServiceLocator
<bean id="googleEngineServiceLocator" class
      class="com.ifm.util.GoogleEngineServiceLocator">
</bean>

<bean id="indiagoogleEngine" factory- instance of serviceLocator
      bean="googleEngineServiceLocator"
      factory-method="getIndiaGoogleLocator">
instance method in the SL
</bean>
<bean id="usgooleEngine" factory-
      bean="googleEngineServiceLocator"
      factory-method="getUsGoogleLocator" >
</bean>
```

If a instance method of serviceLocator taking an parameter then how we going to call that method in application-context . Actually IOC container will execute the virtual constructor to inject the parameter.

ex:

```
<bean id="indiagoogleEngine" factory-
bean="googleEngineServiceLocator"
factory-method="getDerection">
<constructor-arg value="india"></constructor-arg>
</bean>

<bean id="usgooleEngine" factory-bean="googleEngineServiceLocator"
factory-method="getDerection">
<constructor-arg value="us"></constructor-arg>
```


Instance factory method instantiation

```
public class GoogleMapRenderer {
    private IGoogleEngine engine ;

    public void getDirection(String
source,String destinations){
        String []direction=null;
        direction = engine.getDirections(source,
destinations);
        System.out.println("Direction :");
        for(String direct : direction){
            System.out.println(direct);
        }
    }

    public void setEngine(IGoogleEngine engine)
    {
        this.engine = engine;
    }
}
Main class
```

```
public class GoogleEngineServiceLocator {

    /*public IGoogleEngine
getIndiaGoogleLocator(){
        //lookup logic to get the remote
object from the remote application.

        return new IndiaGoogleEngineImpl();
    }
    public IGoogleEngine getUsGoogleLocator(){
        //lookup logic to get the remote
object from the remote application.
        return new UsGoogleEngineImpl();
    }
}*/

    public IGoogleEngine getDerection(String
countryCode){
        IGoogleEngine engine = null ;
        if(countryCode.equals("india")){
            engine=new
IndiaGoogleEngineImpl();
        }else if (countryCode.equals("us"))
            engine= new
UsGoogleEngineImpl();
        }
        return engine;
    }
}
ServiceLocator class which perform the
```

```
public interface IGoogleEngine {
    String[] getDirections(String source,String destination);
}
```

```
public class IndiaGoogleEngineImpl implements IGoogleEngine{

    @Override
    public String[] getDirections(String source, String
destination) {
        return new String[]
{"(indx1,indy2)","(indx1,indy2)"};
    }
}
```

```
public class UsGoogleEngineImpl implements IGoogleEngine{

    @Override
    public String[] getDirections(String source, String
destination) {
        return new String[]
{"(usx1,usy2)","(usx1,usy2)"};
    }
}
```

Remote application which is available on other machine

Normal instantiation

```
<bean id="googlemaprender" class="com.ifm.beans.GoogleMapRenderer">
    <property name="engine" ref="usgoogleEngine"></property>
</bean>
<bean id="googleEngineServiceLocator"
class="com.ifm.util.GoogleEngineServiceLocator">
</bean>
<bean id="indiagooogleEngine" factory-bean="googleEngineServiceLocator"
factory-method="getIndiaGoogleLocator">
</bean>
<bean id="usgoogleEngine" factory-bean="googleEngineServiceLocator"
factory-method="getUsGoogleLocator" >
</bean>
```

application-context.xml

lookup to locate the remote object

parameter instance factory method instantiation

```
<bean id="indiagooogleEngine" factory-bean="googleEngineServiceLocator"
factory-method="getDerection">
    <constructor-arg value="india"></constructor-arg></bean>
<bean id="usgoogleEngine" factory-bean="googleEngineServiceLocator"
factory-method="getDerection" >
    <constructor-arg value="us"></constructor-arg></bean>
```

By Mr.Sachin Gaikwad

- **Factory bean:**

- Factory bean is the old concept which a part of the spring 1.x version. At the initial day onwards this concept used by the spring.
- Actually there are some cases where IOC container unable to create an object for those classes.
- IOC container internally call the default constructor to create the object. Means it will get the object by calling new operator.
- but some time it is not possible for IOC container to create the object then we can use the concept called factory bean.
- Spring has provided one interface called FactoryBean which contains three method.

1)public Object getObject()

2)public Class getObjectType()

3)public boolean isSingleton()

Lets see the example for better understanding

```

public class Alarm {
    private Calendar calendar ;
    public void sayTime() {

        System.out.println(calendar.getTime())
    };
    }
    public void setCalendar(Calendar
calendar) {
        this.calendar = calendar;
    }
}

```

Java.util.Calendar

```

public class AlarmFactoryBean implements
FactoryBean {
    @Override
    public Object getObject() throws
Exception {
        return Calendar.getInstance();
    }
    @Override
    public Class getObjectType() {
        return Calendar.class;
    }
    @Override
    public boolean isSingleton() {
        return false;
    }
}

```

Annotations and return types are numbered for reference:

- 2: `@Override` on `getObject()`
- 1: `Class` in `getObjectType()`
- 3: `boolean` in `isSingleton()`

The return type of `getObject()` is `Object`, which is a superclass of `Calendar`.

```

<bean id="alarm" class="com.fb.beans.Alarm">
    <property name="calendar" ref="calendar"></property>
</bean>
<bean id="calendar" class="com.fb.util.AlarmFactoryBean"/>

```

=>IOC container will not create the object for Calender class bz Calandar class doesn't have default constructor.but programmer know how to create the object of Calender class so spring has provide one interface we have to implements that interface and override the all method into our class and provide required set of information, so IOC container will all our class and he will create the object.

- While creating the object of AlarmFactoryBean IOC container will identify the implemented interface and he will call the internal method of the AFB .
- Factory bean create a another bean and place into the IOC container.
- Actually factory bean for creating an bean were implementation provided by programmer and place that bean into the IOC container.
- Internals
 - If the programmer provided class is singleton then IOC container will check were object is available with him or not, if it is not then it will call the getObjectType() method and after that it will call the getObject() to get the Object with corresponding data returns type.
 - If programmer provided class is prototype then every time IOC will create new object for corresponding classes, Even we specified internal method as isSingleton as true.

- Lifecycle of the object:

- In general every existence has a life or state in there life.
- In programming world also build on top up such kind state which gives more things.
- Every programming language has a support for lifecycle of the object.
- Actually object is represent the structure of the class. And it will allow us to perform some operation.
- There are two state of the object
 - Creation /Born state (initial state)
 - Destruction /Die state (ending state)
- Every state represent the specific role throughout the life of the object.
- By using new operator we will usually create the object, but some time just after the creation and before used by other one we want to perform some initialization logic, then java has provided one method called as constructor.
- Constructor is the object management lifecycle method.
- It is used for assigning the state of the object for performing the task.
- Bz of that java has provided two method to handle the life cycle of the object.
 - Constructor
 - Finalize (it is used for realizing the resources which are occupied by the object) this method execute when program about to exit / end.

- Why servlet has its own lifecycle? Why it doesn't use java lifecycle methods?
 - There are some scenario where programmer want to initialize the values at the time of start up of the servlet, but the problem is that we can't take parameterized constructor in servlet, even if we take we can not pass the values to the servlet because everything is managed by servlet container.
 - Because of that we can not use same java object life cycle in servlet.
 - Servlet has its own life cycle to manage the object lifecycle.
 - There are two methods used by servlet container to handle lifecycle.
 - Init()
 - Destroy()
 - Service() => service is the request processor method, it is not for handling the lifecycle of the servlet.

- Init() method is separate for every servlet.
- Init is the method which is used for providing the dynamic input to the servlet.
- We can configure our input data into init-param tag with name and value, which is read by servletConfig object, and assign to the init method to use into servlet.

– Ex: `servletConfig config;`
`Init(servletConfig config){`
`this.config = config;`
`}`

- By destroy method we can release the

- **Bean Lifecycle:**
- Spring support both the way of initialization i.e. default and dynamic initialization.
- i.e. constructor initialization and setter initialization. Even though spring not supporting java lifecycle.
- Why spring has different lifecycle? Why spring not use java lifecycle ?
 - There are some situation programmer wanted to do some operation with constructor passes value and setter passed value then by using java lifecycle they can not do.
 - spring also has two attribute which will take care of executing the method after creating the object.
 - Init-method
 - Destroy method
 - In spring we can give any name to the method but we have to add that addition configuration into spring bean configuration file.


```

public class Math {
    private int i;
    private int j;
    private int sum;
    public Math(int j) {
        this.j = j;
        this.sum= this.i + this.j;
    }
    public void setI(int i) {
        this.i = i;
    }
    public void init(){
        this.sum= this.i + this.j;
    }
    @Override
    public String toString() {
        return "Math [i=" + i + ", j=" + j + ",
            sum=" + sum + "]";
    }
    public void destroy(){
        System.out.println("shutdown.....");
    }
}

```

```

public static void main(String[]
args) {
    BeanFactory factory =
new XmlBeanFactory(new
ClassPathResource("com/bl/common/app
lication-context.xml"));
    Math m1 =
factory.getBean("math", Math.class);
    System.out.println(m1);
}

```

```

<bean id="math" class="com.bl.beans.Math" init-method="init"
    destroy-method="destroy">
    <constructor-arg value="20"></constructor-arg>
    <property name="i" value="10"></property>
</bean>

```

By Mr.Sachin Gaikwad

- Garbage Collector
- Why jvm is not responsible for calling the GC
- What is the internal process to invoke GC
- What happen when GC under control of JVM
- At what time GC will execute
- What will be the impact if GC called each and every request?
- What is doamen thread what is the use of it?

- **Dependency-check:**

- Most of the time we going to inject the values via setter and constructor and constructor inject is mandatory.
- Means if a bean contains a constructor then we have to pass the values then only a object of bean will be created.
- But come to the setter injection it is optional if we will not provide any value also, IOC container will create the object.
- If we want to make setter also mandatory then spring has provided dependency-check as a additional configuration.
- Dependency-check will make setter inject mandatory by adding additional configuration. There are three modes available which make primitive and object to be mandatory.

- **MODES:**

- Simple :- it is for primitives
- Object :- it is for Objects
- All :- it is for primitives and objects

- IOC container will follow sort of procedure while creating an object when we configured as dependency-check.
 - IOC container will take an ID of the configured bean and it will check into in-memory metadata any bean has been configured with given id or not. If ID will be there then it will check the scope of the bean, if it is singleton then it will check the IOC container whether object is there or not, if object is not there then it will check for circular dependency.
 - After it will inject all the constructor parameter and it will check setter injection.
 - If it is configured with dependency-check then it will check the mode of the dependency-check and along with that it will check corresponding attributes or object will be available or not along with there setters.
 - If attributes are there and setter will not be there then it will not inject, as same object also.
 - For attributes or object setters are mandatory.
 - If setters are there then only it will inject the value.
 - If setter has taken but we didn't provided the value so the attributes and object get initialized with default value, then IOC container will not create complete object. We will get exception.

Spring 70,71

- Depends on:
- Cache:
 - Cache is used for storing the data in it. So we are avoiding the round trips to visiting the DB every time and storing the data into the cache.
 - Cache improve the performance of the application.
 - When to use the cache?
 - There are several places we can use the cache.
 - If data is common for throughout the application then go to cache concept.
 - If we want to shared the data among the application then use cache.
 - Every class with in the application can access the cache class and can use the data.
 - Most of the time cache must be singleton only.
 - B'z data remain same for every request then there is no need to create an multiple object of the class.
 - Using cache we going to avoid duplication logic as part of the application.
 - Actually in cache data will be stored in the form of key and value.
 - It may help in organizing the data.
 - A cache contains other member methods also which going to shared the state of the object in it.
 - Cache can has multiple method to add, retrieve and to check the data.
 - Cache should not contains business related logic.

- If 5 classes are there they want to use the data which is available into the cache.
- Each class has to check data is available into cache or not, if not he has to load the data, means each class end up with writing the checking the data is available or not and if not it will load the data.
- Means the same code going to write into all the classes which is not good best practice.
- To avoid such kind of problems we can write the same code in one place which is going to shared across the application i.e. cache.
- If we going to write the code within cache is it wrathful or not ? Actually every class get the data from cache then it is wrathful, but there are bunch of problems when we write the code into cache.

Why we should not write loading logic into cache?

- There are several problems available when we going to write logic into cache
 - We going to expose the internal resource system .
 - If there is change into resource system then whole application going to impact, b'z cache is the one of the place were every one going to access the data from the cache.
 - If there are multiple outsources are available then to get the data from multiple sources and writing the logic for loading it is very clumsy/hard.
 - Even though we prepared to write the logic, we have to face problems like if there is problem with one outsource will going to impact all the class which are related to other outsource also.
 - We should not write the business logic into the cache. Actually cache made for storing the data and retrieving the data.

- One of the best practice is should not write the logic into constructor because once constructor will execute we unable to call next time, to call the constructor we have to re-execute application. We can avoid above problem by writing code into one of the method and call that method from the constructor, if there is change into the data we can recall that method to get latest data.

– Ex:

```
Cache{  
    init();  
}  
Public void init(){  
    //logic for getting data  
}
```

- So up to this we got we should not write the loading logic into the cache.
- But we have to write logic some where else where all the application can use it.
- So CacheManager came to feature.

- **CacheManager:**
 - CacheManager is the class who going to manage all the problems. But even CacheManager also has to write same loading logic in it.
 - Again cacheManager also has to fall into the same problems like cache class, but CacheManager not only manages the loading logic it has to manage other prospective logic also, ex: making sure data should be in correct manager ...
 - cacheManager will perform verification, validation, accuracy and all other jobs also.
 - CacheManager will not load the data from the corresponding source system it will call other classes to load the data from resource system.
 - Once they loaded, CacheManager will take the data and it will perform some operation to make sure data in usable format and it will stored into the cache.
 - Means now other classes need not be worry about parsing the data in each and every class they can use directly.

```
class cache{
    private Cache(){
        //logic loading data from properties
        //logic loading data from xml
        //logic loading data from excel
        //logic loading data from database
        init();
    }

    public void init(){
        //logic loading data from properties
        //logic loading data from xml
        //logic loading data from excel
        //logic loading data from database
    }

    public void put(.,.){}
    public Object get(.){}
    public boolean containsKey(.){}
}
```

- To solve the above problem we have to use one of the design pattern which is accessor design pattern.
- **Accessor Design Pattern :**
 - It is the one of the Design Pattern which going to interact with the underlying database to load the data.
 - ADP is one of the way we can improve the performance of the application, actually it will used with core requirement.
 - There are multiple accessor which are talking to underlying databases or filesystem.
 - So cacheManager has to talk to every accessor to get the data from the particular accessor. If cacheManager talking to every accessor then he has to check every time data is available into the cache or not it not it has to get the data from accessor.
 - To avoid repeated checks we have to use one abstract class which has common requirement logic for checking and loading the data into the cacheManager and checkManager will massage the data and load into cache.
 - even we are using ADP and CacheManager still my class tightly coupled with the cache class.
 - Bz my class is getting the data from cache and it is getting instance into my class.

- To avoid above problem profiler design pattern came into feature.
- **Profile Design Pattern:**
 - cacheManager going to talk to the profiler and profiler itself check the data into cache and it load the data into cache also.
 - Using profiler design pattern we can improve the core performance of the application.
 - Profiler works as on demand, depends on demand it will call the appropriate accessor to get the data and load to the cache.
 - CacheManager job is to massage the data only.
 - We can improve the core usability by creating the baseprofiler which is going to handle common checks in cache and accessor.
 - Let see the program to get more details

- **Depends on:**
 - Depends on just the one of the attribute in bean level elements but it plays the very crucial role while developing the application.
 - While developing an application one class is depends on another class directly, and we can inject directly no problems .
 - But some time indirect dependency also be there means before creating the object of myclass other class should be loaded because myclass using the data from other class.
 - To fulfill the requirement we can easily use the depends on addition configuration.
 - Ex:

LoanCalculator class what the data from cache but cache data loaded by cacheManager so before myclass object created I want the data available into the cache, means loanCalculator indirectly depends on cacheManager. So before created loanCalculator cacheManager has to load the data into cache, so loadCalculator can easily use the data.

Use depends on and configure loanCalculator with depends-on="cacheManager" attribute.

- Beanfactory vs ApplicationContext:
- **Beanfactory:**
 - Beanfactory is the interface which has multiple implementation to create IOC container.
 - IOC is the logical memory in the JVM memory.
 - When we going to execute the XMLBeanFactory() method this memory will be created by XMLBeanFactory to keep an object .
 - IOC also called as Core Container.
 - IOC container memory having two parts
 - In memory METADATA
 - Empty(for storing the object (key=value format)).
 - XMLBeanFactory() method will take an object resource from the ClassPathResource() and create the object and keep it in to the IOC Container.
 - Before creating IOC container XmlBeanFactory will check for well form ness and validation then only it will create the object.

- Beanfactory is the lazy initializer because beanfactory will going to create the object on demand only.

**BeanFactory factory = new XmlBeanFactory(new
ClassPathResource("application-context.xml"));**

- Above line classpathResource will take the application-context.xml file and create the object of the resource and pass as the input to the XmlBeanFactory, here it will check for well formness and validation and it will create the IOC container.
- It will not generate any bean object into the IOC container.
- When we say factory.getBean("name",class_name.class); then only ti will create the object for corresponding class.
- **Drawback:**
 - Even if we configure wrong beans also it will not throw error, when we call the bean then only it will throws an exception which will going to terminated the application.

- **ApplicationContext:**
 - ApplicationContext also an interface which has number of implementation classes, it is also used for creating an IOC container.
 - ApplicationContext is the eager initializer.
 - applicationContext has number of implementation classes
 - XmlBeanFactoryApplicationContext
 - ClassPathXmlApplicationContext
 - When we create the applicationConext reference it will instantiate all the beans which are configured as part of the application-context.xml.
 - While create the object it will check bean id along with definition into inmemory metadata, along with that
 - Circular dependency
 - Constructor injection
 - Scope of the bean
 - Setter injection
 - Dependency-check
 - Direct references
 - Depends-on
 - And so on..
 - After checking all the permutation and combination it will create the object, for every object it will do the same process and create the object.
 - If there is the problem then it will not create the IOC container, bz applicationContext is the eager initializer.
 - The benefits of applicationContext is it will check all the configured beans at initiate time only and if it is problem then it will throw an exception.

Spring

- 76 – Resume builder information
- 77 - interview tips and fake experience overview
- 78- industry awareness
- 79-techniquial rounds and HR round
- 80-after offer letter release to you what next ? Guidelines
- 81 – usecase with team lead and Rule Engine and method Replacement.

- Once we completed the industry awareness now lets see the overview about RULE ENGINE.
- Most of the project has some critical development module which going to crack the developer has mostly decision making statement.
- We can not be on the one decision while developing an module there would be many decision are there so depends on the decision we can to perform some business operation.
- Take an example providing and insurance policy to the customer
 - Ex: If Govt. has given a permission to every insurance vendors to accept the insurance for poor stage people whose income is less than 1 lacs rupees.
 - So insurance vendors has to take all the required document and according to the rule they have to provide the insurance, after that they have to send those document details to the Govt. for verification. Once verification has completed Govt. will provide money to the insurance vendors.
 - But while filling the insurance form they have to take age, in-network treatment, out-network treatment, address, after that they have to check any discount available or not,... for this they have to perform multiple decision making which lead to the huge problem.
 - We can not avoid writing if-else , else if , condition which makes many confusion.

- If there are four parameter available then we can write 24 decision making condition, it is so difficult to handle such kind of situation.
- To avoid such kind of confusion we should have to use RULE ENGINE.
- **RULE ENGINE:**
 - Rule Engine is the tool or framework which will help to make decision making easy.
 - It is another approach we of programmatic approach.
 - Rule Engine is unlike the java programming decision making approach, actually it will provide the ENGLISH like decision approach any one can read and any one can modify depends up on the requirement.
 - We can write the decision making statement in sentence format only and even business people can also easily modify the rule files.
 - Actually all the decision making statement will going to put into the rule file, which will dynamically modify by programmer or business people and there is no need to re-compile, re-deploy, re-test and so on.
 - It will automatically update and run with latest changes. But if we use general if-else and else-if condition then it is every hard to change the decision because it will lead to the maintenance.

- **When to use Rule Engine?**
- when there is no satisfactory traditional programming approach to solve the problem.
- The problem is just too fiddle for traditional code.
- The problem may not be complex, but you can't see a non-fragile way of building a solution for it.
- The problem is beyond any obvious(clear) algorithmic solution.
- It is a complex problem to solve, there are no obvious(clear) traditional

- There are several tools available in the market who provide the Rule Engine tools.
 - **JBOSS Drools**
 - **JRuleEngine**
 - **Mandarax**
 - **ILisa**
 - **JEOPS - The Java Embedded Object Producti
on System**
 - **Prova language**
 - **OpenRules**
 - **Open Lexicon**
 - **SweetRules**
 - **Zionis**
 - **Hammurapi Rules**
- Again there are so many open source rule engines tools are available.

- **Method Replacement:**

- It is the feature of the spring which is available in spring framework only.
- This feature makes so many sense in industry bz it will reduce maintenance cost, cost, time, So on.
- Without touching to the method or without modify the existing method we can replace one method to another method, using method replacement feature.

Ex:

There are some situation we can not solve the bug in the module but we wanted to solve that bug ,actually we tried number of time but in production environment it will failing but still it is working, it will arise after 10000 request, but still it is there then we can not modify the current working class method code, but still we wanted to modify then we can easily can use method replacement feature.

- To use method replacement spring has provide one interface called as MethodReplacer, which has a method and we have to override that method i.e. reimplement().
- Reimplement() method has three parameter which going to represent to whom he is replacing.
- Actually every parameter has it own data which is required for replacing the original method to new method without modifying the existing one.

– Ex:

```
Public Object reimplement(Object OrgObj,Method method, Object[] agrs){  
    //logic  
}
```

- If we look at the above method it going to return the Object bz if original method has returning some thing means this method also has to return the appropriate value. If it is not there then assign null as the return type.
- There are three parameters are there first one is original class object were we can get the instance data of the original one and on which class object we are calling the method we will come to know.

- Second one is the Method which tells about the method on which we are performing the operation, method parameter contains the original method name.
- Third one is the Object[] args, it will give all the required set of value which are passed as parameter to the original one. Here we can extract the Object[] and we will get all the parameters.
- Replacing the method is not a easy task bz we can not touch to the original method, moreover all the object not with us, they are available with the IOC container.
- Now we have to tell to the IOC container to replace the original method with new method. For that we have to provide the additional configuration to the IOC container.
- Once we configured all the information then requests come to the original class method but internally IOC container manages and it will execute the new method i.e. reimplement().
- Actually we are calling original method only but internally IOC container will check is there any additional configuration has been provided or not if it is there IOC container will manages dependencys.

- For method replacement below additional configuration will be provided

– Ex:

```
<bean id="a" class="A">  
    <replaced-method="m1"  
replacer="b"/>  
</bean>  
<bean id="b" class="B"/>
```

- Replaced-method tag will talk about which method we want to be replace and replacer tag talks about by whom we are replacing.

- Once a class implements the MethodReplacer interface and when configured with replaced-method and replacer then IOC container will internally going the create a proxy class which extends the original class and override the replaced-method and internally call the new implemented method which is in new class.
- Just see the below diagram we can understand how internally proxy will generate and how it is calling the reimplement method.
- If we want to use method replacement then the original class should not be final and method should not be static, then only it is possible because interally original class will extends by proxy class.
- To get all the details of the method internally proxy class uses stackTrace techniques.
- stackTrace will keep all the records of current thread which is currently executing.

```

class planfinder{
    public String findPlan(int ageGroup, int coverage, int network)
    {
        //original logic
        return new String("jeevan bharathi");
    }
}

class FindPlanReplacer implements MethodReplacer{
    public Object reimplement(Object target,Method method,Object[] args){
        //new logic
        return "jeevan abhaya";
    }
}

//At runtime IOC container will generate one proxy class for planFinder
and that proxy name may be planFinder$poxy, lets see the internal

class PlanFinder@proxy extends PlanFinder{
    MethodReplacer replacer;
    PlanFinder@proxy(MethodReplacer replacer ){
        this.replacer = replacer;
    }
    @override
    public String findPlan(int ageGroup,int coverage,int network){

        return new repalcer.reimplements(int ageGroup,int coverage,int
network);
    }
}

```

MethodReplacement

- **Property Editors:**
 - We can inject the values to the attribute in spring by two ways
 - **Setter injection**
 - **Constructor injection**
 - Using spring bean configuration file we can easily inject value to the properties.
 - By default spring considered all the value as string, but spring is intelligence in converting string to primitives. Implicit casting it will use and convert. but there are some situation we can not convert object, arrays, file , URL ,date and so on.
 - So to convert different objects into corresponding format spring has provided a feature called property Editors.
 - **Property editors** mean to edit the property and convert into corresponding format for operation.
 - Spring has provided number of property editors class internally which will help in directly declaring the values for that property.

- While creating the object of the bean IOC container will check the scope of the bean if it is singleton then it will check object already available into container or not, after it will check constructor injection is there or not, after it will check any object value directly injected to the property or not if it is there then it will check what is the property type and it will take the property and look into property editors Registry which is available with IOC container.
- Property editor Registry contains key as the type and value as the object type.

Object Area

property editor registry

key

value

java.util.Date

Date object

j.u.File

File Object

j.net.URL

URL object

Array

Array Object

InMemory MetaData

IOC Container

```
<bean id="person" class="Person">
    <property name="DOB" ref="date"/>
</bean>
<bean id="date" class="java.util.Date"/>
```

In the above eg. how we are injecting the date to the DOB it is also valide but it is not spring has prvided implicit propertyEditor for date type i.e. DatePropertyEditor means we can directly inject the value to the date type property also. e.g

```
<bean id="person" class="Person">
    <property name="DOB" value="01/01/2015">
</bean>
```

internally spring will take the property type and it will call the appropriate propertyEditors class to convert the value.

- There are some situation where we have to create our own custom property editors , internally spring given number of property editors we can easily use it.
- As per above discussion if there is an object as attribute and if we are passing string as value then while creating an object IOC container will check any corresponding property editor available or not if available it will convert that string format to appropriate format, neither it will throw an exception saying no matching editors found.
- So to create our own custom property editor spring has provided a abstract class called `PropertyEditorSupport`.
- We can override one method called `setAsText(String value)` and we can write the converting logic in it and set to the corresponding constructor.
- Lets see the example By Mr.Sachin Gaikwad

```
public class ComplexNumberPropertyEditors
extends PropertyEditorSupport{

    @Override
    public void setAsText(String value)
throws IllegalArgumentException {
        String splits[] =
value.split(",");
        ComplexNumber cn = null;
        cn = new
ComplexNumber(Integer.parseInt(splits[0]),
Integer.parseInt(splits[1]));
        setValue(cn);
    }
}
```

- Writing property editor class is not enough we have to register that custom property editor propertyEditorRegistry in IOC Container.
- To register into IOC container we have to talk to the PropertyEditorRegistrar interface by overriding one method called **public void registerCustomEditors(PropertyEditorRegistry registry)** and we have to register into the propertyEditorRegistry.
- Lets see snippet of code.

- **public class CustomPropertyEditors implements PropertyEditorRegistrar{**
- **@Override**
- **public void registerCustomEditors(PropertyEditorRegistry registry) {**
- **registry.registerCustomEditor(ComplexNumber.class, new ComplexNumberPropertyEditors());**
- **}**
- **}**
- Once we write the above code we can register customPropertyEditors into the propertyEditorRegistry but to add into IOC container we have to write the snippets of code i.e.
- **BeanFactory factory = new XmlBeanFactory(new ClassPathResource("com/cpe/common/application-context.xml"));**
- **((ConfigurableListableBeanFactory)factory).addPropertyEditorRegistrar(new CustomPropertyEditors());**
- BeanFactory is immutable we can not modify the IOC container, to modify or add we have to use the configurableListableBeanFactory

```

public class Calculator {
    private ComplexNumber complexNumber ;
    public ComplexNumber getComplexNumber() {
        return complexNumber;
    }
    public void setComplexNumber(ComplexNumber
complexNumber) {
        this.complexNumber = complexNumber;
    }
    @Override
    public String toString() {
        return "Calculator [complexNumber=" +
            complexNumber + "]";
    }
}

```

```

public class ComplexNumber {
    private int base;
    private int expo;
    public ComplexNumber() {}
    public ComplexNumber(int base, int expo) {
        this.base = base;
        this.expo = expo; }
    public int getBase() {
        return base;}
    public void setBase(int base) {
        this.base = base;
    }
    public int getExpo() {
        return expo; }
    public void setExpo(int expo) {
        this.expo = expo; }
    @Override
    public String toString() {
        return "ComplexNumber [base=" + base
+ ", expo=" + expo + "]"; }
}

```

```

public class ComplexNumberPropertyEditors extends
PropertyEditorSupport{
    @Override
    public void setAsText(String value) throws
IllegalArgumentException {
        String splits[] = value.split(",");
        ComplexNumber cn = null;
        cn = new
ComplexNumber(Integer.parseInt(splits[0]),Integer.parseInt(sp
lits[1]));
        setValue(cn);
    }
}

```

```

public class CustomPropertyEditors implements
PropertyEditorRegistrar{
    @Override
    public void
registerCustomEditors(PropertyEditorRegistry registry) {
        registry.registerCustomEditor(ComplexNumber.class, new
ComplexNumberPropertyEditors());
    }
}

```

```

public class CPETest {
    public static void main(String[] args) {
        BeanFactory factory = new XmlBeanFactory(new
ClassPathResource("com/cpe/common/application-context.xml"));
        ((ConfigurableListableBeanFactory)
factory).addPropertyEditorRegistrar(new
CustomPropertyEditors());
        Calculator calculator =
factory.getBean("calculator", Calculator.class);
        System.out.println(calculator);
    }
}

```

<beans>

```

<bean id="calculator"
    class="com.cpe.beans.Calculator">
    <property name="complexNumber"
        value="25,30"/>

```

</bean>

</beans>

application-context.xml

- **What is Internationalization?**
- How internationalization works with core java?
- How internationalization works with JEE?
- There are three ways we can develop internationalization
 - Data internationalization(collation)
 - NLS Programming
 - Content internationalization
- How data internationalize will work ?
- What is NLS programming internationalization?
- Content internationalization .
- How to render static context or static templet on jsp pages.
- Drawback with static text file static context?
- Property file approach and drawbacks with property file approach.
- What is mean by Locale?
- How to work with locale?
- What is the use of ResourceBundle?
- How to use ResourceBundle?
- Drawbacks with resourcebundle and example of resource bundle?
- Resourcebundle example using servlet and dofilter.
- How to optimize the solution of the resourcebundle using locale and baseName.

- Why ResourceBundle will take the bundle from src??
- Hard Coding the static content in any application will not make it Internationalized, so, write the content in some other source, i.e., Properties.
- Why properties means, we can read any content using Key, so we can access them easily. If we use Text file, we can't read a specific portion of it.

index.jsp

```
<%@page import="java.io.FileInputStream"%>
<%@page import="java.io.File"%>
<%@page import="java.io.InputStreamReader"%>
<%@page import="java.util.Properties"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<%Properties props = null;
    props = new Properties();
    props.load(new FileInputStream(new File("E:\\Sriman\\Spring\\08102015\\SpringCore\\I18N_Practice2
\\WebContent\\messages.properties")));
%>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Jet Airways</title>
    <h1><%out.println(props.getProperty("index_header")); %></h1>
</head>
<body>

    <br/><br/><center><h1><font color="red"><%out.println(props.getProperty("index_body")); %></font></h1>
</center>

    <br/><br/><br/><br/><center><h1><font color="purple"><%out.println(props.getProperty("index_footer")); %>
</font></h1></center>
</body>
</html>
```

ResouceBoudle Example

Messages.properties

index_header=Welcome to AirTicketing System

index_body=You can Book the tickets here

index_footer=@copy_right Sandeep

- Here, we are reading the static content (or) template text from the properties file, but it display only one language, the end user may send request from any Locale , so write multiple properties files for multiple languages

```
messages.properties
```

```
-----
```

```
index_header=Welcome to AirTicketing System
```

```
index_body=You can Book the tickets here
```

```
index_footer=@copy_right Sandeep
```

```
messages_hindi.properties
```

```
-----
```

```
index_header=Welcome to AirTicketing System (Hindi)
```

```
index_body=You can Book the tickets here (Hindi)
```

```
index_footer=@copy_right Sandeep (Hindi)
```

index.jsp

```
-----  
<%  
    Locale locale = Locale.getDefault();  
    Locale locale2 = request.getLocale();  
    out.print(locale2);  
    out.print(locale2.getDisplayLanguage());  
    Properties props = null;  
    props = new Properties();  
    if(locale2.getDisplayLanguage().equals("Hindi")) {  
        props.load(new FileInputStream(new File("E:\\Sriman  
\\Spring\\08102015\\SpringCore\\I18N_Practice2\\WebContent  
\\messages_hindi.properties")));  
    }  
    else{  
        props.load(new FileInputStream(new File("E:\\Sriman  
\\Spring\\08102015\\SpringCore\\I18N_Practice2\\WebContent  
\\messages.properties")));  
    }  
    Usage to Local class  
%>
```

□ Here for every Locale specific language, we have to provide one if condition, so to avoid that, append locale to base name (like messages), and name the properties file in same convention

```
messages.properties
```

```
-----
```

```
index_header=Welcome to AirTicketing System  
index_body=You can Book the tickets here  
index_footer=@copy_right Sandeep
```

```
messages_hi_IN.properties
```

```
-----
```

```
index_header=Welcome to AirTicketing System (Hindi)  
index_body=You can Book the tickets here (Hindi)  
index_footer=@copy_right Sandeep (Hindi)
```

```
<%
```

```
/* Locale locale = Locale.getDefault(); */
```

```
Locale locale = request.getLocale();
```

```
out.println(locale);
```

```
out.println(locale.getDisplayLanguage());
```

```
Properties props = null;
```

```
props = new Properties();
```

```
props.load(new FileInputStream(new
```

```
File("E:\\Sriman\\Spring\\08102015\\SpringCore\\I18N_Practice2\\WebContent\\  
messages_"+ locale+".properties"))));
```

```
props.load(request.getServletContext().getResourceAsStream("resources//me  
ssages_"+locale+".properties"));
```

```
%>
```

- We are Writing the logic to append the locale to the File, but, J2EE is Provided a class ResourceBundle to read properties file bundle.
- We have to pass Basename and locale, so it will good enough to identify and read the messages from properties file, and returns as resourceBundle Object.
-
- `<%`
- `/* Locale locale = Locale.getDefault(); */`
- `Locale locale = request.getLocale();`
- `ResourceBundle rb = ResourceBundle.getBundle("messages", locale);`
- `%>`
-
- □ But, Writing Scriptlet logic inside a JSP is not recommended, so write this logic of RecourceBundle inside a ServletFilter, whenever servlet container get the request its handover that request to FilterManager and handover that request to that filter based on mapping.
- □ Filter is the one which perform pre-processing and post-processing on request.

- `@WebFilter("//*")`
- **public class** `I18NFilter` **implements** `Filter`{
- **public void** `destroy()` {}
- **public void** `init(FilterConfig arg0)` **throws** `ServletException` {}
- **public void** `doFilter(ServletRequest req, ServletResponse resp,`
- `FilterChain chain)` **throws** `IOException, ServletException` {
- `Locale locale = req.getLocale();`
- `ResourceBundle rb = ResourceBundle.getBundle("messages",`
- `locale);`
- `req.setAttribute("rb", rb);`
- `chain.doFilter(req, resp);`
-
- `}}`
-
- □ The `ResourceBundle` Object is set to request scope.
- □ If we want to access multiple Bundles we need to use multiple `ResourceBundle`'s.
- □ And we have to set them to request scope.

- @Override
- **public void** doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) **throws** IOException, ServletException {
- Locale locale = req.getLocale();
- ResourceBundle rb1 = ResourceBundle.getBundle("messages", locale);
- ResourceBundle rb2 = ResourceBundle.getBundle("errors", locale);
- req.setAttribute("rb1", rb1);
- req.setAttribute("rb2", rb2);
- chain.doFilter(req, resp);
- }
-
- <% ResourceBundle rb1 = (ResourceBundle) request.getAttribute("rb1");
- ResourceBundle rb2 = (ResourceBundle) request.getAttribute("rb2");
- %>
- <%out.println(rb1.getString("index_header"));%>
- <%out.println(rb2.getString("unable_to_login")); %>
-
- □ If we have multiple properties bundles we need to write multiple ResourceBundles's and we have to set it to request scope to access in JSP page.
- □ But, to access different bundles we need to know their names, this is difficult, so retrieve the data from ResourceBundle Objects and store them in a Collection Object then forward it to JSP page.

- I18N Using Spring
- Spring has provided a class called **ResourceBundleMessageSource** which implements **MessageSource** interface.
- **messageSource** interface there are two implementation classes are available
 - **ResourceBundleMessageSource**
 - **ReloadableResourceBundleMessageSource**
- The bean id must be and should be **messagesource** only.
- **beanFactory** will not support internationalization but **ApplicationContext** will support internationalization.
- And the constructor of **resourceBundleMessageSource** will take
 - Property file name i.e. **baseName, replacement object, locale**
 - Internal process of Spring **resourceBundleMessageSource**

Spring 91

- **Lookup method Injection:**
- How many object will going to created by servletContainer?
- If it is one then why and how actually it is manage all the request by only one object?
 - In JEE environment all the application are distributed application so multiple request will come to the corresponding servlet at a time. But servletContainer will create only one object for all the request.
 - If servletContainer will going to create new object for every request then for 1000 request, 1000 object has to created by servletContainer. it will cause big performance issue and JVM method will feed up with objects only.
 - Because of above problem servletContainer will going to create only one object only and that object will shared to all the corresponding requests.
 - Actually internally servetCotainer will call run() method which will internally going to call the service() method of servlet, and it will generate one thread for per request.
 - servletContainer class is not a singleton class but its behave like singleton.

- servletContainer will generate multiple thread for per request but there is a problem if object state is none-sharable then data inconsistency will encounter.
- We will discuss one use case which encounter the problem when object state

```
class TransferFounServlet extends HttpServlet{  
    private int fromAc;  
    private int toAc;  
    private float amount;  
  
    p v service(req,res){  
  
        fromAc = req.getParam("fromAc");  
        toAc = req.getParam("toAc");  
        amonut =  
            Float.parseFloat(req.getParam("amount"));  
  
        //business operation  
        //display the success page  
    }  
}
```

For every thread,
object state will change

- As per the above example servletContainer will create one thread for first request and that thread will start execution of servlet.
- It read the fromAc, toAc and amount and perform the operation, while performing business operation new request came and servletContainer will created one more thread and first thread get suspended.
- Now second thread complete the business operation now again first thread came and he want to execute but thread first value overlapped with second thread values which lead the data inconsistency .
- When object state is sharable but not final then for every thread it will assigned with new value and prior data gets erase.

- Now how we can make our application works with multi-thread environment without data inconsistency.
- There is one way available i.e. put business logic into synchronized block.
- Synchronized block will allow only one thread to enter into the block and it will lock the block until and unless block execution complete. After that only it will release the lock.
- If multiple request are coming then thread scheduler make them to wait into the thread scheduler queue.
- But here also an problem i.e. performance issue problem.
- The above solution will is not much good so we have to see the alternative.
- Lets first observe when the problem raising when we read the data from the object state. To remove the problem just remove the object state and place it in to method level.
- Lets see the example.

```

class TransferFounServlet extends HttpServlet{
    p v service(req,res){

        private int fromAc;
        private int toAc;
        private float amount;

        fromAc = req.getParam("fromAc");
        toAc = req.getParam("toAc");
        amonut =
            Float.parseFloat(req.getParam("amount"));

        //business operation
        //display the success page

    }
}

```

- Now see the above all attributes are available into service() method.
- When first thread will create then all attributes will assigned with there corresponding values and perform the operation but while performing first thread execution, second thread will gets created now first thread get suspended but while suspending first thread, it will keep track of current line and and all the corresponding variables and there values.
- When second thread enter into service method, it will start with new values and attribute, after some time thread first will called, now second thread will track the current line where actually it suspended and it will keep all the attributes and values into thread stack.
- Now first thread will restore all the information what was the current-line, attributes and corresponding value. And it will start execution where at last time he has left.

- For every thread one stack will be there where it will store all the information, to restore the execution at same line with same data .

- **ThreadLocal Usage:**
- As per the above example we can avoid to write the synchronized block by making those attributes local to that method. But in typical application one class contains multiple method which need same data to perform the business logic.
- So we can't call every time different method to pass the same data within the class. It is some what clumsy. We can do that process but it is not a good programming practice.
- To avoid above problem if we declared these attributes at object level but we seen what is the problem when we declared at object level in above example.
- Even we declared at object level and we made corresponding method as synchronized which accessing those attributes then performance impact will arise. Bz in multi-thread environment it will allow only one thread to access that method so performance impact will arise.
-

- So we can not write at object level, we can not write at method level and we can not make corresponding method as synchronized then where to declare and how we will solve this problem.
- Actually at what time these problems are coming when multiple thread are interacting with the object or method of the class , So maintain these values at thread level itself.
- To solve and to maintain values at thread level we have to use a concept called ThreadLocal.
- ThreadLocal is the class which used to manage thread scope, for per thread it will manage the stack and it will never let one thread to talk to another thread.
- But ThreadLocal class will allow us to store only one value in it. We can set the value to the threadlocal,
 - Ex: `ThreadLocal<integer> tl = new ThreadLocal<Integer>();`
 - `tl.set(10);`
 - `tl.get();`
- By this we can solve the above problems lets see the example.

```

package com.te.threadlocal;
public class Ticker {

    private static Ticker instance;
    private int seconds;
    private ThreadLocal<Integer> local ;
    private Ticker(){
        local = new ThreadLocal<Integer>();
    }
    public static synchronized Ticker getInstance(){
        if(instance == null){
            instance = new Ticker();
        }
        return instance;
    }
    public void oscillate(){
        if(local.get()==null){
            seconds = 0;
        }else{
            seconds = local.get();
            seconds++;
        }
        local.set(seconds);

        System.out.println(Thread.currentThread().getName (
            )+" "+seconds);

    }
}

```

```

public class Clock implements Runnable {

    ThreadLocal<Integer> local ;
    @Override
    public void run() {
        Ticker ticker = Ticker.getInstance();
        local = new ThreadLocal<Integer>();
        for(int i=0;i<100;i++){
            /*local.set(10);*/
            ticker.oscillate();
        }
    }
}

```

```

public class TLTest {
    public static void main(String[] args) {
        Clock clock = new Clock();
        Thread t1 = new Thread(clock, "Thread-1");
        Thread t2 = new Thread(clock, "Thread-2");
        t1.start();
        t2.start();
    }
}

```

ThreadLocal usage and manage Thread safety

Target	Dependent	Status
singleton	singleton	yes
Non-singleton	Non-singleton	Yes
singleton	Non-singleton	no (no thread safe)

Non-singleton

Singleton

- As per the above table we can take sharable data into singleton class, if it is non-singleton then we can take non-sharable data it work but when we have singleton class and if we take non-sharable data then it will work but there is no thread safety.
- Why we need Lookup method injection lets see the reason with an example

```
class A{
    B b;
    //setter and getters
}
```

```
class B{
    int i;
    //setters and getters
}
```

application-context.xml

```
<bean id="a" class="A" scope="singleton">
  <property name="b" ref="b"/>
</bean>
<bean id="b" class="B" scope="prototype">
  <property name="i" value="10"/>
</bean>
```

- As per the above example we configured both A class and B class as beans into IOC container.
- A class is configured as singleton class but class B configured as prototype.
- When we created a IOC container using beanFactory factory both the classes place as in-memory metadata into IOC container.
- When we call `A a = factory.getBean("a",A.class);`
- IOC container check the bean scope if it is singleton it will check object all ready available into IOC container or not if it is not then it will create the object and place into IOC container. When next time we will try to get the A `a1 = factory.getBean("a",A.class)` then it will check the bean scope if it is singleton it will check object is available in IOC container or not , if it is there it will return the same object reference.

- Now the problem is class B scope is prototype even though we will get same object of B class, means if we inject prototype class into singleton class then prototype class became singleton.
- It is one of the limitation with injection.
- Every time we can not use injection to manage the dependency some time we need lookup injection also.
- One of the core feature of the spring is non-invasiveness : means without spring also we can make our application loosely coupled.

- In spring we can manage dependency in two ways
 - Dependency lookup
 - Lookup method injection
 - Contextual lookup injection
 - Dependency injection
 - Setter injection
 - Constructor injection
 - Mostly people use dependency injection for managing the dependency, but there are some limitation are available.
 - We can not use prototype scope bean in to singleton scope, bz that prototype class behave like singleton only.
 - We can not injection dynamic or runtime injection using dependency injection , we can inject static dependency injection.

- To make use of dependency lookup we will discuss a previous concept called as IDREF. Which will help better understand of look-up method injection .
- Some people call lookup method injection as a method injection and some people call method replacement as method injection.
- Let discuss IDREF concept ..
- Make our classes loosely coupled how can use dependency pulling concept. One of the concept is **IDREF** which will make our classes loosely coupled.
- **IDREF word itself describes, it refers to the id of another bean.**
- By using IDREF attribute of spring we can make our classes loosely coupled,

Ex:

```
public class Car {  
  
    private IEngine engine;  
    public void run()  
    {  
        engine.start();  
        System.out.println("Car is running.....");  
    }  
    public void setEngine(IEngine engine) {  
        this.engine = engine;  
    }  
}
```

- There are two ways available to get the values from other class
- 1). Make our method to get the value from other class

For example

```
public void m1( int i){}
```

But it is specific to the method only.

- 2).Declared Attribute at class level

if a value used by through out the class then declared that variable as class level and initialize into the constructor, for example

```
class A{  
    private int i;  
    A(int i){  
        this.i=i;  
    }  
}
```

we know the above procedure but there are some problems available.

1. `<bean id="car" class="com.idr.beans.Car">
 <property name="engine" ref="suzukiengine"></property>
</bean> -->`
2. `<!-- <bean id="car" class="com.idr.beans.Car">
 <property name="beanId" value="yamahaengine"></property>
</bean> -->`

- By using dependency injection we can inject the other bean in to the target bean but here i dont want to use dependency injection, rather my target class pull corresponding object from the Spring bean configuration file, we can make our target class to pull the correspond object by two ways
- 1. by declared one attribute in target class and inject a perticular bean id to the target attribute
- 2. by using IDREF attribute
- if we use value="yamahaengine" it may not give a correct context about what kind of value we are injecting. other developer may get confuse, but if we use IDREF it will clearly specify Idref attribute using other bean id for injection.

- So we can configure the bean using property attribute of bean and clearly specify the idref to the bean.

```
<bean id="car" class="com.idr.beans.Car">
  <property name="beanId">
    <idref bean="suzukiengine"/>
  </property>
</bean>
<bean id="yamahaengine1" class="com.idr.beans.YamahaEngine"></bean>
<bean id="suzukiengine" class="com.idr.beans.SuzukiEngine"></bean>
</beans>
```

- Let see the example idref.
- **public class Car {**
- **private String beanId;**
- **public void run()**
- {
- IEngine engine = **null**;
- */*System.out.println(beanId);*/*
- BeanFactory factory = **new XmlBeanFactory(new**
- **ClassPathResource("com/idr/common/application-context.xml"));**
- engine = factory.getBean(beanId, IEngine.**class**);
- engine.start();
- System.***out.println("Car is running.....")***;
- }
- **public void setBeanId(String beanId) {**
- **this.beanId = beanId;**
- }
- }

- As per the above example now my application became loosely coupled but still there are bunch of problems are there :
 - All the beans are available into one IOC container even though my car class creating IOC container to get the bean from the IOC container .
 - all beans are in same IOC container and to avoid creating another bean we can use BeanFactoryAware interface. Which will help to use same factory object to get the beans.
 - When we use the BeanFactoryAware interface then this procedure is called as contextual dependency injection.
 - To get the factory object we have to follow some contract then only we will get factory object.
 - See the below example how we will implements BeanFactoryAware interface and usage of it.

Ex:

```
Class Car implements BeanFactoryAware{
String beanId;
BeanFactory factory;
public void run()
{
    IEngine engine = null;
    engine = factory.getBean(beanId, IEngine.class); //engine = getEngine();
    engine.start();
    System.out.println("Car is running.....");
}
public void bread(){
    IEngine engine = null;
    engine = factory.getBean(beanId, IEngine.class); //engine = getEngine();
    engine.stop();
    System.out.println("car stoped .....");
}
public void setFactory(BeanFactoty factory){
this.factory = factory;
}
public Iengine getEngine(){
    return factory.getBean(beanId,"Iengine.class");
}
}
```

- We are duplicating the logic for getting the bean in multiple method to avoid duplication take one method which will get the bean and return it, wherever it required can refer direct that method.

- As per above example we make our code some what loosely coupled but its not totally loosely coupled, still my `getEngine()` method talking to the IOC container to get the bean.
- To make totally loosely coupled my car class should not implements `BeanFactoryAware` interface and it should not write pulling logic, rather declare one abstract method in car class and make car class as abstract class and configure that class in IOC container.
- Ex:

```
<bean id="car" class="Car">  
  <look-up method="getEngine" bean="yamahaEngine"/>  
</bean>  
<bean id="yamahaEngine" class="YamahaEngneImpl"  
  scope="prototype"/>
```
- once we configured that class into IOC container, container will check the scope the bean if it is singleton then it will check object already available or not if it is not then it will start creating the object and it observed that the bean is configured with look-up method i.e. `getEngine()` and the corresponding class as abstract class then IOC container will generate one proxy which is extends from the Car class.

- Ex:

```
Class Car$Proxy extends Car{  
    public IEngine getEngine(){  
        // logic for returning corresponding bean object  
    }  
}
```

- Whenever we try to get the object of Car class then IOC container will generate runtime proxy class and return the proxy class object to the car class object.
- IOC Container will do this process when he look at the look-up configuration and it will inject the corresponding bean to the corresponding method.
- In generate we can not configure abstract class as bean, but when we are dealing with look-up method injection then only it is possible bz of look-up method configuration.
- While configuring that class and method into IOC container those should not be final and static.
- Let see the example

Look-Up method Injection

```
IRequestProcessor{  
    setData(String data);  
    processRequest();  
}
```

```
class HttpRequestProcessorImpl implements IRequestProcessor{  
  
    p v processRequest(){  
  
        System.out.println("processing data:"+data+"hashCode  
                            :"+this.hashCode().toString());  
    }  
    p v setData(String data){  
        this.data = data;  
    }  
}
```

```
abstract class Container{  
    p v request(String data){  
        IRequestProcessor process = null;  
        process = getRequestProcess();  
        process.setData(data);  
        process.processRequest();  
    }  
  
    abstract p IRequestProcessor getRequestProcess();  
}
```

```
<bean id="container" class="Container">  
    <look-up name="getRequestProcess" bean="httpRequestProcessor"/>  
</bean>  
<bean id="HttpRequestProcessor" class="HttpRequestProcessorImpl"/>
```

```
class LMTest{  
    p s v main(String[] args)  
    {  
        BF f = new XmlBF(new CPR("application-context.xml")  
        Container container =  
        f.getBean("container", Container.class);  
        container.request("lookup method injection")'  
    }  
}
```

Here we will get Container\$proxy class object

```
class Container$Proxy extends Container{  
  
    public IRequestProcessor getRequestProcess(){  
  
        return // configured bean object  
    }  
}
```

Spring 94,95,96,97,98

- While developing a project there are so many requirements are there which will happen after deployment and some has to happen before the deployment.
- Actually in project every task depends on object there are some situations where we can using empty object to accomplish the task, but there are some situation without additional data we can not perform the task.
- Before use that object we have to manually add the data and give to the programmer to use it. Means before using that object if we want to perform some operation we can do using the postProcessor concept.
- Spring has provided below concept which help us to do custom manipulation into Object or Beanfactory.
- **postProcessor : After creating an object and before use that object to performing some operation called as postProcessor work.**
 - BeanFactoryPostProcessor
 - BeanPostProcessor

- **BeanFactoryPostProcessor:**
 - Using BeanFactory we will create an IOC container which contains numbers of beans.
 - But there are some situation before using that IOC container , if we want to perform the some additional configuration on IOC container then we can do using BeanFactoryPostProcessor.
- **BeanPostProcessor:**
 - One bean represent one object, when we call getBean method then we will get the corresponding bean object but before use that object if we want to perform some operation then we can use BeanPostProcessor to add the extra behavior to the object.

- Bug handler process
- States of the bugs
 - Open:
 - When bugs encounter into the project then the state of the bugs is open or new.
 - Not producible :
 - If bugs are not resolvable then that state is called not producible.
 - Not a bug
 - While deploying the project into QA env. There are several problems may encounter, some of them platform comparability , database related issued which is not known to the taster, so without verifying they may raise a bugs.
 - So after verification developer going to state that it's not a bug.
 - Fixed
 - One the bug has been fixed then that state called as fixed state.
 - Verified
 - Before fixing the bugs developer has to check its really bugs or not, then only they going fix .
 - Closed
 - Once the bug has been resolved then tester will check bug has been fixed or not and once they got clarification then they will state that bug closed state.
- How to deploy the project into Quality assurance environment?
 - Developer don't have authority to deploy the project in QA environment.
 - One of the QA engineer only can deploy the project into QA Env. But QA engineer has to follow some sort of procedure while deployment.
- What are the problems when project deployed by the developers ?
 - Given a chance developer can deploy a project into QA Env. But its security bridge.
 - While testing an application if a module encountering more bugs then developer can easily can modify the code and update into the SVN repository. Again there are no. of problems are their.

- What are the reasons developer will not get permission to access the QA Env.?
 - The project has been developed by the developer then they don't want more bugs encounter into the application, if more bugs are encountering then developer may can modify the code easily, if he has authentication permissions.
 - A application contains bunch of classes and these are interconnected with each other, so one class change will going to affect whole application.
 - Without considering other classes if developer modify one class which collapse whole application.
- What are the drawbacks when we provide installment procedure by document?
 - Actually developer never get a chance to deploy the project into the QA env. But QA people also unable to deploy the project bz they don't have any idea about how to deploy a project.
 - It developer job to prepares document which clearly stated that how to deploy the project, how to checkout the code from SVN repository, how to configure the server and several things.
 - But it is very hard to QA people to follow the document and perform the deployment procedure. Its time consuming process and they may encounters some problems.....

- It is not easy job to deploy the project into QA Env. There are several configuration files are available which contains several beans are registered, finding corresponding bean and modifying that beans are every difficult job.
- Tester may by mistakenly change the other beans or configured wrong data into configuration file.
- when they run the build tool, tool will deploy the project with wrong configuration which lead savior problems. It will waste the tester time as well as developer time...and so on.
- Even developer has provided property file for resolving the above problem even though there may chance to lead to problems.
- To overcome the above tools problems spring has provided as feature called **BeanFactoryPostProcessor**.
- To overcome the above problems building tools came into the market.
- There are some building tools
 - **Ant**
 - Ant is the building tool which is used for automatic building the project.
 - Ant provide the flexibility for make our application automatic, a one of the developer has write the ant script for configuring the server, getting the code from the repository, deploying the project and so on.
 - We can make our application completely automated by build tools. Only QA people job is to run the build tool.
 - But there are some drawback are available with the automated tools also, if we partially build the building script then it is problematic.

– **Maven**

- In industry they are to follow some standard directory structure which help every one while deploying the project.
- No will get distract while placing database related files, tools related files, configuration related files and so on.

- **Drawbacks with tools when they partially written:**

- For example a project has been developed by the developer along with that one of the developer has written building script for ant tools, for making deployment automated.
- but as per the above discussion tools also has problems.
- To overcome the above tools problems spring has provided as feature called **BeanFactoryPostProcessor**.

- **BeanFactoryPostProcessor:**

- Using BeanFactory we will create an IOC container which contains number of beans.
- But there are some situation before using that IOC container , if we want to perform the some additional configuration on IOC container then we can do using BeanFactoryPostProcessor.
- Lets see the example first we will discuss that example.

```

public class ConnectionFactory {
    private String url;
    private String driverClassName;
    private String userName;
    private String password;
    public void setUrl(String url) {
        this.url = url;
    }
    public void setDriverClassName(String driverClassName) {
        this.driverClassName = driverClassName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}

```

db.properties

```

url=jdbc\:oracle\:thin\:@localhost\:1521\:xe
driverClassName=oracle.jdbc.driver.OracleDriver
userName=sachin123
password=sachin@123

```

```

public class BFPPTest {
    public static void main(String[] args) {
        BeanFactory factory = new XmlBeanFactory(new
        ClassPathResource("com/bfpp/common/application-context.xml"));
        BeanFactoryPostProcessor bfpp =
        factory.getBean("bfpp", BeanFactoryPostProcessor.class);
        bfpp.postProcessBeanFactory((ConfigurableListableBeanFactory) factory);

        ConnectionFactory connectionFactory =
        factory.getBean("connectionfactory", ConnectionFactory.class);
        System.out.println(connectionFactory);
    }
}

```

```

<bean id="connectionfactory" class="com.bfpp.beans.ConnectionFactory">
    <property name="url" value="${url}"/>
    <property name="driverClassName" value="${driverClassName}"/>
    <property name="userName" value="${userName}"/>
    <property name="password" value="${password}"/>
</bean>
<bean id="bfpp"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location" value="file:A:\Sriman\Spring Data\Spring Data
\STSWorplaceForSpring\AdvanceSpring\BeanfactoryPostProcesser\src\db.properties">
    </property>
</bean>

```

Place Holder/Expression tokens in spring

BeanFactoryPostProcessor

- As per the above example we can change the property values at runtime, after creating IOC container spring will perform runtime injection.
- Spring has given expression tags which help in holding the place values and it will going to change at runtime.
- Place holders will take key as value and that key will change with original value at runtime, but we have to configure the property file to the spring bean configuration file.
- In property we have to place key and corresponding values which will going to change at runtime in configuration file, wherever place holder is available with key , spring will find that key into property file if it is available then replace that key with corresponding value which is available into property file.

- Spring has provided a interface called BeanFactoryPostProcessor, By help this interface we can write the class which implements BeanfactoryPostprocessor and we will get one method **called public void postProcessBeanFactory(ConfigurableListableBeanFactory object) throws BeansException {..}**
- By this we can write the logic and perform the runtime changes into IOC container.
- But spring has provided predefined class which is concrete class just we can to configure that class into spring bean config. File and pass the property file location and that class is **PropertyPlaceholderConfigurer** which going to take location and property file as input for next operation.
- IOC container can identify and inject corresponding value for particular place holder.
 - **Location**: it is predefined attribute which will take **classpath** or **file** as a input.
 - **File** : if we use file then we have to give the absolute path of the file. Even we can give outside system location also where actually our file is available.
 - **Classpath**: if we use classpath then we can give relative path of the file.

- Configuring PropertyPlaceholderConfigurer is not enough we have to add that bean into IOC container, then only IOC container will perform the operation.
- After creating IOC container we have add this extra configuration into IOC container .
- To add PropertyPlaceholderConfigurer we have to use ConfigurableListableBeanFactory.
- Let see
- **BeanFactoryPostProcessor bfpp = factory.getBean("bfpp",BeanFactoryPostProcessor.class);**
- **bfpp.postProcessBeanFactory((ConfigurableListableBeanFactory)factory);**
- Now IOC container automatically update place holder with property file values.

- **BeanPostProcessor:**
 - One bean represent one object, when we call getBean method then we will get the corresponding bean object but before use that object if we want to perform some operation then we can use BeanPostProcessor to add the extra behavior to the object.
- After creating an object into IOC container and before use that object we want to perform some initialization or customization then we can easily use Bean Life cycle method. i.e. init-method and destroy-method.
- So what is the need for creating an new feature called BeanPostProcessor.
- In bean lifecycle there are some problems are available.
- For example if I want to know how many object are available into IOC container , by using bean lifecycle we can find but in every class we have to write the same sort of logic and every bean we have configure init-method.
- Lets see the example. By Mr.Sachin Gaikwad

```

public class Statistics {
    protected static AtomicInteger instanceCounter = new AtomicInteger(0);

    public static AtomicInteger getInstanceCounter() {
        return instanceCounter;
    }

    public static void setInstanceCounter(AtomicInteger instanceCounter) {
        Statistics.instanceCounter = instanceCounter;
    }
}

```

manually counting the beans from IOC container

```

public class A extends Statistics {
    public void init(){
        Statistics.getInstanceCounter().incrementAndGet();
        System.out.println("class A working");
    }
}

```

```

public class B extends Statistics {
    public void init(){
        Statistics.getInstanceCounter().incrementAndGet();
        System.out.println("class A working");
    }
}

```

```

public class C extends Statistics {
    public void init(){
        Statistics.getInstanceCounter().incrementAndGet();
        System.out.println("class A working");
    }
}

```

application-context.xml

```

<bean id="a" class="com.blc.beans.A"
    init-method="init"/>
<bean id="b" class="com.blc.beans.B"
    init-method="init"/>
<bean id="c" class="com.blc.beans.C"
    init-method="init"/>
<bean id="statistics"
    class="com.blc.beans.Statistics"/>

```

- **AtomicInteger:**
 - An int value that may be updated atomically. An AtomicInteger is used in applications such as atomically incremented counters, and cannot be used as a replacement for an [java.lang.Integer](#). However, this class does extend Number to allow uniform access by tools and utilities that deal with numerically-based classes.
 - It has provided number of methods which make programmer life easy.
- As per the above example we end up with writing duplicate logic in each and every class, to avoid and to add more extra capabilities into beans we can use

```

public class ValueObjectBeanPostProcessor implements BeanPostProcessor {

    AtomicInteger beanCount= new AtomicInteger(0);
    public Object postProcessAfterInitialization(Object bean, String beanName)
        throws BeansException {
        beanCount.incrementAndGet();
        return bean;
    }
    public Object postProcessBeforeInitialization(Object bean, String beanName)
        throws BeansException {
        return bean;
    }
    public AtomicInteger getBeanCount(){
        return beanCount;
    }
}

```

```

public class BPPTTest {
    public static void main(String[] args) throws SQLException {
        BeanFactory beanFactory = new XmlBeanFactory(new
        ClassPathResource("com/bpp/common/application-context.xml"));
        ValueObjectBeanPostProcessor postProcessor =
        beanFactory.getBean("valueObject", ValueObjectBeanPostProcessor.class);
        ((ConfigurableListableBeanFactory)
        beanFactory).addBeanPostProcessor(postProcessor);

        ValueObjectBeanPostProcessor bpp =
        beanfactory("bpp", ValueObjectBeanPostProcessor.class);
        sysout(bpp.getBeanCount());
    }
}

```

BeanPostProcessor beanCount example

- As per the above example we can configure multiple beans into spring bean configuration file and we can get object also.
- But before giving that requested object to the programmer IOC container will perform BeanPostProcessing logic, we can write bean counting logic or different logic also.
- In One of the use case we can use BeanPostProcessor i.e. Auditing.
- **What is mean by Audit?**
 - In business data matter a lot its very important to keep track of the data in daily basis.
 - Auditing is use for tracking , avoiding the fraud and keeping daily record in it.
 - Auditing are used for security purpose also.
 - By this use can track login details and performed activities.
 - In industry every employee has there own userId and password to use the system and depends up on the employee position company will give special kinds of authentication.
 - By using logic details audit will track what is going on under that logic and keep track.
 - Audit table is the separate table which allow duplicate records to be inserted.
 - By this we can track what activities going on under corresponding logic details.
 - every typical application has to provide auditing as part of application. For example in bank applications , they has to track what activities going on under corresponding login.

- Audit are different type some of them implements on the role which are available into the application, some of them application level , and some of user level.
- Beanpostprocessor will help in sharing the common behavior and we can restrict that behavior for specific bean also. which will going to create inside the IOC container.
- For easy understanding lets see one of the example how we can restrict the behavior for specific bean.

By Mr.Sachin Gaikwad

• Audit plays vital role in industry explore

```

abstract public class Edit_Employee {
    private EmpDao empDao;

    public void edit(int empId,String firstName,
                    String lastName,String gender,
                    float salary)throws SQLException{

        EmpVo empVo = null;
        empVo = lookUpEmpDao();
        empVo.setEmpId(empId);
        empVo.setFirstName(firstName);
        empVo.setLastName(lastName);
        empVo.setSalary(salary);
        empVo.setGender(gender);
        empDao.Update(empVo);
    }

    public EmpDao getEmpDao() {
        return empDao;
    }

    public void setEmpDao(EmpDao empDao) {
        this.empDao = empDao;
    }

    abstract public EmpVo lookUpEmpDao();
}

```

```

public class EmpVo extends ValueObjectBaseClass{
    int empId;
    String firstName;
    String lastName;
    String gender;
    float salary;
    //setter and getters
}

```

Sharing created date and last Modified date at bean creation

```

public class ValueObjectBaseClass {

    protected Date created_dt;
    protected Date last_Mod_dt;
    //setter and getters
}

```

```

public class ValueObjectBeanPostProcessor implements
BeanPostProcessor {
    @Override
    public Object postProcessAfterInitialization(Object
bean, String beanName)throws BeansException {
        if(bean instanceof ValueObjectBaseClass){
            ((ValueObjectBaseClass) bean).setCreated_dt(new Date());
            ((ValueObjectBaseClass) bean).setLast_Mod_dt(new Date());
        }

        return bean;
    }

    @Override
    public Object postProcessBeforeInitialization(Object
bean, String beanName)throws BeansException {
        return bean;
    }
}

```

```

p s v main(..){
    BeanFactory beanFactory = new XmlBeanFactory(new
ClassPathResource("com/bpp/common/application-context.xml"));
    ValueObjectBeanPostProcessor postProcessor =
beanFactory.getBean("valueObject",
ValueObjectBeanPostProcessor.class);
    ((ConfigurableListableBeanFactory)
beanFactory).addBeanPostProcessor(postProcessor);
    Edit_Employee edit_Employee =
beanFactory.getBean("editEmp", Edit_Employee.class);
    edit_Employee.edit(1,"sachin", "gaikwad", "Male",
150000);}}

```

- **Event Processing :**
- Before we discuss event processing we have to understand how many ways we can process the event processing in the programming world.
 - There are two ways
 - **Synchronous :**
 - Synchronous driven approach is the linear execution approach one block of execution relay on other block.
 - Collee will going to call the caller and Collee has to wait until and unless caller execution will finish, after getting the cursor it will start execution.
 - In generate one method will call the other class method then first method has to wait until caller method execution finishes.
 - Drawback with synchronous approach is :
 - 1)if one class contains multiple method and they are independent from each other, but one of the method calling other class method then other method has to wait up to completion of that method.
 - 2)It is time consuming approach.
 - 3)performance will not be good.
 - 4)In Synchronous one class or method tightly coupled with each other.
 - **Asynchronous :**
 - Asynchronous driven approach is the simultaneous execution approach one block not relay on other one.
 - Collee and Caller both will going to execute parallel.
 - Advantages :
 - 1)One method can not block other method both are independent in asynchronous approach..
 - 2) In Asynchronous classes and method are loosely coupled.
 - 3)performance will high or throughput will be high.

- What is Event driven approach?
 - Event processing design always loosely coupled, always in event processing design there are four components are involved.
 - Source: source is responsible for triggering the action/event.
 - Here source mean any component which going to trigger the event.
 - Event
 - Listener
 - Handler

Spring 106,107

- Annotation:
- 1) journey spring 2.0 has been started.
- 2)annotation has polymorphic behavior.
- 3)in industry people usage annotation
- Need
- =====
- 1)why annotation
- 2)why should learn annotation
- 3)configuration is there what is the need of annotation
- class xservlet extends HttpServlet{
- p v service(req,res){
- }
- }
- =>Only just creating an servlet is not important because we are not eligible to call the servlet method.
- servletContainer is the person we take care of calling the servlet method.

- => being a programmer we can write the servlet but servlet the server side program which run on network to fulfil the requirement of the clients.
- => we are not responsible for calling servlet, servlet is called by the ServletContainer but until and unless we configure our servlet to the ServletContainer it will not call the servlet method.
- => to configure our servlet to the ServletContainer configuration approach came into feature.
- => in core java we only manage, creating the object for the program.
- => Java has provided API's for network programming but programmer can not write the network logic to execute program on network, it's bit difficult to write.
- => Bz of that JEE has provided prebuild API's which has prebuild network logic for executing the program, so we can write program by obeying the rule which is given by JEE API's .
- => the program will execute by other container or other environment so tell them the information about the servlet or program java has provided configuration file.
- => a configuration file will go to describe the information about the class, like what is the class name, what are the methods, attributes of the class and so on.
- => a file which goes to give the information about the class called as METADATA of the class.

- => **why sunMC has chosen xml only?**
- => As we know the benefits of the xml
- => wellformedness is there, we can present our data in structured manner.
- => easy to understand. => we can easily validate our xml. => easily interpretable.
- => xml is interoperable language. => any environment can easily understand.
- => we will get well structure using XSD or DTD to define our class information.
- => we can apply restriction what to write and how to write.
- => java is the platform independent means compile once and run anywhere, the exact principle available with the xml, i.e. xml is the interoperable, we can use anywhere.
- => Java started journey with XML but over the time most of the programmer finding difficulty to write the XML.
- => Most of the java developer not showing their interest in XML file they are finding difficult to write it.
- because of that Annotation notion came into feature.
- => previously we used to write the configuration file for providing the information to the containers, which is called as configuration metadata.
- => to provide information about the classes we use configuration file, but in annotation driven approach still we are providing the information using annotation, there are write into configuration file here we have to write with source code.
- => with source code only we have to write the annotation.

- ==> **QQ.can Annotation replace the configuration approach or not?**
- There are two way of providing the information to the container
- **1)configuration file** => Here also we can provide the information, it is not executable logic or code, it is only used for providing the information about the class metadata.
- **2)annotation** => Annotation also one of the approach used for providing the information about the classes to the underlying container. Annotation also not executable code it will not executed by container it will provide the information about the metadata of the class.
- **QQ. what makes people to go away from the xml ? and what are the problems they faced while working with xml?**
- ==>In Core java also having internal annotation which will provide the information to the JVM ,clonable,Marker interfaces and so on. which is going to give additional information to the JVM. while running this class what kind of additional capabilities added to the perticular class.
- ==>In industries people are irriteted by xml being a java developer why we should we xml as part of the java application they feeling so difficult.
- ==>many time they approached to SunMC also but java hasn't provided any support to the developer.
- ==>bz of that other third party venders are came forward for providing the annotation support to the java developer.
- ==>XDoclet is the first tool how came up with annotation support in for java. eventually it fails also.
- ==>there are lot's of third parties started providing the support to annotation.most of the people started annotation provided libraries, framework and IDE.
- ==>over the time java has

• XML drawbacks :=>

- =====
- 1)
- => speak more / web bost
- => not so easy to remember tags
- => order of the tags we have to follow
- => remember the order of tags
- => to work with xml java developer has to has complete picture of xml.
- 2)
- => for a complete java developer it's very difficult to work with xml.
- 3)
- => java developer has to write xml file which is not a java part.
- => we can easily compile the class and we can easily validate our logic is right or wrong, written syntax, type method right or wrong, but coming to xml we can not validate declared class and package name or other information is right or wrong.
- 4)
- => there is no prosen we can validate our configuration is right, until and unless we deploy that application and access that application then only we can validate declated configuration is right. XSD and DTD will validate the structure but not the data, what kind of data you have provided validated after deployment if the application only,
- bz of that java developer irritated while working with xml.
- 5)
- => while working with xml validation became repeatative process within one short we can not validate all configured classes are right or wrong, until and unless we access that class corresponding container will not check it is right or wrong.
- 6)
- => To develop an application there are several IDE's available which pramote rapid application development and which make java developer life easy, but coming to xml there is not IDE's which is supporting rapid application deveopment .

- **Annotation :=>**
- =====
- 1)Annotation speak less.(means one word only will discribe whole configuration file)
- 2)Annotation writes along with the source code itself.
- 3)Annotation validation happen at the time of compilation.
- 4)Annotation are the java code every developer feel more comformtable while working with annotation.
- 5)it is every easy for developer to validate java code and provided additional information.
- 6)There are number of IDE's supports annotation based approach.
- **XML Advantages: when we configured our classes using XML:**
- =====
- 1) XML will give wholestic picture of the all the classes which are available into application.
- It will give all classes information,dependencies,properties, references and moreover. By looking at XML file we can easily understand whole application and there corresponding information.
- **Ex:**
- if new developer hired into the project, He no need to visit all the classes to understand the dependencies, properties, classes and so on , by looking at Xml configuration file only he will going to understand.
- 2)If there are changes into classes or information into the class then there is no need to change the classes, just we can change the configuration information automatically changes are rollback to the classes. there is no need to compile the classes if there are change also.
- 3)If there are any bugs are available then we can easily fixed the bug by looking at xml file.

- **DrawBacks With Annotations:**

- =====

- 1)Unlike XML, Annotation are written under each and every class, we can not easily derive which class is depends on which other class. A perticular class will describe itself rather than other dependent classes or attributes.
- To get wholestic picture of the application we have to visit each and every class and understand the relationship between the other classes. It is every difficult to the new developer to understand the application.
- 2)All the annotations are written along with the source code,If there is change in configuration of the classes then we have to change into class, if there is change into class means we have to recompile the code then only the changes gets affected. it's lead to maintainence of the application(recompile,redeploy,time,...).
- 3)if there is bug then we have to understand the complete picture and we have to visit each and every class which is related to the corresponding bug.
- 4)annotations are every short it may not give clear picture.

- **Annotation Types:**

- =====

- 1)document assistors or code assistors
- =>
- 2)Runtime assistors
- 3)compiler assistors

- @Autowired
 - Autowiring :
 - In general programmer is responsible for managing the dependency between the classes.
 - Actually all the managing beans are available into the IOC container, so we can tell to the IOC container to manage the dependency itself by providing additional configuration into spring bean configuration file.
 - To make automated there are different modes are provided by spring people we have to use them.
 - There are four modes available
 - 1)ByName
 - 2)ByType
 - 3)Constructor
 - 4)autodetect (which is deprecated)
 - By enabling the autowired mode we can manage the dependency.
 - The above approach is related to the xml configuration but there is another way available by this we can manage the dependency.
 - Annotation Driven approach.

- We can enable Autowired by two ways
 - 1) Configuration approach (ie xml driven approach)
 - 2) annotation driven approach

2) Annotation driven approach

- @Autowired annotation has a default mode called as Type only.
- It will work on type of the attributes, it will not support other kind of modes which are available into xml configuration.
- When we write @Autowired then by default one attribute becomes true i.e. required = true. If we use @Autowired annotation then that setter or constructor or attributes become mandatory.
- Without using @Required we can impose mandatory constraint on setter injection by using @Autowired.
- We can write Autowired annotation in four places into the class.
 - **1) Attribute level:** we can write @Autowired at attribute level IOC container automatically injects dependent bean to the target bean, it will take attribute name and take the type of the attribute and check whether corresponding attribute type bean is available into the in-memory metadata if it is there then it will inject the bean object directly.
 - **2) constructor level :** we can write @Autowired at constructor level which becomes mandatory.
 - **3) setter level:** we can write @Autowired at setter level which becomes mandatory. To make it optional we can make required as false. Ex: @Autowired(required=false).
 - **4) Orbiter level: means we can take any general method and we can write @autowired , IOC container will inject corresponding beans.**
- We cannot write two constructors having @autowired as annotation; it will not work; we will get an exception. To make it work we can make required as false (@Autowired(required=false)).
- If two constructors are available then make required as false; then it will work as per the rules – max parameters and if every constructor contains the same number of parameters then it works as per hierarchy from top to bottom.

- Orbetry level:

- Ex:

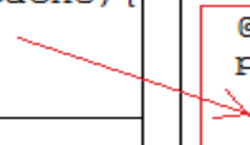
```
class CacheManager{
    private Cache cache;
    private CacheManager(Cache cache){
        this.cache = cache;
        init();
    }
    public void init(){

        //fetching data from DB and
        loading to cache

    }
}
```

```
class CacheManager{
    private Cache cache;

    @Autowired
    public void init(){
        //fetching data from DB and
        loading to cache
    }
}
```



Usage of Autowired, at a time creation of CacheManager init() method will load the data into cache

- Lets see the example of @Autowired Annotation with all the permutation and combination.

```
public class Sensor {
    private int id;
    private String type;
    //setter and getter
}
```

```
public class AWTest {

    public static void
main(String[] args) {
    ApplicationContext
context = new
ClassPathXmlApplicationContext("com
/aw/common/application-
context.xml");
    Toy toy =
context.getBean("toy", Toy.class);

    System.out.println(toy);
}
}
```

```
//application-context.xml
-----
<bean id="sensor"
class="com.aw.beans.Sensor">
    <property name="id"
value="01"/>
    <property name="type"
value="infranet"/>
</bean>
<bean id="toy"
class="com.aw.beans.Toy"></bean>
```

```
<context:annotation-config/>
```

```
public class Toy {
    /*
     * @Autowired : Attribute level Internally IOC container
will use reflection API to create the object of
corresponding attribute type
     */
    /*private Sensor sensor;
    /*
     * @Autowired : it will take argument type and it will inject
     *
     * @Autowired(required=false): We can make Autowired as option also by making
     * required attribute as false.
     */
    public Toy(Sensor sensor) {
        this.sensor = sensor;
    }
    //getter method
    /*
     * @Autowired : when we declared as setter level then
     * it will take parameter type and it will inject.
     */
    public void setSensor(Sensor sensor) {
        this.sensor = sensor;
    }
    /*
    @Autowired : IOC container will call virtual constructor to inject the corresponding
beans.
    //Arbitrary Method : which is called by taking attribute type and inject it.
    public void newSensor(Sensor sensor){
        this.sensor = sensor;
    }
}
```

[@Autowired example with all the ways](#)

Spring 109

- There are some problems with @Autowired Annotation to overcome the problems we have to use another annotation i.e. @Qualifier.
 - Actually @Autowired works on Type of the attributes if there are number of beans are available into spring bean configuration file with the same type then it is very difficult to identifies.
 - Ex:
 - `<bean id="a" class="A">`

 - `</bean>`
 - `<bean id="b" class="A">`

 - `</bean>`
 - IOC container get ambiguity which bean has to inject, and we will get ambiguity exception.
 - We can resolve this problem by two ways
 - Configure one attribute into spring bean configuration file and
 - Annotate the corresponding bean into class with @Qualifier annotation.

application-context.xml

```
<bean id="chain1" class="Motor">  
  <qualifier value="Metalchain"/>  
</bean>  
<bean id="chain2" class="Motor"/>
```

```
class Motor{  
  
    @Autowired  
    @Qualifier("Metalchain")  
    Chain chain;  
  
    //setter and getter  
}
```

@Qualifier_Example

- We can use @Qualifier at three places into the program.
- If we just annotate the @Qualifier into the class then by default it will assume attribute name as Qualifier name and it will search bean into Spring bean Configuration File with attribute name and inject it.
- If we specify the qualifier name then IOC container will search for specified bean only.
- We can give bean ID as a qualifier name but there is a problem, we are hard coding the bean id, So in future we can not change the bean id Bz of that we can not refer Bean id as the Qualifier name.

- We can use @Qualifier annotation at three places into the class.
 - 1) Attribute level
 - 2) Constructor level
 - 3) setter level
 - Lets see the example

```

public class Person {
    private int personId;
    private String name;
    private int age;
    /*@Autowired
    @Qualifier*/ It will take attribute name as bean name and inject it.
    private Address address;
    /*@Autowired
    @Qualifier*/It will take attribute name as bean name and inject it.
    private Passport passport ;
    @Autowired    manually we can set the argument name
    public Person(@Qualifier("address1")Address address,
    @Qualifier("passport1") Passport passport) {
        this.address = address;
        this.passport = passport;
    }
    //setters and getters
}

```

@Autowired with @Qualifier annotation

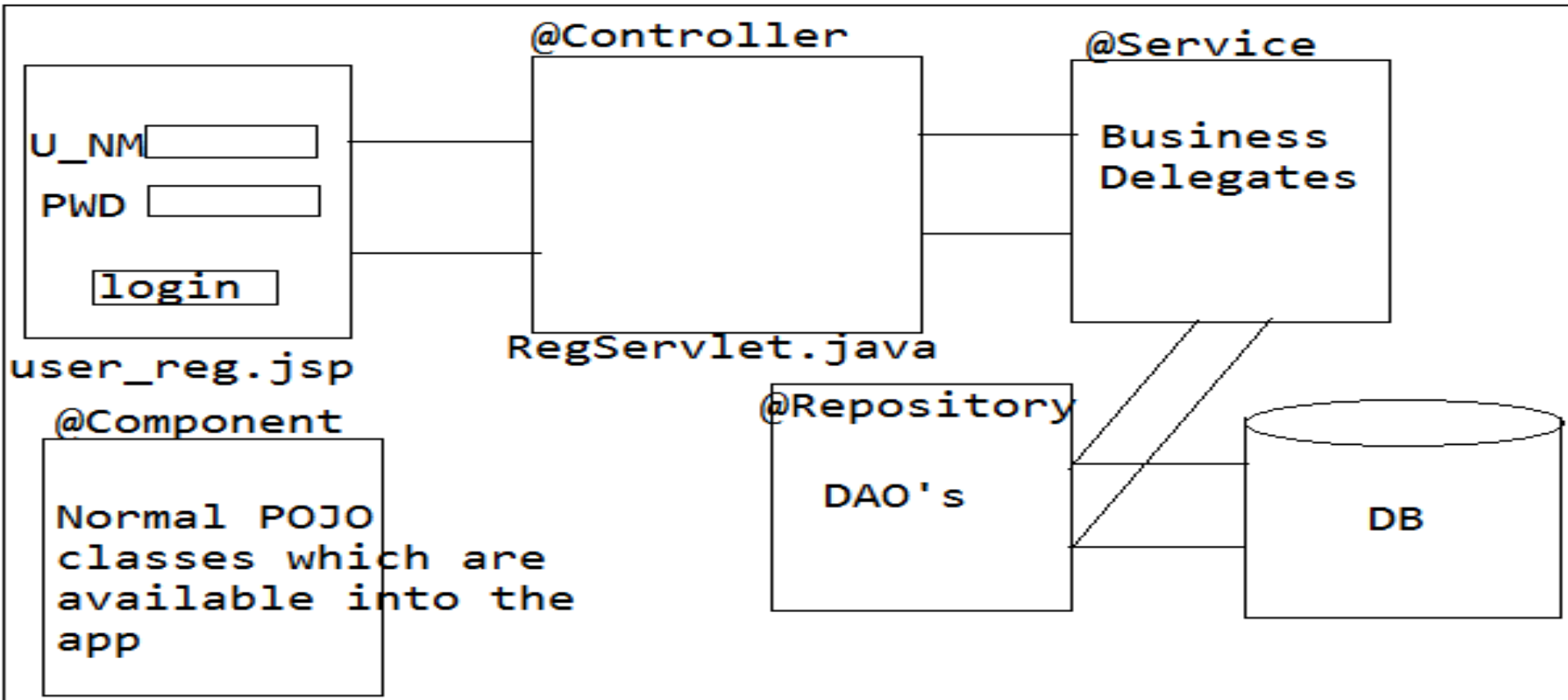
```

<bean id="passport1" class="com.awq.beans.Passport">
    <property name="passportNo" value="101"/>
    <property name="type" value="normal"/>
    <property name="renewalDate" value="10/10/2015"/>
</bean>
<bean id="person" class="com.awq.beans.Person">
    <property name="personId" value="101"/>
    <property name="name" value="sachin"/>
    <property name="age" value="25"></property>
</bean>
<bean id="date" class="java.util.Date"/>
<bean id="address1" class="com.awq.beans.Address"/>
<bean id="address2" class="com.awq.beans.Address"/>
</bean>

```

- **Stereotype Annotations:**
- Spring allows us to create IOC container in two ways
 - 1) **BeanFactory**
 - 2) **Application context**
 - 1) To ensure beans are available into IOC container we have to write spring bean configuration file and we have to configure all the beans. After that pass configuration file as input to the Beanfactory or ApplicationContext. Beanfactory or ApplicationContext read all the beans which are configured into configuration file and place into IOC container as bean.
 - 2) Another way to make our class as bean and place into IOC container we have to use **Stereotype Annotation**.
 - Stereotype annotation are developed by considering typical web application development.
 - There are four stereotype annotations available
 - **@Component** : By using @Component we can make our class as a bean and IOC container will take the class and places into IOC container as bean.
 - **@Controller** : @Controller annotation also used for making our class as bean into IOC container.
 - **@Service** : @Service annotation also used for making our class as bean into IOC container.
 - **@Repository**: @Repository annotation also used for making our class as bean into IOC container.

- A web application divided into different layers
 - Presentation layer
 - Business layer
 - Persistency layer
- To represent and to identify particular layer spring has given four stereotype annotations.
- **@Component**: A general POJO class or a General class represented by @Component annotation.
- **@Controller** : A class which control the request processing and replay accurately that class represented by @Controller.
- **@service** : This annotation used to represents the service class which will going to provide the services to the other classes.
- **@Repository** : This annotation use to represent the persistency related classes.
- But all the four stereotype Annotation are creating the beans into IOC container.
- Lets see the below diagram which will give more clarity.



Spring 110

- **@Scope** :we can specify scope of the bean by @scope annotation. By default scope of the bean is singleton.
 - we can use prototype and other scope also.(ex. Session , request, global) .
- **@Lazy** : Application context used to create the IOC container but it will instantiate all the bean at the time of creation IOC container whose scope is singleton.
 - But we can make it lazy also by annotating that class as @Lazy annotate.
- **@DependsOn**: one class is not directly depends on other class rather indirectly they are dependent, so we can use @DependsOn annotation.
 - Ex: Calculator class want data but data is not available into cache to load data into the cache, CacheManager is responsible, means until and unless data is available into cache calculator class will not work.but to load the data CacheManager is required, means calculator indirectly depends on CacheManager.
 - So we can achieve this by @DependsOn annotation in annotation driven approach .
- **@Value("#{}"): ("#{}")** this is the spring expression which is used to read the data dynamically or from property file and so on.
 - we can directly can assign the values but in future we can not change that value easily. Which is hard coded into the program.
 - Bz of that spring has provided a expression place holder.
- Lets see the example

```

@Component // placing bean into IOC container with same class name ex:"car".
//@Component("car1") we can give another name also.
@Scope("prototype")//create multiple object for the same bean.
public class Car {
    //@Value("10")
    @Value("#{appProps.Car_carId}")
    private int carId;
    //@Value("Red")
    @Value("#{appProps.Car_color}")
    private String color;
    //@Value("Tata")
    @Value("#{appProps.Car_manufacturer}")
    private String manufacturer;
    //@Value("5565566")
    @Value("#{appProps.Car_price}")
    private float price;
    //setters and getters
}

```

@Component
@Scope
@Value(both ways hard code and dynamic)

```

@Configuration
public class AppConfig {
    @Bean(name="appProps")
    public Properties newProps() throws IOException{
        Properties properties = new Properties();

        properties.load(this.getClass().getResourceAsStream("AppValue.p
                                                                    roPERTIES"));

        return properties;
    }
}

```

```

public class SCTest {
    public static void main(String[] args) {
        //ApplicationContext context = new AnnotationConfigApplicationContext("com.scope.annon");
        ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
        Car car2 = context.getBean("car",Car.class);
        System.out.println(car1);
    }
}

```


- **@Configuration**: As we going to write spring bean configuration file like that here also we going to write the configuration class which is responsible for creating beans and placing into IOC container.
 - if we write @configuration annotate class that class acts as configuration file, more-ever we can give name to the bean using name attribute which is available into @Bean annotation.
 - Other attributes also available which help in adding extra capabilities to the corresponding beans.
- **@Bean(name,autowire,initmethod,destorymethod)**
 - @Bean annotation used to create the bean and placing that bean into IOC container. We can add extra capabilities also.
- **@ComponentScan("packageName")** :
 - It is newly added annotation in spring 3.2.x which help in scanning the and placing beans into IOC container.
 - Ex: if a package contains 10 class annotate with @Component and there are some class which are in @Configuration class, while loading configuration class we wanted to create other classes as bean and places IOC container then we can use @ComponentScan which take package name contains annotated classes.
 - While loading @Configuration class IOC container will load able to read other component class which are available into given package.

Spring 112

- **@Import(AppConfig.class, DBUtil.class):** A project may contains multiple class and every class may has its own identity like controller classes, utility classes, services classes, DAOclasses, pojo classes.
 - So we can not write all classes at single place to place into IOC container, actually we can write but it become clumsy to programmer.
 - We can write multiple configuration classes related to particular concern layer ex service related, DAO related , utility related etc.
 - To avoid the confusion spring has provided one annotation called @Import which help in importing multiple configuration at one place.
- **@profile("dev"):** one environment configuration will not support to other environment , ex: development environment platform will not be same as testing environment so it is very hard to migrate from one env. to other env..
 - So to manage spring has provided one annotation called as @profile, which will take Name of the env. Ex: @Profile("dev"), @Profile("test")
- **Java configuration annotation Support**
 - @Inject
 - @Resource(name="")
 - @Named
 - @postConstruct
 - @preDestroy
 - Lets see the example

```

@Configuration
@Profile("dev")
public class Dev_config {
    @Bean(name="connection")
    public Connection_Class createConnection()
    {
        Connection_Class class1=new
        Connection_Class();
        return class1;    }
    @Bean(name="dbProps")
    public Properties create_prop() throws IOException
    {
        Properties properties=new Properties();

        properties.load(this.getClass().getResourceAsStream(
        "oracle.properties"));
        return properties;    }
}

```

```

@Configuration
@Profile("test")
public class Test_config {
    @Bean(name="connection")
    public Connection_Class createConnection_test()
    {
        Connection_Class class2=new
        Connection_Class();
        return class2;
    }
    @Bean(name="dbProps")
    public Properties create_prop() throws IOException
    {
        Properties properties=new Properties();

        properties.load(this.getClass().getResourceAsStream(
        "mysql.properties"));
        return properties;
    }
}

```

```

public class Connection_Class {
    @Value("#{dbProps.env}")
    String environment;
    @Value("#{dbProps.driverClassName}")
    private String driverClassName;
    @Value("#{dbProps.url1}")
    private String url;
    @Value("#{dbProps.username}")
    private String username;
    @Value("#{dbProps.password}")
    private String password;

    //setter and getters
}

```

```

@Import({Dev_config.class,Test_config.class})
@Configuration
public class App_config {
    @Bean(name="environment")
    public DBEnvironment newEnvironment(){
        return new DBEnvironment();
    }
    @Bean(name="dbEnv")
    public Properties create_prop() throws
    IOException
    {
        Properties properties=new
        Properties();

        properties.load(this.getClass().getResource
        AsStream("env.properties"));
        return properties;
    }
}

```

```

mysql.properties
=====
driverClassName=mysql
url1=localhost.mysql
username=thulasi
password=tiger
env=test

```

```

oracle.properties
=====
driverClassName=oracle.jdbc
url1=localhost.thin
username=hima
password=uma

```

```

@Configuration
@Profile
@Import

```

Java config annotation support

```
@Named
public class Bicycle {
    // @Inject
    Chain chain;
    public void ride(){
        System.out.println("riding.....");
    }
    @Resource
    public void setChain(Chain chain) {
        this.chain = chain;
    }
    @PostConstruct
    public void init(){
        System.out.println("initialization.....");
    }
    @PreDestroy
    public void destroy(){
        System.out.println("releasing.....");
    }
}
```

```
@Named("chian")
public class Chain {

    private int id;
    private String type;
    //setters and getters
}
```

```
public class JATest {
    public static void main(String[] args) {
        ApplicationContext context = new AnnotationConfigApplicationContext("com.javaan.annon");
        ((ConfigurableApplicationContext)context).registerShutdownHook();
        Bicycle bicycle = context.getBean("bicycle",Bicycle.class);
        bicycle.ride();
        System.out.println(bicycle);
    }
}
```

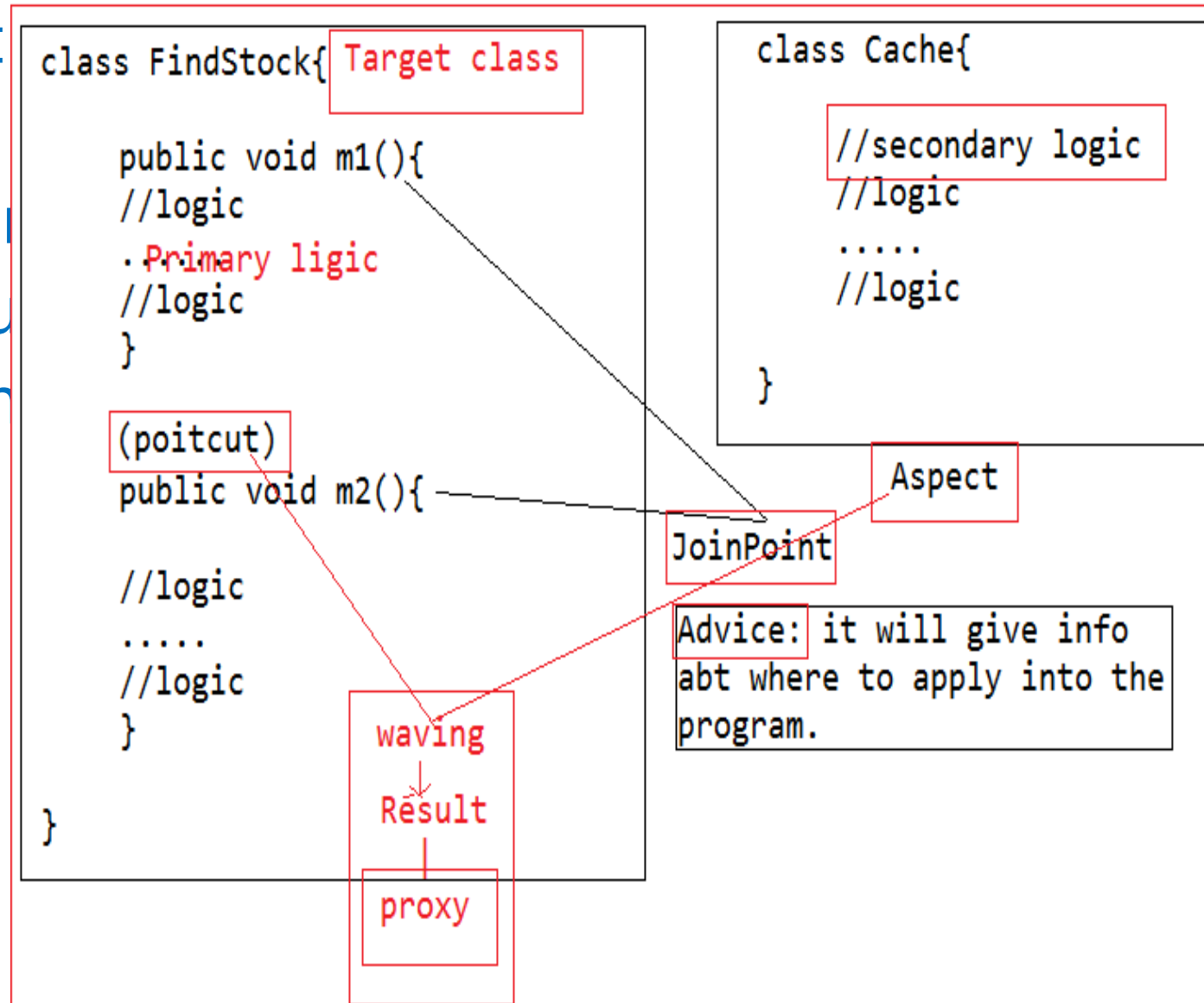
AOP(Aspect Oriented Programming):

- AOP is the programming methodology or paradigm, it has number of principles they talk about how to use the AOP into OOP to complement the OOP's.
- AOP is the process of separating the primary logic from the secondary logic.
- We can say it is the methodology which will separate the crosscutting logic from the primary logic or core logic.
- There are number of pitfalls when we use the secondary logic with primary logic.
 - Crosscutting logic is the logic which can be used in many places into the application, if we mix both primary and secondary logic we end up with duplicating the code across the application.
 - If we write crosscutting logic with primary logic and there is change in secondary logic it may cause the primary logic also.
 - If both are written in one single place programmer must worry about secondary rather than primary logic.
 - If both are written in one place, we cannot easily separate them.
 - Some time if we don't want to execute secondary logic then, we cannot escape, even though if we place condition to evaluate the condition and escape, but the presence of the code will be there into the application. We cannot eliminate permanently.

- **AOP (Aspect oriented Programming)**
 - AOP is the methodology which can not replace the OOP rather it will complements the OOP's.
 - **Primary business logic** : It is the core business logic which has to execute at any cost.
 - **Secondary logic**: It is the crosscutting logic which is optional, As per condition or requirement we can apply and execute.
 - Both should not be written in one single place.
- **Crosscutting logic concern:**
 - AOP talks about crosscutting concern logic, Actually its depends up on the set of principles which help in building the crosscutting logic.
 - To build the crosscutting logic we have to obey the AOP principles which are described below.

• Principles of AOP:

- 1)Aspect
- 2)Advise
- 3)Joinpoint
- 4)Pointcut
- 5)Weaving
- 6)Target
- 7)Proxy



- **Aspect**: it is piece of code which will be separate from the primary logic.
- **Aspect** is represents the secondary logic or crosscutting logic.
- **Advice** : This principle talks about where actually we can apply that aspect.
 - Before advice
 - After advice
 - Around advice
 - Exception (throw) advice
- **Joinpoint**: This principle will tells about how many places we can advice the aspect.
- **Pointcut**: set of joinpoint where actually we advice the aspect will describe by pointcut.
- **Weaving**: The combination of aspect and pointcut called as weaving.
- **Target** : On which class we are going to apply the aspect.
- **Proxy**: After Weaving we will get the proxy as the result.
- Third party AOP vendors
 - Jboss AOP
 - AspectJ AOP
 - Spring AOP module

- OOPs paradigm:
 - OOPs is the methodology and every one has to obey the rules of the OOPs then only they can make the programming language oops supported.
 - There are different concepts are given by OOPs standard org.
 - Polymorphism
 - Inheritance
 - Abstraction
 - Encapsulation
 - Message passing
 - Class
 - Object and so on.

- Spring AOP + AspectJ integration

- 1)JoinPoint:

- 1)Aspectj : constructor,method,static,destroy,....*

- 1)Spring AOP: method

- 2)PointCut:

- 2)Aspectj : static Pointcut

- 2)Spring AOP:

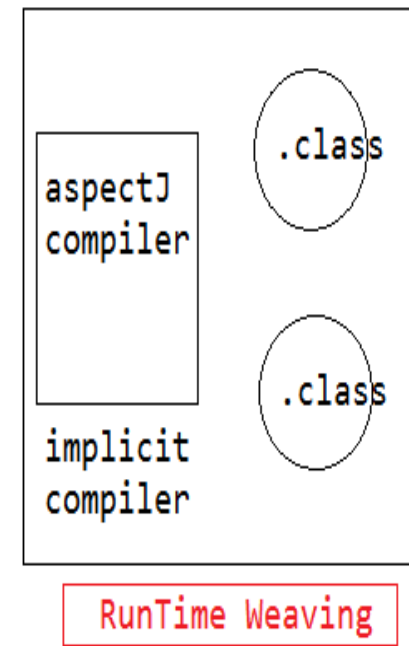
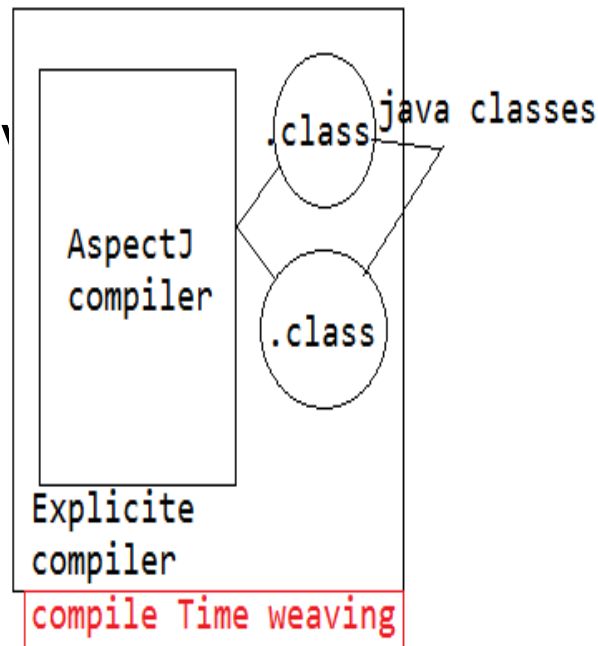
- a) **Static pointcut**: There are number of joinpoint are available into the target class but there are set of pointcut are available which we will expect to execute at compile time called as static pointcut.

- At the compile time if we want to advice the aspect on the specific pointcuts, this process called static pointcut.

- b) **Dynamic pointcut** : There are number of joinpoint are available into the target class, we want to advice the aspect at dynamically by passing some input to the pointcut.

- When we pass the dynamic input it will advice the aspect on the pointcut.

- Spring support three ways of AOP
 - Programmatic AOP development
 - Declarative Aspectj AOP development
 - Annotation Aspectj AOP development
- 3)Weaving:
 - 3)Aspectj : compile-Time Weaving
 - 4)Spring AOP :
 - Compile-Time Weaving
 - Runtime Weaving



– 1)Compile Time Weaving:

- In this approach we will use third party tool/software for compiling the aspect which we written in another place.
- along with that we have to compile our primary business logic and both aspect compiled classes and java compiled classes are given to the Jcompiler to apply the AOP principles.
- At compile time only the third party tool / software/compiler will identify the pointcut among the joinpoint and it will apply the crosscutting logic on target classes.
- In compile Time weaving we have to explicitly install the compiler to handling the crosscutting logic.

– 2)Runtime Weaving:

- In this approach there is no need to install the explicit software/compiler to handle the crosscutting logic, just give external related jar to the JVM and JVM internally bootstrap the compiler. And keep the programming running in to the JVM while we executing the target class automatically JVM will communicate with the other program which is running inside him and it will handle the crosscutting logic.
- As per the above diagram we can get more clear idea how the crosscutting happening in both way.

- Every application have two kind of logic primary business logic and secondary logic.
- Actually primary logic is the core logic which is expected to be an execute at any cost, but secondary logic is the option which will compliment the primary logic.
- Compliments mean adding extra behavior and improving the performance of the application.
- There are different ways we can apply our aspect on target class joinpoints.
 - Around Advice
 - Before advice
 - After advice
 - Throw advice
 - Lets see the Around Advice first.

- 1)Around Advice

- We wanted to advice the aspect before the method execution and after the method execution finished then use Around advice.
- Around advice play crucial role in project because of this we can easily track the flow of the application we can easily identify the problems in the application.
- For providing the logging facilities to the application Around advice will plays vital role.
- As the above discussion we come to know how the spring has the support for AOP, to handle every aspect of the AOP, spring has provided different classes and interface.
- To work with Around advice spring has provided one of the interface called **MethodInterceptor**.
- MethodInterceptor has one method called invoke, which will used for applying the crosscutting logic on target class. And method looks like:

Public Object invoke(MethodInvocation methodInvocation){}

- Method Interceptor means Around the method and we can apply the crosscutting logic.

- To apply the crosscutting logic we can all the information about the target class without target class information we can not apply the crosscutting logic.
- So invoke method has one parameter called MethodInvocation which able grab all the information of the target class.
- Means by help MethodInvocation can we get all information of the target class easily, to get all the information which methods we have to use by seeing the example we can get it.
- Once we got the all information about the target class now we can perform the before advice and can proceed the execution of the target class by calling `methodInvocation.proceed()` method.
- `methodInvocation.proceed()` method will call the target class corresponding method, once target method execution has been finished it will return the cursor to the invoke method then again invoke method can apply the crosscutting logic after the method execution.
- Invoke method not only apply the crosscutting logic, it can modify the input data, even it can modify the return output also.
- Bz target class method called by invoke method and target method will return output to the invoke method and before returning the original output to the callee invoke method can modify it.

- There are three control points available while working with the Around advice.
 - 1)Once we got all information we can modify the the input and proceed.
 - 2)Method invocation under control invoke method as per requirement we can call or not.
 - 3)Once we call the target class method, target class method will return output to the invoke method , means again control comes to the invoke method.
- Callee will call target class method before execute target method invoke method gets executed this sentence is wrong, Actually Instead of target class method first Aspect class method gets execute and the control given to the target class by invoking `methodinvocation.proceed()` method.

- Lets see the example
 - We can not write both primary business logic and secondary logic in one single class.
 - Secondary logic means crosscutting logic should be written in separate class only.
 - Bz some time its require or sometime it will not we can easily get read from the crosscutting logic.

```
class Calculator{
    joinpoint
    public float add(int a,int b){
        return a+b;
    }
}
```

Target class

```
class LoggingAspect implements
MethodInterceptor{
    public Object invoke(MethodInvocation
methodInvocation){
```

if req we can modify

```
String methodName =
methodInvocation.getMethod().getName();
Object[] args = methodInvocation.getArguments();
args[0] = (Integer)args[0]+1;
args[1] = (Integer)args[1]+1;
```

```
Object output=methodInvocaton.proceed();
```

```
output = (float)output+1;
return output;
}
```

Control Points