# Spring-Core Material-2016

# 1. What is spring?

The spring is a Framework, by using spring we develop multiple types of application. Spring supports managing the dependencies between components.

## 2. What is difference between spring and struts?

➔Spring is better than all framework that there in the market. But there is another successful framework in market is struts.

➔By using struts we can develop only Web application.

➔By using spring we can develop multiple applications that are Web Application, Standalone Application, Distributed Application, Enterprises Application etc.

➔But Struts provides support only for Web aspect Layer but not Business Layer (we use Java Classes), Persistence Layer (we use JDBC).it doesn't permits to develop end to end web application.

➔Spring provides a very clean division between controllers, Java Bean models, and views (Server Handler, Business Logic, and Persistence Logic), it permits to develop end to end web application.

➔. Which make spring superior than struts?

## 3. What is difference API and Framework?

➔**API** (Application Programming Interface) **contains Abstract classes and Interfaces**, concrete class is not there in as part of API's so it is partial and it is not complete.

➔**For Example** JDBC is a partial it is not complete, there will be implementation that is being provided by the Database vendors which are called JDBC Jars. **Similarly** JavaEE is an API which is partial it contains Interfaces and Abstract classes so somebody has to provided implementation for it. That is being provided by the Server vendors.

Ex: - Apache: Tomcat
IBM : Websphere
Oracle : WebLogic

➔**API's are huge in nature that** means the numbers of components are more there as part of the API. And it not easy to learn it takes more time to learn.

➔**The worst part of API is the class that are there with an API are not easy to understand because these are interlinking with each other.** That means to learn one class we need to know some other class. Due to the classes are interlinking with other classes. For example to execute simply SQL query we have to write following 4 lines of the code.

```
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection(URL,USN,PSW);
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("sql query");
```

➔Because of class are interlink with each other one has to understand all the classes that are provided with in that API, and **we can't start working with that without knowing complete knowledge of that API.**

➔**API will not provide Boilerplate code** *(ex: To fulfil some operation we may have write some code, the piece of logic that I am writing to fulfil that operation not only I any one has to write the same code for such type of requirement, such type of logic that we are trying to write is seems to be common on across all the project across world. This is called boilerplate logic)* so **programmer has to write Boilerplate Logic.**

➔If Boilerplate logic is not there in API so programmer has to write more lines of the code so the **time** required for developing the application is more, the **efforts** that we will be invest in developing the application is more, the **manpower required** to develop the application is more the **cost** involved in the application will be more.
➔As we write more number of lines of code the chances of **increasing bugs** as part of our lines of code will be more.
➔ As we write more number of lines of code the maintenance **time required to maintaining the application** is more and the **cost of maintaining** the application will be high.
➔ As we write more number of lines of code the time required for testing the application will be more, the amount of time we need to spend in insure the quality code is delivered out of our application will be more.
➔Considering all the things that if the API not provides the Boilerplate Logic all the above drawbacks that we are going to face if we are working with API so API's are lots of limitations are there.

***Framework***

➔Framework has provided a bunch of classes, spring framework developers has provided some classes for us that are concrete classes. And the classes that are provide by spring that anyone can directly instantiate the class object and can call the function, In case of API, API is partial and it is not complete.
➔The number classes that are provided by framework are less, and they design the classes in such a way that the minimal set of classes itself we can use to get the outcomes.

*For example: if we want to write JDBC program we need minimum 4 classes while the same thing if we want to do in spring JDBC instead of using*

*Java JDBC then only one class we have to use and one method we need to call. That means the no of classes will be less.*

➔The classes that provided by the framework are not interlink with each other.

➔ One can directly work with the specific class that means it is easy to understand the framework, when compared with API those are less complicated

➔And the amount of time that we need to learn the framework will be less and very easy to learn because interlink between classes are less.

➔We can learn a part of a framework that is required for fulfil our requirement. That means framework will support rapid application development.

➔Framework is providing concrete class and inside the concrete class boilerplate logic has been written by the developer itself. So programmer no needs to write boilerplate logic.

➔In this case we need to write less amount of lines of code, if we are writing less no of lines of code the time required for developing the application will be less.

➔Cost involved in developing the application will be less, manpower required for the developing the application will become less.

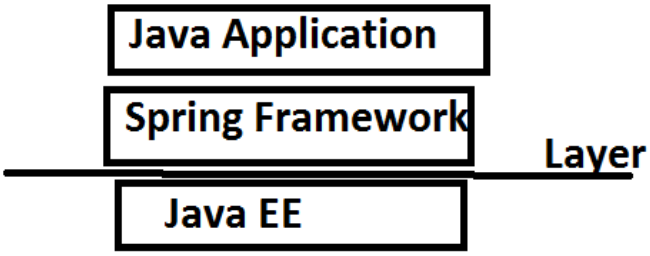
➔Due to writing less no of lines of code the chances of bugs will be very very less.

➔The time required for testing the application will be less. When compared with API.

➔The logic that has been provided by framework people has pretested. That is completely free of bugs ,that means half of the application that we have developed is already pretested and many of the bugs has been avoided and quality code is delivered out of our application will be more.

## 4. Why Spring is not replacement of JavaEE?

Spring is layer that is built on top of the java EE it is not replacement of JavaEE, internally spring uses JavaEE. We can't replace spring with JavaEE.




## 5. Why Spring is being called as Lightweight Framework?

➔If u look at the architecture of the spring framework one can easily understand why it is being called as light weight,

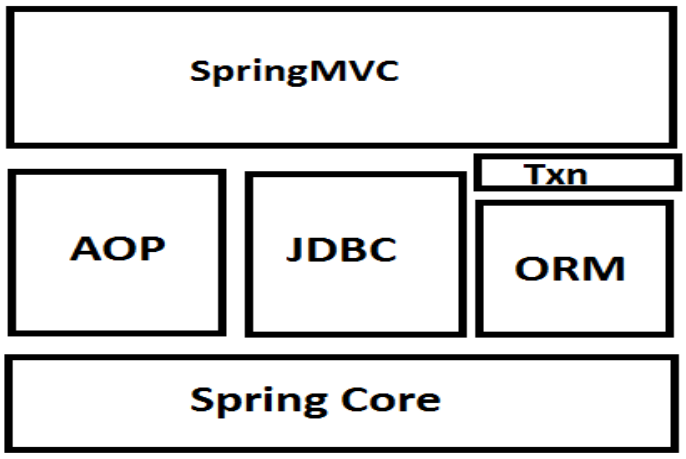
➔If you see the numbers of modules that are there with the spring framework no two modules are having intersection or a crosslink between each other.

➔That itself tell you every module within the spring framework is completely independent of the other module that is there with the spring framework apart from the core module acts as foundation for the entire spring.

➔That itself tells you that any developer can quickly can jump and start in learning the part of the framework and can use it and can quickly build the application.

➔One don't need to worry about what all the rest of the module that there with the framework to learn to build the application.

➔We just need to find the relevant module that is required for release the application that itself tell you it could be easy and quick to learn the framework and build the application using the spring, that's why spring is being called as the lightweight application development framework.




**Spring Architecture**

# 6. Why Spring Framework so much popular in the market?

There are so many reasons but the most popular and most strong and unique features are

1. Versatile application development.
2. Non-invasiveness app development.

**Versatile:**

The word versatile makes you to understand.

➔Versatile means Flexible.

For example:

➔If an organization developing an application using struts there are so many drawbacks available in the struts. It's not a complete framework to develop an application. It doesn't have business layer and persistency layer to develop a complete end to end application.

➔Spring Framework provides 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 spring framework**.

➔We can integrate any technology with the spring framework. It's too flexible.

**Non-Invasiveness**

➔Non-Invasiveness means it does not affect your code even though some one doesn't want to use the spring framework. **It means with zero lines of code changes you can remove the spring from your project**.

➔It remain same at any point of your project you can remove the spring package, after removed, it will compile but spring provided features are not there, now you have to write additional logic if you work with other framework.

➔**It don't force you to rebuild your application, it seems like an interesting features that is being provided by spring people that is non-invasive**.

➔That makes spring superior than other frameworks.

# 7. Introduction of Spring Core Module

➔Spring has been architected in such a way the spring has been broken down into several parts these parts are called modules, there are several modules are there with in the spring framework.

➔ Spring people design the modules in such a way no two modules are having intersection or a crosslink between each other.

➔ In that One module called **spring core which is base module or infrastructure module that's why unless until we learn the spring core we can't learn rest of the module** that are there with in the spring framework.

➔ *It supports managing the dependencies and complexity that are there with the components.*

To develop an application in a single class is not recommended because our logic will become difficult and complexity will be more so to **develop an application we should use multiple types of classes based on functionality.**

Those are:

**→POJO**

**→Java Bean**

**→Bean / Component**

## POJO (Plain Old Java Object)

The class which is not refer to any 3rd party vender provided classes or interfaces, and it can be compliable and it can be executable underlying JDK then that class is called as POJO Classes.

## Java Bean

A class contains attributes with accessor methods (getter & setter methods) and the class should not contain any other method that class is called as java been classes.

## Bean / Component

A class contains attributes and methods with business logic to perform Some processing called as component class or been class, it can be refer to any 3rd party vender provided classes or interfaces.

## 8. What is Design Pattern?

➔When people are developing the application people are using Object Oriented programming principle there are certain drawbacks people are going to face.

➔And the applicable solutions for these problems are called Gang of Four (GOF) Design pattern.

➔Out of which one of the design pattern is strategy design pattern.

## 9. Strategy Design pattern (GOF)

➔ Spring core supports managing the complexity of components and dependencies that are there with the components.

➔To get more benefits of spring framework, recommended to follow a design pattern called Strategy Design pattern.

➔ Strategy design pattern is provided by **Gang of Four (GOF)**

➔In strategy Design pattern there are three principles:

- **Favor composition over inheritance.**
- **Always design to interfaces never design to concrete classes**
- **Code should be open for extension & close for modification** (Open Close principle)

❖ **Favor composition over inheritance.**

There are two ways to access the behaviour of one class to another class.

➔Inheritance

➔Composition

➔If all behaviour of super class is common across all the sub class then **we should go for inheritance.**

➔ If some behaviour of a class is required for another class then we **should go for composition**.

**Inheritance**

➔Inheritance is always called IS-A Relationship

➔ If a class want the functionality of another class instead of rewriting the same functionality within another class we can reuse the functionality of other class. By using inheritance our class can get the property/ quality of another class.

**Drawbacks with Inheritance**

➔In an application if a class wants to use few methods of other class, just sake of using few methods of other class we should go for composition rather than inheritance is recommended.

**Example.1 (just sake of using few methods of other class we should go for composition rather than inheritance is recommended, because there is no use of other methods)**

```
public class A{
        public int m1(){
                ;;;;;;;;;;;;;;;;
                return 35;
        }
        public float m2(){
                ;;;;;;;;;;;;;;;;
                return 12.35f;
        }
}
```

```
class B extends A{
        public void m3(){
                int x=super.m1();
                ;;;;;;;;
        }
}
```

Note: class B required only m1() method

```
class C extends A{
        public void m4(){
                float f=super.m2();
        }
}
```

Note : class C required only m2() method of class A

But inheritance Force us to use another methods also

➔In the project it may not possible a class can use only one class behaviour, it may possible to use multiple classes, but most of the Object oriented programming language **not supporting multiple inheritance**.

**Example.2 (not supporting multiple inheritances)**

➔If we are using inheritance components will become **fragile** (easily broken), **if suddenly super class method return type will change** then all subsequent classes will give compilation error , and it enforce to modify the subsequent classes also.

```
class A{
        public void m1() {

        }
        public void m2(){

        }
}
```

```
class B{
        public void m3(){

        }
        public void m4(){

        }
}
```

```
class C extends A,B{

        public void m5(){
                m1();
                m2();
                m3();
                m4();
        }
}
```

**Note: C class can't extends multiple classes**

## Example.3 (Fragile (easily broken)

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

```
class B extends A{
    public int m1(){
        int i=super.m1();
        //add some logic....
        return i;
    }
}
```

CE: The return type is incompatible with A.m1()

```
class C extends B{
    public  void m4() {
        int x=super.m1();
        //logic....
    }
}
```

If return type of m1() method will change 'int' to 'float' then all subsequent class will be effected.

**if return type change in class A it will be like which is not possible in java (same Method signature with different return type not possible)**

```
class B extends A{
    public int m1(){;;;}
    public float m1(){;;;}
}
```

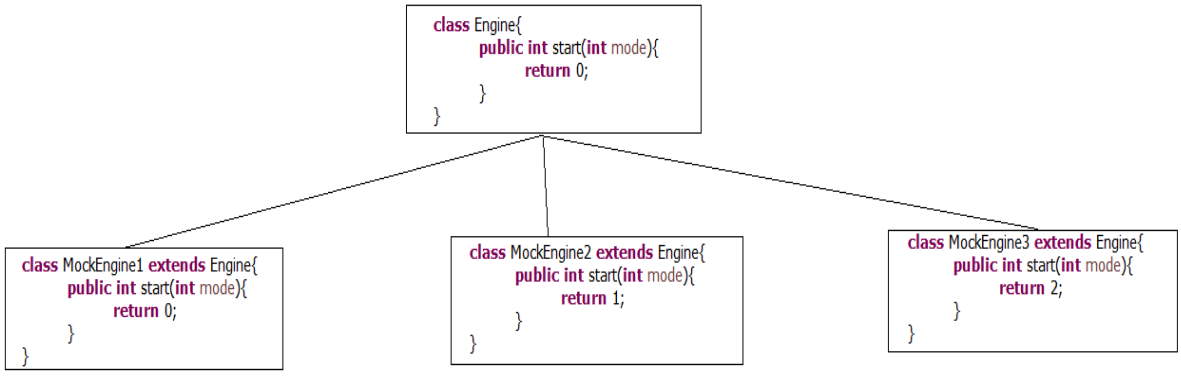
➔In **testability** also we will face the problem if we use inheritance. In an organization it may not possible to develop an application by one developer. There are number of developers are available to construct the application.

## Example.4 (Facing Problem during Testability of Application)

Suppose a developer developing an application on Car Class, to complete Car Class Engine class is required, here Car class depends on Engine Class.

Suppose Car class has completed its implementation now it has to wait until Engine class completion. But in company we can't wait for any other class, in this case **we can't perform the testability of the Car class**.

**Here solution for this problem is** we have to create a mock class (Dummy class) which is act as an Engine class. By help of Mock Engine class we can perform the testability of the Car class.




We have to change the code again and again to get the all possible output .which will not possible while testing the application. bcz we have to give only .class files to testing team

```
                MockEngine3
                MockEngine2
class Car extends MockEngine1{
        public void drive(){
                int result=super.start(1);
                if (result==0) {
                        System.out.println("Engine Not Started");
                }
                else if (result==1) {
                        System.out.println("Engine Started in Manual Mode");
                }
                else if(result==2) {
                        System.out.println("Engine Started in Automatic Mode");
                }
        }
}
```

```
public class Test {
        public static void main(String[] args) {
                Car car=new Car();
                car.drive();
        }
}
```

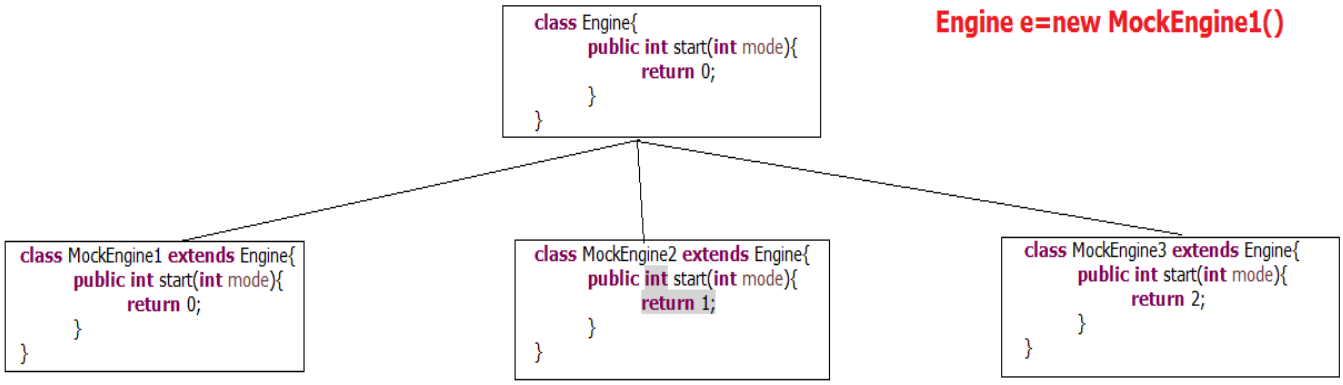
O/p= 1st time Engine Not Started
2nd time Engine Started in Manual Mode
3rd time Engine Started in Automatic Mode

But in case of composition testability problem is not there...

**Composition**

**Here we are using Polymorphism its like**

**Engine e=new MockEngine1()**



```
class Engine{
        public int start(int mode){
                return 0;
        }
}
```

```
class MockEngine1 extends Engine{
        public int start(int mode){
                return 0;
        }
}
```

```
class MockEngine2 extends Engine{
        public int start(int mode){
                return 1;
        }
}
```

```
class MockEngine3 extends Engine{
        public int start(int mode){
                return 2;
        }
}
```

```
class Car {
        private Engine e;
        public Car(Engine e) {
                this.e=e;
        }
        public void drive(){
                int result=e.start(1);
                if (result==0) {
                        System.out.println("Engine Not Started");
                }
                else if (result==1) {
                        System.out.println("Engine Started in Manual Mode");
                }
                else if(result==2) {
                        System.out.println("Engine Started in Automatic Mode");
                }
        }
}
```

```
public class Test {
        public static void main(String[] args) {
                Car car=new Car(new MockEngine1());
                 car.drive();
                Car car1=new Car(new MockEngine2());
                  car1.drive();
                Car car2=new Car(new MockEngine3());
                 car2.drive();
        }
}
```

**O/p** Engine Not Started
Engine Started in Manual Mode
Engine Started in Automatic Mode

**Composition**

➔ **Composition is also called as Has-A Relationship.**

➔If a class want the functionality that is there in another class instead of rewriting the same functionality within another class we can use as it the functionality of other class is there in our class itself.

➔**Declare other class as an attribute in our class and use the functionality of another class.**

❖ **Always design to interfaces never design to concrete classes**

```
class A {
        public int m1(){
        //  Complex Logic
                return 24;
        }
}
```

```
class C{
        public int m1(){
                //  Boost Logic
                        return 24;
        }
}
```

```
class B{
        private A a;
        public int m2(){
                a=new A();
                int i=a.m1();
                //add some Logic
                return i;
        }
}
```

**To get new Boost Logic of class C instead of Complex logic of class A , we can't easily integrate C class with B class without change in the existing code in class B,so here degree of dependency is more between class A and class B and it is called tightly coupling**

➔Here class B is dependent on class A, because class B is holding the attribute of class A, so the level of dependency between class A and class B is more, here two classes are tightly coupled with each other,

➔**If we want to replace the class A with class C total existing code inside the class B we have to change** because coupling encourage to redevelopment. Due to this redevelopment cost, maintenance cost, testability cost will be high. **So never Design Concrete Class always Design Interface**

**Some Other Approach:**

```
class A{
        public int m1() {
                //logic 1000 Lines of code
                return 100;
        }
}
```

```
class B extends A{
        public int m1() {
                //logic 600 Lines of code
                return 100;
        }
}
```

```
class C extends A{
        private A a;
        public void m5(){
                a=new B();
                int i=a.m1();
                //logic
        }
}
```

Dynamic (run time) polymorphism

It will work but the code inside class A m1() method will become Dead Code

```
abstract class A{
        public abstract int m1();
        public void m2(){

        }
        public void m3(){

        }
}
```

```
class B extends A{
        public int m1() {
                //logic 600 Lines of code
                return 100;
        }
}
```

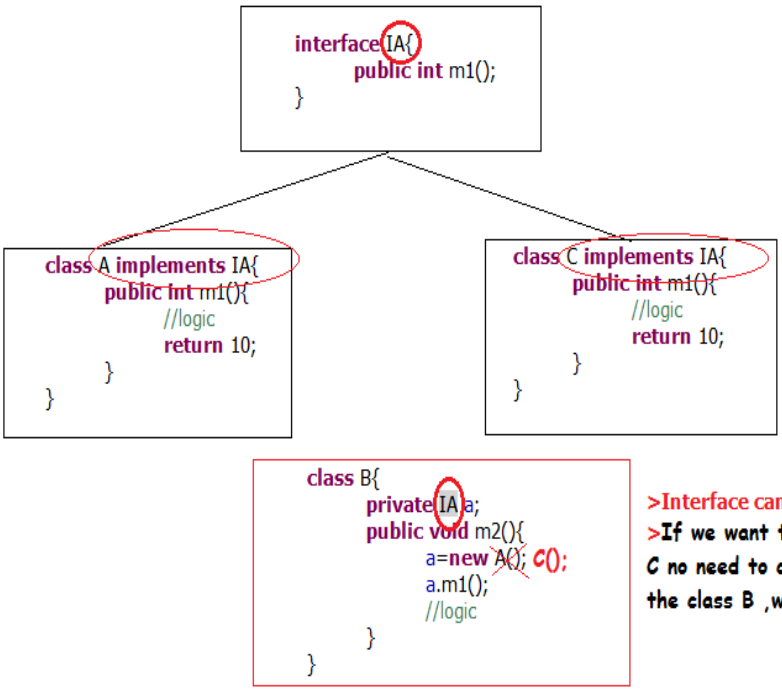
```
class C extends B{
        private A a;
        public void m5(){
                a=new B();
                int i=a.m1();
                //logic
        }
}
```

1. Abstract classes can be partially or fully implemented.

2.Just sake of override few methods of abstract class we should go for interface rather than abstract class is recommended, because there is no use of other methods.

3. better depends on Interfaces are useful because Java classes will not support multiple inheritance but interfaces do.

If we use Interface See the Example:




>Interface can hold its implemented class Object
>If we want to replace the class A with class C no need to change total existing code inside the class B ,we just change the runtime Object .

➔To develop an application if we use composition as part of our application then it must and **should use interface as service provider.**
➔**Without service provider if we develop an application** then **it lead to many problems in the future.**
➔We can't manage our application because **it will become tightly coupled.**
➔It may lead to huge loss to the client.
➔When we **use interface we can easily change our code and migrate with other classes, so never Design Concrete Class always Design Interface**

```
interface BankService{
        void deposite();
        int withDraw();
        void transfer();
}
```

```
final class SBIPaymentImpl implements BankService{

        private SBIPlugIn sbi=new SBIPlugIn();
        @Override
        public void deposite() {
                sbi.deposite();
        }
        @Override
        public int withDraw() {
                int i=sbi.withDraw();
                return i;
        }
        @Override
        public void transfer() {
                sbi.transfer();
        }
}
```

SBI Classes Implementation

```
final class HDFCPaymentImpl implements BankService{

        private HDFCPlugIn hdfc=new HDFCPlugIn();
        @Override
        public void deposite() {
                hdfc.deposite();
        }
        @Override
        public int withDraw() {
                int i=hdfc.withDraw();
                return i;
        }
        @Override
        public void transfer() {
                hdfc.transfer();
        }
}
```

HDFC Classes Implementation

1. My Class is not talk to Vendor provided class directly .
2.So my application will become Loosely coupled

```
public class OdishaSSB{
        private BankService service;
        public void payment(int i) {
                service=new SBIPaymentImpl();
                service.deposite();
                System.out.println("Rs."+i+"\tSuccessfully Deposited");
        }
        public static void main(String[] args) {
                OdishaSSB ossb=new OdishaSSB();
                ossb.payment(200);
        }
}
```

```
class SBIPlugIn{

        public void deposite(){
                //Logic
        }
        public int withDraw(){
                //Logic
                return 0;
        }
        public void transfer(){
                //Logic
        }
}
```

SBI Provide Classes

```
class HDFCPlugIn{

        public void deposite(){
                //Logic
        }
        public int withDraw(){
                //Logic
                return 0;
        }
        public void transfer(){
                //Logic
        }
}
```

HDFC Provide Classes

1. If i do my Application Like this way i can switch SBI-HDFC or HDFC-SBI easily .
2.Just only Runtime Object I have to change.
3.Here My code Open extension but not for Modification because my classes are final classes

```
interface FlyableI {
        void fly();
}
final class FlyableImpl implements FlyableI{
        public void fly(){
                System.out.println("Fly--------->");
        }
}
```

```
interface DriveableI {
        void drive();
}
final class DriveableImpl implements DriveableI{
        public void drive(){
                System.out.println("drive--------->");
        }
}
```

```
class Vehicle {
  protected FlyableI f;
  protected DriveableI d;
        void honk(){

        System.out.println("Honk--------->");
        }
}
```

```
class Car extends Vehicle{
        public void m1(){
                d=new DriveableImpl();
                super.honk();
                d.drive();
        }
}
```

```
class Bus extends Vehicle{
        public void m1(){
                d=new DriveableImpl();
                super.honk();
                d.drive();
        }
}
```

```
class Jet extends Vehicle{
        public void m2(){
                f=new FlyableImpl();
                f.fly();
                super.honk();
        }
}
```

```
class Robot extends Vehicle{
        public void m2(){
                f=new FlyableImpl();
                d=new DriveableImpl();
                f.fly();
                d.drive();
                super.honk();
        }
}
```

```
interface MessageProduce{
        String convertMessage(String message);
}
```

```
final class HtmlMessageProduceImpl implements MessageProduce {
        public String convertMessage(String message) {
                return message+"\tHTML Print";
        }
}
final class PdfMessageProduceImpl implements MessageProduce {
        public String convertMessage(String message) {
                return message+"\tPDF Print";
        }
}
```

```
class messageWritable{
        private MessageProduce msgProduce;
        public void print(){
                msgProduce=new HtmlMessageProduceImpl();
                msgProduce.convertMessage("Hello");
        }
}
```

Here we can easily change HTML to PDF and PDF to HTML with changing only one line of code. It means like Zero changes we can switch one class to another class.

## Eclipse Shortcuts.

| SL. No | Description | Key |
|---|---|---|
| 1 | F2 | Rename |
| 2 | Ctrl + space | AutoComplete |
| 3 | Ctrl + D | Delete Line |
| 4 | Ctrl+ shift + F | Auto Formatting |
| 5 | Ctrl + M | Maximize/Minimize |
| 6 | Ctrl +. | Next Error Point |
| 7 | Ctrl + shift+ G | See Workspace |
| 8 | Alt + ➔ ,Alt + ← | Previous/next visited point |
| 9 | Alt + Shift + S → S | toString() Method |
| 10 | Alt + Shift + S →R | Setter() & getter() methods |
| 11 | Alt + Shift + S →O | Constructor (--) |
| 12 | Alt + Shift + S →C | Constructor () |
| 13 | Alt + Shift + S →H | equals & hashCode() methods |
| 14 | Alt + Shift + S →V | Override Message |
| 15 | main → ctrl+space | Public static void main(String[]args){} |
| 16 | Syso → ctrl+space | System.out.println(); |
| 17 | Ctrl + Shift + T | See Predefined Class(add source 1st) |
| 18 | Ctrl + F11 or<br>Alt + Shift + X → J | Run main() method |
| 19 | F11 ,F6,F5,F7,F8 | Debug keys |
| 20 | Ctrl + Shift + / | Comment multiple line |
| 21 | Ctrl + Shift + \ | Uncomment multiple line |
| 22 | Ctrl + Shift +O | Import Classes |

**Note:**

➔As per the strategy design pattern we developed above examples. But using Strategy Design Pattern we can't develop our application completely loosely coupled.

## ➔Problems with strategy design pattern

**There are two problems are generated.**

1. While instantiating an class we are using concrete class which can affect our application, if we change one concrete class to other concrete class we have to change all code which is refer to that concrete class.
2. while instantiating an class we have to know all the information about the corresponding class

## Factory Design Pattern

➔Factory pattern is one of most used design pattern in Java. This type of design pattern comes under creational pattern, **to avoid complexity of creation object we should go for Factory Design pattern** as this pattern provides one of the best ways to create an object.

➔In factory design pattern we can create **object without exposing the creation logic to the client** and refer to newly created object using a common interface.

```
public interface IMessageProducer {
        String convertMessage(String message);
}
```

```
public class HtmlMessageProducerImpl implements IMessageProducer{
    @Override
    public String convertMessage(String message) {
        return "HTML Prints..." +message;
    }
}
```

```
public class PdfMessageProducerImpl implements IMessageProducer {
    @Override
    public String convertMessage(String message) {
        return "PDF Print..."+message;
    }
}
```

```
public class MessageProducerFactory {
    public static IMessageProducer createMessage(String type){

        IMessageProducer messageproducer=null;
        if (type.equals("html")) {
            messageproducer=new HtmlMessageProducerImpl();
        }else if(type.equals("pdf")){
            messageproducer=new PdfMessageProducerImpl();
        }
        return messageproducer;
    }
}
```

```
public class MessageWriter {
    private IMessageProducer messageProducer;
    public void writeMessage(){
        String msg=null;
        messageProducer=MessageProducerFactory.createMessage("pdf");
        msg=messageProducer.convertMessage("Startegy Design Pattern is Working !!!...");
        System.out.println(msg);
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        MessageWriter messageWriter =new MessageWriter();
        messageWriter.writeMessage();
    }
}
```

➔Factory class contains static method only because there is no need to create an Object for Factory classes.

➔ And also we never use any attributes inside factory class so Object is not required.

---

## Problems with Factory Design Pattern

```
IMessageproducer  messageProducer =
                    MessageProducerFactory.createMessage("html");
```

➔As we used factory class for making our classes completely loosely coupled **here physically we solved it but still logically we have to provide the type of requirement to create an object of the particular class**.

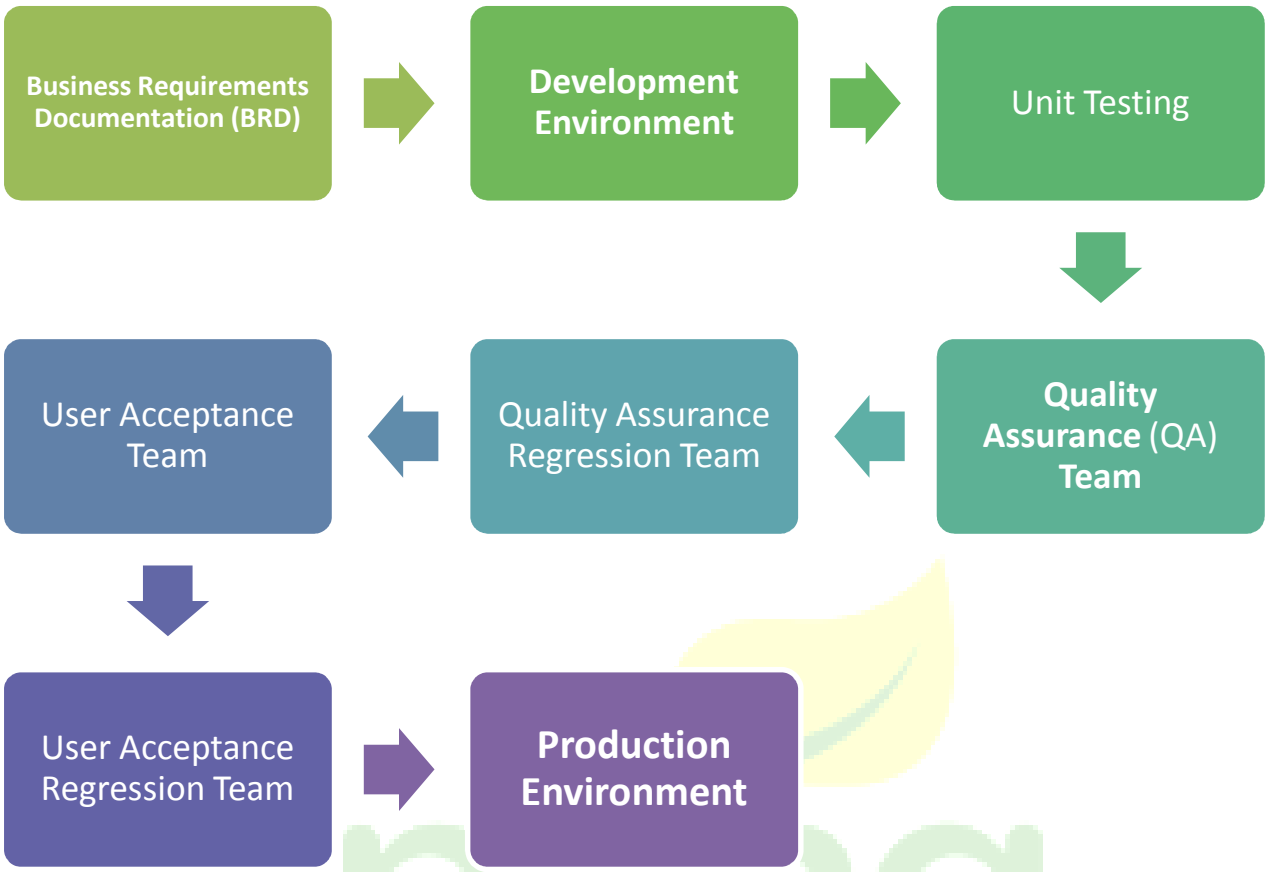
➔If we want PDFMessageProducer then **we have to manually change the logic**.

If we are using Factory class approach what kind of problem will occur let's see...

---

# RealTime Project Flow

## Delivery Handler Quality Assurance Process (DHQA)




### BRD(Business Requirement Documentation)

- It is the first phase where client will give the all requirement.
- Company 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.
- Everyone has to follow the Business Requirement Documentation only to meet the determination of the application.
- Single mistake may lead big problems in the project, understanding the Business Requirement Documentation is very important.
- After the design only project given to the development environment.

### Development Environment

- Development Environment is the place where company will provide the system to the all the developers to develop an application.
- Developer job is to follow the design and develop an application. After completion of application, **Developer has to test it to make it as executable, this procedure called as Unit Testing.**

- Developer has to write some test cases also for understanding to the other people.

## Quality Assurance Team (QAT)

- After unit testing application will give to the QA through SCM(Source Code Managing Repository) ,QA team going to check whether any bug are available in the project or not , if bug are found by QA team then they report the Development Environment to fixed the bugs using bug reporting tools and asking progress report.

| Bug ID | Incident Name | Type | Status | Priority | Severity | Creation Date | Detected By |
|---|---|---|---|---|---|---|---|
| IN7 | Cannot add a new Id to the System | Bug | **Open** | **1-Critical** | 3-Medium | 13-10-2016 | Dhananjaya |
| IN2 | Search button not working | Bug | **Fixed** | **3-Medium** | 3-Medium | 13-10-2016 | Dhananjaya |
| | | | **ReOpen** | | | | |
| | | | **Not a Bug** | | | | |
| | | | **Not Re- Produced** | | | | |
| | | | **InProcess** | | | | |
| | | | **Close** | | | | |

- Until and unless they get zero bugs they will repeat the above procedure to remove the bugs.

## Quality Assurance Regression

- After the QA team there is other phase will again check whether application will contains any bugs or not.
- If bugs are available again the same procedure will done by QA regression phase.

## User Acceptance Team (UAT)

- It will perform business operation to check whether the application working as per client requirement or not. By help of Business Requirement Documentation they can analysis the application.

## User Acceptance Regression Team

- After User Acceptance Team again there is a phase called User Acceptance Regression Team which cross checks the entire requirement and so on.

## Production Environment

- The production environment is also known as *live*, it is a term used mostly by developers to describe the setting where software are actually put into operation for their intended uses by end users.

➔Here we are asking to client 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.**

➔ Here we are approaching to factory class to create an object of particular class and **return to the MessageWriter class called as Dependency pulling.**

➔Here we are making our class loosely coupled but whenever we switch to one class to another class we have to make few changes in our class. So that again cost involved in our application will be high because to certify our class, that our class is free of bugs again DHQA Process will required.

➔We can solve this problem, here don't create any object, even don't ask anyone to create an object, who wants the functionality of our class let them to create an object and set to our class.

Ex:-

```
public interface IMessageProducer {
        String convertMessage(String message);
}
```

```
public class HtmlMessageProducerImpl implements IMessageProducer{
        @Override
        public String convertMessage(String message) {
                return "HTML Prints..." +message;
        }
}
```

```
public class PdfMessageProducerImpl implements IMessageProducer {
        @Override
        public String convertMessage(String message) {
                return "PDF Print..."+message;
        }
}
```

```
public class MessageWriter {
        private IMessageProducer messageProducer;

        public void setMessageProducer(IMessageProducer messageProducer) {
                this.messageProducer = messageProducer;
        }
        public void writeMessage(){
                String msg=null;
                msg=messageProducer.convertMessage("Startegy Design Pattern is Working !!!...");
                System.out.println(msg);
        }
}
```

```
public class Test {
        public static void main(String[] args) {
                MessageWriter messageWriter =new MessageWriter();
                IMessageProducer messageProducer=new HtmlMessageProducerImpl();
                messageWriter.setMessageProducer(messageProducer);
                messageWriter.writeMessage();
        }
}
```

➔Here advantages are no two classes will talk to each other that carry the business logic will not being modified.

➔No two classes are managing the dependency between each other. Now all the components that are there in our application are loosely coupled.

➔Test class is going to take an object from concern IMessageProducer and he is passing that object to the IMessageWriter, but the problem is if Test class wants to switch from html to pdf, then Test has to change its logic to get particular object.

➔**Test has to pass which MassageProducer he wants. 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 Test class. Still there is some amount of coupling is there.

➔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.

Ex-1

```
interface ImessageProducer{
    String convertMessage(String message);
}
```

```
class HtmlMessageProducerImpl implements ImessageProducer{
    public String convertMessage(String message) {
        return "<html><body>"+message+"</body></html>";
    }
}
```

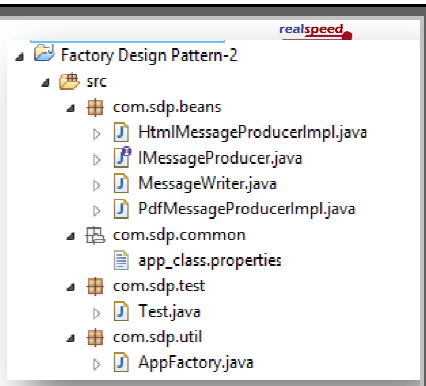
```
class PdfMessageProducerImpl implements ImessageProducer{
    public String convertMessage(String message) {
        return "<pdf>"+message+"</pdf";
    }
}
```

```
class MessageWriter{
    ImessageProducer messageProducer=null;

    public void setMessageProducer(ImessageProducer messageProducer) {
        this.messageProducer = messageProducer;
    }
    public void createMessage(String message){
        String cMessage=messageProducer.convertMessage(message);
        System.out.println(cMessage);
    }
}
```

```
abc.txt

HtmlMessageProducerImpl
```

```
public class Test {
    public static void main(String[] args) throws Exception{
        MessageWriter messageWriter=new MessageWriter();

        FileInputStream fis=new FileInputStream("abc.txt");
        byte[] bs = new byte[40];
        fis.read(bs);
        String className=new String(bs).trim();

        ImessageProducer messageProducer=(ImessageProducer) Class.forName(className).newInstance();
        messageWriter.setMessageProducer(messageProducer);
        messageWriter.createMessage("Welcome Startegy Design Pattern");
    }
}
```

Ex-2




```
public interface IMessageProducer {
        String convertMessage(String message);
}
```

```
public class HtmlMessageProducerImpl implements IMessageProducer{
        @Override
        public String convertMessage(String message) {
                return "HTML Prints..." +message;
        }
}
```

```
public class PdfMessageProducerImpl implements IMessageProducer {
        @Override
        public String convertMessage(String message) {
                return "PDF Print..."+message;
        }
}
```

```
public class AppFactory {
        public static Object createObject(String logicalName) throws Exception{
                Properties  props=null;
                Object obj=null;
                String className=null;

                props=new Properties();
                props.load(new FileInputStream("........app_class.properties"));//Get Location   Alt + Enter
                className=props.getProperty(logicalName);
                obj=Class.forName(className).newInstance();
                return obj;
        }
}
```

app_class.properties

```
messageWriter.class=com.sdp.beans.MessageWriter
messageProducer.class=com.sdp.beans.PdfMessageProducerImpl
```

```
public class MessageWriter {
        private IMessageProducer messageProducer;
        public void setMessageProducer(IMessageProducer messageProducer) {
                this.messageProducer = messageProducer;
        }
        public void writeMessage(String message){
                String msg=null;
                msg=messageProducer.convertMessage(message);
                System.out.println(msg);
        }
}
```

```
public class Test {
        public static void main(String[] args) throws Exception {
                MessageWriter messageWriter=null;
                IMessageProducer messageProducer=null;

                messageWriter=(MessageWriter)AppFactory.createObject("messageWriter.class");
                messageProducer=(IMessageProducer)AppFactory.createObject("messageProducer.class");
                messageWriter.setMessageProducer(messageProducer);
                messageWriter.writeMessage("Welcome to startegy Design Pattern");

        }
}
```

**Approach: 2**

➔In above example wherever object required for another class we are calling static factory method which loads the properties file multiple numbers of times, which is not recommended.

➔If we create the object of Properties class at the time of class loading then performance will become improve because it loads the properties file only once, even though we call static factory method multiple number of times it will return only one Properties object.

```
public class AppFactory2 {
        static      Properties  props;
        static{
                try {
                        props=new Properties();
                        props.load(new FileInputStream("...\src\\com\\sdp\\common\\app_class.properties"));
                } catch (IOException e) {e.printStackTrace();}//Get Location   Alt + Enter
        }
        public static Object createObject(String logicalName) throws Exception{
                Object obj=null;
                String className=null;
                className=props.getProperty(logicalName);
                obj=Class.forName(className).newInstance();
                return obj;
        }
}
```

➔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 source code which is in the AppFactory.java class.
➔It is not possible to run our application in other systems & follow the same path it may be different.
➔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.**
➔Elipse 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 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

```
AppFactory3.class.getClassLoader().getResourceAsStream("com\\sdp\\common\\app_class.properties"));
                                                        Relative Path
```

**Ex:**

```
public class AppFactory3 {
        static Properties  props;
        static{
           try {
                props=new Properties();
                props.load(AppFactory3.class.getClassLoader().getResourceAsStream("com\\sdp\\common\\app_class.properties"));
                } catch (IOException e) {e.printStackTrace();}//Get Location   Alt + Enter
        }
        public static Object createObject(String logicalName) throws Exception{
                Object obj=null;
                String className=null;
                className=props.getProperty(logicalName);
                obj=Class.forName(className).newInstance();
                return obj;
        }
}
```

➔Using property file we can manage the dependency between the classes but if there are multiple classes are available as part of application, 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 are going to write boiler plate logic in every application while developing.

➔**To manage it effectively we have to give our classes to the spring and ask to the spring to manage the dependency.**

➔**Let see the example to understand**

```java
package com.sfp.beans;
public interface IMessageProducer {
	String convertMessage(String message);
}
```

```java
package com.sfp.beans;
public class HtmlMessageProducerImpl implements IMessageProducer{
	@Override
	public String convertMessage(String message) {
		return "<html><body>" +message+"</body></html>";
	}
}
```

```java
package com.sfp.beans;
public class PdfMessageProducerImpl implements IMessageProducer {
	@Override
	public String convertMessage(String message) {
		return "<pdf>"+message+"</pdf>";
	}
}
```

```java
package com.sfp.beans;
public class MessageWriter {
	private IMessageProducer messageProducer;
	public void setMessageProducer(IMessageProducer messageProducer) {
		this.messageProducer = messageProducer;
	}
	public void writeMessage(String message){
		String msg=null;
		msg=messageProducer.convertMessage(message);
		System.out.println(msg);
	}
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
	http://www.springframework.org/schema/beans/spring-beans.xsd">

	<bean id="messageWriter" class="com.sfp.beans.MessageWriter">
		<property name="messageProducer" ref="htmlMesageProducer"/>
	</bean>
	<bean id="htmlMesageProducer" class="com.sfp.beans.HtmlMessageProducerImpl"/>
	<bean id="pdfMesageProducer" class="com.sfp.beans.PdfMessageProducerImpl"/>
</beans>
```

```java
package com.sfp.test;
public class Test {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/sfp/common/application-context.xml"));
		MessageWriter messageWriter=(MessageWriter)factory.getBean("messageWriter");
		messageWriter.writeMessage("Welcome To Spring");
	}
}
```

➔Here we are not creating or managing the dependencies between the components, spring has created and managed the dependencies between the components but we have to provide the information about classes to spring then only spring able to manage and can able to create the object .

➔Spring has provided a standard format, we have to provide the information of our classes to the spring framework through spring bean configuration file.

# Spring Core

### Q.What is Spring Core?

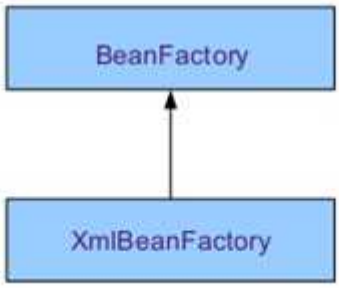
➔ Spring has been architected in such a way the spring has been broken down into several parts these parts are called modules, there are several modules are there with in the spring framework.

➔ In that one module called **spring core which is base module or infrastructure module that's why unless until we learn the spring core we can't learn rest of the module** that are there with in the spring framework.

➔ **It supports managing the dependencies and managing complexity that are there with the components.**

### Q.What is BeanFactory?

BeanFactory is an interface and XmlBeanFactory is it's implementation.




➔BeanFactory supports to creating and managing the dependencies of the components.

### Q. Why Spring has provided BeanFactory as Interface?

➔When we use interface (as BeanFactory) we can easily switch from one configuration classes to another configuration classes (implemented class from one class to another class) to ensure that our application component will become loosely coupled.

➔BeanFactory a common interface between all the configuration classes

**Eg.** Spring reads the configuration about our classes from xml file and even we can configure our classes to the properties file,

**To read the configuration about our classes from xml we need XmlBeanFactory**, to read configuration from properties file we need another class.

Now switching form one configuration to another configuration we need to change the source code within our application to ensure our application components will become loosely coupled spring people provided BeanFactory as interface.

### Q. When we called a class as a Spring Bean?

➔If a class object being created and being managed by spring framework then those kinds of classes are called as spring bean.

### Q.What is Spring-bean-configuration File?

➔Spring has provided a spring-bean-configuration-file, which is used to configure our classes and to manage the dependency between them.

➔In spring-bean-configuration-file, the root tag is **<beans></beans> because it not only contains one class information but also contain multiple classes information, every class should have defined as bean** so we have to configure our classes into the bean-configuration file, **to configure class use <bean> tag which having two Properties i.e 'id' and 'class'.**

➔ It is the convention to write the spring-bean-configuration-file. We have to follow the syntax and rule of the spring. By help of **"id"** we are going to get particular class object, and **"class"** represents the actual class path location for creating an object.

### Q.Why we provide Resource as input to BeanFactory?

We can provide the path by using two ways, by using absolute path or by using relative path both are seems to be stream.

XmlBeanFactory (BeanFactory implemented classes) can't distinguish between absolute path and relative path as both seems to be strange on it.

If it is the case spring has to provide two different different class one is for absolute path and another is for relative path like AbsolutePathXmlBeanFactory and RelativePathXmlBeanFactory.

XmlBeanFactory (BeanFactory implemented classes) never read the file at all .it will not read the file from absolute path or from relative path rather he added two more class

ClassPathResource----for -----➔**Relative Path**
FileSystemResource---for ----➔**Absolute Path**

### Q.What will happen when we execute a spring program?

➔When we execute the spring program our program will starts executing with BeanFactory

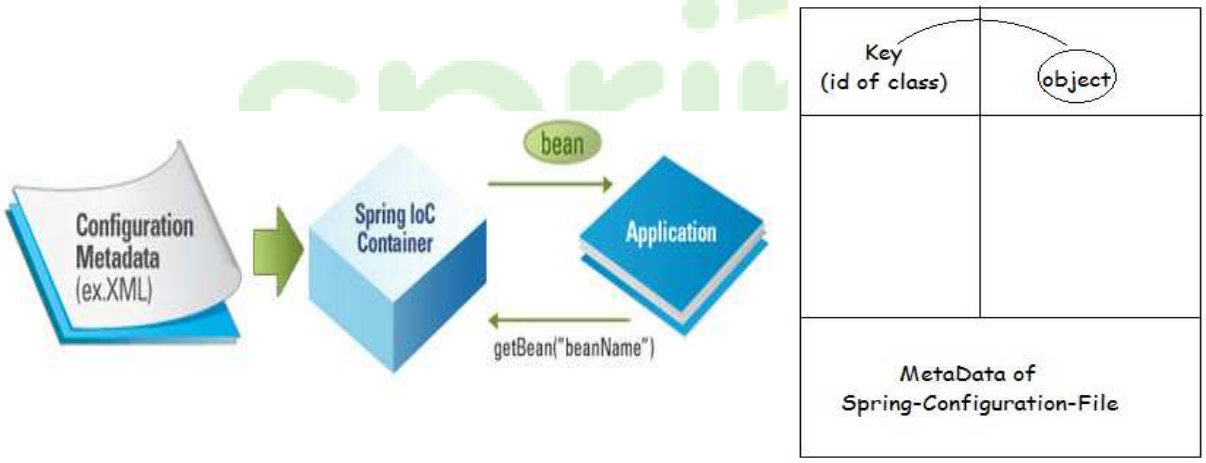
```
BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/sfp/common/application-context.xml"));
```

➔First of all **ClassPathResource object will be created.**
➔ClassPathResource will go to the classpath location of our application (bin) and go to the specified location and try to **identify application-context.xml**
➔Then it reads the configuration file and creates a **Resource Object (Input Stream)** and **pass as an input to the XmlBeanFactory**.
➔ Then XmlBeanFactory will **first check resource is well formed or not** (syntactical errors, means all the start tags are closed or not).
➔If it is well formed then it **will validate all the tags** that are inside configuration file.
➔**Then BeanFactory will start creating an In-Memory location (Logical memory partition) within the JVM Memory this is called IOC container / Core container.**
➔Then quickly reach that physical configuration file that is being passed as an input, it quickly loads that and create a METADATA (information about our classes) in side In-Memory location (Logical memory partition)(to reduce the IO Operation).

```
MessageWriter messageWriter=(MessageWriter)factory.getBean("messageWriter");
```

➔ When we call getBean("idName") now the BeanFactory has to go into the In-Memory location (Logical memory partition) **METADATA** and creates the object of specified class by using reflection .

```
Class.forName(className).newInstance();
```




IOC Container / Core Container

➔If again we call **getBean("idName"), then BeanFactory will not go to the METADATA**, it will go to the MAP and checks the id is already there with MAP or not,

- ⇨ If it is there it returns the same object, it will not create another object for same Id.
- ⇨ If it is not there then it goes to the METADATA and search for the Bean Definition of the class and then creates the object and place in IOC container.

➔After creating the object it keeps the object with it, and goes to the child <property> tag, and checks the attribute having setter or not.
➔If setter is there bean will call the setter by passing the reference of another bean, before passing the reference first it creates the object of that reference and place it in Map.

# IOC Container (Core Container)

## What is IOC?

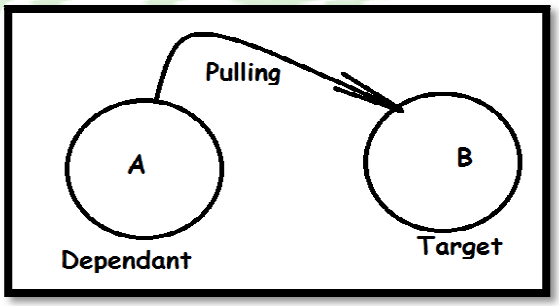
➔**IOC is Standard design principle belongs to Object Oriented programming world and it is not provided by the spring people.**
➔**It collaborating** (injecting dependency) an object and **managing the lifecycle** (create /manage/release) **of the object**.
➔IOC is the logical memory in the JVM memory.
➔IOC container memory having two parts
- ⇨ **In memory METADATA**
- ⇨ **Empty (for storing the object (key (id) & value (Object))).**

➔It is used to managing the dependency between the objects.

➔There are two ways to collaborating (injecting dependency) the object.
- ⇨ **Dependency pull**
  1. Dependency lookup
  2. Contextual dependency lookup
- ⇨ **Dependency Injection**
  1. Constructor injection
  2. Setter injection

### Dependency pull:

A class taking help from other class to perform a task called as dependency.

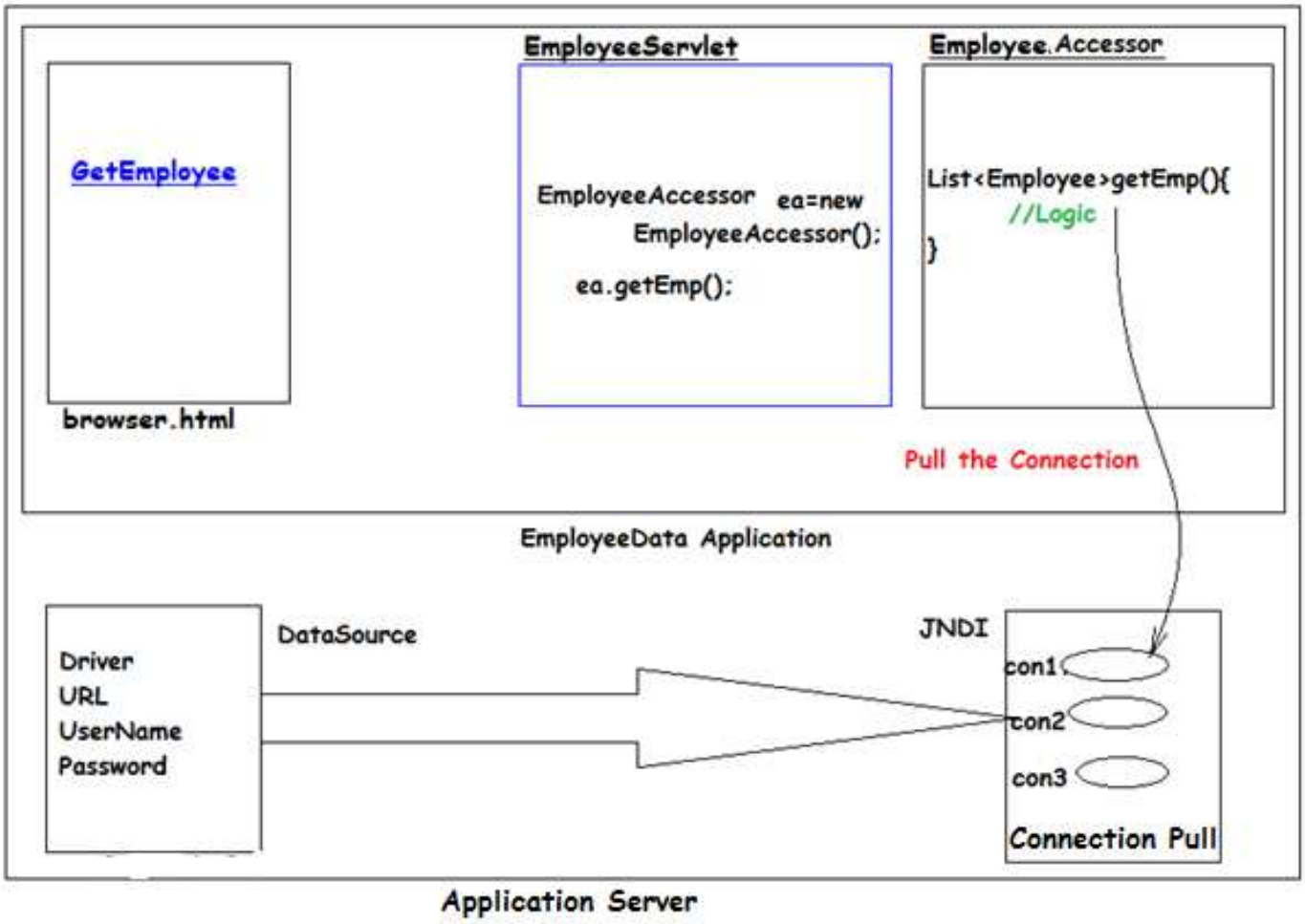


## Dependency Lookup

➔Dependency Lookup is a very old technique which is already exists in most of the JavaEE applications use.

**Ex:**

➔A programmer wants to persist data into the database through Application Server, for persisting the data **we have to pull the connection from the JNDI** registry and then we can persist the data into database.(it's like factory class ref:17 page)




```
Context ctx=new InitialContext();
DataSource ds=ctx.lookup("con1");
Connection con=ds.getConnection();
```

**To get the connection EmployeeAccessor class has to look up the JNDI registry to get the connection...**

### What is JNDI registry?

➔It is global memory where sharable data can be placed. To exposing sharable resources to the user or client we are 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 information to the application server and server is going to create the connection with database. And it will place into the connection Pool.

## Contextual dependency lookup

➔In this technique our component will follow some rules, through which dependant object will be injected into your code.

Ex:
➔If we want to get the internal details of the servlets, version, absolute path, Port No, ip address, 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 in our class.

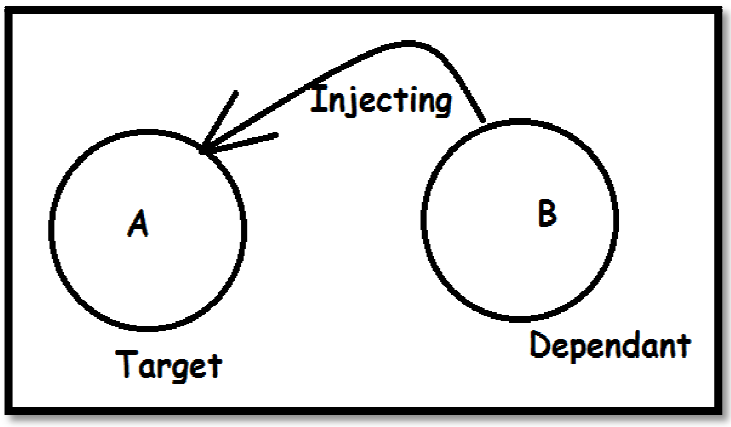
Let's see the procedure…

```
public class MyServlet extends HttpServlet implements Servlet {
        public void init(ServletConfig config) throws ServletException {
                ServletContext context=config.getServletContext();
                //Logic
                        Ex.  context.getContextPath();
                .....
        }
}
```

➔ **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.

## Dependency Injection

➔In dependency pull a class is totally depends on the other class or other runtime environment.
➔To make our classes completely loosely coupled, we have to use dependency injection concept from IOC principle.
➔While performing a 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.

## 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 id=" " class=" ">
    <property name="<attribute_name>" ref="<class_name>">
</bean>
```

Ex:

```
package com.ssp.beans;
public interface IReceiver {
        void tuneUp();
}
```

```
package com.ssp.beans;
public class Receiver10Hz implements IReceiver{
        @Override
        public void tuneUp() {
                System.out.println("Tune up...");
        }
}
```

```
<beans>
        <bean id="radio" class="com.ssp.beans.Radio">
                <property name="receiver" ref="receiver"/>
        </bean>
        <bean id="receiver" class="com.ssp.beans.Receiver10Hz"/>
</beans>
```

```
package com.ssp.beans;
public class Radio {
        private IReceiver receiver;
        public void setReceiver(IReceiver receiver) {
                this.receiver = receiver;
        }
        public void  listen(){
                receiver.tuneUp();
                System.out.println("Listening...");
        }
}
```

```
public class Test {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/ssp/common/application-context.xml"));
                Radio radio=factory.getBean("radio",Radio.class);
                radio.listen();
        }
}
```

## 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>
```

Ex.

```
package com.ssp.beans;
public interface IReceiver {
        void tuneUp();
}
```

```
package com.ssp.beans;
public class Receiver10Hz implements IReceiver{
        @Override
        public void tuneUp() {
                System.out.println("Tune up...");
        }
}
```

```
package com.ssp.beans;
public class Radio {
        private IReceiver receiver;
        public Radio(IReceiver receiver) {
                super();
                this.receiver = receiver;
        }
        public void  listen(){
                receiver.tuneUp();
                System.out.println("Listening...");
        }
}
```

```
<beans >
        <bean id="radio" class="com.ssp.beans.Radio">
                <constructor-arg ref="receiver"/>
        </bean>
        <bean id="receiver" class="com.ssp.beans.Receiver10Hz"/>
</beans>
```

```
public class Test {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/ssp/common/application-context.xml"));
                Radio radio=factory.getBean("radio",Radio.class);
                radio.listen();
        }
}
```

## Q.What is IOC Container?

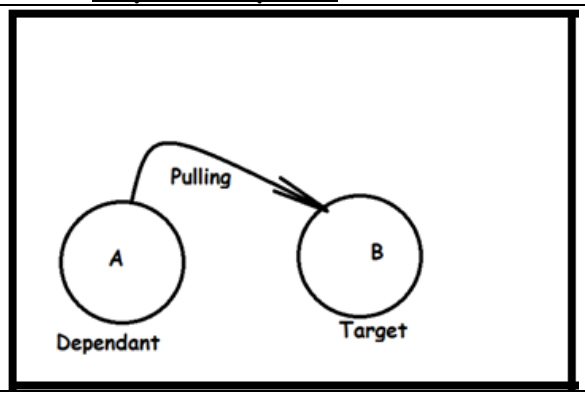
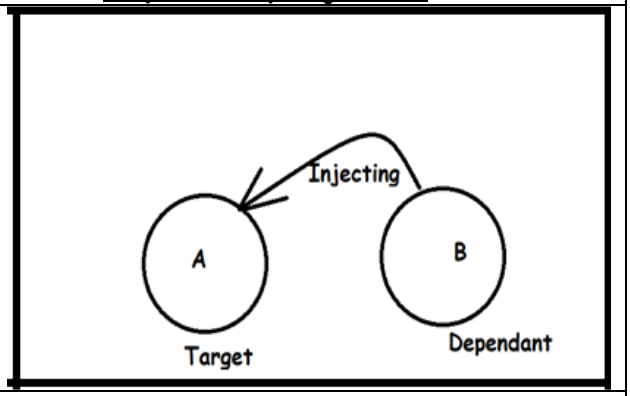
➔IOC is a design principle,

➔ **It talks about collaborating** an object and **managing the lifecycle of the object.**

➔ **IOC going to support all the four ways of the managing the dependency.**

- ⇨ **Dependency pull**
  1. Dependency lookup
  2. Contextual dependency lookup
- ⇨ **Dependency Injection**
  1. Constructor injection
  2. Setter injection

## Q.Why IOC called as Inversion of control.

| Dependency Pull | Dependency injection |
|---|---|
|  |  |
| ❖ In traditional ways we used dependency pull to get the | ❖ This is the Process of injecting a dependent object in to the target |

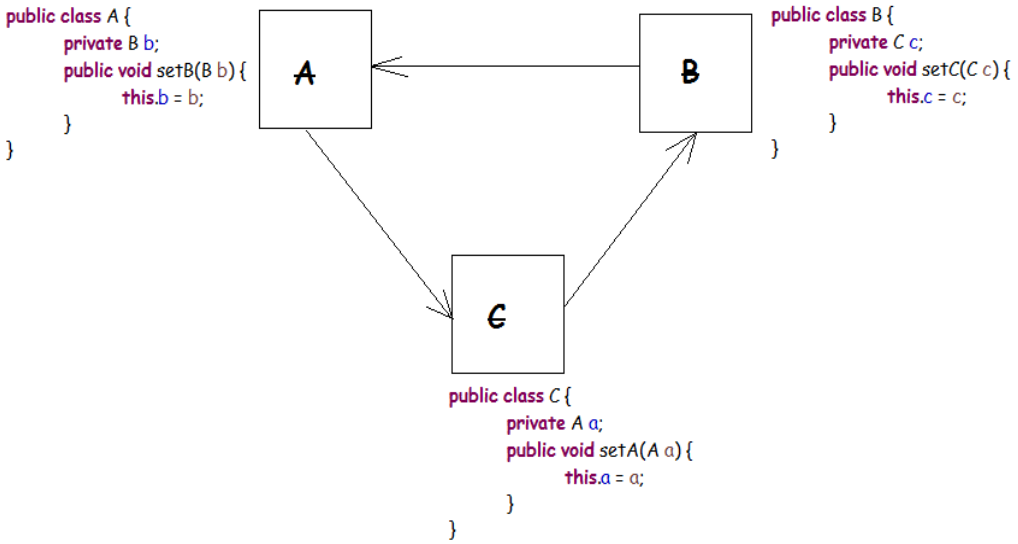
| object of another class | component. |
|---|---|
| ❖ A class is going to getting the object of B ,<br>❖ Here B is Target Class and A is Dependant class here A is totally depends on B.<br>❖ **Here A is pulling the object of B and performing the task.** | ❖ B class is going to inject with class A.<br>❖ Here A is Target class and B is Dependant class.<br>❖ **Here B is injecting to A and performing the task.** |

**That's why this is called Inversion of control**

## Q-What is Difference between Setter Injection and Constructor Injection?

| Setter Injection | Constructor Injection |
|---|---|
| 1. In case of setter injection always a dependency is injected after creating the object of target class. | 1. In case of constructor injection always dependency is injected while creating the object of target class. |
| 2. In case of Setter injection <property> tag is optional. Means we may write <property> tag or we may not write the <property>tag still object will be created. | 2. In case of Constructor <constructor-arg> is mandatory we must have to write <constructor>tag otherwise we will get exception.RE: BeanCreationException: |
| 3. We can handle circular dependency using the setter injection. | 3. We cannot handle circular dependency using the constructor injection. |

## Q-What is Circular dependency and how it is recognized by IOC Container?



➔When having a circular dependency, spring cannot decide which of the beans should be created first, since they depend on one another. In these cases, Spring will raise a *BeanCurrentlyInCreationException*
Requested bean is currently in creation: Is there an unresolvable circular reference? while loading context.

➔It can happen in spring when using **constructor injection**; if you use setter injection you should not find this problem since the dependencies will be injected when they are needed and not on the context loading.

➔We cannot handle circular dependency using the constructor injection because its leads to the deadlock. One is totally depends on other one in circular manner.

➔**When we use constructor injection IOC will create a Graph and maintain all beans as NODE and dependent as point to that NODE. In this way IOC can recognize circular dependency**.

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

➔ In constructor injection always dependency is injected while creating the object of target class so **if you want to access dependent object in side constructor then you should go for constructor.**

## Primitive Injection

```
public class Employee {
        private int empId;
        private String empName;
        public Employee(int empId, String empName) {
                super();
                this.empId = empId;
                this.empName = empName;
        }
        @Override
        public String toString() {
                return "Employee [empId=" + empId +
                        ", empName=" + empName + "]";
        }
}
```

***Constructor Injection***

```
<beans>
        <bean id="employee" class="com.pi.beans.Employee">
                <constructor-arg value="101"/>
                <constructor-arg value="David"/>
        </bean>
</beans>
```

***Setter Injection***

```
<beans>
        <bean id="employee" class="com.pi.beans.Employee">
                <property name="empId" value="101"/>
                <property name="empName" value="David"></property>
        </bean>
</beans>
```

```
public class Test {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/pi/common/application-context.xml"));
                Employee employee=factory.getBean("employee",Employee.class);
                System.out.println(employee);
        }
}
```

## Collection Injection

➔When we configure any of the collection classes as beans in Bean-Configuration file then always collection object will be created and **will be initialize as empty** because all the collection class that are provided by API **will not have Setter / Constructor to populating object as values by configuring** as beans **which seems to be an biggest limitation.**

➔If we want to inject collection type properties in to target class, by default it is not possible when we configure API provided collection class as bean, so spring provided some extra tags to populate data when we use collection type properties. This is called Collection dependency injection

Spring support four kinds of list collection injection dependency.

- **List**
- **Set**
- **Map**
- **Properties**

```
public class College {
        private Map<String,Course> hodToCourse;
        private Properties CourseTopper;

        public void setHodToCourse(Map<String, Course> hodToCourse) {
                this.hodToCourse = hodToCourse;
        }
        public void setCourseTopper(Properties courseTopper) {
                CourseTopper = courseTopper;
        }
        @Override
        public String toString() {
                return "College [hodToCourse=" + hodToCourse
                        + ", CourseTopper=" + CourseTopper + "]";
        }
}
```

```
<bean id="college" class="com.cdi.beans.College">
        <property name="hodToCourse">
                <map key-type="java.lang.String" value-type="com.cdi.beans">
                        <entry key="Dhananjaya" >
                                <ref bean="cse1stSem"/>
                        </entry>
                </map>
        </property>
        <property name="courseTopper">
                <props>
                        <prop key="CSE">Ravi</prop>
                </props>
        </property>
</bean>
```

```
public class Test {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/cdi/common/application-context.xml"));
                College college=factory.getBean("college",College.class);
                System.out.println(college);
        }
}
```

```java
public class Course {
	private List<String> subjects;
	private Set<String> faculties;
	public Course(Set<String> faculties) {
		this.faculties = faculties;
	}
	public void setSubjects(List<String> subjects) {
		this.subjects = subjects;
	}
	@Override
	public String toString() {
		return "Course [subjects=" + subjects
			+ ", faculties=" + faculties + "]";
	}
}
```

```xml
<bean id="cse1stSem" class="com.cdi.beans.Course">
	<property name="subjects">
		<list value-type="java.lang.String">
			<value>C</value>
			<value>D.S</value>
			<value>DBMS</value>
		</list>
	</property>
	<constructor-arg>
		<set value-type="java.lang.String">
			<value>Rama</value>
			<value>Sita</value>
			<value>Laxman</value>
		</set>
	</constructor-arg>
</bean>
```

```java
public class Test {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/cdi/common/application-context.xml"));
		Course course=factory.getBean("cse1stSem",Course.class);
		System.out.println(course);

	}
}
```

➔We are using collection tags (<list>, <set>, <map>, <properties>) inside bean which is not available to another bean it seems to be anonymous object we can't use it in other bean.

➔If we want to reuse same data across multiple beans then we should go for Util Name Space

➔ When we are using collection tags then

<list>tag by default internally creates object of :java.util.ArrayList

<set> tag by default internally creates object of :java.util.LinkedHashSet

<map> tag by default internally creates object of :java.util.LinkedHashMap

➔It is not possible to create object for other implementation classes of collection interfaces when we are using <list>, <set>,<map>,tags , then must we should go for Util NameSpace.

Ex:

```
public class Course {
        private  List <String> subjects;
        private  Set <String> faculties;
        private Map<String,String> hod;
        private Properties topper;
        //setter
        //toString()
        }
}
```

```
public class Test {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/uns/common/util-namespace.xml"));

                Course
firstSemCSE=factory.getBean("firstSemCSE",Course.class);
                System.out.println(firstSemCSE);

        }
}
```

```
<beans>
        <util:list id="list" value-type="java.lang.String" list-class="java.util.Vector">
                <value>C</value>
                <value>DS</value>
        </util:list>
        <util:set id="set" value-type="java.lang.String" set-class="java.util.HashSet">
                <value>Rama</value>
                <value>Sita</value>
        </util:set>
        <util:map id="map" key-type="java.lang.String" value-type="java.lang.String" map-
class="java.util.HashMap">
                <entry key="CSE"  value="Dhananjaya"/>
                <entry key="EEE"  value="Arjun"/>
        </util:map>
        <util:properties id="properties">
                <prop key="CSE">Haria</prop>
                <prop key="ECE">Baya</prop>
        </util:properties>


        <bean id="firstSemCSE" class="com.uns.beans.Course">
                <property name="subjects" ref="list"/>
                <property name="faculties" ref="set"/>
                <property name="hod" ref="map"/>
                <property name="topper" ref="properties"/>
        </bean>

        <!--
        <bean id="firstSemIt" class="com.uns.beans.Course">
                <property name="subjects" ref="list"/>
                <property name="faculties" ref="set"/>
                <property name="hod" ref="map"/>
                <property name="topper" ref="properties"/>
        </bean>
         -->
</beans>
```

## Q.What is bean aliasing?

➔Providing more than one name for a bean is called bean alias.
➔Apart from id of a bean if you want to give alternate name for a bean then we should go for bean alias.
➔If I want to give my own convention (friendly name) to access the bean, it will not be possible when I use ID, then we should go for alias name.
➔There are two ways to giving the alias name to bean

- ⇨ By using 'name' attribute(initial version)
- ⇨ By using <alias> tag **(spring 2.0)**

## Q-If name attribute is there what use of <alias> tag is.

➔When we using name attribute inside <property> tag it is possible give multiple alias names to by using ','comma separator.
➔But sometimes our requirement is give the alias name to bean which contain "," comma (like" 505, Ameerpet"), then we should go for <alias>tag

# Use-Case of aliasing.

```
public interface ICourierService {
        String courier(List products , String address);
}
```

```
public class BlueDartCourierImpl implements ICourierService{
        private BlueDartService service;
        public void setService(BlueDartService service) {
                this.service = service;
        }
        @Override
        public String courier(List products , String address) {
                String status=service.courier(products, address);
                return status;
        }
}
```

```
public class DtdcCourierImpl implements ICourierService{
        private DtdcService service;
        public void setService(DtdcService service) {
                this.service = service;
        }
        @Override
        public String courier(List products , String address) {
                String status=service.parcel(products, address);
                return status;
        }
}
```

```
public class BlueDartService {
        public String courier(List products , String address){
                return "Order received by BlueDart";
        }
}
```

```
public class DtdcService {
        public String parcel(List products , String address){
                return "Order received by DTDC";
        }
}
```

```
public class AmazonOrder {
        private List<String> products;
        private String address;
        private String city;
        private int zipCode;
        private String country;

        //Setters
        //Getters
}
```

```
public class AmazonOrderManager {
        private AmazonOrder order;
        private ICourierService dtdcCourierImpl;
        private ICourierService blueDartCourierImpl;

        //Setters

        public String setOrder(){
                String status=null;
                if (order.getZipCode()<=70000 && order.getZipCode()>=60000) {
                        status=dtdcCourierImpl.courier(order.getProducts(), order.getAddress());
                }
                else{
                        status=blueDartCourierImpl.courier(order.getProducts(), order.getAddress());
                }
                return status;
        }
}
```

```
<beans>

        <bean id="amazonOrder" class="com.as.beans.AmazonOrder">
                <property name="products">
                        <list value-type="java.lang.String">
                                <value>Samsung Galaxy 3309</value>
                                <value>Sony Cyber shot</value>
                        </list>
                </property>
                <property name="address" value="Ameerpet"/>
                <property name="city" value="Hyderabad"/>
                <property name="zipCode" value="90050"/>
                <property name="country" value="India"/>
        </bean>

        <bean id="amazonOrderManager"
class="com.as.beans.AmazonOrderManager">
                <property name="order" ref="amazonOrder"/>
                <property name="dtdcCourierImpl" ref="dtdcCourierImpl"/>
                <property name="blueDartCourierImpl"
ref="blueDartCourierImpl"/>
        </bean>

        <bean id="blueDartService" class="com.as.beans.BlueDartService"/>
        <bean id="dtdcService" class="com.as.beans.DtdcService"/>

        <bean id="blueDartCourierImpl"
class="com.as.beans.BlueDartCourierImpl">
                <property name="service" ref="blueDartService"></property>
        </bean>
        <alias name="blueDartCourierImpl" alias="dtdcCourierImpl"/>
        <!--
        <bean id="dtdcCourierImpl" class="com.as.beans.DtdcCourierImpl">
                <property name="service" ref="dtdcService"></property>
        </bean>
         -->
</beans>
```

```
public class AsTest {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/as/common/application-context.xml"));
                AmazonOrderManager amazon=factory.getBean("amazonOrderManager",AmazonOrderManager.class);
                System.out.println(amazon.setOrder());
        }
}
```

If we want to switch one class to another class without changing in source code, we should go for aliasing, but not only this bean some other beans also referring to this bean Id , So it is not possible to change in everywhere.

## Constructor Confusion

➔Even though we are giving the appropriate value to the bean, IOC container will not successfully inject the value well, because IOC Container may not understand and incorrectly try to inject. In such how can we manage the constructor injection let's see.

### Index Attribute

```
public class Product {
        private int pid;
        private String pname;
        private double price;

        public Product(int pid, String pname) {
                this.pid = pid;
                this.pname = pname;
        }
        public Product(String pname, double price) {
                this.pname = pname;
                this.price = price;
        }
        @Override
        public String toString() {
                return "Product [pid=" + pid + ",
                        pname=" + pname + ", price=" + price + "]";
        }
}
```

```
<beans>
        <bean id="product" class="com.cc.beans.Product">
                <constructor-arg value="chocolate"/>
                <constructor-arg value="10" index="0"/>
        </bean>
</beans>
```

```
public class Test {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/cc/common/application-context.xml"));
                Product product=factory.getBean("product",Product.class);
                System.out.println(product);
        }
}
```

### Type Attribute

```
public class Product {
        private int pid;
        private String pname;
        private String type;

        public Product(int pid, String pname) {
                this.pid = pid;
                this.pname = pname;
        }
        public Product(String pname, String type) {
                this.pname = pname;
                this.type = type;
        }
        @Override
        public String toString() {
                return "Product [pid=" + pid + ",
                        pname=" + pname + ", type=" + type + "]";
        }
}
```

```
<beans>
        <bean id="product" class="com.cc.beans.Product">
                <constructor-arg value="chocolate"/>
                <constructor-arg value="10" type="int"/>
        </bean>
</beans>
```

```
public class Test {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/cc/common/application-context.xml"));
                Product
product=factory.getBean("product",Product.class);
                System.out.println(product);
        }
}
```

### Name Attribute

```
public class Product {
        private int pid;
        private String pname;
@ConstructorProperties({ "pid", "pname"})
        public Product(int pid, String pname) {
                this.pid = pid;
                this.pname = pname;
        }
        @Override
        public String toString() {
                return "Product [pid=" + pid + ",
                                pname=" + pname + "]";
        }
}
```

```
<beans>
        <bean id="product" class="com.cc.beans.Product">
                <constructor-arg value="chocolate" name="pname"/>
                <constructor-arg value="10" name="pid"/>
        </bean>
</beans>
```

```
public class Test {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/cc/common/application-context.xml"));
                Product product=factory.getBean("product",Product.class);
                System.out.println(product);
        }
}
```

## Inner Bean

➔If without exiting a bean if there is no use of another bean then we should go for inner bean.

➔In our application there are multiple beans are there as part of our application , but the dependant bean which we are trying to inject to target bean, **if there is no bean accessing the dependant bean instead of using that bean as separate bean declare that as inner bean inside the target bean** then it will be easy to manage the bean and the complete information will be in a one single bean, so we can avoid programmatic mistakes or configuration mistakes while managing such kind of bean , its recommended to go for inner Bean

```
public class Bicycle {
	private String manufacturer;
	private Chain chain;
	public void setManufacturer(String manufacturer) {
		this.manufacturer = manufacturer;
	}
	public void setChain(Chain chain) {
		this.chain = chain;
	}
	@Override
	public String toString() {
		return "Bycycle [manufacturer=" +
		manufacturer + ", chain=" + chain + "]";
	}
}
```

```
<beans>
	<bean id="bicycle" class="com.ib.beans.Bicycle">
		<property name="manufacturer" value="hero"/>
		<property name="chain" >
			<bean class="com.ib.beans.Chain">
				<property name="type" value="Metal"/>
			</bean>
		</property>
	</bean>
</beans>
```

```
public class Test {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/ib/common/application-context.xml"));
		Bicycle bicycle=factory.getBean("bicycle",Bicycle.class);
		System.out.println(bicycle);
	}
}
```

## Bean Inheritance

➔ A Bean contains configuration, bean inheritance means how to reuse the configuration one bean inside another bean rather than write configuration for another bean.
➔To get bean inheritance we have to use "parent" attribute inside <bean> tag, **but in this case strictly speaking the corresponding classes will not get inherited.**

➔ Means the **configuration value that we have configure for parent bean will be just copied in to the child bean.**

```
public class Car {
        private int id;
        private String model;
        private String manufacturer;
        private String color;
        private String fuelType;
        private float price;
        //Setters
        //toString();
        }
}
```

```
public class BITest {
        public static void main(String[] args) {
        BeanFactory factory=new XmlBeanFactory(new ClassPathResource
        ("com/bi/common/application-context.xml"));

        Car car=factory.getBean("car",Car.class);
        System.out.println(car);

        Car car1=factory.getBean("car1",Car.class);
        System.out.println(car1);

        Car car2=factory.getBean("car2",Car.class);
        System.out.println(car2);

        }
}
```

```
Car [id=1, model=swift, manufacturer=Maruti Suzuki, color=red, fuelType=petrol, price=4564665.0]
Car [id=2, model=swift, manufacturer=Maruti Suzuki, color=White, fuelType=petrol, price=4564665.0]
Car [id=3, model=swift, manufacturer=Maruti Suzuki, color=Black, fuelType=petrol, price=4564665.0]
```

```
<beans>
        <bean id="car" class="com.bi.beans.Car">
                <property name="id" value="1"/>
                <property name="model" value="swift"/>
                <property name="manufacturer" value="Maruti Suzuki"/>
                <property name="color" value="red"/>
                <property name="fuelType" value="petrol"/>
                <property name="price" value="4564665"/>
        </bean>
        <bean id="car1" class="com.bi.beans.Car" parent="car">
                <property name="id" value="2"></property>
                <property name="color" value="White"></property>
        </bean>
        <bean id="car2" class="com.bi.beans.Car" parent="car">
                <property name="id" value="3"></property>
                <property name="color" value="Black"></property>
        </bean>
</beans>
```

### Q-What will happen when we use "parent" attribute?

➔Bean Factory goes to the IOC container and then goes to the in memory meta data and search the bean definition of the class, after that immediately it will not create the object of corresponding class first it will checks is there any parent or not, if parent is there it directly go to the parent bean and copy all the configuration properties and paste it in child bean.

➔If **any property declared inside child bean then parent bean data will overwrite that property by child bean data.**

➔At that time only one time bean factory will inject the properties.

➔We **can use any type bean for inheritance** but make sure that **all properties of parent bean must same as properties of child bean.**

## Q-What is use of abstract bean?

➔We can make a bean as abstract bean by using "abstract" attribute.
➔ **When we are using "abstract" attribute Bean Factory will not able create the object of that bean definition.**
➔So when we are using **bean inheritance** then only we are using abstract bean concept.
➔Because when we are using a non-abstract/ used bean as parent bean, any change in parent bean will be effected to the child beans, so recommended to use abstract bean/non-used bean because it will never instantiate the object for the bean.
➔As it never instantiate the object for the bean then no need to use "class" attribute for the bean.
**Ex:**

```
<beans >
        <bean id="baseCar"  abstract="true">
                <property name="model" value="swift"/>
                <property name="manufacturer" value="Maruti Suzuki"/>
                <property name="color" value="red"/>
                <property name="fuelType" value="petrol"/>
                <property name="price" value="4564665"/>
        </bean>
        <bean id="car" class="com.bi.beans.Car" parent="baseCar">
                <property name="id" value="1"/>
                <property name="color" value="red"/>
        </bean>
        <bean id="car1" class="com.bi.beans.Car" parent="baseCar">
                <property name="id" value="2"></property>
                <property name="color" value="White"></property>
        </bean>
        <bean id="car2" class="com.bi.beans.Car" parent="baseCar">
                <property name="id" value="3"></property>
                <property name="color" value="Black"></property>
        </bean>
</beans>
```

## Bean Inheritance through Constructor.

➔When we are using bean inheritance through <constructor-args> must we have to specify 'name' attribute.

➔Otherwise it seems to be an extra argument to constructor when we want to over write any argument of that constructor.

```
<beans>

        <bean id="baseCar"  abstract="true">
                <constructor-arg value="BMW" name="model"/>
                <constructor-arg value="BMW Company" name="manufacturer"/>
                <constructor-arg value="Petrol" name="fuelType"/>
                <constructor-arg value="4566731" name="price"/>
        </bean>
        <bean id="car" class="com.bi.beans.Car" parent="baseCar">
                        <constructor-arg value="01" name="id"/>
                        <constructor-arg value="RED" name="color"/>
        </bean>
        <bean id="car1" class="com.bi.beans.Car" parent="baseCar">
                        <constructor-arg value="02" name="id"/>
                        <constructor-arg value="Black" name="color"/>
        </bean>
        <bean id="car2" class="com.bi.beans.Car" parent="baseCar">
                        <constructor-arg value="03" name="id"/>
                                <constructor-arg value="Yellow" name="color"/>
        </bean>
</beans>
```

## Bean Auto wiring

➔When ever we declare two classes as a bean, spring by default will not able managing the dependency between those beans, Programmer has to provide the additional configuration to managing the dependency between those bean definitions.

**➔If we don't want to declaring <property> or <constructor-arg> tag rather IOC container will automatically identify such kind of dependency and has to manage it. Then this is called Bean Auto wiring.**

➔By default the bean auto wiring is not enabled, we need to write one attribute called **autowire="mode"** in bean tag level.

➔Here **mode** tell the IOC container how to identify the dependency and the possible values for the mode are:

- ⇨ **byName**
- ⇨ **byType**

  } **Setter Injection**

- ⇨ **constructor**
- ⇨ **autoDetect (Deprecated from 3.0 onwards)**

## 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 and then it returns the target class object.

```
public class Chip {
        private int id;
        private String type;

        //Setters
        //toString()
}
```

```
public class Sensor {
        private int id;
        private String range;

        //Setters
        //toString()
}
```

```
public class Robot {
        private Chip chip;
        private Sensor sensor;

        //Setters
        //toString()
}
```

```
public class AWTest {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/aw/common/application-context.xml"));
                Robot robot=factory.getBean("robot",Robot.class);
                System.out.println(robot);
        }

}
```

```
<beans>
        <bean id="robot" class="com.aw.beans.Robot" autowire="byName"/>
        <bean id="chip" class="com.aw.beans.Chip">
                <property name="id" value="10"/>
                <property name="type" value="P3"/>
        </bean>
        <bean id="sensor" class="com.aw.beans.Sensor">
                <property name="id" value="1131"/>
                <property name="range" value="5.5m"/>
        </bean>
</beans>
```

## byType

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

```
public class Chip {
        private int id;
        private String type;

        //Setters
        //toString()
}
```

```
public class Sersor {
        private int id;
        private String range;

        //Setters
        //toString()
}
```

```
public class Robot {
        private Chip chip;
        private Sensor sensor;

        //Setters
        //toString()
}
```

```
public class AWTest {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/aw/common/application-context.xml"));
                Robot robot=factory.getBean("robot",Robot.class);
                System.out.println(robot);
        }
}
```

```
<beans>
        <bean id="robot" class="com.aw.beans.Robot" autowire="byType"/>
        <bean id="chip" class="com.aw.beans.Chip">
                <property name="id" value="10"/>
                <property name="type" value="P3"/>
        </bean>
        <bean id="sensor" class="com.aw.beans.Sensor">
                <property name="id" value="1131"/>
                <property name="range" value="5.5m"/>
        </bean>
</beans>
```

## Constructor

➔When we set autowire="constructor", the **IOC container will manage the dependencies via constructor, while creating the object of the target class**,
➔It will check the autowire mode, if it is constructor then it create the target object and then checks the target class attributes which having parameter to constructor and then it **checks the parameter Type with the bean Type,**

➔If both are matched, then it will inject the object of dependent bean and then it returns the target class object.

```java
public class Chip {
        private int id;
        private String type;

        public void setId(int id) {
                this.id = id;
        }
        public void setType(String type) {
                this.type = type;
        }
        //toString();
}
```

```java
public class Sensor {
        private int id;
        private String range;

        public void setId(int id){
                this.id = id;
        }
        public void setRange(String range) {
                this.range = range;
        }
        //toString()
}
```

```java
public class Robot {
        private Chip chip;
        private Sensor sensor;

        public Robot(Sensor sensor) {
                this.sensor = sensor;
        }
        public Robot(Chip chip) {
                this.chip = chip;
        }
        //toString();
}
```

```java
public class AWTest {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/aw/common/application-context.xml"));
                Robot robot=factory.getBean("robot",Robot.class);
                System.out.println(robot);
        }
}
```

```xml
<beans>
    <bean id="robot" class="com.aw.beans.Robot" autowire="constructor"/>
    <bean id="sensor" class="com.aw.beans.Sensor">
        <property name="id" value="1131"/>
        <property name="range" value="5.5m"/>
    </bean>
        <bean id="chip" class="com.aw.beans.Chip">
        <property name="id" value="10"/>
        <property name="type" value="P3"/>
    </bean>
</beans>
```

## Use Cases1:

```xml
<bean id="robot" class="com.aw.beans.Robot" autowire="constructor"/>
<bean id="sensor1" class="com.aw.beans.Sensor">
        <property name="id" value="23"/>
        <property name="range" value="5.5m"/>
</bean>

<bean id="chip" class="com.aw.beans.Chip">
        <property name="id" value="101"/>
        <property name="type" value="IR"></property>
</bean>
```

```java
public class Robot {
        private Sensor sensor;
        private Chip chip;

        public Robot(Chip chip) {
                super();
                this.chip = chip;
        }
        public Robot(Sensor sensor) {
                this.sensor = sensor;
        }
        @Override
        public String toString() {
                return "Robot [sensor=" + sensor + ", chip=" + chip + "]";
        }
}
```

➔ **If every constructor is having same parameters then highest priority given to the first configured constructor in the class in top to bottom manner**

## Use Cases2:

```
<bean id="robot" class="com.aw.beans.Robot" autowire="constructor"/>
    <bean id="sensor" class="com.aw.beans.Sensor">
        <property name="id" value="23"/>
        <property name="range" value="5.5m"/>
    </bean>
    <bean id="sensor2" class="com.aw.beans.Sensor">
        <property name="id" value="14"/>
        <property name="range" value="5.5m"/>
    </bean>
```

```
public class Robot {
    private Sensor sensor;

    public Robot(Sensor sensor) {
        this.sensor = sensor;
    }
    @Override
    public String toString() {
        return "Robot [sensor=" + sensor + "]";
    }
}
```

➔If name of the parameter in constructor same as any bean id then even though there are two beans it will consider the same matched name and inject.

## Use Cases3:

```
<bean id="robot" class="com.aw.beans.Robot" autowire="constructor"/>
<bean id="sensor1" class="com.aw.beans.Sensor">
    <property name="id" value="23"/>
    <property name="range" value="5.5m"/>
</bean>
<bean id="sensor2" class="com.aw.beans.Sensor">
    <property name="id" value="14"/>
    <property name="range" value="5.5m"/>
</bean>
```

```
public class Robot {
    private Sensor sensor;

    public Robot(Sensor sensor) {
        this.sensor = sensor;
    }
    @Override
    public String toString() {
        return "Robot [sensor=" + sensor + "]";
    }
}
```

```
NoUniqueBeanDefinitionException:
expected single matching bean but found 2: sensor1,sensor2
```

➔ If name of the parameter in constructor doesn't same as any bean id and if there are multiple beans of same class it will throws an Exception.

## Use Cases4:

```
<bean id="robot" class="com.aw.beans.Robot" autowire="constructor"/>
<bean id="sensor1" class="com.aw.beans.Sensor">
        <property name="id" value="23"/>
        <property name="range" value="5.5m"/>
</bean>
<bean id="sensor2" class="com.aw.beans.Sensor">
        <property name="id" value="14"/>
        <property name="range" value="5.5m"/>
</bean>
<bean id="chip" class="com.aw.beans.Chip">
        <property name="id" value="101"/>
        <property name="type" value="IR"></property>
</bean>
```

```
public class Robot {
        private Sensor sensor;
        private Chip chip;

        public Robot(Chip chip) {
                super();
                this.chip = chip;
        }
        public Robot(Sensor sensor) {
                this.sensor = sensor;
        }
        @Override
        public String toString() {
                return "Robot [sensor=" + sensor + ", chip=" + chip + "]";
        }
}
```

➔ Even though there are multiple beans of same class it will not throws an Exception because there is another constructors available in the class with matched type then that will be considered.

## Use Cases5:

```
<bean id="robot" class="com.aw.beans.Robot" autowire="constructor"/>
<bean id="sensor1" class="com.aw.beans.Sensor">
        <property name="id" value="23"/>
        <property name="range" value="5.5m"/>
</bean>
<bean id="chip" class="com.aw.beans.Chip">
        <property name="id" value="101"/>
        <property name="type" value="IR"></property>
</bean>
```

```
public class Robot {
        private Sensor sensor;
        private Chip chip;

        public Robot(Chip chip) {
                super();
                this.chip = chip;
        }
        public Robot(Sensor sensor) {
                this.sensor = sensor;
        }
        public Robot(Sensor sensor, Chip chip) {
                super();
                this.sensor = sensor;
                this.chip = chip;
        }
        @Override
        public String toString() {
                return "Robot [sensor=" + sensor + ", chip=" + chip + "]";
        }
}
```

➔If multiple constructors are there, then it gives the highest priority to the maximum parameter constructor.

## Q-What are the drawbacks by using bean auto wiring?

➔Bean Auto wiring is there in the spring, but it is not highly recommended to use bean auto wiring as part of our application and no one is using it within the market.

➔Whenever we are using the bean auto wiring there are several problems are there.

**Drawbacks**

Whenever we enable the autowire to manage the dependency between the beans it could make lot more **complicated for people,**

- **To understanding the dependency information between our components will become more difficult and quite complicated.**
- It is not easy to debugging our application because first of all we need to know which class is talking to which class, to derive such kind of dependencies it will take more number of time to find the relevant bean.

**Steps:**

Identify mode➔go to the relevant class➔find the attribute➔and understand which are all bean are available for those attribute➔and derive which bean is being injected.

It makes more completed to derive the dependencies between those beans, that itself tell you that **debugging the problem that we are encounter in the development because one don't know which class is talk to which class ,so we can't quick jump start the debugging.**

- If we are enabling the **autowire byname** then it will check the attribute name with bean id , because of one don't know which class is talk to which class , **if one has incorrectly modified the id of bean with some other name then that change will make our application completely broken down.** That mean a additional care has to be taken while we are working with autowire byname.

- If we are using **autowire byType** and if multiple beans having same class type , then autowire will not work it will runs it **to ambiguity** , even though there is a way to manage by using **"autiwire-candidate="false" " attribute then we have to do the manual injection only.**

- If we are using auto wiring mode with **autodetect**, while creating the object for our target **class first it identify the dependency and injecting that via constructor then again it identify the dependency and injecting that via setter**. That means to it will more complicate to understand which bean will be injected with which bean, **it gives double confusion when we use autodetect** so it has been completely deprecated from spring 3.0.

**Q-If we are not using autowiring why it is there as part of spring?**

➔ It is not highly recommended to use bean auto wiring as part of our application and no one is using it within the market. But still there are some circumstances under which we can go for using bean autowiring.

➔ If you want to do some add some additional new feature to your existing application, **it is not recommended to directly use it in your existing application** instead of that create a **(Peace of code(POC) application**) sample project and test that new feature in that project, in this case to **reduce the time for testing the sample project, no need to manually configure dependencies use autowire because there are less no of classes will be there as part of our sample project.**

## Null String

**Q-How to inject null value to constructor?**

➔By using **<null/>** tag in side <constructor-arg> </constructor-arg> we can pass null value to the constructor arguments.

➔In setter it is not required because setter is optional to declare the <property> tag.

```
<bean id="toy" class="com.ns.beans.Toy">
   <constructor-arg value="01"/>
   <constructor-arg>
      <null/>
   </constructor-arg>
</bean>
<bean id="remote" class="com.ns.beans.Remote">
</bean>
```

## Nested Bean Factory

**Parent**

```
public class B {
        private String name;
        public void setName(String name) {
                this.name = name;
        }
        @Override
        public String toString() {
                return "B [name=" + name + "]";
        }
}
```

**Child**

```
public class A {
        private B b;
        public void setB(B b) {
                this.b = b;
        }
        @Override
        public String toString() {
                return "A [b=" + b + "]";
        }
}
```

```
public class Test {
        public static void main(String[] args) {
                BeanFactory parentFactory=new XmlBeanFactory(new ClassPathResource("com/nbf/common/parentBean.xml"));
                BeanFactory childFactory=new XmlBeanFactory(new ClassPathResource("com/nbf/common/childBean.xml"),parentFactory);

                A a=childFactory.getBean("a",A.class);
                System.out.println(a);
        }
}
```

parentBean.xml

```
<bean id="b" class="com.nbf.beans.B">
        <property name="name" value="ChildObject"/>
</bean>
```

childBean.xml

```
<bean id="a" class="com.nbf.beans.A">
        <property name="b" ref="b"/>

    <!--        <property name="b">
                        <ref bean="b"/>
                        <ref parent="b"/>
                </property> -->

    </bean>
```

There are three attributes are available:

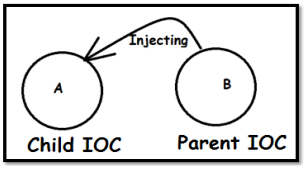
- ➔ **ref / <ref bean="b">:** it will check first in local context, if required bean available in local then it will inject local, if it is not there then it will check into parentBean.xml.
- ➔ **<ref local ="b">**: it will check in local only.
- ➔ **<ref parent="b">**: it will check in parent Bean.

### Q-What is Nested Bean Factory?

➔Spring can manage the dependency between two beans if both beans are as part of one IOC container. But nested bean factory is the concept where spring can manage beans from two different IOC containers.

➔ Actually Nested BeanFactory means we can create one IOC container inside another IOC container.

➔ An IOC container going to inject into another IOC container called **parent IOC** container, and another one called as Child IOC Container.



## How to Create?

➔First we have to configure parent class into the spring bean configuration file and give name as parentBean.xml.

➔Then Configure child class into the spring bean configuration file and give name as childBean.xml.

➔After creating both configuration file, now create parentBeanFactory, and then create childBeanFactory and pass parentBeanFactory reference to the ChildBeanFactory as argument.

```
BeanFactory parentFactory=new XmlBeanFactory(new ClassPathResource("com/nbf/common/parentBean.xml"));
BeanFactory childFactory=new XmlBeanFactory(new ClassPathResource("com/nbf/common/childBean.xml"),parentFactory);
```

## Collection Merging

➔**If I have a parent bean it contains attribute of collection type injected with collection of values**, I wanted a one more bean acting as a child bean with same attribute as a collection type into which **I wanted to inject the collection of values in addition to the parent collection values completely in my child bean then we have to go for collection merging**.

If we want to work with collection merging there are three rules we have to follow.

➔Collection merging we are using **when we use bean inheritance**.

- ⇨ If there is collection property type in parent bean then only we can merge with child bean collection property type.

➔ If Both Parent and Child Property has **same collection type with same name** then only collection merging will be work.

➔ If Both Parent and Child Property has **same collection generic type** then only collection merging will be work.

```
public class Test {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/cm/common/application-context.xml"));
		Course course=factory.getBean("btech1styrece",Course.class);
		System.out.println(course);
	}
}
```

```
public class Course {
	private List<String> subjects;

	public void setSubjects(List<String> subjects) {
		this.subjects = subjects;
	}
	@Override
	public String toString() {
		return "Course [subjects=" + subjects + "]";
	}
}
```

```
<bean id="btech1stYrcsc" class="com.cm.beans.Course" >
	<property name="subjects">
		<list value-type="java.lang.String" >
			<value>C</value>
			<value>D.M.S</value>
		</list>
	</property>
</bean>
<bean id="btech1styrece" class="com.cm.beans.Course" parent="btech1stYrcsc">
	<property name="subjects" >
		<list value-type="java.lang.String" merge="true">
			<value>RT</value>
			<value>E.D.C</value>
		</list>
	</property>
</bean>
```

## IDRef

To make our classes loosely coupled there are two ways

- ➔ Dependency pull
- ➔Dependency injection

➔Dependency injection makes our classes loosely coupled, but there are some limitations are available with the dependency injection.

### Limitations of Dependency Injection

- ➔ If we are using dependency injection it is good about managing the dependency between those components which are having the static references with each other.
- ➔ If we have some dynamic dependency to be manage means the class whom I talk to will be decided by runtime instead of development time. Then dependency injection will not work.
- ➔ We cannot use prototype scope bean in to singleton scope, because that prototype class behave like singleton only.

➔By using dependency injection we can inject the other bean in to the target bean but if I don't want to use dependency injection, rather my target class pull corresponding object from the spring bean configuration file then I should go for dependency pull.

➔We can make our target class to pull the correspond object by two ways

- ⇨ By declared one attribute in target class and inject a particular bean id to the target attribute.
- ⇨ By using IDREF attribute.

**When we go for IDREF?**

➔ idref is used for pointing to name of the bean

➔<idref bean="suzukiEngine"> is exactly the same as just the string value.

➔If our requirement is inject a string value which is interlink with a bean id then we should go for idref.

```
<bean id="car" class="com.idref.beans.Car">
        <constructor-arg value="yamahaEngine"/>
</bean>
<bean id="yamahaEngine" class="com.idref.beans.YamahaEngineImpl"/>
<bean id="suzukiEngine" class="com.idref.beans.SuzukiEngineImpl"/>
```

```
<bean id="car" class="com.idref.beans.Car">
        <constructor-arg>
                <idref bean="suzukiEngine"/>
        </constructor-arg>
</bean>
<bean id="yamahaEngine" class="com.idref.beans.YamahaEngineImpl"/>
<bean id="suzukiEngine" class="com.idref.beans.SuzukiEngineImpl"/>
```

➔ what are the problems are there If we are using **value="yamahaEngine"** instead of **<idref bean=" yamahaEngine"/>**

Example:

⇨ Our requirement is develop the logic which read the String value (id of a bean) from configuration and using that value pull the object from IOC container.

⇨ **If a some has newly joined in our project development team**, that person doesn't know about bean configuration which has already configured by another team member. And if that person incorrectly change the id of bean as per his requirement and he has also **renamed bean id (ref=" yamahaEngine")** which has referred to that bean but **he did not change (value=" yamahaEngine")** which has interlinked with that bean .**because no one don't know the logic which has written for read the value (id type) from configuration and using that value pull the object from IOC container.**

**Problems:**

➔We have not given any right perceptions configuration. (Not understandable by other person), Configuration will become inconsistent stage (no clarity around the value we are trying to pass).

➔Not only have we, IOC container also not understood if we are using the value attributed .because IOC treat it as value only. But internally it link with id of a bean.

➔IOC container will create the object but when we call the method by using that reference then Runtime Exception will occur because of incorrect configuration.

**To solve this problem we should use IDREF.**

```
public interface IEngine {
        int startup();
}
```

```
public class YamahaEngineImpl implements IEngine{
        @Override
        public int startup() {
                return 1 ;
        }
}
```

```
public class SuzukiEngineImpl implements IEngine {
        @Override
        public int startup() {
                return 1;
        }
}
```

```
public class Car {
        private String beanId;
        public Car(String beanId) {
                this.beanId=beanId;
        }
        public void drive(){
                int status=0;
                IEngine engine=null;
                BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/idref/common/application-context.xml"));
                engine=factory.getBean(beanId,IEngine.class);
                status=engine.startup();
                if (status==1) {
                        System.out.println(beanId+"\t Engine Started");
                }else{
                        System.out.println("Engine start failed");
                }
        }
}
```

```
<beans>
        <bean id="car" class="com.idref.beans.Car">
                <constructor-arg>
                        <idref bean="suzukiEngine"/>
                </constructor-arg>
        </bean>
        <bean id="yamahaEngine" class="com.idref.beans.YamahaEngineImpl"/>
        <bean id="suzukiEngine" class="com.idref.beans.SuzukiEngineImpl"/>
</beans>
```

# Bean Scope

**Singleton Design pattern:**

**If a class allow to creating only one object of a class and the same object is being used by all the classes within (Class Loader) scope of an application then that class is called as singleton class.**

There are several reasons available, why we are creating singleton class.

➔In some cases an object or a configuration will be common to whole application.

➔If everyone going to create the object of common thing then it is 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 throughout the application

Ex:

There are some use cases where we are using Singleton class

- **If a class doesn't have any state (attribute), then object my class is not representing any state and the methods of class are using parameter of that methods and performing the operation, the outcome of these methods will not affect when we are calling any of the object of our class because all the object are empty, so multiple objects are not recommended in such case we should go for Singleton. (Utility classes like cashing, currency convert, date format, Loan Calculator.....)**
- **If a class have read only attribute (final) and member method ,in this case all the object will carry same state(read only), if we call the method in any of the object then all the object will represent the same state , in such case multiple objects are not recommended go for singleton class.**

➔In our application if we are using any master data (fixed data) like city, state, country in HTML pages, then it is not recommended to write in every page, rather we need to store these master data's in database and retrieve those from database when it required.

➔But HTML will not able to give dynamic response in this case we have to use JSP page, and write the logic to retrieve the cities, states, countries from database using Database logic.

**Use-cases:**

➔There is a JSP page available in my application and my JSP page have dropdown list, which indicates cities, states, country.

How to add the cities, states, and country into the JSP.

```
<select name="cities">
    <option value="hyd">Hyderabad</option>
    <option value="chn">Channai</option>
    <option value="rkl">Rourkela</option>
</select>
```

## Problems with hardcode source code

➔If we write the above procedure to add the data into the dropdown list then it will be hardcoded in our application.
➔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 Data base and my JSP page want the data rendered into the dropdown list, so how we can inject that data to the dropdown list.

**Second way is write scriptlet into the JSP page, by writing the scriptlet we can load data into the Dropdown list but there are bunch of problems.**

➔ **JSP is the view module so we cannot mix business logic with view logic.**
➔ If there is change into the business logic may impact our view logic also.
➔ To managing such kind of code into the JSP page with is fall into management problems, maintenance problems.
➔ **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.**
➔ 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.
➔So we should not write the java code into the JSP page.

## Solve:

➔If we cannot 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.
➔ 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 cannot create 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 throughout the application, upto the application die the data should be available.
➔ Servlet easily shared the data from servlet to JSP page using the different scopes.
➔ **Actually my jsp page want the data means first one of the servlet has to execute and send the data into the jsp page. Because my jsp don't want to write the business logic 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 dropdown list with data.

➔So there are two approaches available we add data into the list.

- **Properties file**

  **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.

  ➔But into the property file we can not specify the relationship between the data.

- **Database**

  ⇨ Actually there are two types of tables present into the database in application prospective.

### 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.

### Operational Tables:

➔Operational tables are general table which is field by end user.

➔These tables are changeable, it's up to the requirement.

```
20  protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
21      try {
22          Class.forName("oracle.jdbc.driver.OracleDriver");
23          Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
24          Statement st=con.createStatement();
25          ResultSet rs=st.executeQuery("select cities from tbl_master");
26
27
28              List<String>cities =new ArrayList<String>();
29
30          while(rs.next()){
31              cities.add(rs.getString(1));
32          }
33          request.setAttribute("cities",cities);
34          request.getRequestDispatcher("/register.jsp").forward(request, response);
35      } catch (ClassNotFoundException|SQLException e) {
36          e.printStackTrace();
37      }
38  }
```

register.jsp

```
12      <select name="cities">
13          <%
14          List<String>cities=(List)request.getAttribute("cities");
15              for(String city: cities){
16                  out.println("<option value='"+city+"'>"+city+"</option>");
17              }
18          %>
19      </select>
```

**But still problem is there**

➔There are multiple users can access my JSP page each and every request my servlet has to call and pre-populate the data into drop down list.

➔ Each and every request my 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 Database and fetching the same data and returning it. If data is same then why we are fetching data from database every time, it is waste of time and we are killing the application performance, scalability problems also are there.

**Approach-1**

```
@WebServlet("/city")
public class RegisterServlet extends HttpServlet {
	private Connection con=null;
	private Statement stmt=null;
	private ResultSet rs=null;
	private List<String> cities;
	protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		try {
			if (cities==null) {
				Class.forName("oracle.jdbc.driver.OracleDriver");
				con=DriverManager.getConnection("jdbc:oracle:thin:@localHost:1521:xe","system","manager");
				stmt=con.createStatement();
				rs=stmt.executeQuery("select* from tbl_master");
				System.out.println("go to data base");
				cities=new ArrayList<String>();
				while(rs.next()){
					cities.add(rs.getString(1));
				}
			}
			request.setAttribute("city", cities);
			request.getRequestDispatcher("register.jsp").forward(request, response);;
		} catch (ClassNotFoundException | SQLException e) {
			e.printStackTrace();
		}
	}

}
```

**Approach-2**

```
17 public class RegisterServlet extends HttpServlet {
18
19    private List<String>cities;
20    public void init(){
21       try {
22          cities=new ArrayList<String>();
23          Class.forName("oracle.jdbc.driver.OracleDriver");
24          Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
25          Statement st=con.createStatement();
26          ResultSet rs=st.executeQuery("select cities from tbl_master");
27
28          while(rs.next()){
29             cities.add(rs.getString(1));
30          }
31       } catch (ClassNotFoundException|SQLException e) {
32          e.printStackTrace();
33       }
34    }
35    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
36
37       request.setAttribute("cities",cities);
38       request.getRequestDispatcher("/register.jsp").forward(request, response);
39    }
40 }
```



➔ So we should go to the database 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

database and then 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.

➔ It's seem to be good but there also have a problem, actually session object shared within the JEE classes but apart from that we cannot 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 dropdown list.

➔ Because of the above problems Cache came into feature.

## Caching Design pattern

➔ Cache is used for storing the data in it. So we are avoiding the round trips to visiting the database every time and storing the data into the cache.

➔ Cache improves 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 share the data among the application then use cache.

➔ Most of the time cache must be singleton only

➔ Because data remain same for every request then there is no need to create 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.

➔ A cache contains other member methods also which going to shared the state of the object in it.

➔ Cache can has multiple methods to add, retrieve and to check the data. Because in future if cache technology will change the we can easily able to change so we are not using map object directly.

### Approach-3

```
6  public class Cache {
7     private static Cache instance;
8     private Map<String,Object> dataMap;
9     private Cache(){
10       dataMap=new HashMap<String,Object>();
11    }
12    public static synchronized Cache getInstance(){
13       if (instance==null) {
14          instance=new Cache();
15       }
16       return instance;
17    }
18    public void put(String key,Object value){;
19       dataMap.put(key, value);
20    }
21    public Object get(String key){
22       return dataMap.get(key);
23    }
24    public boolean containsKey(String key){
25       return dataMap.containsKey(key);
26    }
27 }
```

```
19 private List<String>cities;
20 protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
21    try {
22       Cache cache=Cache.getInstance();
23       cities=new ArrayList<String>();
24       if(cache.containsKey("cities")==false){
25          Class.forName("oracle.jdbc.driver.OracleDriver");
26          Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
27          Statement st=con.createStatement();
28          ResultSet rs=st.executeQuery("select cities from tbl_master");
29          while(rs.next()){
30             cities.add(rs.getString(1));
31          }
32
33          cache.put("cities", cities);
34       }
35       else{
36          cities=(List)cache.get("cities");
37       }
38       request.setAttribute("cities",cities);
39       request.getRequestDispatcher("/register.jsp").forward(request, response);
40    } catch (ClassNotFoundException|SQLException e) {
41       e.printStackTrace();
42    }
```

**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.

➔ Because it may contains duplicate data also or cache may fail to decide which data as other class asking for.

➔ We can't justify a particular data into the Map, because 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.

- 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 creating 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**. Other classes can create as many numbers of objects for the corresponding class, so to create multiple objects for using the same method it waste of JVM memory.

➔Another thing is if we want to extend functionality of methods we can't override static methods.

➔And it will kill the performance of the application, because of that we have to make our class as singleton only

Approach-4

```
6  public class Cache {
7     private static Cache instance;
8     private Map<String,Object> dataMap;
9     private Cache(){
10       dataMap=new HashMap<String,Object>();
11    }
12    public static synchronized Cache getInstance(){
13       if (instance==null) {
14          instance=new Cache();
15       }
16       return instance;
17    }
18    public synchronized void put(String key,Object value){;
19       dataMap.put(key, value);
20    }
21    public synchronized Object get(String key){
22       return dataMap.get(key);
23    }
24    public synchronized boolean containsKey(String key){
25       return dataMap.containsKey(key);
26    }
27 }
```

➔When a class read the map object there may be another class put or check the data in to map object so data inconsistency problem (race condition) may occur so we have to make every method as synchronized in our class.

## How to create Singleton class in optimized way:

➔**First of all constructor should be private.**

**Reason:**

Because we are restricting other classes to creating the object of a class. To create an object default constructor is mandatory if we make it as private then other people unable to create the object of the class.

➔**Take a static method**

**Reason:**

Because without creating object we can able to call the method of that class by using static method.

➔**Take a static Singleton class type variable**

**Reason:**

To store the object in this variable so that we can able to check whether the object is existed or not, we take static variable because we are using that in side static method.

```
public class SingletonDemo {
        private static SingletonDemo instance;
        private SingletonDemo() {
                //No Code
        }
        public static SingletonDemo getInstance(){
                if (instance==null) {
                        instance=new SingletonDemo();
                }
                return instance;
        }
}
```

The code that we are writing it will not work in multi threaded environment it will create problem.

## What is Multi Threading?

➔**To execute different different part an application simultaneously is the concept of multi threading.**

➔***A* logical independent part of execution of application is called as multi threading.**

➔If you have 5 multiple independent logical part are there then JVM can't know that those are exists within your application, **so programmer has to separate those as a Thread.**

## When we use multi Threading?

**For ex:**

Suppose our program contains 1000 lines of code and that contains 2 separate independent parts .then JVM will execute program after executing 1st part 2nd part will be executed.

If in 1st part first 50 lines of code there is an IO Operation then CPU will kept in Ideal until the IO operation has completed. Due to that performance problem will be occurred.

In this case we have to go for multi threading. Because when a thread executes 1st part of program then simultaneously 2nd part also executed.

## What is Thread?

**➔Execution of a logical part of a program linearly is known as Thread.**

There are two ways to create a Thread only in one way

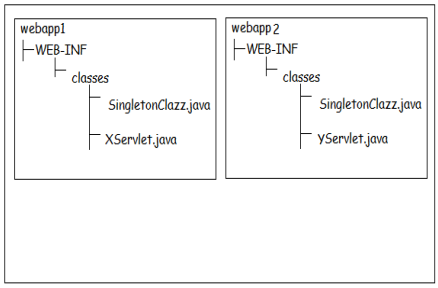
➔ By extending from Thread

**By implementing by Runnable (I) we are not creating a Thread we have to pass it to thread class constructor then only thread will be created.**

```
public class SingletonDemo {
        private static SingletonDemo instance;
        private SingletonDemo() {
                //No Code
        }
        public static synchronized SingletonDemo getInstance(){
                if (instance==null) {
                        instance=new SingletonDemo();
                }
                return instance;
        }
}
```

```
public class Test implements Runnable{
        public void run(){
                SingletonDemo s1=SingletonDemo.getInstance();
                System.out.println("Hashcode : "+s1.hashCode()+"\t "+Thread.currentThread().getName());
        }
        public static void main(String[] args)throws InterruptedException {

                Test r1=new Test();
                Test r2=new Test();
                Thread t1=new Thread(r1);
                Thread t2=new Thread(r2);
                t1.start();
                t2.start();
        }
}
```

**In Web application there may be a chance of multiple object will be created in JVM if there is multiple web application running on same container.**




Servlet Container

```
public class SingletonClazz {
        private static SingletonClazz instance;
        public static synchronized SingletonClazz getInstance(){
                if (instance==null) {
                        instance=new SingletonClazz();
                }
                return instance;
        }
}
```

```
public class YServlet extends HttpServlet {
        protected void service(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
                SingletonClazz s1=SingletonClazz.getInstance();
                response.getWriter().println("HashCode :"+s1.hashCode());
        }
}
```

```
public class SingletonClazz {
        private static SingletonClazz instance;
        public static synchronized SingletonClazz getInstance(){
                if (instance==null) {
                        instance=new SingletonClazz();
                }
                return instance;
        }
}
```

```
public class XServlet extends HttpServlet {
        protected void service(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
                SingletonClazz s1=SingletonClazz.getInstance();
                response.getWriter().println("HashCode :"+s1.hashCode());
        }
}
```

**In this case we have to create a executable Jar file and set it to environment variable.**

**MANIFEST.MF**

```
Manifest-Version: 1.0
Main-Class: A
```

**How to create Executable Jar file**

**Step-1.Create MANIFEST.MF file**

**Step-2.Compile Java Program ⇨ javac A.java**

**Step-3. Create ExecutableJar ⇨ jar cfm A.jar MANIFEST.MF A.class**

**Then it will give only one hashcode.**

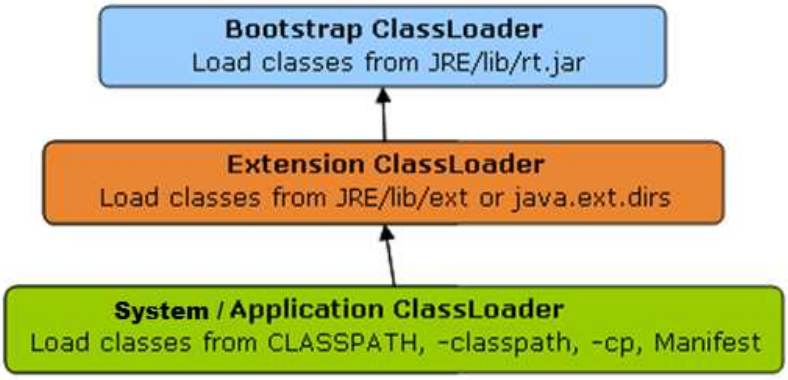
## Class Loader:

➔Class loader is a subsystem of JVM that is used to load class files from physical file location.

➔Actually JVM will call the Class loader to load the byte code into the JVM memory

➔Class loader will take the .class file byte codes 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




➔These class loaders not independent class loader they are related with each other within hierarchical way.
➔These three class loaders are loading the class from different different location.

## BootStrap ClassLoader(Native class loader):

➔BootStrap class Loader is the root class loader (Ultimate class loader / promotial class loader.
➔Actually BootStrap class loader is the parent class loader for remaining two class loader.

➔Bootstraps class Loader is native class loader that is being written in C language many of the times.

➔Actually platform to platform this BootStraps class loader will be changes. **Because it will load 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.

➔BootStrap class loader directory is "**Java_Home/jdk/bin**".

## Q-Why BootStrap is native class loader?

➔When JVM can execute anything if that byte code of class will be loaded.
➔Unless until byte code loaded in to JVM it will not execute.
➔There are predefined classes in JAVA libraries (JDK), and those classes are also being compiled to the byte code.
➔To start the echo system of java all the java libraries has to be loaded.
➔That means java has to run also **there should be someone who should be executing under the platform** who loads the binary library (byte code of the java classes (JDK)).

➔Here Bootstrap class loader is one who loads the Java because Bootstrap class loader is executable code (because it is **native compiled class loader**).
➔This is the class loader that loads other class loaders otherwise no class loader will work.

## Extension Class Loader(Java class loader):

➔It is the child class loader of Bootstrap class loader.
➔It's not a native class loader it is a java class loader by implementation from Bootstrap class loader.
➔It loads the classes from two different locations.
➔JRE/lib/ext
➔Java.ext.dirs (System variable / System Properties)

```
//How to set System Variable
-------------------------------------------------------------------------
public class Test {
        public static void main(String[]args){
                String i=System.getProperty("i");
                System.out.println(i);
        }
}
-------------------------------------------------------------------------
javac Test.java
java -D i=10 Test
```

## What will happen when a bytecode of a class loads to JVM?

➔When a java file compiled then a **.class file will be generated**.
➔When a class file is loaded by Class Loader **then Class class object will be created.**
Means **it reads the file and converts into class definition (METADATA of actual class).**
➔ METADATA contains **which attribute and which methods within this class.**
➔And within that class **two extra attribute will added** at the time of Class class objection.

- ⇨ **ClassLoader type attribute** (private ClassLoader classLoader)
- ⇨ **Class class type attribute** (private **static Class class**) refer to current object

Ex.

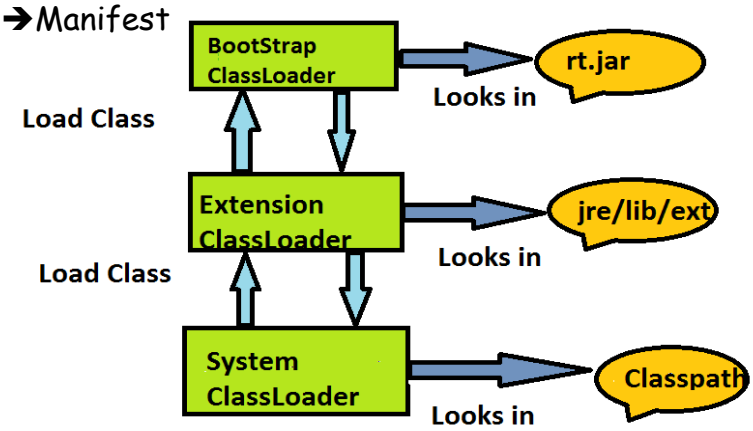
```
public class Test {
        public static void main(String[] args) {

        System.out.println(Test.class.getClassLoader().getClass().getName());
        }
}
```

**Class class Object will Contains**

```
Class class{
        private ClassLoader classLoader;
        private  Class class;

        Class Test=new Class();
        public ClassLoader getClassLoader(){
                return classLoader;
        }
}
```

## Application ClassLoader(java Class Loader)

➔➔It is the child class loader of Extension class loader.
➔It's not a native class loader it is a java class loader by implementation from Extension class loader.
➔It loads the classes from
- ➔-classpath
- ➔-cp
- ➔set CLASSPATH
- ➔Manifest



# Principles of classloaders:

- PRINCIPLE OF DELEGATION
- PRINCIPLE OF VISIBILITY
- 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.

## Principle of Delegation

➔ Actually byte code loaded by Application class loader but to check it is available throughout the cycle or not principle of delegation will be used first it will check with least class loader.
➔ First Application class loader will check into cache of the application class loader , if it is available it will not load same byte code into the JVM memory.
➔ If it not available then it will delegate to the extension class loader to check it is available or not, Then extension class loader will check into cache, if it is available it will load the same byte code neither it will delegate to the BootStrap class loader.
➔ Bootstrap class loader will check into the cache if it available it will load same, if it is not available, it will again delegates to the Extension class loader and then Extension classloader will delegates to the Application Class Loader. And Application class loader will load the byte code into the memory.

## Principle to visibility:

➔ Here parent-child relation available like inheritance.
➔ Child can access all the properties of the parent but parent cannot access child properties.
➔ Here parent class properties are visible to the child class loader but child class loader properties cannot be visible to the parent.

- ➔ Application Class loader can see the Extension Class loader and Bootstrap class loader loaded classes.
- ➔Extension class loader cannot see the Application class loader loaded byte code.
- ➔ Bootstrap class loader cannot see the Extension class loader and Application class loader byte code.
- ➔ But Extension class loader can see the Bootstrap class loader loaded byte code.

## Principle of Uniqueness:

➔ JVM memory is very expensive we cannot use anywhere.
➔ Uniqueness principle takes care of loading unique classes only. Mean no one class loader load duplicate byte code into the memory.
➔ Here principle to delegation plays vital role, it will always delegate and check byte code has been loaded or not.
➔ It will not allow duplication of byte code into the JVM memory.

# Examples

## How to Create an Executable Jar File

**C:/work>**

```
public class Game{
        public static void main(String[] args) {

                //Get the ClassLoader of the Game Class
                System.out.println("Game Executed By:"+Game.class.getClassLoader().getClass().getName());

                DropBox db=new DropBox();
                int coupon=db.getCoupon();
                System.out.println("Coupon Code:"+coupon);

                //Get the ClassLoader of the DropBox Class
                System.out.println("DropBox Executed By:"+DropBox.class.getClassLoader().getClass().getName());
        }
}
```

```
public class DropBox {
        public int getCoupon()
        {
                System.out.println("Get Coupon Method Executed");
                return 456;
        }
}
```

**manifest.mf**

```
Main-Class: Game
```

**Steps to create Manifest File**

C:/work>javac *.java

C:/work>jar -cvfm Game.jar manifest.mf *.class

**//If you want to Run through System/Application ClassLoader**

C:/work>java -cp Game.jar Game (System ClassLoader)
(or)
C:/work>java -jar Game.jar (System ClassLoader)

**//If you want to Run through Extension ClassLoader**

C:/work>java -Djava.ext.dirs=C:\work Game (Extension ClassLoader)

Put the Game.jar in JRE_Home\lib\ext
|— Game.jar

C:/work>java Game (Extension ClassLoader)

# //Create an Executable jar file with dependant jar

**C:/work>**

```
public class Game{
        public static void main(String[] args) {

                //Get the ClassLoader of the Game Class
                System.out.println("Game Executed By:"+Game.class.getClassLoader().getClass().getName());

                DropBox db=new DropBox();
                int coupon=db.getCoupon();
                System.out.println("Coupon Code:"+coupon);

                //Get the ClassLoader of the DropBox Class
                System.out.println("DropBox Executed By:"+DropBox.class.getClassLoader().getClass().getName());
        }
}
```

```
public class DropBox {
        public int getCoupon()
        {
                System.out.println("Get Coupon Method Executed");
                return 456;
        }
}
```

**manifest.mf**

```
Main-Class: Game
Class-Path: ./lib/DropBox.jar   (Press Enter)
```

**Steps to create Manifest File**

```
C:/work>javac *.java
C:/work>jar -cvf DropBox.jar DropBox.class
C:/work>jar -cvfm Game.jar manifest.mf  Game.class
```

**//Place the DropBox jar in lib folder**

```
C:/work>
    |
    |---lib
    |     |
    |     |--DropBox.jar
```

**//If you want to Run through System/Application ClassLoader**

```
C:/work>java -cp Game.jar Game        (System ClassLoader)
            (or)
C:/work>java -jar Game.jar            (System ClassLoader)
```

**//If you want to Run through Extension ClassLoader**

```
C:/work>java -Djava.ext.dirs=C:\work Game     (Extension ClassLoader)
```

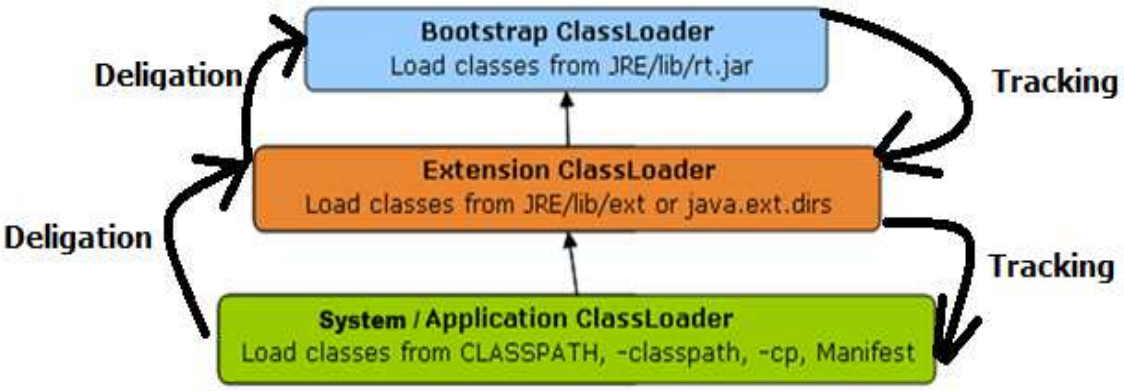
# Principle Delegation

```
public class Test{
        public static void main(String[] args) {
                //Get the ClassLoader of the Game Class
                System.out.println("Test Executed By:"+Test.class.getClassLoader().getClass().getName());
        }
}
```

```
C:/work>javac Test.java
C:/work>jar -cvf Test.jar Test.class

C:/work>set classpath=C:/work/Test
```

```
C:/work>java Test                    (System/Application ClassLoader)
```

```
Place it in
C:\Program Files\Java\jdk1.7.0_79\jre\lib\ext

C:\work>java Test                    (Extension ClassLoader)
```



# Principle Visibility

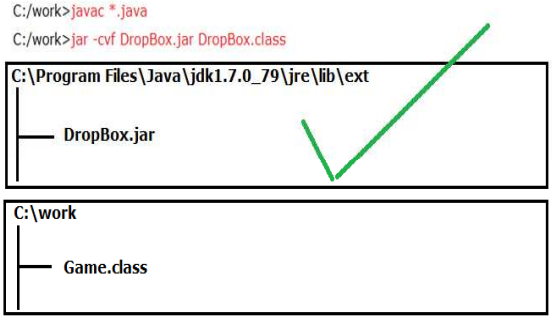
C:/work>

```
public class Game{
        public static void main(String[] args) {

                //Get the ClassLoader of the Game Class
                System.out.println("Game Executed By:"+Game.class.getClassLoader().getClass().getName());

                DropBox db=new DropBox();
                int coupon=db.getCoupon();
                System.out.println("Coupon Code:"+coupon);

                //Get the ClassLoader of the DropBox Class
                System.out.println("DropBox Executed By:"+DropBox.class.getClassLoader().getClass().getName());
        }
}
```

```
public class DropBox {
        public int getCoupon()
        {
                System.out.println("Get Coupon Method Executed");
                return 456;
        }
}
```

C:/work>javac *.java

C:/work>jar -cvf DropBox.jar DropBox.class





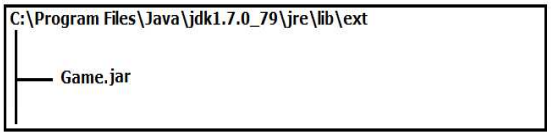


C:work>java Game (System/Application ClassLoader)

Visibile

C:work>javac *.java

C:work>jar -cvf Game.jar Game.class





C:work>java Game (Extension ClassLoader)

RE:ClassNotFoundException

Not Visible

# Principle of Uniqueness

```
public class Game{
	public static void main(String[] args)throws Exception {

		//Get the ClassLoader of the Game Class
		System.out.println("Game Executed By:"+Game.class.getClassLoader().getClass().getName());

		DropBox db=new DropBox();
		int coupon=db.getCoupon();
		System.out.println("Coupon Code:"+coupon);

		//Get the ClassLoader of the DropBox Class
		System.out.println("DropBox Executed By:"+DropBox.class.getClassLoader().getClass().getName());

		Class c=Class.forName("DropBox",false, Game.class.getClassLoader()[redacted]);
		System.out.println("Again DropBox Executed By:"+c.getClassLoader().getClass().getName());

	}
}
```

```
public class DropBox {
	public int getCoupon()
	{
		System.out.println("Get Coupon Method Executed");
		return 456;
	}
}
```

**C:/work>javac *.java**

**C:/work>jar -cvf DropBox.jar DropBox.class**

**//Place DropBox.jar in C:\Program Files\Java\jdk1.7.0_79\jre\lib\ext**

**DropBox.jar**

**C:/work>java Game**

**OutPut:**

```
C:\work>java Game
Game Executed By:sun.misc.Launcher$AppClassLoader
DropBox Executed By:sun.misc.Launcher$ExtClassLoader
Get Coupon Method Executed
Coupon Code:456
Again DropBox Executed By:sun.misc.Launcher$ExtClassLoader

C:\work>
```

## ClassLoader Name Space

➔When a class has been loaded by a class loader , then the class will be identified by the JVM memory by (ClassLoader.PackaGename.ClassName);

Ex: AppClassLoader.com.app.Game

➔But when we are importing that class we have to write (PackageName.ClassName) because a class should be loaded by only one class Loader in the hierarchy so we no need to write class loader name.

### What is Difference between .equals() , == ,hashcode()

**➔ Whether two objects are pointing to the same memory location or not we can find by using ==.**
**➔Whether two object contents are same or not we can find by using hashCode().**
**➔Whether two object belongs to same class and the contents are same or not you can check using equals()**

➔HashCode is a number that is unique no for every object of a class depends on the content of that object.
➔ The equals() will not compare the content of the String rather it computes the hash value for both the String and checks whether the hash value that is being computed is same or not.

- ⇨ **Object class hashCode() method always gives random numbers.**
- ⇨ So if we override equals method its recommended to override hashCode() also.

**Contract between equals() and hashCode()**
➔If .equals () returns true then hashCode of that objects must be same.
➔If .equals () returns false then hashCode may not be different.

## Phases of Class Loading:

➔Loading
➔Linking
- ⇨ Verifying
- ⇨ Preparing
- ⇨ Resolving

➔Initializing

- ❖ **Loading**

➔JVM call Class loader to load the .class file into the JVM memory. While loading the byte code of the class, it follows principles of class loaders.

➔Once loading has been completed by one of Class Loader, it will send to the linking phase.

- **Linking**

Linking phase has classified into three parts.

- ✓ **Verifying:**
  - ⇨ It will verify the byte code compatibility means generated byte code 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, and 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 class loader 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

➔Implicit Class loading

```java
public class Receiver{
    public double tuneUp(){
        System.out.println("Tunned Up");
        return 92.7;
    }
}
```

```java
public class Radio{
  public static void main(String[] args){
    Receiver receiver=new Receiver();
    receiver.tuneUp();
  }
}
```

```
C:\work\class>javac Radio.java

C:\work\class>del Receiver.class

C:\work\class>java Radio
Exception in thread "main" java.lang.NoClassDefFoundError: Receiver
        at Radio.main(Radio.java:3)
Caused by: java.lang.ClassNotFoundException: Receiver
        at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
        ... 1 more

C:\work\class>
```

## ➔ Explicit Class loading:

```
1  public class Receiver{
2      public double tuneUp(){
3          System.out.println("Tunned Up");
4          return 92.7;
5      }
6  }
7
```

```
1  public class Radio{
2      public static void main(String[] args)throws Exception{
3          Receiver receiver=(Receiver)Class.forName("Receiver").newInstance();
4          receiver.tuneUp();
5      }
6  }
7
```

```
C:\work\class>javac Radio.java

C:\work\class>del Receiver.class

C:\work\class> java Radio
Exception in thread "main" java.lang.ClassNotFoundException: Receiver
        at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:191)
        at Radio.main(Radio.java:3)

C:\work\class>
```

| ClassNotFoundException | NoClassDefFoundError |
|---|---|
| ➔When we are creating Object of Concrete class By using newInstance() method,If corresponding .class file is not there in location then we will get RE: **ClassNotFoundException** | ➔When we are creating Object of Concrete class By using new Operator ,If corresponding .class file is not there in location then we will get CE:NoClassDefFoundError |
| ➔It is checked Exception | ➔It is Unchecked Exception |

- **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: public class Receiver{
        static{
                System.out.println("static Block Executed...");
        }
    }
   public class Radio{
        public  static void main(String[]args){
                Class.forName("Receiver",false,Radio.getClassLoader);
        }
    }
```

## When should we go for Customized Class Loader?

➔If we want to secure our byte code we have to encode our .class files, but JVM can't read encoding format from physical file location, we have to write logic for decoding the byte code, in such case we should go for Customized Class Loader.(Banking Sector)

➔ If .class files are available in some remote location, but JVM can read from class path file location only not from remote location, we have to write logic for read the data from remote location, in such case we should go for Customized Class Loader.

➔All server vendors using their own class loader to read the data from a specific location. (Tomcat-----webapps)

# Create Custom Class Loader

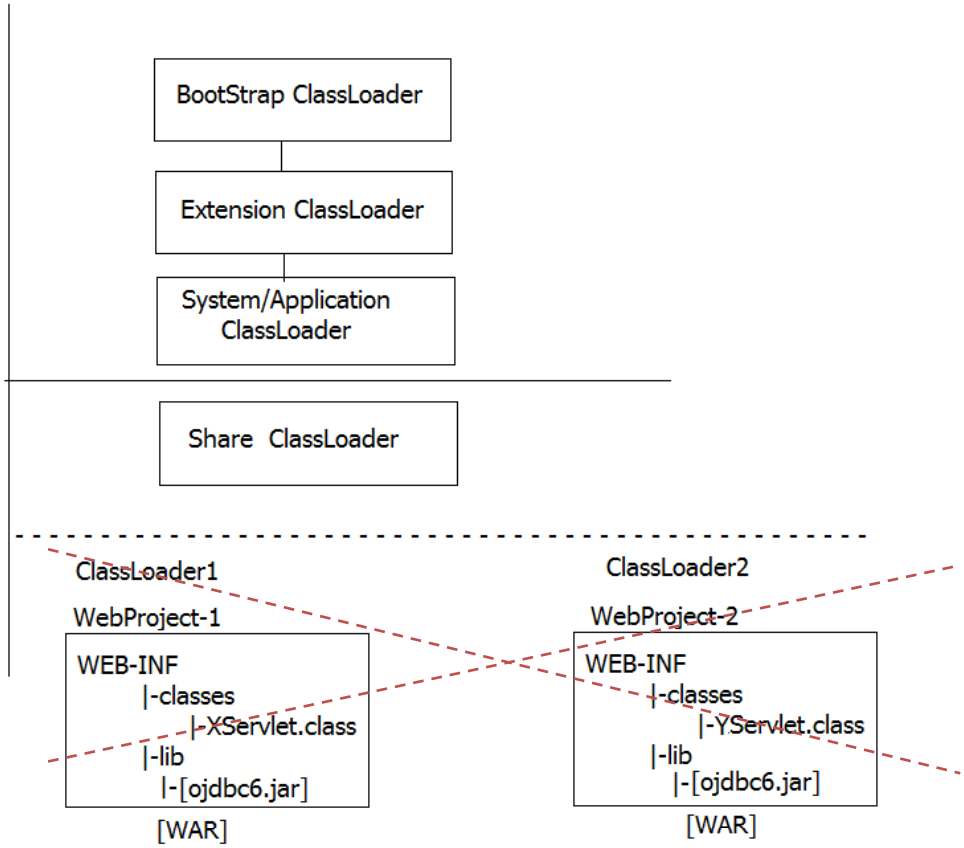
```
public class Toy {
        public String getName(){
                return "Iron Man";
        }
}
```

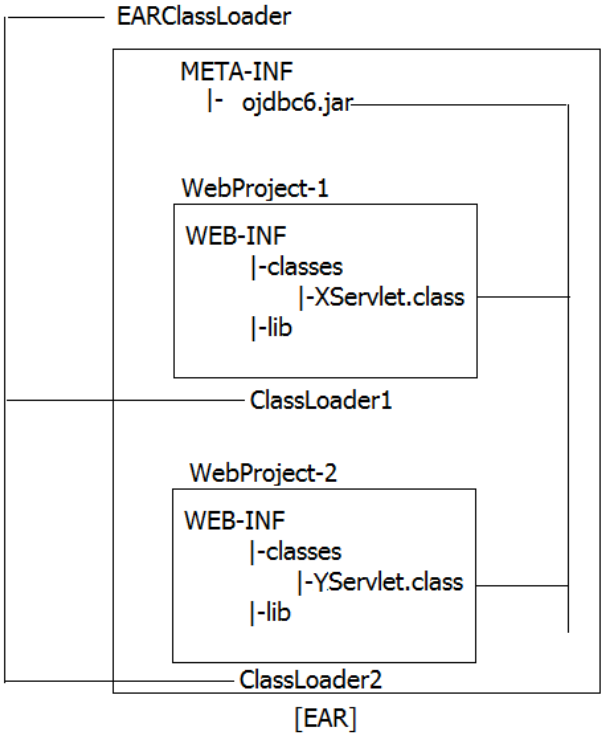
```
D:\work
        |- com
                |- ccl
                        |- Toy.class
```

```
public class Test {
        public static void main(String[] args) throws Exception {
                CustomClassLoader customClassLoader=new CustomClassLoader(Test.class.getClassLoader());
                Object toy=Class.forName("com.ccl.beans.Toy",true,customClassLoader).newInstance();
                System.out.println("Toy Class Executed By:"+toy.getClass().getClassLoader().getClass().getName());

        }
}
```

```
public class CustomClassLoader extends ClassLoader {

        //Constructor
        public CustomClassLoader(ClassLoader parent) {
        super(parent);
                System.out.println("Test Class Loaded By: "+parent.getClass().getName());
        }

        public Class<?> loadClass(String name) throws ClassNotFoundException{
                Map<String,Class<?>> classes=new HashMap<String,Class<?>>();
                //Checks From Cache
                if (classes.containsKey(name)==true) {
                        return classes.get(name);
                }
                try{
                        return findSystemClass(name);
                }
                catch(ClassNotFoundException e){
                        System.out.println(e);
                }
                byte[] byteCode=getClassDetails(name);
                        Class <?> c=defineClass(name,byteCode,0,byteCode.length);
                        resolveClass(c);
                        classes.put(name, c);
                        return c;
                }
        public byte[] getClassDetails(String name) throws ClassNotFoundException {
                        try{
                                //Logic for Custom Class Loader
                                String baseDir ="D:\\work\\";
                                String classFileName=name.replace(".", "\\").concat(".class");

                                File file=new File(baseDir+classFileName);
                                FileInputStream fis=new FileInputStream(file);
                                ByteArrayOutputStream bos = new ByteArrayOutputStream();

                                int i =0;
                                while((i=fis.read())!=-1){
                                        bos.write(i);
                                }
                                fis.close();
                                byte[] byteCode=bos.toByteArray();
                                bos.close();
                                return byteCode;
                        }catch (Exception e) {
                                throw new ClassNotFoundException();
                                }
        }
}
```

**➔All application sever vendors are provided their own customized class loaders to load classes from specific locations.**



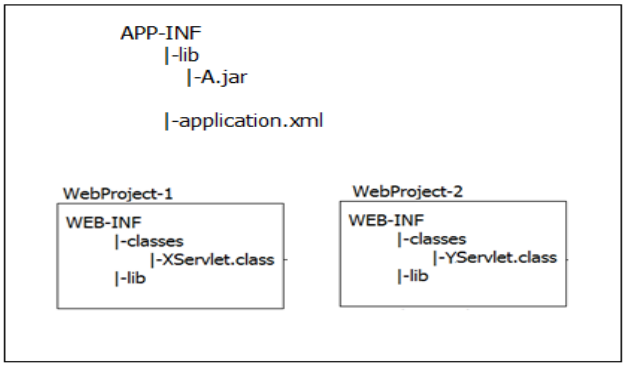



Example:1

**SingleTon Class**

```
1  public class A{
2     private static A instance;
3     public static synchronized A getInstance(){
4        if(instance==null){
5           instance=new A();
6        }
7        return instance;
8     }
9  }
```

**Folder Structure**




**webapps**

```
1 import java.io.*;
2 import javax.servlet.*;
3 import javax.servlet.http.*;
4 public class XServlet extends HttpServlet{
5    public void doGet(HttpServletRequest request,HttpServletResponse response ) throws  ServletException , IOException{
6       A a1=A.getInstance();
7       response.getWriter().println("hashCode :"+a1.hashCode());
8    }
9 }
```

**webapps2**

```
1 import java.io.*;
2 import javax.servlet.*;
3 import javax.servlet.http.*;
4 public class YServlet extends HttpServlet{
5    public void doGet(HttpServletRequest request,HttpServletResponse response ) throws  ServletException , IOException{
6       A a1=A.getInstance();
7       response.getWriter().println("hashCode :"+a1.hashCode());
8    }
9 }
```

**application.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
- <application version="1.4" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/j2ee">
    <description>Test EAR Example</description>
    <display-name>TestEAR Example</display-name>
  - <module>
    - <web>
        <web-uri>webapps.war</web-uri>
        <context-root>/x</context-root>
      </web>
    </module>
  - <module>
    - <web>
        <web-uri>webapp2.war</web-uri>
        <context-root>/y</context-root>
      </web>
    </module>
</application>
```

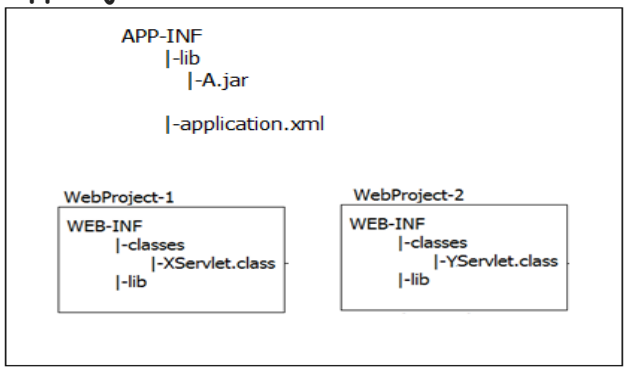
## How to create war file

➔ C:\Tomcat 7.0\webapps\WebProject1> **jar –cvf WebProject1.war * ;**

➔ C:\Tomcat 7.0\webapps\WebProject2> **jar –cvf WebProject2.war * ;**

➔ C:\Tomcat 7.0\webapps\AppProject> **jar –cvf AppProject.ear * ;**

**AppProject**

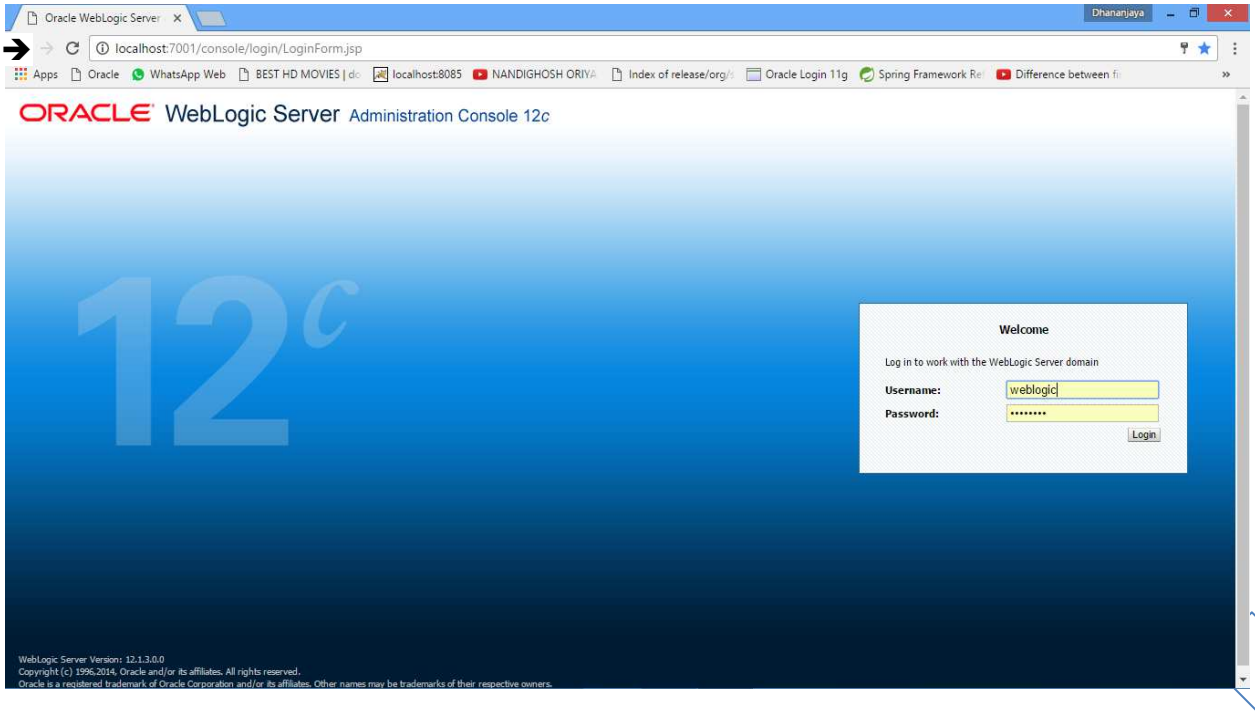



**How to install Weblogic server?**

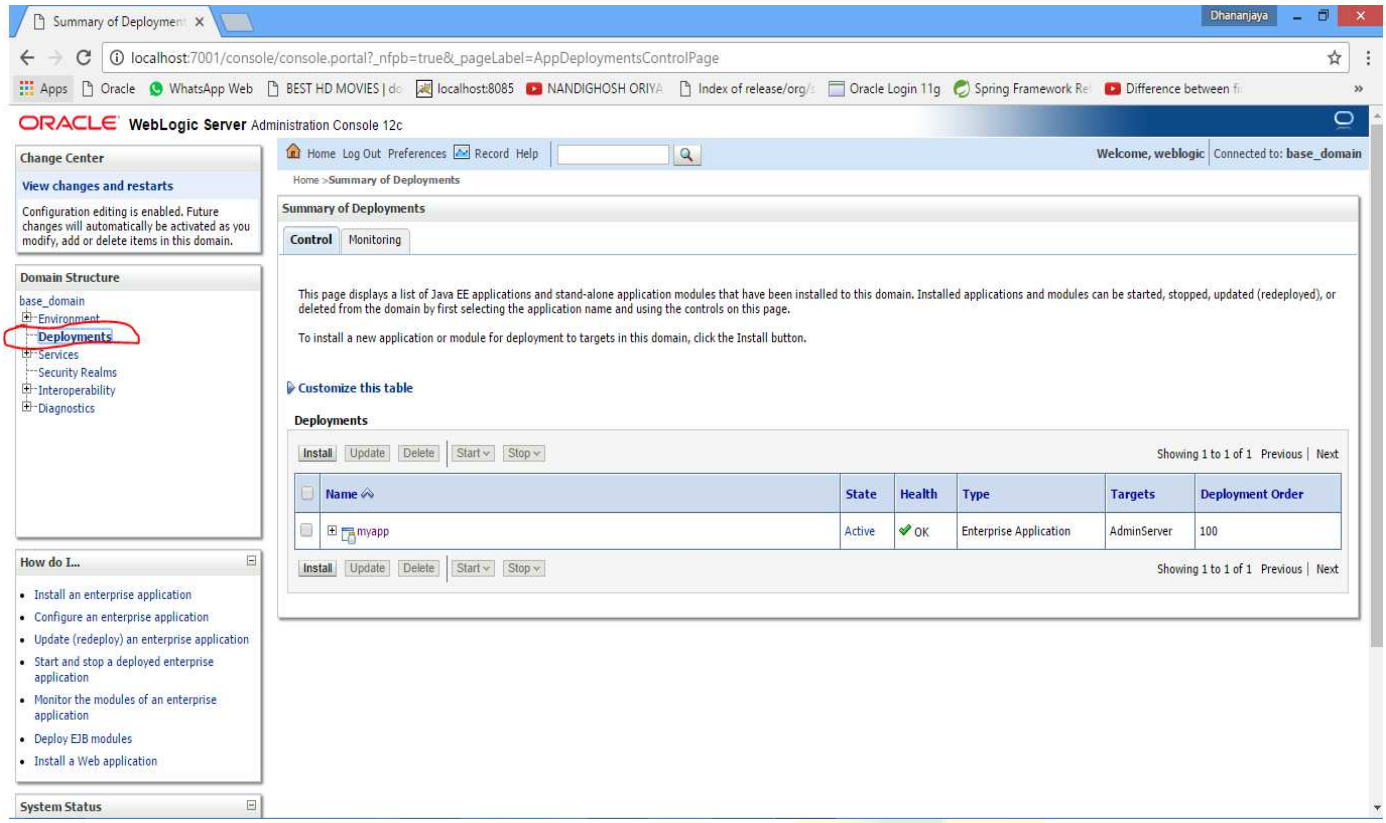
➔Install oepe-12.1.3-kepler-installer-win32_2.exe
➔Go to start ➔ Configuration Wizard➔create a domain
➔Go to ➔C:\Oracle\Middleware\Oracle_Home\user_projects\domains\base_domain\
➔Click on **startWebLogic.cmd** (to start the server)

```
<28 Nov, 2016 11:43:44 PM PST> <Notice> <Server> <BEA-002613> <Channel "Default[6]" is now listening on 0:0:0:0:0:0:0:
<28 Nov, 2016 11:43:44 PM PST> <Notice> <Server> <BEA-002613> <Channel "Default[2]" is now listening on fe80:0:0:0:d95
<28 Nov, 2016 11:43:44 PM PST> <Notice> <WebLogicServer> <BEA-000331> <Started the WebLogic Server Administration Serv
<28 Nov, 2016 11:43:44 PM PST> <Notice> <WebLogicServer> <BEA-000360> <The server started in RUNNING mode.>
<28 Nov, 2016 11:43:44 PM PST> <Notice> <WebLogicServer> <BEA-000365> <Server state changed to RUNNING.>
```

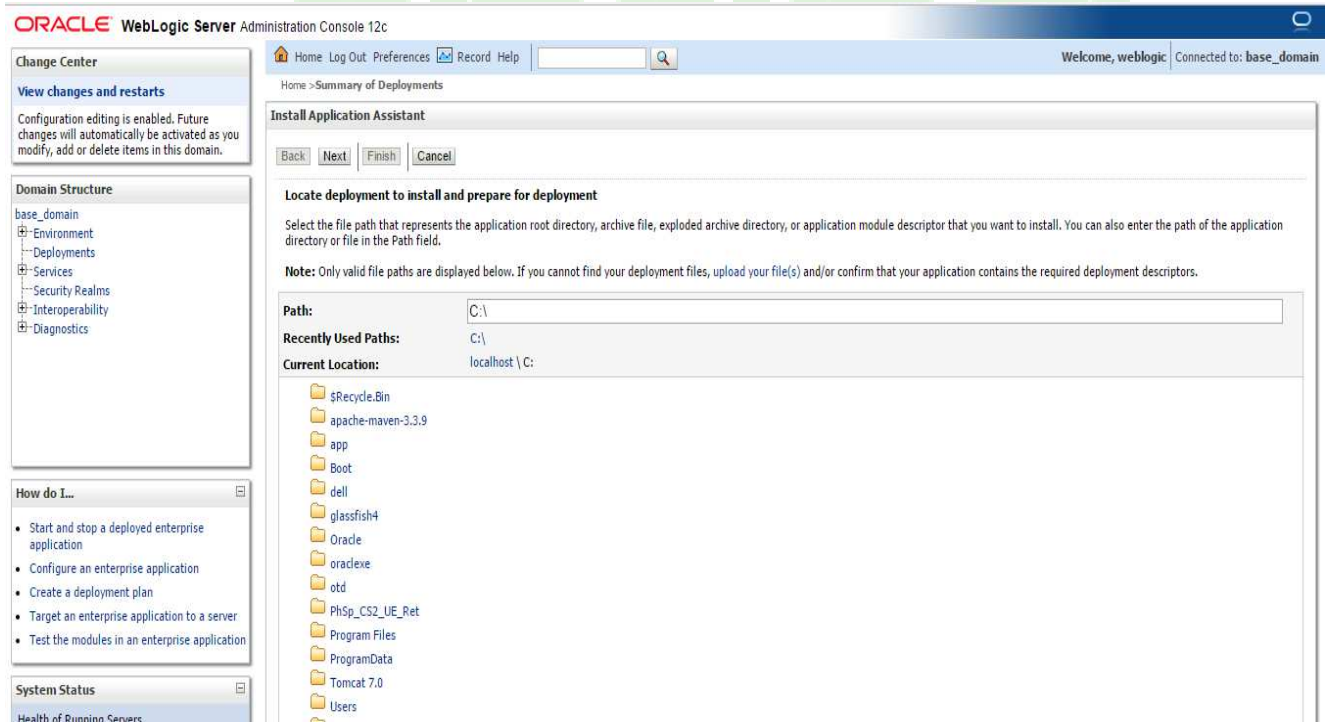
➔Open the browser ➔http://localhost:7001/console.
➔Enter user name and password.



➔Go to Deployments




➔Go to install




➔Select path of ear file

➔Finish

## Bean Scope

Whenever we define a class as a bean in spring bean configuration file only one bean definition object will be created in the IOC container because by default the scope of the bean is singleton, if you want multiple object for that bean definition then we can control the instantiation of a bean using scope of a bean.

In spring there are 4 types of scopes are available.

- SINGLETON
- PROTOTYPE
- REQUEST
- SESSION
- ~~GLOBAL SESSION~~ [port late application] deprecated and removed from spring 3.0.

### SINGLETON

➔ Actually in spring every bean by default scope is singleton.

➔ If we try to create multiple objects also we will get same object as part of the bean.

### PROTOTYPE

➔ Prototype is another bean scope which is used for creating multiple objects as part of the bean.

➔ IOC container will read the scope and depends on the scope it will create or return the object.

### When we should go for singleton or prototype

- If a class doesn't have any state (attribute), then object my class is not representing any state and the methods of class are using parameter of that methods and performing the operation, the outcome of these methods will not affect when we are calling any of the object of our class because all the object are empty, so multiple objects are not recommended in such case we should go for Singleton. (Utility classes like cashing, currency convert, date format, Loan Calculator.....)

- If a class have read only attribute (final) and member method ,in this case all the object will carry same state(read only), if we call the method in any of the object then all the object will represent the same state , in such case multiple objects are not recommended go for singleton class.

❖ When state of the object is there and it will sharable into the Member method. (Caching design pattern)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="a" class="com.bs.beans.A"/>
    <bean id="b" class="com.bs.beans.B" scope="prototype"/>
</beans>
```

```java
public static void main(String[] args) {
    BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bs/common/application-context.xml"));
    A a1=factory.getBean("a",A.class);
    A a2=factory.getBean("a",A.class);
    System.out.println("a1==a2: ? "+(a1==a2));  // true

    B b1=factory.getBean("b",B.class);
    B b2=factory.getBean("b",B.class);
    System.out.println("b1==b2: ? "+(b1==b2));  // false
```

O/P

```
A()
a1==a2: ? true
B()
B()
b1==b2: ? false
```

**P and C NameSpaces:**

➔ It is the shortcut procedure to use the property and constructor attributes into the bean.

➔ Because if there are multiple properties are available in 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 name spaces.

```java
public class Address {
    private String addressLine1;
    private String addressLine2;
    private String city;
    private String state;
    private int zip;
    private String country;

    public Address(String addressLine1, String addressLine2, String city, String state, int zip, String country) {
        super();
        this.addressLine1 = addressLine1;
        this.addressLine2 = addressLine2;
        this.city = city;
        this.state = state;
        this.zip = zip;
        this.country = country;
    }

    @Override
    public String toString() {
        return "Address [addressLine1=" + addressLine1 + ", addressLine2=" + addressLine2 + ", city=" + city
                + ", state=" + state + ", zip=" + zip + ", country=" + country + "]";
    }
}
```

```java
public class Person {
   private String name;
   private Address address;
   public void setName(String name) {
      this.name = name;
   }
   public void setAddress(Address address) {
      this.address = address;
   }
   @Override
   public String toString() {
      return "Person [name=" + name + ", address=" + address + "]";
   }
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:c="http://www.springframework.org/schema/c"
   xmlns:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd">

   <bean id="person" class="com.pcns.beans.Person"
      p:name="Dhananjaya Samanta Singhar"
      p:address-ref="address"
   />
   <bean id="address" class="com.pcns.beans.Address"
      c:addressLine1="505"
      c:addressLine2="Mytrivanam"
      c:city="Hyderabad"
      c:state="Telengana"
      c:zip="500038"
      c:country="India"
   />

</beans>
```

## Dependency-check:

➔ Most of the time we going to inject the values via setter or constructor and constructor inject is mandatory.

➔Means if a bean contains a constructor then we have to pass the values then only an object of bean will be created.

➔ But in case of 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 an additional configuration.

➔There are three modes available which make primitive and object to be mandatory.

**MODES**

➔ Simple (it is only for primitives)
➔ Object (it is only 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 their 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.

**Deprecated from 3.0 onwards instead if it @Required came in to picture**

```
<beans >
	<bean id="person" class="com.dc.beans.Person"  dependency-check="all">
		<property name="name" value="Dhananjaya"/>
		<property name="address" ref="address"/>
	</bean>

	<bean id="address" class="com.dc.beans.Address" dependency-check="simple">
	<property name="addressLine1" value="505"/>
	<property name="addressLine2" value="Mytrivanam"/>
	<property name="city" value="Hyderabad"/>
	<property name="state" value="Telengana"/>
	<property name="zip" value="500038"/>
	<property name="country" value="India"/>
	</bean>
</beans>
```

```
Exception in thread "main" org.springframework.beans.factory.BeanCreationException:
Caused by: org.springframework.beans.factory.UnsatisfiedDependencyException:
Unsatisfied dependency expressed through bean property 'country':
Set this property value or disable dependency checking for this bean.
```

## Static Factory Method Instantiation:

➔By default bean scope is singleton only but when there are some classes will not allow creating the object directly (abstract classes, Predefined Abstract classes).
➔If you want to create the object then these classes providing static factory method to create an object.

<bean id =“cal” class=“java.util.Calendar”/>

➔Here we cannot create the object of the calendar because calendar is an abstract class.
➔Actually calendar has one static factory method which going to return the Calendar class object (Implementation class Object i.e java.util.GregorianCalendar).
➔To get the object we have to factory-method attribute in side bean which going to return the object.

```
<bean id="calender" class="java.util.Calendar" factory-method="getInstance"/>
```

```
import java.util.Calendar;
public class Remainder {
        private String eventName;
        private Calendar eventDate;
        public void setEventName(String eventName) {
                this.eventName = eventName;
        }
        public void setEventDate(Calendar eventDate) {
                this.eventDate = eventDate;
        }
        @Override
        public String toString() {
                return "Remainder [eventName=" + eventName +
                        ", eventDate=" + eventDate + "]";
        }
}
```

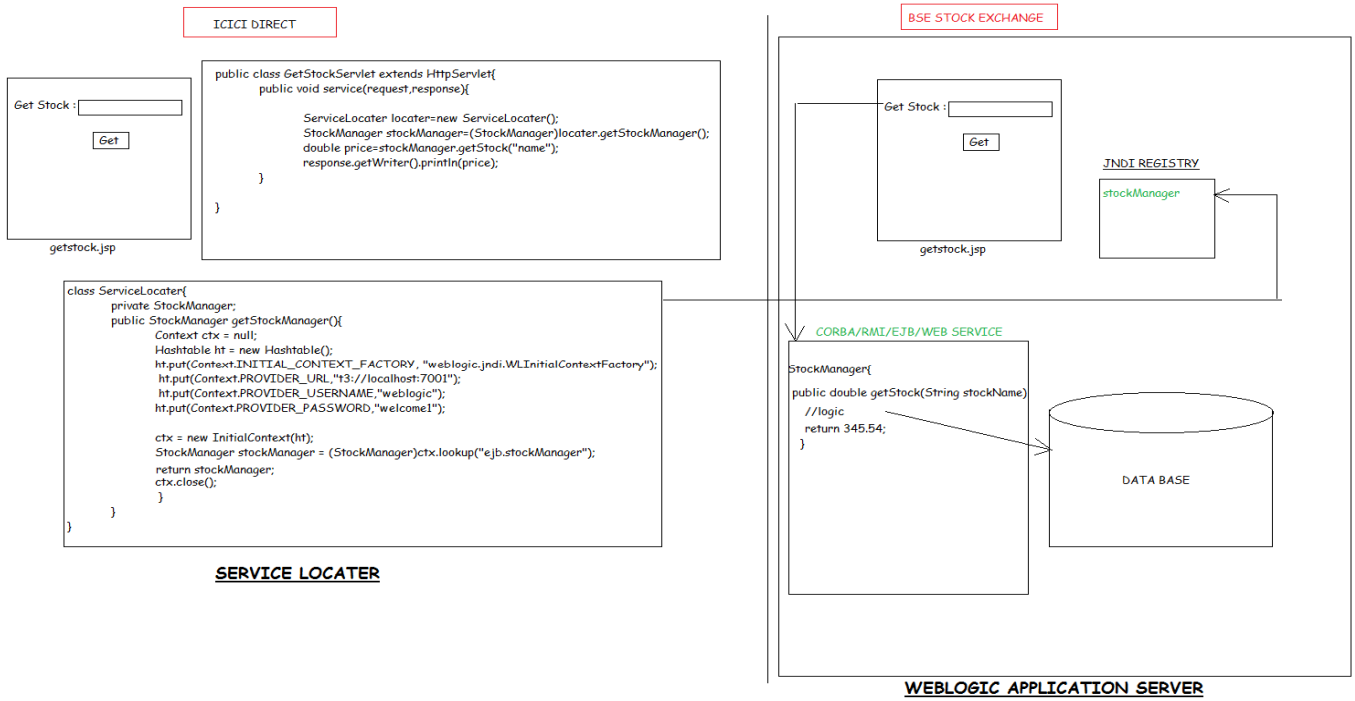
```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

        <bean id="remainder" class="com.sfm.beans.Remainder">
                <property name="eventName" value=""></property>
                <property name="eventDate" ref="calender"/>
        </bean>
        <bean id="calender" class="java.util.Calendar" factory-method="getInstance"/>
</beans>
```

```
public class Test {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new
        ClassPathResource("com/sfm/common/application-context.xml"));
                Remainder remainder=factory.getBean("remainder",Remainder.class);
                System.out.println(remainder);
        }
}
```

**Q-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 cannot create the object of abstract class, but we want the object then we can use static factory method instantiation strategy to get the object.

## Instance Factory Method Instantiation



➔BSE should not allow anyone to direct access to database because of Security Bridge.
➔BSE should not provide Database details.
➔BSE should not provide source code.
➔BSE should not provide class file.
➔Rather BSE has to expose object over the network for providing the service. So application should be build on CORBA |RMI | EJB |WEBSERVISES
➔And object should be in application container because any one can access easily.

**Why should we use service locator?**

➔BSE has to put the object in the JNDI registry because other partner can lookup the object and they can easily access the methods of StockManager class.
➔**If GetStockServlet writing the logic for getting the object of remote application** i.e. get StockManager object then it is seem to be good if only one class is writing the logic, but ICICI DIRECT application contains **multiple classes they want to talk to the BSE Stock Exchange then every class has to write the lookup logic for getting the object.**
➔Then the **first problem** is duplication of logic.
➔**Second problem** is if there is **change in technology (CORBA |RMI | EJB |WebService)** driven in provider side then again all the classes has to rewrite the lookup logic for getting the remote object from JNDI registry, because every technology specific lookup jars are different.
➔**Third problems** is if there is **change in application server (JBOSS to WEB Logic)** in provider side then again we have to rewrite whole logic for connecting to the other application server.
➔Fourth problem is **specific to the environment platform** if there change in environment **(Machine change)** then again we have to rewrite the logic for connecting to the machine (like Port no and other).

To overcome from these problems we should use a design pattern called Service Locator design pattern.

**Service Locater**
➔Service Locator is the class which will avoid all the above problems which are available in our application.
➔ Actually Service Locator class get the reference object from the remote location **by performing lookup and it will provide this locator reference service to the current application** because of that is it called as service Locator.
➔ Service Locator provides transference of the remote location. Actually my other classes don't know from where we are accessing the information, they think that the accessing class as part of our application only (abstraction).
➔ Service Locator also optimizes the performance of the application (by using Connection pooling).

***Instance factory method instantiation***

➔ It is available from spring 2.0 version.

➔It is the special kind of factory method instantiation.

➔ Actually IOC container use 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 ServiceLocator will get the object from the remote location and it give us to use in our application as normal object.

➔ But IOC container will create an object for a class which is available in the application, but here Service Locator class is going to locate the object which is available on the remote location.

➔ So IOC container cannot create the object for remote application classes. Because those class is not with the current application.

➔ Here Service Locator class have one method which is responsible for getting the object from the target class and method name is getStockManager().

➔ getStockManager() 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 let's see.

CONSUMER

```
public class Test {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/ifmi/common/application-context.xml"));
		ICICIGetStock getStock=factory.getBean("iciciGetStock",ICICIGetStock.class);
		getStock.getStockPrice("cipla");
	}
}
```

```
public class ICICIGetStock {
	private StockManager stockManager;

	public void setStockManager(StockManager stockManager) {
		this.stockManager = stockManager;
	}
	public void getStockPrice(String stockName){
		double price=stockManager.getPrice(stockName);
		System.out.println(price);
	}
}


public class ICICIServiceLocator {
	private static ICICIServiceLocator instance;
	private StockManager stockManager;

	public static synchronized ICICIServiceLocator getInstance(){
		if (instance==null) {
			instance=new ICICIServiceLocator();
		}
		return instance;
	}
	public StockManager getService(){
		//logic to look up in JNDI Registry
		stockManager=new BSEStockManagerImpl();
		return stockManager;

	}
}
```

PROVIDER

```
public interface StockManager {
	public double getPrice(String stockName);
}
```

```
public class BSEStockManagerImpl implements StockManager {

	@Override
	public double getPrice(String stockName) {
		double price=0.00;
		if(stockName==null || stockName.trim().length()==0){
			price=346.89;
		}
		else if (stockName.equals("cipla")) {
			price=678.45;
		}
		else{
			price=458.22;
		}
		return price;
	}
}
```

```
<beans>
	<bean id="iciciGetStock" class="com.ifmi.beans.ICICIGetStock">
		<property name="stockManager" ref="stockManager"/>
	</bean>

	<bean id="serviceLocator" class="com.ifmi.beans.ICICIServiceLocator" factory-method="getInstance"/>
	<bean id="stockManager" factory-bean="serviceLocator" factory-method="getService"/>

</beans>
```

➔Here IOC container will get the object of Service Locator class and it will call the instance factory lookup method of the Service Locator.

➔If a instance method of Service Locator 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.

```
public class Test {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/ifmi/common/application-context.xml"));
		ICICIGetStock getStock=factory.getBean("iciciGetStock",ICICIGetStock.class);
		getStock.getStockPrice("norway");
	}
}
```

```
public interface StockManager {
	public double getPrice(String stockName);
}
```

```
public class ICICIGetStock {
	private StockManager stockManager;

	public void setStockManager(StockManager stockManager) {
		this.stockManager = stockManager;
	}
	public void getStockPrice(String stockName){
		double price=stockManager.getPrice(stockName);
		System.out.println(price);
	}
}
```

```
public class BSEStockManagerImpl implements StockManager {

	@Override
	public double getPrice(String stockName) {
		double price=0.00;
		if(stockName==null || stockName.trim().length()==0){
			price=346.89;
		}
		else if (stockName.equals("cipla")) {
			price=678.45;
		}
		else{
			price=458.22;
		}
		return price;
	}
}
```

```
public class ICICIServiceLocator {
	private static ICICIServiceLocator instance;
	private StockManager stockManager;

	public static synchronized ICICIServiceLocator getInstance(){
		if (instance==null) {
			instance=new ICICIServiceLocator();
		}
		return instance;
	}
	public StockManager getService(String exchangeName){
		//logic to look up in JNDI Registry
		if (exchangeName.equals("BSEstockManager")) {
			stockManager=new BSEStockManagerImpl();
		}
		else if(exchangeName.equals("NSEstockManager")){
			stockManager=new NSEStockManagerImpl();
		}
		return stockManager;

	}
}
```

```
public class NSEStockManagerImpl implements StockManager {

	@Override
	public double getPrice(String stockName) {
		double price=0.00;
		if(stockName==null || stockName.trim().length()==0){
			price=6.89;
		}
		else if (stockName.equals("norway")) {
			price=8.45;
		}
		else{
			price=8.22;
		}
		return price;
	}
}
```

```
<beans >
	<bean id="iciciGetStock" class="com.ifmi.beans.ICICIGetStock">
		<property name="stockManager" ref="NSEstockManager"/>
	</bean>

	<bean id="serviceLocator" class="com.ifmi.beans.ICICIServiceLocator" factory-method="getInstance"/>
	<bean id="BSEstockManager" factory-bean="serviceLocator" factory-method="getService" >
		<constructor-arg value="BSEstockManager"/>
	</bean>

	<bean id="NSEstockManager" factory-bean="serviceLocator" factory-method="getService" >
		<constructor-arg value="NSEstockManager"/>
	</bean>

</beans>
```

## 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 constructor.

➔But some time it is not possible for IOC container to create the object by calling constructor then we can use the concept called factory bean

➔Spring has provided one interface called FactoryBean which contains three methods.

- ⇨ **public Object getObject(){}**
- ⇨ **public Class getObjectType(){}**
- ⇨ **public boolean isSingleton(){}**

```
public class Reminder {
	private String event;
	private Calendar calendar;

	public void setEvent(String event) {
		this.event = event;
	}
	public void setCalendar(Calendar calendar) {
		this.calendar = calendar;
	}

	public String getEvent() {
		return "Reminder [event=" + event + ", calendar=" + calendar + "]";
	}
}
```

```
public class CalendarFactoryBean implements FactoryBean {

	@Override
	public Class getObjectType() {
		System.out.println("getObjectType()");
		return Calendar.class;
	}
	@Override
	public Object getObject() throws Exception {
		System.out.println("getObject()");
		return Calendar.getInstance();
	}

	@Override
	public boolean isSingleton() {
		System.out.println("isSingleton()");
		return true;
	}
}
```

```
isSingleton()
getObject()
Reminder [event=john's Birthday, calendar=java.util.GregorianCalendar
calendar1==calendar2 : ?true
factoryBean1==factoryBean2 : ?true
```

```
<beans>

	<bean id="reminder" class="com.fb.beans.Reminder">
		<property name="event" value="john's Birthday"/>
		<property name="calendar" ref="calendar"/>
	</bean>

	<bean id="calendar" class="com.fb.beans.CalendarFactoryBean" scope="singleton"/>

</beans>
```

```
public class Test {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/fb/common/application-context.xml"));

		Reminder reminder1=factory.getBean("reminder",Reminder.class);
		System.out.println(reminder1.getEvent());


		Calendar calendar1=factory.getBean("calendar",Calendar.class);
		Calendar calendar2=factory.getBean("calendar",Calendar.class);

		System.out.println("calendar1==calendar2 : ?"+(calendar1==calendar2));


		CalendarFactoryBean factoryBean1=factory.getBean("&calendar",CalendarFactoryBean.class);
		CalendarFactoryBean factoryBean2=factory.getBean("&calendar",CalendarFactoryBean.class);

		System.out.println("factoryBean1==factoryBean2 : ?"+(factoryBean1==factoryBean2));
	}

}
```

OutPut

```
isSingleton()
getObject()
Reminder [event=john's Birthday, calendar=java.util.GregorianCalendar

calendar1==calendar2 : ?true

factoryBean1==factoryBean2 : ?true
```

```
<beans >

	<bean id="reminder" class="com.fb.beans.Reminder">
		<property name="event" value="john's Birthday"/>
		<property name="calendar" ref="calendar"/>
	</bean>

	<bean id="calendar" class="com.fb.beans.CalendarFactoryBean" scope="prototype"/>

</beans>
```

```
public class Test {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/fb/common/application-context.xml"));

		Reminder reminder1=factory.getBean("reminder",Reminder.class);
		System.out.println(reminder1.getEvent());


		Calendar calendar1=factory.getBean("calendar",Calendar.class);
		Calendar calendar2=factory.getBean("calendar",Calendar.class);

		System.out.println("calendar1==calendar2 : ?"+(calendar1==calendar2));


		CalendarFactoryBean factoryBean1=factory.getBean("&calendar",CalendarFactoryBean.class);
		CalendarFactoryBean factoryBean2=factory.getBean("&calendar",CalendarFactoryBean.class);

		System.out.println("factoryBean1==factoryBean2 : ?"+(factoryBean1==factoryBean2));
	}
}
```

OutPut

```
isSingleton()
getObject()
Reminder [event=john's Birthday, calendar=java.util.GregorianCalendar

isSingleton()
getObject()

isSingleton()
getObject()

calendar1==calendar2 : ?false


factoryBean1==factoryBean2 : ?false
```

➔IOC container will not able to create the object of Calendar class because Calendar class doesn't have default constructor.
➔But we know how to create the object of Calendar class so spring has provided one interface.
➔We have to implement that interface and override all the three method into our class and provide required set of information.
➔So IOC container will able to call our class and IOC container will able to create the object.

## Internals

### 1st time call

➔If the programmer provided class is singleton then IOC container will check whether object is available with him or not, if it is not then it will call the getObjectType ()method (from java 1.5v onwards this method will not call by IOC because of Generic) and after that it will call the isInstance() then getObject() to get the Object with corresponding data returns type.

### 2nd time call

➔If the programmer call 2nd time then IOC container will check whether object is available with him or not, if it is there then it will check bean scope is singleton/prototype if it is prototype then it will call the isSingleton() and getObject() to get the Object with corresponding data returns type.
➔If we want to change scope of Calendar class without any change in source code then just modify **CalenderFactoryBean scope as prototype** , so that even though **isInstance() return' true' it will create multiple object**

## BeanFactoryAware (Interface Injection / Contextual Dependency Lookup)

➔In spring we can manage dependency in two ways

➔Dependency lookup

- ⇨ Lookup method injection
- ⇨ Contextual lookup injection

➔Dependency injection

- ⇨ Setter injection
- ⇨ Constructor injection

➔By using dependency injection we can inject the other bean in to the target bean but if I don't want to use dependency injection, rather my target class pull corresponding object from the spring bean configuration file then I should go for dependency pull.

➔Mostly people use dependency injection for managing the dependency, but there are some limitations are available with dependency injection.

### Limitations of Dependency Injection

- ➔ If we are using dependency injection it is good about managing the dependency between those components which are having the static references with each other.
- ➔ If we have some dynamic dependency to be manage means the class whom I talk to will be decided by runtime instead of development time. Then dependency injection will not work.

**➔Make our classes loosely coupled how can use dependency pulling concept. One of the concepts is IDREF which will make our classes loosely coupled.**

➔We can make our target class to pull the correspond object by two ways

- ⇨ By declared one attribute in target class and inject a particular bean id to the target attribute.
- ⇨ By using IDREF attribute.

**When we should go for IDREF?**

➔➔IDREF word itself describes, it refers to the id of another bean.

➔ idref is used for pointing to name of the bean

➔<idref bean="suzukiEngine"> is exactly the same as just the string value.

➔If our requirement is inject a string value which is interlink with a bean id then we should go for idref.

**Ex:1**

```
public class Car {
	private Engine engine;
	private BeanFactory factory;
	private String beanName;

	public Car() {
		System.out.println("Constructor.....");
	}
	public void setBeanName(String beanName) {
		System.out.println("SetterBeanName().....");
		this.beanName = beanName;
	}

	public void drive(){

		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bfa/common/application-context.xml"));
		engine=factory.getBean(beanName,Engine.class);

		if (engine.start()==1) {
			System.out.println("Driving.....");
		}
		else{
			System.out.println("Engine faild to start...");
		}
	}
}
```

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.

➔ Whenever we are using 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

➔To get BeanFactory object we have to implementing our class from BeanFactoryAware Interface and override setbeanFactory() method ,This is called contextual dependency lookup.

```
public class Car implements BeanFactoryAware {
	private BeanFactory factory;
	@Override
	public void setBeanFactory(BeanFactory factory) throws BeansException {
		System.out.println("setBeanFactory()....");
		this.factory=factory;
	}
}
```

```java
public interface Engine {
	int start();
}
```

```java
public class SuzukiEngine implements Engine {
	@Override
	public int start() {
		System.out.println("Engine Started...");
		return 1;
	}
}
```

```java
public class YamahaEngine implements Engine {
	@Override
	public int start() {
		System.out.println("Engine Started....");
		return 1;
	}
}
```

```java
public class Car implements BeanFactoryAware {
	private Engine engine;
	private BeanFactory factory;
	private String beanName;

	public Car() {
		System.out.println("Constructor.....");
	}
	public void setBeanName(String beanName) {
		System.out.println("SetterBeanName().....");
		this.beanName = beanName;
	}

	@Override
	public void setBeanFactory(BeanFactory factory) throws BeansException {
		System.out.println("setBeanFactory()....");
		this.factory=factory;
	}

	public void drive(){
		engine=factory.getBean(beanName,Engine.class);

		if (engine.start()==1) {
			System.out.println("Driving.....");
		}
		else{
			System.out.println("Engine faild to start...");
		}
	}
}
```

O/P

```
Constructor.....
SetterBeanName().....
setBeanFactory()....
Engine Started....
Driving.....
```

```java
public class Test {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bfa/common/application-context.xml"));
		Car car=factory.getBean("car",Car.class);
		car.drive();
	}
}
```

```xml
<bean id="car" class="com.bfa.beans.Car">
	<property name="beanName">
		<idref bean="yamahaEngine"/>
	</property>
</bean>
<bean id="suzukiEngine" class="com.bfa.beans.SuzukiEngine"/>
<bean id="yamahaEngine" class="com.bfa.beans.YamahaEngine"/>
```

➔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.

# Bean Lifecycle

**Lifecycle of an Object**

➔In general every object has a life or state in their 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 (dead 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 special method called as constructor.
➔Constructor is the object lifecycle management method.
➔It is used for assigning the state of the object for performing the task.
➔ Because of that java has provided two methods 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 end/dead.

**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 cannot pass the values to the servlet because object creation is managed by servlet container, servlet will call only no-argument constructor.

**For Example:**

```
public class XServlet extends HttpServlet {

        private String driver;
        private String url;
        private String usn;
        private String psw;

        public XServlet(String driver,String url,String usn,String psw) {
                this.driver=driver;
                this.url=url;
                this.usn=usn;
                this.psw=psw;
        }
}
```

WRONG!

```xml
<servlet>
  <servlet-name>XServlet</servlet-name>
  <servlet-class>com.swp.servlets.XServlet</servlet-class>
  <init-param>
      <param-name>driver</param-name>
      <param-value>oracle.jdbc.driver.OracleDriver</param-value>
  </init-param>
  <init-param>
      <param-name>url</param-name>
      <param-value>jdbc:oracle:thin:@localhost:1521:xe</param-value>
  </init-param>
  <init-param>
      <param-name>usn</param-name>
      <param-value>system</param-value>
  </init-param>
  <init-param>
      <param-name>psw</param-name>
      <param-value>manager</param-value>
  </init-param>
</servlet>
```

```
response.getWriter().println(getInitParameter("driver"));
response.getWriter().println(getInitParameter("url"));
response.getWriter().println(getInitParameter("usn"));
response.getWriter().println(getInitParameter("psw"));
```


➔Because of that we cannot 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() method is the request processer 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.

➔By destroy method we can release the hold objects.

```
@Override
        public void init(ServletConfig config ) throws ServletException {
                this.config = config;
        }
}
```

## What is Bean Lifecycle?

Def.1

➔If we want to perform some post construct activity or manage the birth of a bean that has been created by the IOC container after the object of my bean has been created.

➔ And I want to perform some pre destruction activity or death of a bean that has been performed before the bean is being removed from the IOC container.

➔To handle these activities by IOC container, IOC container has provided bean life cycle management methods.

Def.2

➔Object for my bean is being created by the IOC container so the IOC container has to provide the mechanism for managing the birth of a bean and managing the death of a bean, so spring bean has provided life cycle methods for post construct activity and pre destroy activity on every bean, so bean life cycle comes in to picture.

## Why do we need a spring bean life cycle, why we can't use the constructor?

➔In java if we want to do any post construct activity then the only mechanism for passing the data as input while creating the object for the class is by using the constructor.

➔**But in spring bean there are multiple ways we can pass the values as input to the spring bean during time it is being created. Some data could be passing via constructor, some data could pass via setter or some data can be passing via other ways.**

➔So if I wanted to perform initialization with all the supplied value we cannot do initialization inside constructor of my bean.

➔**Spring bean needs the separate lifecycle methods because as all the data that we have supplied in creation of spring bean is not available within the constructor like a normal java object we can't initialize the state of an object with all the data inside the constructor, So we need some mechanism to manage post construct activity or pre destroy activity after all the data is being initialized, so spring has Provided lifecycle management methods.**

## So how can I perform initialization of a spring bean object with all the user supplied value?

We need to perform initialization

➔After object of Calculator bean has been created.

➔After performing the constructor injection.

➔After performing the setter injection.

➔After performing the aware injection

I wanted to perform the initialization whatever I pass everything, after is being injected with all the input data I wanted to do initialization.

So spring has provided bean lifecycle management **method like init () for birth of a spring bean** and **destroy () for death of a spring bean**.

Ex:

➔Suppose I have a calculator class and I want to add two numbers, here I want to inject one value via constructor and another value want to inject via setter.

```
public class Calculator {
                        private int num1;
                        private int num2;
                        private int sum;

                        public Calculator(int num1) {
                                this.num1=num1;
                                this.sum=this.num1+this.num2;
                        }
                        public void setNum2(int num2) {
                                this.num2 = num2;
                        }
                        //some other ways are also there like BeanFactoryAware => setBeanfactory()

                        public void init(){
                                this.sum=this.num1+this.num2;
                        }
                }
```

➔If I want to initialize all the state of calculator class while creating the object. It is not possible to initialize all states while creating the object because constructor will call while creating the object and setter will call after creating the object.

➔In this case I have to perform the initialization in init () method because init () will call by IOC container automatically, after injecting all the states of an object.

**There are some standard signature to write the init () and destroy ()**

➔Method return type should be void

➔We can give any name to the method but method should be no parameter.

**To tell IOC container this is the init method or this is the destroy method We have to configure this information in spring bean configuration file by using two attributes these are...**

```
<bean id="calculator" class="com.bl.beans.Calculator"
                init-method="init"  destroy-method="shutdown" >
```

## How Bean Lifecycle works internally?

➔When we call factory.getBean ("calculator", Calculator.class)

➔Bean Factory will goes to the InMemeory Meta data of IOC container searches for the bean definition.

➔Once the bean definition has been found it checks whether the object has created or not.

➔Then it checks the scope of the bean if the scope is singleton then it checks whether the object is already created or not, if not then it goes to the class and start instantiate the object of my bean.

➔While creating it looks for <constructor-arg> tag if it is there then it going to check for cyclic dependency check if there is no circular dependency problem.
➔Then it will check IDREF for dependency check.
➔If all the value has been passed, then immediately it starts creating the object for my bean by performing the constructor injection.
➔Then it will check for <property> tag if it is there it performs setter injection.
➔Then looks for whether the class is implementing form Aware interface or not, if class implementing from Aware interface it calls the corresponding method to inject the internal object of that IOC container.
➔When all the injection has been happened on that bean then before the object is be placed in the IOC container it checks did I configure any init-method or not to handle the post construct activity.
➔If I configured then before placing the object in IOC container it will call init-method and completes the post construct activity of my bean and then finally the bean definition object will be placed in the IOC container.
➔Then the destroy method will be called when garbage collector will destroy the object from Heap memory

```
public class Calculator implements BeanFactoryAware {
	private int num1;
	private int num2;
	private int sum;

	public Calculator(int num1) {
		System.out.println("constructor Called...");
		this.num1=num1;
	}
	public void setNum2(int num2) {
		System.out.println("setter() Method  Called...");
		this.num2 = num2;
	}
	public void init(){
		System.out.println("init() Method  Called...");
		this.sum=this.num1+this.num2;
		System.out.println("Sum of Num1 and Num2 is :"+sum);
	}
	public void shutdown(){

		System.out.println("shutdown() called...");
	}
	@Override
	public void setBeanFactory(BeanFactory factory) throws BeansException {
		System.out.println("setFactory() Method  Called...");
	}

}
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
	<bean id="calculator" class="com.bl.beans.Calculator" init-method="init" destroy-method="shutdown">
		<constructor-arg value="12"/>
		<property name="num2" value="30"/>
	</bean>
</beans>
```

```
public class Test {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bl/common/application-context.xml"));
		Calculator calculator=factory.getBean("calculator",Calculator.class);
	}
}
```

```
constructor Called...

setter() Method  Called...

setFactory() Method  Called...

init() Method  Called...
```

**In this example destroy method will not call to know why it is not called, we have to know about Garbage collector of java.**

## Q. What is Garbage Collector

➔Garbage collector is a component of JVM that is responsible for memory management.
➔Garbage collector will not be invoking each and every object when the object has not any reference.
➔This is a daemon thread that is executed periodically.
➔When garbage collector executed then the huge amount of performance of the application will be impacted because it holding the whole memory blocks which is allocated for application and identified the free memory spaces to make available for application.

**Q-When object is qualified for Garbage collector (dangling pointer)? How the garbage collector is going to manage the memory management?**

➔If a object not having any reference with any of the part of our application then the object is being qualified for garbage collector or dangling point.

➔Garbage collector identified such kind of object and executed periodically and that is goes to JVM memory and issuing **'stop the world'** command.

➔In this stage all the threads are in freeze state then the garbage collector is start the calling on each and every object and identify whether object available for GC or not, this **algorithm is called mark and sweep algorithm**

➔**At the time of removing the object it will call the finalize method on that object.**

➔ Before 1.6 version of java the GC followed mark & sweep algorithm but from 1.6v onwards some other algorithm came.

Note: we can increase the memory of Heap by using prom gram argument

**Q.Why finalize() method is called by the Garbage collector why not called by JVM?**

➔If JVM is repeatedly pulled up with memory management activities then performance of the application is being impacted.

➔The primary meaning of executing the application is not memory management that is the second thing.

➔So memory management has to be given the least priority so JVM will not involve in memory management aspects.

➔So garbage collector is takes care of memory management.

➔We can't never call the garbage collector but we can request JVM to run the invoke garbage collector by using two ways.

- ⇨ System.gc();
- ⇨ Runtime.getRuntime().gc();

**Q.How to make an object eligible for GC?**

➔If an object no longer required then we can make eligible for GC by assigning "null" to all its reference variables.

➔If an object no longer required then reassign all its reference variable to some other object then old object is by default eligible for GC.

➔Object created inside a method are by default eligible for GC once method completes.

**Q. Why destroy method is not called in bean object**

➔When the bean is being removed then the IOC container will call the destroy method Because of the bean is singleton so it is exist in the IOC container up to end of the IOC container.

➔It will call destroy method when IOC container going to die but IOC container never knows when he is going to die.

➔So IOC container never call destroy method on my object.

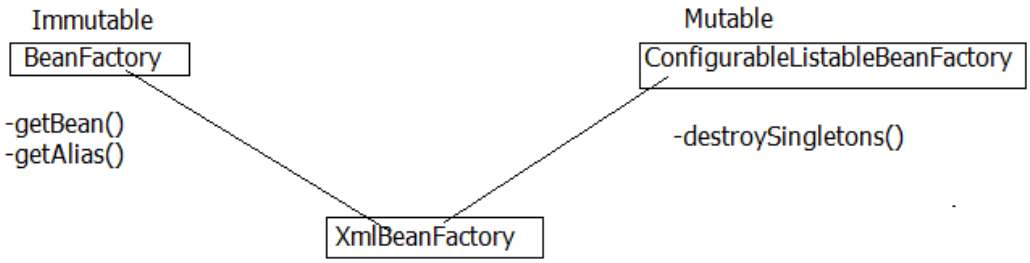
**Q. How we can call destroy method on bean object explicitly?**

➔ My Bean is going to die when IOC container itself is going to die

➔IOC container is going to die when JVM is going to die.

➔If I know when the JVM is going to die then I can tell to IOC container you are dying.

➔As IOC container knows then IOC container will call the destroy-method on Bean object.
➔JVM will go to terminate at the end of the application




```
public class Test {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bl/common/application-context.xml"));
		Calculator calculator=factory.getBean("calculator",Calculator.class);

		((ConfigurableListableBeanFactory)factory).destroySingletons();
	}
}
```

➔If we call destorySingletons() method of **ConfigurableListableBeanFactory** then IOC container will release all the singleton beans before it is going to die.

➔But there is a chance JVM may shutdown in any moment(before ending of program also ) so we have to optimize the this application and design our class with a design structure before JVM is going to die it will inform to IOC container so that IOC container will do pre destroy activity by calling destroy-method.

➔**When we call detroySingletons() method IOC container will not destroy rather it release all the object which are there within IOC container**(we can call again factory.getBean("calculator") then calculator object will come ).

```java
public class Test {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bl/common/application-context.xml"));

		Calculator calculator=factory.getBean("calculator",Calculator.class);
		ShutDownHookUp shutdown=factory.getBean("shutdownhookup",ShutDownHookUp.class);
		Runtime runtime=Runtime.getRuntime();
		runtime.addShutdownHook(new Thread(shutdown));
	}
}
```

```java
public class Calculator implements BeanFactoryAware {
	private int num1;
	private int num2;
	private int sum;

	public Calculator(int num1) {
		System.out.println("constructor Called...");
		this.num1=num1;
	}
	public void setNum2(int num2) {
		System.out.println("setter() Method  Called...");
		this.num2 = num2;
	}
	public void init(){
		System.out.println("init() Method  Called...");
		this.sum=this.num1+this.num2;
		System.out.println("Sum of Num1 and Num2 is :"+sum);
	}
	public void shutdown(){
		System.out.println("shutdown() called...");
	}
	@Override
	public void setBeanFactory(BeanFactory factory) throws BeansException {
		System.out.println("setFactory() Method  Called...");
	}
}
```

```java
public class ShutDownHookUp implements Runnable, BeanFactoryAware{

	private BeanFactory factory;
	@Override
	public void setBeanFactory(BeanFactory factory) throws BeansException {
		this.factory=factory;
	}
	@Override
	public void run() {
		((ConfigurableListableBeanFactory)factory).destroySingletons();
	}
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
	<beans xmlns="http://www.springframework.org/schema/beans"
		xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
		xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
		<bean id="calculator" class="com.bl.beans.Calculator" init-method="init"
destroy-method="shutdown" >
			<constructor-arg value="12"/>
			<property name="num2" value="30"/>
		</bean>
		<bean id="shutdownhookup" class="com.bl.hookup.ShutDownHookUp"/>
	</beans>
```

## O/P

```
constructor Called...
setter() Method  Called...
setFactory() Method  Called...
init() Method  Called...
Sum of Num1 and Num2 is :42
shutdown() called...
```

There are three ways to can work with bean Lifecycle

- ⇨ Configuration approach
- ⇨ Programmatic approach
- ⇨ Annotation-driven approach

**Q. If Configuration approach is there why should I go for Programmatic approach?**

**1st problem**

➔There are some problems with configuration approach due to which we should go for programmatic approach.

➔If we are configuring multiple no beans (to get different configuration value for my bean) of a same class then we have to configure init-method and destroy-method in every <bean> so there may be a chance we might forget to configure init-method, destroy-method.

➔In such case initialization and destruction process may not work properly so unexpected result will come when we use this bean.

**2nd problem**

➔ If we are configuring multiple no beans of a same class then configuration will become duplicated.

➔For managing such configuration will become difficult.

➔If I change the int () to initialize () then in every place I have to modify manually which seems to be a difficult job.

➔Programmer has to always identify this is the init () and this is the destroy () when he use the class.

**Programmatic Approach:**

➔In programmatic approach no need to configure init() and destroy() method.

➔we have to implementing our class from standard interfaces and override the standard methods so at the time of creating the object for bean ,IOC container will detect the class implementing from standard interfaces then performing the life cycle methods.

➔We need to implement our class from two interfaces

- ⇨ **InitializingBean**
  **|-afterPropertiesSet();**

- ⇨ **DisposableBean**
  **|-destroy()**

**Q. Why spring has provided two interfaces for programmatic approach of life cycle.**

➔Spring has provided two interfaces because a bean may have birth lifecycle it may not have death life cycle. So if spring provides tow methods in one interface the always we need to manage birth and death life cycle of a bean.

Ex-2

```
public class Calculator implements BeanFactoryAware,InitializingBean,DisposableBean {
	private int num1;
	private int num2;
	private int sum;

	public Calculator(int num1) {
		System.out.println("constructor Called...");
		this.num1=num1;
	}
	public void setNum2(int num2) {
		System.out.println("setter() Method  Called...");
		this.num2 = num2;
	}
	@Override
	public void setBeanFactory(BeanFactory factory) throws BeansException {
		System.out.println("setFactory() Method  Called...");
	}
	@Override
	public void afterPropertiesSet() throws Exception {
		System.out.println("afterPropertySet() Method  Called...");
		this.sum=this.num1+this.num2;
		System.out.println("Sum of Num1 and Num2 is :"+sum);
	}
	@Override
	public void destroy() throws Exception {
		System.out.println("destroy method called...");

	}
}
```

```
public class ShutDownHookUp implements Runnable, BeanFactoryAware{

	private BeanFactory factory;
	@Override
	public void setBeanFactory(BeanFactory factory) throws BeansException {
		this.factory=factory;
	}
	@Override
	public void run() {
		((ConfigurableListableBeanFactory)factory).destroySingletons();
	}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
	<beans xmlns="http://www.springframework.org/schema/beans"
		xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
		xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
		<bean id="calculator" class="com.bl.beans.Calculator" >
			<constructor-arg value="12"/>
			<property name="num2" value="30"/>
		</bean>
		<bean id="shutdownhookup" class="com.bl.hookup.ShutDownHookUp"/>
	</beans>
```

## Property Editors

**What is property editor?**

➔Property editor is the classes which takes care of edit the property value of target class before those are injected in to the target class variables.

➔**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 is some situation we cannot convert object, arrays, file, URL, date and so on.**

➔There are lots of predefined class types in java which are using frequently used class types.

➔For those class types if I wanted to inject dependencies then I have to configure those classes as bean.

➔So to convert different non-primitive types into corresponding format spring has provided a feature called property Editors.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
       <bean id="student" class="com.pe.beans.Student">
              <property name="id" value="1"/>
              <property name="name" value="Dhana"/>

              <!--  <property name="dob" ref="date"/>  -->

              <property name="dob" value="10/11/2016"/>
              <property name="marks" value="30,40,50,60" />
              <property name="image" value="d:/work/image1.jpg"/>
       </bean>

       <bean id="date" class="java.util.Date">
              <constructor-arg value="10"/>
              <constructor-arg value="11"/>
              <constructor-arg value="2016"/>
       </bean>
</beans>
```

➔Property editors mean to edit the property and convert into corresponding format for operation.

➔Spring has provided number of property editors classes internally which will help in directly declaring the values for that property.

```
private int id;
private String name;
private Date dob;
private int[] marks;
private File image;
public void setId(int id) {
       this.id = id;
}
public void setName(String name) {
       this.name = name;
}
public void setDob(Date dob) {
       this.dob = dob;
}
public void setMarks(int[] marks) {
       this.marks = marks;
}
public void setImage(File image) {
       this.image = image;
}
@Override
public String toString() {
       return "Student [id=" + id + ", name=" + name + ",
       dob=" + dob + ", marks=" + Arrays.toString(marks)
                     + ", image=" + image + "]";
}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
       <bean id="student" class="com.pe.beans.Student">
              <property name="id" value="1"/>
              <property name="name" value="Dhana"/>
              <property name="dob" value="10/11/2016"/>
              <property name="marks" value="30,40,50,60" />
              <property name="image" value="d:/work/image1.jpg"/>
       </bean>
</beans>
```

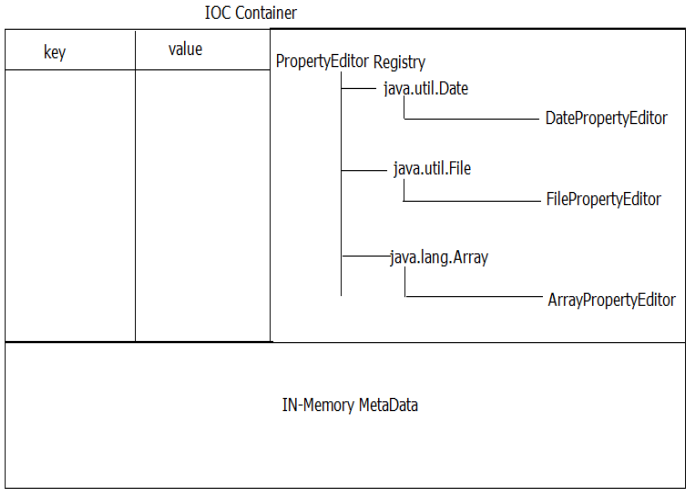
```
public class Test {
       public static void main(String[] args) {
              BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/pe/common/application-context.xml"));
              Student student=(Student)
factory.getBean("student",Student.class);
              System.out.println(student);
       }

}
```

➔While creating the object of the bean IOC container will check the scope the 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 in 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.

➔Spring has provided implicit property editor for Date, URL, Array, and File... etc.

➔When we pass the value as string type spring will take property type and it will call the appropriate Property Editor class to convert the value



## How to create Custom Property Editor

➔There are some situation were have to create our own custom property editors.

➔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**.

```
public class ComplexNumber {
	private int base;
	private int expo;

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
	<bean id="calculator" class="com.pe.beans.ScientificCalculator">
		<property name="complexNumber" value="10,78"/>
	</bean>

</beans>
```

```
public class ComplexNumberPropertyEditor extends PropertyEditorSupport {

	@Override
	public void setAsText(String value) throws IllegalArgumentException {
		int base=0;
		int expo=0;
		String [] token=null;
		ComplexNumber complexNumber=null;

		token=value.split(",");
		base=Integer.parseInt(token[0]);
		expo=Integer.parseInt(token[1]);
		complexNumber=new ComplexNumber();
		complexNumber.setBase(base);
		complexNumber.setExpo(expo);
		setValue(complexNumber);
	}
}
```

```
public class CustomPropertyEditor implements PropertyEditorRegistrar{
	@Override
	public void registerCustomEditors(PropertyEditorRegistry registry) {
		registry.registerCustomEditor(ComplexNumber.class , new ComplexNumberPropertyEditor());

	}
}
```

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

```
public class ComplexNumberPropertyEditor extends PropertyEditorSupport {

	@Override
	public void setAsText(String value) throws IllegalArgumentException {
		int base=0;
		int expo=0;
		String [] token=null;
		ComplexNumber complexNumber=null;

		token=value.split(",");
		base=Integer.parseInt(token[0]);
		expo=Integer.parseInt(token[1]);
		complexNumber=new ComplexNumber();
		complexNumber.setBase(base);
		complexNumber.setExpo(expo);
		setValue(complexNumber);
	}
}
```

➔We have to override one method called **setAsText(String value)** and we can write the converting logic in it and set to the corresponding constructor

➔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 class CustomPropertyEditor implements PropertyEditorRegistrar{
	@Override
	public void registerCustomEditors(PropertyEditorRegistry registry) {
		registry.registerCustomEditor(ComplexNumber.class , new ComplexNumberPropertyEditor());

	}
}
```

➔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.

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

➔BeanFactory is immutable we can not modify the IOC container, to modify or add we have to use the **configurableListableBeanFactory**.

## Method Replacement

➔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 insurance plan ), which lead to the huge problem.

```
3  public class PlanFinder {
4     public String[] findPlan(int ageGroup , int copay , String maritalStatus){
5
6
7        if (ageGroup<=3 && copay>=10 && maritalStatus.equals("single")) {
8           return new String[]{"Plan-1","Plan-2","Plan-3"};
9        }
10       else if (ageGroup<=2 && copay>=10 && maritalStatus.equals("married")) {
11          return new String[]{"Plan-4","Plan-5","Plan-6"};
12       }
13       else if (ageGroup<=1 && copay>=20 && maritalStatus.equals("married")) {
14          return new String[]{"Plan-7","Plan-8","Plan-9"};
15       }
16       else if (ageGroup<=1 && copay>=20 && maritalStatus.equals("single")) {
17          return new String[]{"Plan-7","Plan-8","Plan-9"};
18       }
19       else if (ageGroup<=3 && copay<=30 && maritalStatus.equals("married")) {
20          return new String[]{"Plan-10","Plan-11","Plan-12"};
21       }
22       else if (ageGroup<=2 && copay<=30 && maritalStatus.equals("single")) {
23          return new String[]{"Plan-10","Plan-11","Plan-12"};
24       }
25       return null;
26    }
27 }
```

➔In this case developer can not avoid writing if-else and else if conditions , which makes many confusion.
➔If there are four parameter available then we can write 16 decision making condition(which insurance plan ), 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.
➔Rule Engine is unlike the java programming decision making approach.
➔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 problem.

## When to use Rule Engine?

➔when there is no satisfactory traditional programming approach to solve the problem.
➔The problem may not be complex, but you can't see a non-fragile way of building a solution for it.

➔There are several tools available in the market who provide the Rule Engine tools.

- JBOSS Drools
- JRuleEngine
- Mandarax
- JLisa
- JEOPS - The Java Embedded Object Production System
- Prova language
- OpenRules
- Open Lexicon
- SweetRules
- Zilonis
- Hammurapi Rules

Again there are so many open source rule engines tools are available
➔We are using library of Rule engines so out class is tightly coupled with particular rule engine

### 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.**

**Example:**

➔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 times but in production environment it will failing so that client is getting huge revenue lose.
➔But still it is working, because after 10000 request it will arise exception so client wants to fix the bug but if we are trying to fix the bug again it fails in production environment.

➔But still it is there then we can not modify the current working class method code because it leads to be a heavy lose of client.

➔In this case if we wanted to modify the code how we can able to modify without modifing in exting method then we can easily can use method replacement feature which is provided by spring .

➔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.

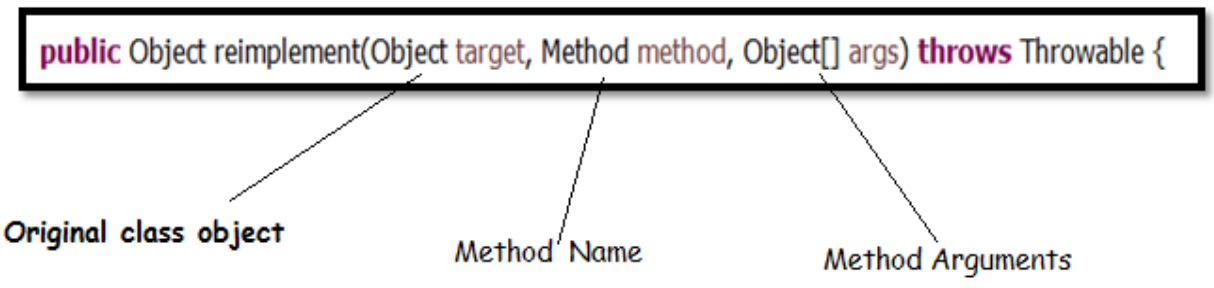
```
public Object reimplement(Object target, Method method, Object[] args) throws Throwable {
```

➔If we look at the above method it going to return the Object because 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 because we can not touch to the original method, moreover all the object are not with us and 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 dependencies.

For method replacement below additional configuration will be provided

```
<beans>

        <bean id= "insurance" class= "com.mr.classes.Insurance">
                <replaced-method name= "findPlan" replacer= "findplanreplacer" />
        </bean>

        <bean id= "findplanreplacer" class= "com.mr.classes.FindPlanReplacer"/>

</beans>
```

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

```
public class PlanFinder {
        public String[] findPlan(int ageGroup , int copay , String maritalStatus){
                String [] s=null;
                s=new String[]{"Jivan Anand","Jivan Saral","Jivan Samrudhi"};
                return s;
        }
}
```

```
import org.springframework.beans.factory.support.MethodReplacer;
public class FindPlanReplacer implements MethodReplacer {
        public Object reimplement(Object target, Method method, Object[] args)
throws Throwable {

                if (method.getName().equals("findPlan")) {
                        int ageGroup=0;
                        int copay=0;
                        String maritalStatus=null;
                        ageGroup=(Integer)args[0];
                        copay=(Integer)args[1];
                        maritalStatus=(String)args[2];
                        return new String[] {"Abhaya Bima Gold","Abhaya Surakhya
Parivar","Abhaya Jivan"};
                }

                if (method.getName().equals("getMaritalStatus")) {
                        System.out.println("Married");
                }
                return null;
        }
}
```

```
<?xml version= "1.0" encoding= "UTF-8"?>
<beans xmlns= "http://www.springframework.org/schema/beans"
        xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation= "http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

        <bean id= "planfinder" class= "com.mr.classes.PlanFinder">
                <replaced-method name= "findPlan" replacer= "findplanreplacer" />
        </bean>
        <bean id= "findplanreplacer" class= "com.mr.classes.FindPlanReplacer"/>
</beans>
```

```
public class MRTest {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new
ClassPathResource("com/mr/common/application-context.xml"));
                PlanFinder finder=factory.getBean("planfinder",PlanFinder.class);
                String[] plans=finder.findPlan(1, 10, "single");
                for (String plan : plans) {
                        System.out.println(plan);
                }
        }
}
```

## Internals of Method Replacer

➔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 original method and** it gives all details of Class object ,Method Name ,and arguments of that method to reimplement method.

➔At runtime IOC will create the object for that proxy class and place it in JVM memory at runtime only.

➔**IOC using native library like (cgilib, asm) to create the class at runtime and place the object at JVM memory at runtime because this feature is not available at java.**

➔If we want to use method replacement then the original class should not be final and method should not be static, and then only it is possible because internally original class will extends by proxy class.

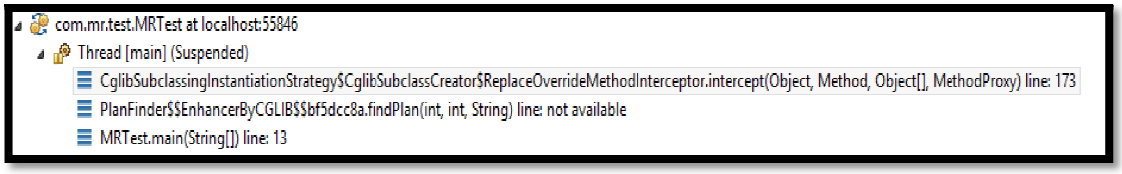
➔To get all the details of the method internally proxy class uses stack Trace techniques. Because Stack Trace will keep all the records of current thread which is currently executing.

```
public class PlanFinder {
        public String[] findPlan(int ageGroup , int copay , String maritalStatus){
                String [] s=null;
                s=new String[]{"Jivan Anand","Jivan Saral","Jivan Samrudhi"};
                return s;
        }
}
```

```
public class FindPlanReplacer implements MethodReplacer {
        public Object reimplement(Object target, Method method, Object[] args) throws Throwable {
                if (method.getName().equals("findPlan")) {
                        int ageGroup=0;
                        int copay=0;
                        String maritalStatus=null;
                        ageGroup=(Integer)args[0];
                        copay=(Integer)args[1];
                        maritalStatus=(String)args[2];
                        return new String[] {"Abhaya Bima Gold","Abhaya Surakhya Parivar","Abhaya Jivan"};
                }
                if (method.getName().equals("getMaritalStatus")) {
                        System.out.println("Married");
                }
                return null;
        }
}
```

```
class PlanFinder$$EnhancedByCGI extends PlanFinder{
        MethodReplacer replacer;
        public PlanFinder$$EnhancedByCGI(MethodReplacer replacer) {
                this.replacer=replacer;
        }
        @Override
        public String[] findPlan(int ageGroup, int copay, String maritalStatus) {
                try {
                        Object obj=this;
                        Method[] method=Thread.currentThread().getStackTrace().getClass().getDeclaredMethods();
                        Object[] args=new Object[]{ageGroup ,copay,maritalStatus};
                        return  (String[]) replacer.reimplement(obj,method[0],args);
                } catch (Throwable e) {}
                return null;
        }
}
```

**Debug Report: of method Replacement**




# Internationalization (I18N)

**What is internationalization?**

➔The process of preparing an application to support more than one language and data format is called **internationalization**. **Localization** is the process of adapting an internationalized application to support a specific region or locale.

➔I want to build a application, will support multiple languages, and the people who wanted to access the application want to be localize for the locale so that adaptability of the application will be high and the reachable of the application will be high so my application has to support internationalization.

**How many levels of Internationalization are there?**

➔There are three types of internationalizations are there

- ⇨ Data Internationalization (Collation)
- ⇨ NLS Internationalization (Native/Natural Language Support)
- ⇨ Content Internationalization

**Data Internationalization**

If we want to store data in database which supports multiple languages then the database should have support for internationalization. To support multiple languages we have to enable collation.

**NLS Internationalization (Native/Natural Language Support)**

If a java application read the data from database with same character set encoding which data has been stored in the database. So java has provided Natural/native language support for read the data with same set of character.

**Content Internationalization**

The static data that is being stored within my application, if I wanted to display data with user specific language format depends on locale, and then this is called content internationalization.

## How to achieve Content Internationalization

➔The user who is accessing the application is accessing it from which language is important. We can determine user language is not sufficient because every country has different different own language.

➔We need to find the country code and language so the combination of two are called locale.

Approach-1

```
<%@page import="java.util.Locale"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    <%
    Locale locale=Locale.getDefault();
    String lang=locale.getLanguage();
    if (lang.equals("en")){

        %><h1>Dhananjaya welcomes you</h1><%
    }
    else if(lang.equals("hi")){
        %><h1>धनंजय आप का स्वागत कर रहे हे</h1><%
    }
    else if(lang.equals("or")){
        %><h1>ଧନଞ୍ଜୟ ଆପଣଙ୍କୁ ସ୍ୱାଗତ କରୁଛି</h1><%
    }
    %>
</body>
</html>
```

➔To Display Content In Jsp Page We Should Not Hardcode Rather Should Read The Text From Message Bundle (Properties File).

## Optimize-2

| | |
|---|---|
| error_en_IN.properties | errorMessage=Password must be contain number!! |
| message_defaultName.properties | welcomeMessage=Dhananjaya welcomes you!! |
| message_en_IN.properties | welcomeMessage=Dhananjaya welcomes you!! |
| message_en_US.properties | welcomeMessage=Dhananjaya welcomes you!! |
| message_hi_IN.properties | welcomeMessage=धनंजय आप का स्वागत कर रहे है!! |
| message_or_IN.properties | welcomeMessage=ଧନଞ୍ଜୟ ଆପଣଙ୍କୁ ସ୍ୱାଗତ କରୁଛି |

```
<%@page import="java.io.FileInputStream"%>
<%@page import="java.util.Locale"%>
<%@page import="java.util.Properties"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
     <%
     Locale locale=Locale.getDefault();
     String lang=locale.getLanguage();
     Properties props=new Properties();
     String message="";
     String baseLocation="message_";
     if (lang.equals("en")){

     props.load(this.getClass().getClassLoader().getResourceAsStream(baseLocation+"en_US.properties"))
;
          message=props.getProperty("welcomeMessage");
     }
     else if(lang.equals("hi")){
          props.load(new FileInputStream(baseLocation+"hi_IN.properties"));
          message=props.getProperty("welcomeMessage");
     }
     else if(lang.equals("or")){
          props.load(new FileInputStream(baseLocation+"or_IN.properties"));
          message=props.getProperty("welcomeMessage");
     }
     byte ptext[] = message.getBytes("ISO-8859-1");
     String value = new String(ptext, "UTF-8");
     out.print("<h1>"+value+"</h1>");
     %>
</body>
</html>
```

## Optimize-3

```
<%@page import= "java.io.FileInputStream"%>
<%@page import= "java.util.Locale"%>
<%@page import= "java.util.Properties"%>
<%@ page language= "java" contentType= "text/html; charset=UTF-8"
  pageEncoding= "UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv= "Content-Type" content= "text/html; charset=UTF-8">
<title>Insert title here</title>

</head>
<body>
      <%
            Locale locale=Locale.getDefault();
            Properties props=new Properties();
            String message="";
            String baseLocation="message_";
            try{
                  props.load(this.getClass().getClassLoader().getResourceAsStream(baseLocation+locale+".properties"));
                  message=props.getProperty("welcomeMessage");
            }catch(Exception e){
                  props.load(this.getClass().getClassLoader().getResourceAsStream(baseLocation+"defaultName.properties"));
                  message=props.getProperty("welcomeMessage");
            }
            byte ptext[] = message.getBytes("ISO-8859-1");
            String value = new String(ptext, "UTF-8");
            out.print("<h1>"+value+"</h1>");
      %>
</body>
</html>
```

➔*To fulfil internationalization operation in our application I have to write above code, the piece of logic that I am writing to internationalization operation not only I any one has to write the same code for such type of requirement, such type of logic that we are trying to write is seems to be common on across all the project across world. This is called boilerplate logic.*

➔So JAVA API has provided internationalization support and given a class called ResourceBundle.

➔So programmers no need to write more no of lines of code to achieve internationalization in his project.

## Resource Bundle

➔ The **ResourceBundle class** is used to internationalize the messages. In other words, we can say that it provides a mechanism to globalize the messages.
➔ The hardcoded message is not considered good in terms of programming, because it differs from one country to another. So we use the ResourceBundle class to globalize the massages. The ResourceBundle class loads these informations from the properties file that contains the messages.

**Optimize-4**

| | |
|---|---|
| error_en_IN.properties | errorMessage=Password must be contain number!! |
| message_defaultName.properties | welcomeMessage=Dhananjaya welcomes you!! |
| message_en_IN.properties | welcomeMessage=Dhananjaya welcomes you!! |
| message_en_US.properties | welcomeMessage=Dhananjaya welcomes you!! |
| message_hi_IN.properties | welcomeMessage=धनंजय आप का स्वागत कर रहे है!! |
| message_or_IN.properties | welcomeMessage=ଧନଞ୍ଜୟ ଆପଣଙ୍କୁ ସ୍ୱାଗତ କରୁଛି |

```
<%@page import="java.io.FileInputStream"%>
<%@page import="java.util.Locale"%>
<%@page import="java.util.ResourceBundle"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Insert title here</title>
    </head>
    <body>
        <%
            ResourceBundle resourceBundle=ResourceBundle.getBundle("message",Locale.getDefault());
            String value=resourceBundle.getString("welcomeMessage");
            out.print("<html><body><h1>"+value+"</h1></body></html>");
        %>
    </body>
</html>
```

➔Here,we are Writing Scriptlet logic inside a JSP which is not recommended, so write this logic of RecourceBundle inside a ServletFilter, whenever servlet container get the request its handover that request to Filter Manager and handover that request to that filter based on mapping.
➔Filter is the one which perform pre-processing and post-processing on request.

## Optimize-5

| | |
|---|---|
| error_en_IN.properties | errorMessage=Password must be contain number!! |
| message_defaultName.properties | welcomeMessage=Dhananjaya welcomes you!! |
| message_en_IN.properties | welcomeMessage=Dhananjaya welcomes you!! |
| message_en_US.properties | welcomeMessage=Dhananjaya welcomes you!! |
| message_hi_IN.properties | welcomeMessage=धनंजय आप का स्वागत कर रहे है!! |
| message_or_IN.properties | welcomeMessage=ଧନଞ୍ଜୟ ଆପଣଙ୍କୁ ସ୍ୱାଗତ କରୁଛି |

```
import java.io.IOException;
import java.util.Locale;
import java.util.ResourceBundle;
import javax.servlet.*;

@WebFilter("/*")
public class I18NFilter implements Filter {

    public void destroy() {
    }
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {

        Locale locale = Locale.getDefault();
        ResourceBundle bundle = ResourceBundle.getBundle("message",locale);
        request.setAttribute("msg", bundle);
        chain.doFilter(request, response);
    }
    public void init(FilterConfig fConfig) throws ServletException {

    }
}
```

```
<%@page import="java.io.FileInputStream"%>
<%@page import="java.util.Locale"%>
<%@page import="java.util.ResourceBundle"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Insert title here</title>
    </head>
    <body>
        <%
            ResourceBundle obj=(ResourceBundle)request.getAttribute("msg");
            String msg=obj.getString("welcomeMessage");
            out.println("<html><body><h1>"+msg+"</h1></body></html>");
        %>
    </body>
</html>
```

➔An application cannot be completed with one properties file; we may need to read the value from multiple properties file.

➔If we have multiple properties file we need to create one Resource Bundle per one properties file to read the values.

## Optimize-6

```
import java.io.IOException;
import java.util.Locale;
import java.util.ResourceBundle;
import javax.servlet.*;
@WebFilter("/*")
public class I18NFilter implements Filter {

     public void destroy() {
     }
     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {

          Locale locale = Locale.getDefault();
          ResourceBundle bundle = ResourceBundle.getBundle("message",locale);
          ResourceBundle bundle1 = ResourceBundle.getBundle("error",locale);
          request.setAttribute("msg", bundle);
          request.setAttribute("err", bundle1);

          chain.doFilter(request, response);
     }
     public void init(FilterConfig fConfig) throws ServletException {
     }
}
```

```
<%@page import="java.io.FileInputStream"%>
<%@page import="java.util.Locale"%>
<%@page import="java.util.ResourceBundle"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
     <title>Insert title here</title>
     </head>
     <body>
          <%
               ResourceBundle obj=(ResourceBundle)request.getAttribute("msg");
               ResourceBundle obj2=(ResourceBundle)request.getAttribute("err");
               String msg=obj.getString("welcomeMessage");
               String error=obj2.getString("errorMessage");
               out.println("<html><body><h1>"+msg+"<br>"+error+"</h1></body></html>");
          %>
     </body>
</html>
```

➔Here The Resource Bundle Object is set to request scope.

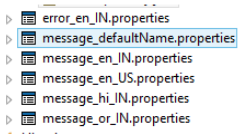
➔If we want to access multiple Bundles we need to use multiple Resource Bundle's.

➔And we have to set them to request scope

➔ If we have multiple properties bundles we need to write multiple Resource Bundles' 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 Resource Bundle Objects and store them in **a Collection Object then forward it to JSP page.**

## Optimize-7




```
public class I18NRepository {
	private Properties props;
	private static I18NRepository instance;
	private I18NRepository(){
		//no-code
	}
	public static synchronized I18NRepository getInstance(){
		if (instance==null) {
			instance=new I18NRepository();
		}
		return instance;
	}

	public Properties getProperty(Locale locale){
		String [] baseNames=new String[]{"error","message"};
		props=new Properties();
		for (String baseName : baseNames) {
			ResourceBundle bundle=ResourceBundle.getBundle(baseName,locale);
			Set <String> keys=bundle.keySet();
			for (String key : keys) {
				props.put(baseName, key);
			}
		}
		return props;
	}
}
```

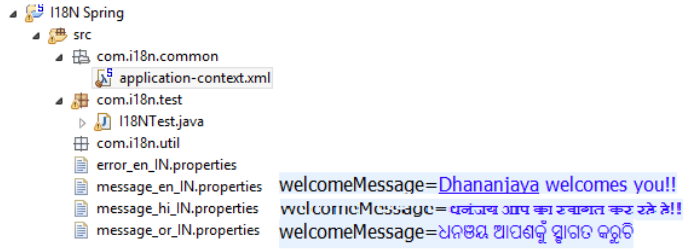
```
@WebFilter("/*")
public class I18NFilter implements Filter {

	public void destroy() {
	}
	public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
		Locale locale=request.getLocale() != null ? Locale.getDefault():request.getLocale();
		I18NRepository repository=I18NRepository.getInstance();
		Properties props=repository.getProperty(locale);
		Set keys=props.keySet();
		for(Object key:keys){
			String baseName=key.toString();
			ResourceBundle bundle = ResourceBundle.getBundle(baseName);
			String msg=bundle.getString(props.getProperty(baseName));
			request.setAttribute(props.getProperty(baseName), msg);
		}
		chain.doFilter(request, response);
	}
	public void init(FilterConfig fConfig) throws ServletException {
	}
}
```

```
<%@page import="java.io.FileInputStream"%>
<%@page import="java.util.*"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
	<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
	<title>Insert title here</title>
	</head>
	<body>
		<%
			out.println(request.getAttribute("welcomeMessage"));
			out.println(request.getAttribute("errorMessage"));

		%>
	</body>
</html>
```

## Spring Internationalization

➔Spring has provided better support for internationalization when compared with JEE internationalization.

➔It has provided multiple classes to support Internationalization.




```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

       <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
              <property name="basenames">
                     <list>
                            <value>message</value>
                            <value>error</value>
                     </list>
              </property>
       </bean>
</beans>
```

```
public class I18NTest {
       public static void main(String[] args) throws UnsupportedEncodingException {
              ApplicationContext context=new ClassPathXmlApplicationContext("com/i18n/common/application-context.xml");
              String message=context.getMessage("welcomeMessage",null,Locale.getDefault());
              byte ptext[] = message.getBytes("ISO-8859-1");
              String value = new String(ptext, "UTF-8");
              System.out.println(value);
       }

}
```

➔Name of the bean should be messageSource otherwise it will not work.

➔ApplicationContext supports MessageSource Directly.

## BeanFactory vs. Application Context

| BeanFactory | ApplicationContext |
|---|---|
| 1.BeanFactory is lazy initialize, means after creating the Bean Factory it will loads the configuration file and create metadata in In-memory metadata inside JVM but it will not create any bean for the bean definition, when I try to fetch the bean using factory.getbean("...") then only Bean Factory will create the bean | 1. Application Context is eager initialize. Means when I first create the application context with configuration, after loading the configuration, it will immediately create all the singleton beans in the configuration. |
| 2.Bean Factory Doesn't support Internationalization directly | 2. ApplicationContext support internationalization. |

| | |
|---|---|
| **3.** Beanfactory Doesn't Support Event Processing | **3.** ApplicationContext Supports Event Processing |
| **4.** BeanFactory will not automatically registers BeanPostProcessor or BeanFactoryPostProcessor, We need to explicitly write the code for registering them. | **4.** ApplicationContext up seeing the BeanPostProcessor and BeanFactoryPostProcessor declarations in the configuration will automatically registers them with the container and applies processing. |

error_en_IN.properties errorMessage=Password must be contain number!!

message_en_IN.properties welcomeMessage=Dhananjaya welcomes you!!

message_hi_IN.properties welcomeMessage=धनंजय आप का स्वागत कर रहे हे!!

message_or_IN.properties welcomeMessage=ଧନଞ୍ଜୟ ଆପଣଙ୍କୁ ସ୍ୱାଗତ କରୁଚି

```
public class i18NTestBeanFactory {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/i18n/common/application-context.xml"));
		MessageSource message=factory.getBean("messageSource",ResourceBundleMessageSource.class);
		System.out.println(message.getMessage("welcomeMessage",null, Locale.getDefault())  );
	}
}
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

	<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
		<property name="basenames">
			<list>
				<value>message</value>
				<value>error</value>
			</list>
		</property>
	</bean>
</beans>
```

➔BeanFactory doesn't support MessageSource directly.
➔We have to create explicitly ResourceBundleMessageSource object and call getMessage() on that object.

## Case:1

error_en_IN.properties errorMessage=Password must be contain number!!

message_en_IN.properties welcomeMessage=Dhananjaya welcomes you!!

message_hi_IN.properties welcomeMessage=धनंजय आप का स्वागत कर रहे हे!!

message_or_IN.properties welcomeMessage=ଧନଞ୍ଜୟ ଆପଣଙ୍କୁ ସ୍ୱାଗତ କରୁଚି

```
public class i18NTestBeanFactory {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/i18n/common/application-context.xml"));
		MessageSource message=factory.getBean("messageSource",ResourceBundleMessageSource.class);
		System.out.println(message.getMessage("welcomeMessageWithName1",null,"Not a valid key",Locale.getDefault())  );
	}
}
```

**If i will pass the wrong argument to the getMessage() as a first argument then third argument will automatically printed**

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

	<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
		<property name="basenames">
			<list>
				<value>message</value>
				<value>error</value>
			</list>
		</property>
	</bean>
</beans>
```

## Case-2

error_en_IN.properties
```
errorMessage=Password must be contain number!!
```

message_en_IN.properties
```
welcomeMessage=Dhananjaya welcomes you!!
welcomeMessageWithName={0} welcomes {1} !!
```

```
public class i18NTestBeanFactory {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/i18n/common/application-context.xml"));
		MessageSource message=factory.getBean("messageSource",ResourceBundleMessageSource.class);
		System.out.println(message.getMessage("welcomeMessageWithName",new Object[]{"Dj","You"},"Not a valid key",Locale.getDefault()));
		//System.out.println(message.getMessage("welcomeMessageWithName1",null,"Not a valid key",Locale.getDefault()) );
	}
}
```

We can dynamically pass the message value to the **message_en_IN.properties** file by using object[]{} as 2nd argument

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

	<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
		<property name="basenames">
			<list>
				<value>message</value>
				<value>error</value>
			</list>
		</property>
	</bean>
</beans>
```

## Case-3

➔If we are using ReloadableResourceBundleMessageResource then we can handle the cache memory timing

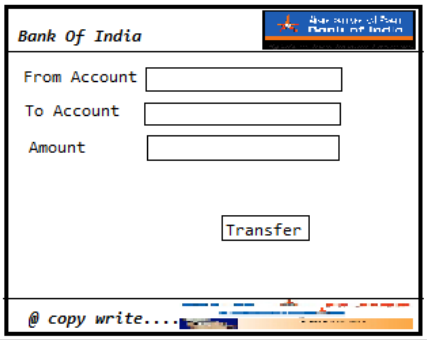
```
public class i18NTest {
	public static void main(String[] args) {
		ApplicationContext context=new ClassPathXmlApplicationContext("com/i18n/common/application-context.xml");
		MessageSource message=context.getBean("messageSource",ReloadableResourceBundleMessageSource.class);
		System.out.println(message.getMessage("welcomeMessageWithName",new Object[]{"Dj","You"},"Not a valid key",Locale.getDefault()));
		//System.out.println(message.getMessage("welcomeMessageWithName1",null,"Not a valid key",Locale.getDefault()) );
	}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

	<bean id="messageSource" class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
		<property name="basenames">
			<list>
				<value>message</value>
				<value>error</value>
			</list>
		</property>
		<property name="cacheSeconds" value="1"/>
	</bean>
</beans>
```

## Lookup-Method Injection

**Q.How many object will going to created by servletContainer?**
**Q. 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 servlet Container will going to create new object for every request then for 1000 request, 1000 object has to created by servlet Container. It will cause big performance issue and JVM will fill up with objects only.
➔ Because of above problem servlet Container will going to create only one object only and that object will shared to all the corresponding requests.
➔ Actually internally **servet Cotainer will call run() method which will internally going to call the service() method of servlet, and it will generate one thread for per request**.
➔ servlet Container class is not a singleton class but its behave like singleton.
➔ **servlet Container will generate multiple thread for per request but there is a problem if object state is none-sharable then data inconsistency will encounter.**




```
public class TransferFormServlet extends HttpServlet {

	private String fromAccount;
	private String toAccount;
	private double amount;
	protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		//receive data from html page

		//business operation

		//display success page
	}

}
```

➔ As per the above example servlet Container will create one thread for first request and that thread will start execution of servlet.
➔ It read the fromAccount, toAccount and amount and performs the operation,but while performing business operation new request came and servlet Container 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 will gets erase.
➔ Now how we can make our application works with multithread 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 a 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 problem occurred.
**➔To remove the problem just remove the object state and place it in to method level.**

```
public class TransferFormServlet extends HttpServlet {
	protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		String fromAccount;
		String toAccount;
		double amount;

		//receive data from html page

		//business operation

		//display success page
	}

}
```

➔ Now see the above all attributes are available into service () method.
➔ When first thread will create then all attributes will assigned with their 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 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 were it will store all the information, to restore the execution at same line with same data.**

---

➔ 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 methods which need same data to perform the business logic.
➔ But when I am declaring attributes in method level it is seem to be local to the method only. It is not accessible to any other method.
➔ 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. Because in multi-thread environment it will allow only one thread to access that method so performance impact will arise.

## ThreadLocal

**Q. we cannot declare attribute at object level, we cannot declare at method level and we cannot 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 threads 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 Thread Local.**

➔ But Thread Local class will allow us to store only one value in it. We can set the value to the Thread local.

Ex: **ThreadLocal<integer> threadLocal = new ThreadLocal<Integer>();**

- ⇨ public void set(Object obj);
- ⇨ public Object get();

By this we can solve the above problems let's see the example.

```
public class Test {
	public static void main(String[] args) throws InterruptedException {
		MyThread r1=new MyThread();
		MyThread r2=new MyThread();
		Thread t1=new Thread(r1);
		Thread t2=new Thread(r2);
		t1.start();
		Thread.sleep(400);
		t2.start();
	}
}
```

```
public class MyThread implements Runnable {
	private CalculateTable cal;

	public MyThread() {
		cal=CalculateTable.getInstance();
	}
	@Override
	public void run() {
			cal.setTableNo(new Random().nextInt(10));
			cal.show();
	}
}
```

```
public class CalculateTable {
	private  static CalculateTable instance;
	private int tableNo;
	private ThreadLocal<Integer>tl;
	public static  CalculateTable getInstance(){
		if (instance==null) {
			instance=new CalculateTable();
		}
		return instance;
	}
	private CalculateTable(){
		tl=new ThreadLocal<Integer>();
	}

	public void setTableNo(int tableNo) {
		tl.set(tableNo);
		this.tableNo = tableNo;
	}
	public void  show(){

		for (int i = 1; i < 10; i++) {
			try {
				Thread.sleep(300);
				System.out.println(Thread.currentThread().getName()+":"+(tl.get()+i));
			} catch (Exception e) {
			}
		}
	}
}
```

| Target | Dependant | Remarks |
|---|---|---|
| Singleton | SingleTon | Allow |
| Prototype | Prototype | Allow |
| Singleton | Prototype | Not Thread safe |
| Prototype | Singleton | Allow |

**➔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 works**
**➔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 let's see the reason with an example

```
public class A {
	private B b;

	public void setB(B b) {
		this.b = b;
	}
	public void show(){
		System.out.println(b);
	}
}
```

```
public class B {
	private int i;

	public void setI(int i) {
		this.i = i;
	}

	@Override
	public String toString() {
		return "B [i=" + i + "]";
	}
}
```

```
<bean id="a" class="com.lum.beans.A" scope="singleton">
	<property name="b" ref="b"/>
</bean>
<bean id="b" class="com.lum.beans.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 an IOC container using BeanFactory will place both the classes inside 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 limitations with injection.**
**➔ Every time we cannot use injection to manage the dependency some time we need lookup injection also.**

➔ One of the core features 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
- ⇨ Dependency injection

➔ Mostly people use dependency injection for managing the dependency, but there are some limitations are available.

- ⇨ **We cannot use prototype scope bean in to singleton scope, because that prototype class behave like singleton only.**
- ⇨ **We cannot 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.

➔ 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.

```
public interface IEngine {
        int startup();
}
```

```
public class YamahaEngineImpl implements IEngine{
        @Override
        public int startup() {
                return 1 ;
        }
}
```

```
public class SuzukiEngineImpl implements IEngine {
        @Override
        public int startup() {
                return 1;
        }
}
```

```
public class Car {
        private String beanId;
        public Car(String beanId) {
                this.beanId=beanId;
        }
        public void drive(){
                int status=0;
                IEngine engine=null;
                BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/idref/common/application-context.xml"));
                engine=factory.getBean(beanId,IEngine.class);
                status=engine.startup();
                if (status==1) {
                        System.out.println(beanId+"\t Engine Started");
                }else{
                        System.out.println("Engine start failed");
                }
        }
}
```

```
<beans>
        <bean id="car" class="com.idref.beans.Car">
                <constructor-arg>
                        <idref bean="suzukiEngine"/>
                </constructor-arg>
        </bean>
        <bean id="yamahaEngine" class="com.idref.beans.YamahaEngineImpl"/>
        <bean id="suzukiEngine" class="com.idref.beans.SuzukiEngineImpl"/>
</beans>
```

➔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.

```
public interface Engine {
        int start();
}
```

```
public class SuzukiEngine implements Engine {
        @Override
        public int start() {
                System.out.println("Engine Started...");
                return 1;
        }
}
```

```
public class YamahaEngine implements Engine {
        @Override
        public int start() {
                System.out.println("Engine Started....");
                return 1;
        }
}
```

```
public class Car implements BeanFactoryAware {
        private Engine engine;
        private BeanFactory factory;
        private String beanName;

        public Car() {
                System.out.println("Constructor.....");
        }
        public void setBeanName(String beanName) {
                System.out.println("SetterBeanName().....");
                this.beanName = beanName;
        }

        @Override
        public void setBeanFactory(BeanFactory factory) throws BeansException {
                System.out.println("setBeanFactory()....");
                this.factory=factory;
        }

        public void drive(){
                engine=factory.getBean(beanName,Engine.class);

                if (engine.start()==1) {
                        System.out.println("Driving.....");
                }
                else{
                        System.out.println("Engine faild to start...");
                }
        }
}
```

O/P

```
Constructor.....
SetterBeanName().....
setBeanFactory()....
Engine Started....
Driving.....
```

```
public class Test {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bfa/common/application-context.xml"));
                Car car=factory.getBean("car",Car.class);
                car.drive();
        }
}
```

```
<bean id="car" class="com.bfa.beans.Car">
        <property name="beanName">
                <idref bean="yamahaEngine"/>
        </property>
</bean>
<bean id="suzukiEngine" class="com.bfa.beans.SuzukiEngine"/>
<bean id="yamahaEngine" class="com.bfa.beans.YamahaEngine"/>
```

➔As per above example we make our code somehow loosely coupled but it's not totally loosely coupled, still my drive() 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.

```
public abstract class Car {
        private Engine engine;

        public abstract Engine getEngine();

        public void drive(){
                engine=getEngine();

                if (engine.start()==1) {
                        System.out.println("Driving.....");
                }
                else{
                        System.out.println("Engine faild to start...");
                }
        }
}
```

```
<bean id="car" class="com.bfa.beans.Car">
                <lookup-method name="getEngine" bean="suzukiEngine"/>
        </bean>
        <bean id="suzukiEngine" class="com.bfa.beans.SuzukiEngine"/>
        <bean id="yamahaEngine" class="com.bfa.beans.YamahaEngine"/>
```

```
public class Test {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bfa/common/application-context.xml"));
                Car car=factory.getBean("car",Car.class);
                car.drive();
        }
}
```

```
public interface Engine {
        int start();
}
```

```
public class SuzukiEngine implements Engine {
        @Override
        public int start() {
                System.out.println("Engine Started...");
                return 1;
        }
}
```

```
public class YamahaEngine implements Engine {
        @Override
        public int start() {
                System.out.println("Engine Started....");
                return 1;
        }
}
```

## Internals

**➔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.**

```
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 looks at the lookup configuration and it will inject the corresponding bean to the corresponding method.
➔ In generate we cannot configure abstract class as bean, but when we are dealing with look-up method injection then only it is possible because of look-up method configuration.

➔ While configuring that class and method into IOC container those should not be final and static.

```
public abstract class Container {
        private RequestHandler requestHandler;
        public abstract  RequestHandler getHandler();

        public void getData(String data){
                requestHandler=getHandler();
                requestHandler.setData(data);
                System.out.println(requestHandler.request(data));
        }
}
```

```
public class RequestHandler {
        private String data;
        public void setData(String data) {
                this.data = data;
        }
        public String request(String data){
                return data+"=="+this.hashCode();
        }
}
```

```
public class Test {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/lum/common/application-context.xml"));
                Container container=factory.getBean("container",Container.class);
                container.getData("Data-1");
                container.getData("Data-2");
                container.getData("Data-3");
        }
}
```

```
<bean id="container" class="com.lum.beans.Container">
        <lookup-method name="getHandler" bean="requestHandler"/>
</bean>
<bean id="requestHandler" class="com.lum.beans.RequestHandler" scope="prototype"/>
```

**➔Now the problem is solved class RequestHandler scope is prototype even though we will get same object of RequestHandler class, means if we inject prototype class into singleton class then prototype nature will not change when we use lookup-method injection.**

## BeanFactoryPostProcessor

➔If we want to perform some post processing on BeanFactory after it has been created and before it instantiate the beans then we need to go for BeanFactoryPostProcessor.

**Use-Case:**

### Bug handler process

| Status | Description |
|---|---|
| **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 environment 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.<br>⇨ 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 it's 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 doesn't have authority to deploy the project in QA environment.

➔One of the QA engineer only can deploy the project into QA environment 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 environment 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. So Again there are no. of problems are their

**What are the reasons developer will not get permission to access the QA environment ?**

➔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 instalment procedure by document?**

➔Actually developer never get a chance to deploy the project into the QA environment But QA people also unable to deploy the project because 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 check out 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 environment. 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.

➔ To overcome the above problems building tools came into the market.

There are some building tools

**Maven**

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

**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.
- ⇨ For example a project has been developed by the developer along with that, one of the developers has written building script for ant tools, for making deployment automated.
- ⇨ But as per the above discussion tools also has problems.
- ⇨ When we are using Maven or ANT if we want to modify the configuration again it demands the rebuild and redeployment
- ⇨ **To overcome the above tools problems spring has provided as feature called BeanFactoryPostProcessor.**

➔ When they run the build tool, tool will deploy the project with wrong configuration which leads big 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.

## BeanFactoryPostProcesser

➔ 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 BeanFactoryPostProcesser.**
**For example:**
➔If we want to change the configuration , we can always change it by modifying the physical configuration file. Instead of if we want to modify the configuration dynamically at runtime within the IOC container metadata then we can use BeanFactoryPostProcessor.
➔When we are using Maven or ANT if we want to modify the configuration again it demands the rebuild and redeployment.
➔ Spring has provided an interface called BeanFactoryPostProcessor By help this interface we can write the class which **implements BeanfactoryPostprocessor** and we will get one method called **postProcessBeanFactory(ConfigurableListableBeanFactory object){..}**
➔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 configuration 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.**
➔ Then only IOC container automatically update place holder with property file values.
➔**PropertyPlaceHolderConfigurer is a post processor which will reads the message from the properties file and replace the ${..} place holders of a bean configuration after the BeanFactory has loaded it.**

---

➔In below example The connectionManager will get create with driverClassName,url, username, password with ${} token values,
➔Instead of this if we want to replace the ${} values with property file key and values, we need to configure PropertyPlaceHolderConfigurer.

➔This Post Processor after loading the configuration by BeanFactory and before factory creates ConnectionManager bean, it will replace the ${} values with property key value.

**Example:**

```
public class ConnectionManager {
        private String driverClassName;
        private String url;
        private String username;
        private String password;
        public void setDriverClassName(String driverClassName) {
                this.driverClassName = driverClassName;
        }
        public void setUrl(String url) {
                this.url = url;
        }
        public void setUsername(String username) {
                this.username = username;
        }
        public void setPassword(String password) {
                this.password = password;
        }
        public Connection getConnection(){
                Connection con=null;
                try {
                        Class.forName(driverClassName);
                        con=DriverManager.getConnection(url,username,password);
                } catch (SQLException | ClassNotFoundException e) {
                        e.printStackTrace();
                }
                return con;
        }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
        <bean id="connectionManager" class="com.bfpp.beans.ConnectionManager">
                <property name="driverClassName" value="${db.driverClassName}"/>
                <property name="url" value="${db.url}"/>
                <property name="username" value="${db.username}"/>
                <property name="password" value="${db.password}"/>
        </bean>

        <bean id="bfpp" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
                <property name="location" value="classpath:db.properties"/>
        </bean>
</beans>
```

file:c:/work/properties/db.properties

Absolute Path

Relative Path

**db.properties**

```
db.driverClassName=oracle.jdbc.OracleDriver
db.url=jdbc:oracle:thin:@localhost:1521:xe
db.username=system
db.password=manager
```

```
public class BFPPTest {
        public static void main(String[] args) throws SQLException {
                //Creating BeanFactory
                BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bfpp/common/application-context.xml"));

                //Applying logic before creating the bean object
                BeanFactoryPostProcessor processor=factory.getBean("bfpp",BeanFactoryPostProcessor.class);
                processor.postProcessBeanFactory((ConfigurableListableBeanFactory)factory);

                //Creating bean object
                ConnectionManager manager=factory.getBean("connectionManager",ConnectionManager.class);
                Connection con=manager.getConnection();
                Statement st=con.createStatement();
                ResultSet rs=st.executeQuery("select * from project");
                while(rs.next()){
                        System.out.println("["+rs.getString(1)+","+rs.getString(2)+","+rs.getString(3)+","+rs.getString(4)+","+rs.getString(5)+"]");
                }
        }
}
```

## Bean PostProcessor

➔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 behaviour 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.
➔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.
➔Instead of writing the same sort of logic in every class we should write the logic in BeanPostProcessor.
➔For example **how many object has created by IOC Container** if we want to know the see the example.

```
public class A {
        private B b;
        public A(B b) {
                System.out.println("A Object Created");
                this.b = b;
        }
}
```

```
public class B {
        private C c;
        public B(C c) {
                System.out.println("B Object Created");
                this.c = c;
        }
}
```

```
public class C {
        public C() {
                System.out.println("C Object Created");
        }
}
```

```
import java.util.concurrent.atomic.AtomicInteger;
public class InstanceStaticstics {
        private static AtomicInteger instances=
                                new AtomicInteger();
        public static void increment(){
                instances.getAndIncrement();
        }
        public static int getInstances(){
                return instances.get();
        }
}
```

```
public class InstanceTrackerBeanPostProcessor implements BeanPostProcessor {
        @Override
        public Object postProcessAfterInitialization(Object bean, String beanName)
                        throws BeansException {
                InstanceStaticstics.increment();
                return bean;
        }
        @Override
        public Object postProcessBeforeInitialization(Object bean, String beanName)
                        throws BeansException {
                return bean;
        }
}
```

```
public class BPPTest {
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bpp/common/application-context.xml"));
                BeanPostProcessor bpp=factory.getBean("instanceTrackerBeanPostProcessor",InstanceTrackerBeanPostProcessor.class);
                ((ConfigurableListableBeanFactory)factory).addBeanPostProcessor(bpp);
                A a=factory.getBean("a",A.class);
                System.out.println("Total Object Created:"+InstanceStaticstics.getInstances());
        }
}
```

```
<beans>
        <bean id="a" class="com.bpp.beans.A">
                <constructor-arg ref="b"/>
        </bean>
        <bean id="b" class="com.bpp.beans.B">
                <constructor-arg ref="c"/>
        </bean>
        <bean id="c" class="com.bpp.beans.C"/>
        <bean id="instanceTrackerBeanPostProcessor" class="com.bpp.util.InstanceTrackerBeanPostProcessor"/>
</beans>
```

```
O/P
C Object Created
B Object Created
A Object Created
Total Object Created:3
```

**Atomic Integer**

➔Present in java.util.concurrent.atomic package
➔It is the Synchronized version of Integer introduced in java 1.6v
➔It has provided number of methods which make programmer life easy.

**One of the use case we can use BeanPostProcessor i.e. Auditing.**

**What is mean by Audit?**

➔In business data matter a lot, it's 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 is used for security purpose also.
➔By this we can track login details and performed activities.
➔In industry every employee has their 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.
➔Every typical application has to provide auditing as part of application. For example in bank applications, they have 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**.
➔Audit plays vital role in industry.

## Employee Management Service

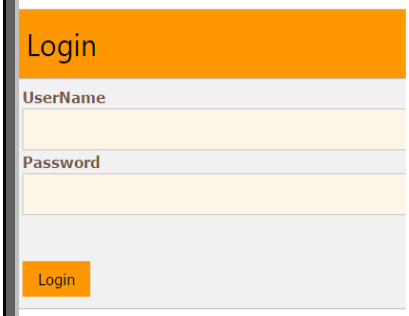
*Show Employees* Register Employee Update Employee Audit Report Track Admin Logout

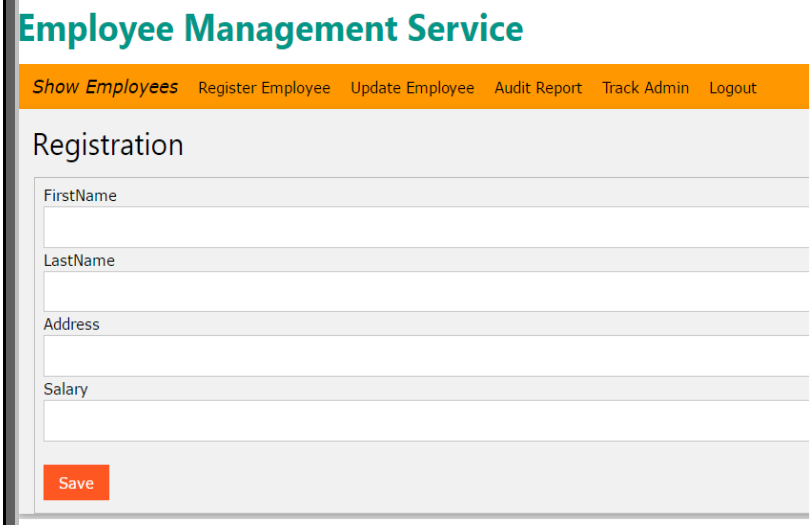
| EMPLOYEE-ID | EMPLOYEE-NAME | EMPLOYEE-ADDRES | EMPLOYEE-SALARY | EMPLOYEE-CREATEDDATE | EMPLOYEE-CREATEDBY | EMPLOYEE-MODIFIEDDATE | EMPLOYEE-MODIFIEDBY |
|---|---|---|---|---|---|---|---|
| 1000 | Dhananjaya Samanta singhar | Bhubaneswar | 50000 | 2017-02-03 16:32:00.0 | user1 | 2017-02-03 16:34:12.0 | user2 |
| 1001 | Kamlesh Ray | Hyderabad | 50000 | 2017-02-03 16:44:43.0 | user4 | 2017-02-03 16:44:43.0 | user4 |
| 1002 | Ashirbad Rout | Bhubaneswar | 600000 | 2017-02-03 16:46:13.0 | user4 | 2017-02-03 16:54:13.0 | user4 |
| 1003 | Swarup Nayak | Hyderabad | 6900 | 2017-02-03 17:48:45.0 | user2 | 2017-02-03 17:48:45.0 | user2 |
| 1004 | swarup nayak | hyderabad | 80000 | 2017-02-03 17:49:33.0 | user2 | 2017-02-03 17:51:29.0 | user2 |

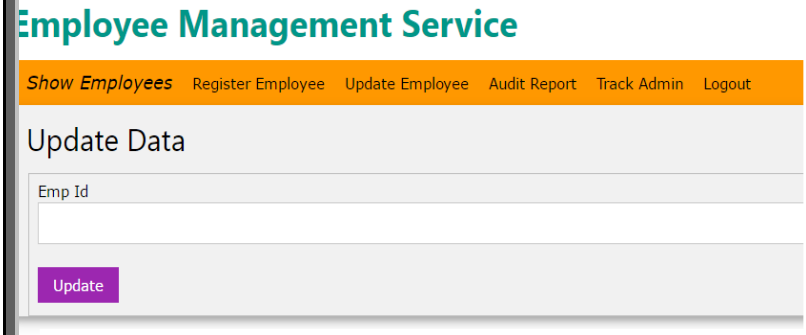
## Employee Management Service

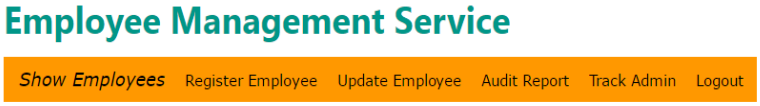
*Show Employees* Register Employee Update Employee Audit Report Track Admin Logout

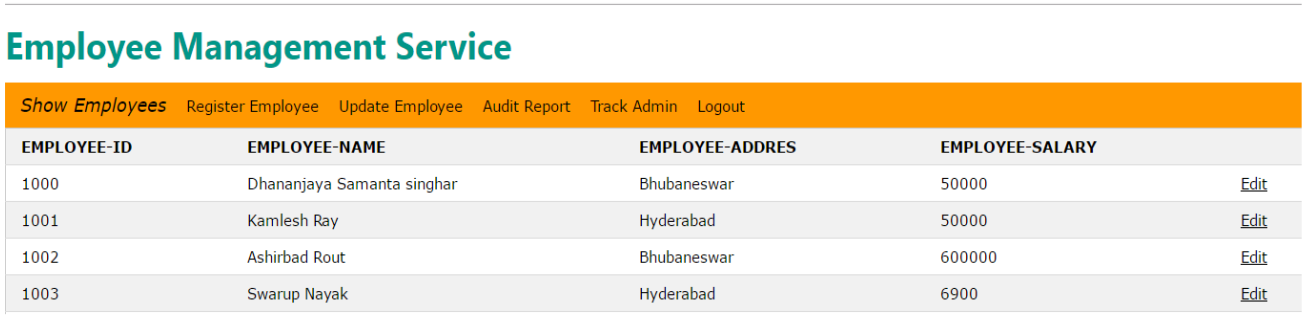
| EMPLOYEE-ID | OPERATION-TYPE | MODIFIED COLUMN | OLD VALUE | NEW VALUE | CREATED BY | CREATED DATE | LAST MODIFIED BY | LAST MODIFIED DATE |
|---|---|---|---|---|---|---|---|---|
| 1000 | insert | All | NA | NA | user1 | 03-02-17 | user1 | 03-02-17 |
| 1000 | update | SALARY | 45000 | 50000 | user1 | 2017-02-03 16:32:00.0 | user1 | 03-02-17 |
| 1000 | update | LAST NAME | Samantas | Samanta singhar | user1 | 2017-02-03 16:32:00.0 | user2 | 03-02-17 |
| 1001 | insert | All | NA | NA | user4 | 03-02-17 | user4 | 03-02-17 |
| 1002 | insert | All | NA | NA | user4 | 03-02-17 | user4 | 03-02-17 |
| 1002 | update | SALARY | 50000 | 600000 | user4 | 2017-02-03 16:46:13.0 | user4 | 03-02-17 |
| 1003 | insert | All | NA | NA | user2 | 03-02-17 | user2 | 03-02-17 |
| 1004 | insert | All | NA | NA | user2 | 03-02-17 | user2 | 03-02-17 |
| 1004 | update | SALARY | 8000 | 80000 | user2 | 2017-02-03 17:49:33.0 | user2 | 03-02-17 |


Login
UserName
Password
Login


Employee Management Service
Show Employees
Register Employee
Update Employee
Audit Report
Track Admin
Logout
Registration
FirstName
LastName
Address
Salary
Save


Employee Management Service
Show Employees
Register Employee
Update Employee
Audit Report
Track Admin
Logout
Update Data
Emp Id
Update


Employee Management Service
Show Employees
Register Employee
Update Employee
Audit Report
Track Admin
Logout


Employee Management Service
Show Employees
Register Employee
Update Employee
Audit Report
Track Admin
Logout
EMPLOYEE-ID
EMPLOYEE-NAME
EMPLOYEE-ADDRES
EMPLOYEE-SALARY
1000
Dhananjaya Samanta singhar
Bhubaneswar
50000
Edit
1001
Kamlesh Ray
Hyderabad
50000
Edit
1002
Ashirbad Rout
Bhubaneswar
600000
Edit
1003
Swarup Nayak
Hyderabad
6900
Edit

## Insert Logic for Track Record

```
public static String insert(String firstname,String lastname,String address,int salary,String createdBy ){
      Connection con=ConnectionManager.getConnection();
      String status=null;
      int count=0;
      try {
            Statement st=con.createStatement();
            ResultSet rs=st.executeQuery("select count(*) from audit_emp");
            if (rs.next()) {
                  count=rs.getInt(1)+1000;
            }

            st.executeUpdate("insert into audit_emp values("+count+",'"+firstname+"','"+lastname+"','"+address+"',
                              "+salary+",SYSDATE,'"+createdBy+"',SYSDATE,'"+createdBy+"')");

            status="inserted";

      } catch (Exception e) {
            status="failed";
      }
      audit_insert(count, "insert", "NA", "NA", createdBy, "SYSDATE", createdBy, "SYSDATE");
      return status;
}
```

```
public static void  audit_insert(int emp_Id,String operation_Type , String oldValue,String newValue,String
                                    createdBy,String createdDate,String modifiedBy,String modifiedDate){
      try {
            Connection con=ConnectionManager.getConnection();
            Statement st=con.createStatement();
            st.executeUpdate("insert into audit_tracking values("+emp_Id+",'"+operation_Type+"','"+oldValue+"','"
                        +newValue+"','"+createdBy+"',SYSDATE,'"+modifiedBy+"',SYSDATE,'All')");
      } catch (SQLException e) {
            e.printStackTrace();
      }
   }
```

## Update Logic for Track Record

```
public static String update(int empId,String firstname,String lastname,String address,int salary,String user){
      Connection con=ConnectionManager.getConnection();
      AuditDetails details=AuditAccessor.fetchAuditEmp(empId);
      String status=null;
      try {
            Statement st=con.createStatement();
            st.executeUpdate("update audit_emp set FIRSTNAME='"+firstname+"',LASTNAME='"+lastname+"',
                        ADDRESS='"+address+"',SALARY="+salary+" ,MODIFIEDDATE=SYSDATE,MODIFIEDBY
                                    ='"+user+"' where EMP_ID="+empId+" ");
      status="updated";
      String modifiedValue=null;
      if (!details.getFirstName().equals(firstname)) {
            audit_update(empId, "update", details.getFirstName(), firstname,details.get
                        CreatedBy(),details.getCreatedDate(),user,"","
                              FIRST NAME");
      }
      if (!details.getLastName().equals(lastname)) {
            audit_update(empId, "update", details.getLastName(),lastname,details.getCreatedBy(),
                                    details.getCreatedDate(),user,"","LAST NAME");
      }
      if (!details.getAddress().equals(address)) {
            audit_update(empId, "update", details.getAddress(),address,details.getCreatedBy(),
                        details.getCreatedDate(),user,"","ADDRESS");
      }
      if (!(details.getSalary()==salary)) {
            audit_updatesal(empId, "update", details.getSalary(),salary,details
                        .getCreatedBy(),details.getCreatedDate(),user,"","SALARY");
      }
      } catch (Exception e) {
            status="failed";
      }
      return status;
   }
```

```
public static void  audit_update(int emp_Id,String operation_Type , String oldValue,String newValue,String createdBy,String
                                    createdDate,String modifiedBy,String modifiedDate,String modColumn){
    try {
        Connection con=ConnectionManager.getConnection();
        Statement st=con.createStatement();
        st.executeUpdate("insert into audit_tracking
                          values("+emp_Id+",'"+operation_Type+"','"+oldValue+"','"+newValue+"','"+createdBy+"',
                        '"+createdDate+"','"+modifiedBy+"',SYSDATE,'"+modColumn+"')");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

```
public static void  audit_updatesal(int emp_Id,String operation_Type , int oldValue,int newValue,String createdBy
                ,String createdDate,String modifiedBy,String modifiedDate,String modColumn){

    try {
        Connection con=ConnectionManager.getConnection();
        Statement st=con.createStatement();
        st.executeUpdate("insert into audit_tracking
                          values("+emp_Id+",'"+operation_Type+"','"+oldValue+"','"+newValue+"','"+createdBy+"'
                          ,'"+createdDate+"','"+modifiedBy+"',SYSDATE,'"+modColumn+"')");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

➔**BeanPostProcessor** will help in sharing the common behaviour and we can restrict that behaviour for specific bean also. Which will go to create inside the IOC container.

➔For easy understanding let's see one of the example how we can restrict the behaviour for specific bean.

```
public class ObjectVO {
        protected Date createDate;
        protected String createBy;
        protected Date lastModifiedDate;
        protected String lastModifiedBy;
                //setters & getters
}
```

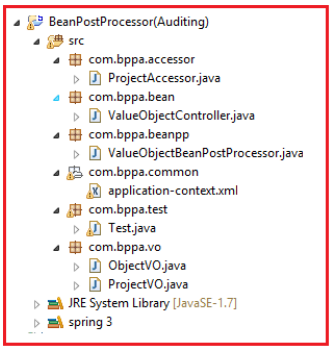
```
public class ProjectAccessor {
        public static void insert(ProjectVO project){
                //database logic to store in database
                System.out.println(project);
        }
}
```

```
public class ProjectVO extends ObjectVO{
        private int projectId;
        private String projectName;
        private String title;
        private String status;
        //Setters & Getters
}
```

```
<beans>
        <bean id="valueObjectController" class="com.bppa.bean.ValueObjectController">
                <lookup-method name="getProjectVO" bean="projectVO"/>
        </bean>
        <bean id="projectVO" class="com.bppa.vo.ProjectVO" scope="prototype"/>
        <bean id="valueObject" class="com.bppa.beanpp.ValueObjectBeanPostProcessor"/>
</beans>
```

```
public abstract class ValueObjectController {
        public void getValue(int projectId,String projectName,String title,String status){
                ProjectVO project=getProjectVO();
                project.setProjectId(projectId);
                project.setProjectName(projectName);
                project.setTitle(title);
                project.setStatus(status);

                project.setCreateBy("user1");
                project.setLastModifiedBy("user4");
                ProjectAccessor.insert(project);
        }
        public abstract ProjectVO getProjectVO();
}
```

```
public class ValueObjectBeanPostProcessor implements BeanPostProcessor {
        @Override
        public Object postProcessAfterInitialization(Object bean, String beanName)throws BeansException {
                if (bean instanceof ObjectVO) {
                        ((ObjectVO)bean).setCreateDate(new Date());
                        ((ObjectVO)bean).setLastModifiedDate(new Date());
                }
                return bean;
        }
        @Override
        public Object postProcessBeforeInitialization(Object bean, String beanName)
                        throws BeansException {
                return bean;
        }
}
```




```
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bppa/common/application-context.xml"));

                BeanPostProcessor processor=factory.getBean("valueObject",BeanPostProcessor.class);
                ((ConfigurableListableBeanFactory)factory).addBeanPostProcessor(processor);

                ValueObjectController con=factory.getBean("valueObjectController",ValueObjectController.class);
                con.getValue(111, "realspeed", "cipla", "in progress");
        }
}
```

In ValueObjectPostProcessor always createdDate (...) and modifiedDate (..) will inject automatically pre construct and post construct of each and every bean which is instance of ObjectVO

## Depends-On

**When we need to go for file system? When we need to go for Properties file? When we need to go for Database?**

➔If our requirement is there is some text data which we wanted to read and use it means these are like paragraph text data then we should go for file system.
➔If the data we have has multiple values then we can't distinguish between separate peace of data if we are using file system then we should go for Properties file so that we can read the data by using key or identifier.
➔The data which we want to store is complex contains set of values then it is not recommended to go for Properties file rather go for Database so that we can easily operation (like validate) on that data.
➔If we have a complex filter mechanism to access the records of data then we should go for Database
➔If we want the relation between the data then we should go for Database

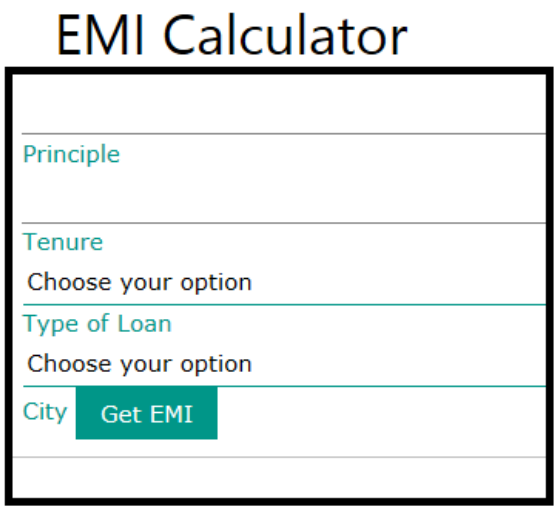
## Cache

➔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 share 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.
➔Because data remain same for every request then there is no need to create a 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 share the state of the object in it.
➔Cache can has multiple methods to add, retrieve and to check the data.
➔Cache should not contains business related logic.
➔If 10 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.

## Optimize-1




```
<body>
    <h1>EMI Calculator</h1>
    <form action="get">
        <fieldset>
            Principle<input type="text" name="principle"><br><br>

            Tenure<input type="text" name="tenure"><br><br>

            Type of Loan
            <select name="type">
              <option value="" disabled selected>Choose your option</option>
              <option value="homeloan">Home Loan</option>
              <option value="carloan">Car Loan</option>
              <option value="personalloan">Personal Loan</option>
            </select><br><br>

            City
            <select name="city">
              <option value="" disabled selected>Choose your option</option>
              <option value="hyderabad">Hyderabad</option>
              <option value="chennai">Chennai</option>
              <option value="bangalore">Bangalore</option>
            </select><br><br>
            <input  type="submit" value="Get EMI">
        </fieldset>
    </form>
</body>
```

```
public class GetLoanDetails extends HttpServlet {
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        int principle=Integer.parseInt(request.getParameter("principle"));
        int tenure=Integer.parseInt(request.getParameter("tenure"));
        String type=request.getParameter("type");
        String place=request.getParameter("city");
        Cache cache=Cache.getInstance();
        Properties props=null;
        PrintWriter out=response.getWriter();

        if (cache!=null) {
            if (cache.containsKey("cities")) {
                props= (Properties) cache.get("cities");
            }
            else{
                props=new Properties();
                props.load(this.getClass().getClassLoader().getResourceAsStream("city.properties"));
                cache.put("cities", props);
                props=(Properties) cache.get("cities");
            }
            if (props!=null) {
                if (props.contains(place)) {
                    double interestRate =Double.parseDouble(props.getProperty(place));
                    double totalAmt=(principle*interestRate/100)* tenure;
                    out.println("<h1><b>Interest Rate: "+interestRate+ " \t and Total Amount in  "+tenure+" months is: "+totalAmt+"<b></h1>");
                }
            }
        }
    }
}
```

```
public class Cache {
    private static Cache instance;
    private Map<String,Object> dataMap;
    private Cache(){
        dataMap=new ConcurrentHashMap<String,Object>();
    }
    public static synchronized Cache getInstance(){
        if (instance==null) {
            instance=new Cache();
        }
        return instance;
    }
    public  void put(String key,Object value){
        System.out.println("Stoting in Cache");
        dataMap.put(key, value);

    }
    public Object get(String key){
        System.out.println("Getting from Cache");
        return dataMap.get(key);

    }
    public  boolean containsKey(String key){
        System.out.println("Check in cache");
        return dataMap.containsKey(key);
    }
}
```

city.properties

```
hyderabad=11.25
chennai=12.35
bangalore=10.19
```

## Optimize-2

```
public class Cache {
        private static Cache instance;
        private Map<String,Object> dataMap;
        private Properties props;
        private Cache(){
                dataMap=new ConcurrentHashMap<String,Object>();
                props=new Properties();
                try {
                        props.load(this.getClass().getClassLoader().getResourceAsStream("city.properties"));
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
        public Properties getProperties(){
                return props;
        }
        public static synchronized Cache getInstance(){}
        public  void put(String key,Object value){}
        public Object get(String key){}
        public  boolean containsKey(String key){}
}
```

```
public class GetLoanDetails extends HttpServlet {
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        int principle=Integer.parseInt(request.getParameter("principle"));
        int tenure=Integer.parseInt(request.getParameter("tenure"));
        String type=request.getParameter("type");
        String place=request.getParameter("city");
        Cache cache=Cache.getInstance();
        Properties props=null;
        PrintWriter out=response.getWriter();
        if (cache!=null) {
            if (cache.containsKey("cities")) {
                props=(Properties) cache.get("cities");
            }else{
                props=cache.getProperties();
                cache.put("cities", props);
                props= (Properties) cache.get("cities");
            }
            if (props!=null) {
                if (props.containsKey(place)) {
                    double interestRate =Double.parseDouble(props.getProperty(place));
                    double totalAmt=(principle*interestRate/100)* tenure;
                    out.println("<h1><b>Interest Rate: "+interestRate+ " \t and Total Amount in  "+tenure+" months is: "+totalAmt+"<b></h1>");
                }
            }
        }
    }
}
```

### 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, because cache is the one of the place where everyone 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.

**Optimize-3**

```
public class Cache {
	private static Cache instance;
	private Map<String,Object> dataMap;
	private Properties props;
	private Cache(){
		dataMap=new ConcurrentHashMap<String,Object>();
		init();
	}
	public void init(){
		props=new Properties();
		try {
			props.load(this.getClass().getClassLoader().getResourceAsStream("city.properties"));
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
	public static synchronized Cache getInstance(){}

	public Properties getProperties(){
		return props;
	}
	public  void put(String key,Object value){}
	public Object get(String key){}
	public  boolean containsKey(String key){}
}
```

➔ So up to this we got we should not write the loading logic into the cache.

➔ But we have to write logic somewhere else where all the application can use it.

➔ So CacheManager came to picture.

**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

**Optimize-4**

```
public class GetLoanDetails extends HttpServlet {
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        int principle=Integer.parseInt(request.getParameter("principle"));
        int tenure=Integer.parseInt(request.getParameter("tenure"));
        String type=request.getParameter("type");
        String place=request.getParameter("city");
        Cache cache=Cache.getInstance();
        Properties props=null;
        PrintWriter out=response.getWriter();
        if (cache!=null) {
            if (cache.containsKey("cities")) {
                props=(Properties) cache.get("cities");
            }else{
                props=cache.getProperties();
                cache.put("cities", props);
                props= (Properties) cache.get("cities");
            }
            if (props!=null) {
                if (props.containsKey(place)) {
                    double interestRate =Double.parseDouble(props.getProperty(place));
                    double totalAmt=(principle*interestRate/100)* tenure;
                    out.println("<h1><b>Interest Rate: "+interestRate+ " \t and Total Amount in  "+tenure+" months is: "+totalAmt+"<b></h1>");
                }
            }
        }
    }
}
```

```
public class Cache {
    private static Cache instance;
    private Map<String,Object> dataMap;
    private Properties props;
    private CacheManager cacheManager;
    private Cache(){
        dataMap=new ConcurrentHashMap<String,Object>();
        cacheManager=CacheManager.getInstance();
        init();
    }
    public static synchronized Cache getInstance(){
        if (instance==null) {
            instance=new Cache();
        }
        return instance;
    }
    public void  init(){
        dataMap=cacheManager.getAllData();
        props=(Properties)dataMap.get("props");
        //Get Database data
        //Get XML Data
        //Get FileSystem Data
    }
    public Properties getProperties(){
        return props;
    }
    public  void put(String key,Object value){
        System.out.println("Stoting in Cache");
        dataMap.put(key, value);

    }
    public Object get(String key){
        System.out.println("Getting from Cache");
        return dataMap.get(key);
    }
    public  boolean containsKey(String key){
        System.out.println("Check in cache");
        return dataMap.containsKey(key);
    }
}
```

```
public class CacheManager {
    private static CacheManager instance;
    private Map<String,Object> dataMap=null;
    private CacheManager() {
            dataMap=new HashMap<String, Object>();
    }
    public static synchronized CacheManager getInstance(){
        if (instance==null) {
            instance=new CacheManager();
        }
        return instance;
    }
    public Map<String,Object> getAllData(){

        Map<String,Object> data=new HashMap<String, Object>();
        try {
            Properties props=null;
            props=new Properties();
            props.load(this.getClass().getClassLoader().getResourceAsStream("city.properties"));
            data.put("props",props);

            /*
            DataBase Logic for retrive the Data
            data.put("db", value)

            File System Operation to retrive the Data
            data.put("file", value)

            XML Parsing Operation to retrive the Data
            data.put("xml", value)
            */
            dataMap.putAll(data);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return dataMap;
    }
}
```

## Optimize-5

```
public class CacheManagerInstantiationFilter implements Filter {
        public void destroy() {
        }
        public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
                                                                throws IOException, ServletException {
                CacheManager.getCacheManager();
                chain.doFilter(request, response);
        }
        public void init(FilterConfig fConfig) throws ServletException {
        }
}
```

```
public class GetLoanDetails extends HttpServlet {
        protected void service(HttpServletRequest request, HttpServletResponse response)
                                                                throws ServletException, IOException {
                int principle=Integer.parseInt(request.getParameter("principle"));
                int tenure=Integer.parseInt(request.getParameter("tenure"));
                String type=request.getParameter("type");
                String place=request.getParameter("city");

                Cache cache=Cache.getInstance();
                Properties props=null;
                PrintWriter out=response.getWriter();
                if (cache!=null) {
                        if (cache.containsKey("cities")) {
                                props=(Properties) cache.get("cities");
                        }else{
                                props= (Properties) cache.get("cities");
                        }
                        if (props!=null) {
                                if (props.containsKey(place)) {
                                        double interestRate =Double.parseDouble(props.getProperty(place));
                                        double totalAmt=(principle*interestRate/100)* tenure;
                                        out.println("<h1><b>Interest Rate: "+interestRate+ " \t and Total Amount in
                                 "+tenure+" months is: "+totalAmt+"<b></h1>");
                                }
                        }
                }
        }
}
```

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

```
public class CacheManager {
        private static CacheManager instance;
        private Cache cache;
        private Properties props;
        private CacheManager() {
                init();
        }
        public static synchronized CacheManager getCacheManager(){}
        public void init(){
                try {
                        cache=Cache.getInstance();
                        props=new Properties();
                        props.load(this.getClass().getClassLoader().
                                        getResourceAsStream("city.properties"));
                        cache.put("cities",props);
                        /*
                        DataBase Logic for retrive the Data
                        cache.put("db", value)

                        File System Operation to retrive the Data
                        cache.put("file", value)

                        XML Parsing Operation to retrive the Data
                        cache.put("xml", value)
                        */
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

➔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 add to cache 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.

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.
➔ AccessorDesignPattern 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 file system.
➔ 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 check Manager will massage the data and load into cache.
➔ Even we are using Accessor Design Pattern and CacheManager still my class tightly coupled with the cache class.
➔ Because my class is getting the data from cache and it is getting instance into my class.
➔if Interviewer will ask about classes are getting tightly coupled then you have to tell in my application all the data which we are accessing from accessor classes those are static data not runtime data(dynamic input) without this data our application will not work, I think in this case ,this is the right way to design our classes with accessor design pattern.

## Optimize-6

```
public interface IAccessor {
        public String getKey();
        public Object getData();
}
```

```
public abstract class AbstractAccessor implements IAccessor{
        private String key;
        public AbstractAccessor() {
                //no-code
        }
        public AbstractAccessor(String key) {
                this.key=key;
        }
        public String getKey() {
                return key;
        }
        @Override
        public abstract Object getData();
}
```

```
public class PropertiesAccessor extends AbstractAccessor {
        public PropertiesAccessor() {
                super("cities");
        }
        public Object getData(){
                Properties props=new Properties();
                try {
                        props.load(this.getClass().getClassLoader().
                        getResourceAsStream("city.properties"));
                } catch (IOException e) {
                        e.printStackTrace();
                }
                return props;
        }
}
```

```
public class DBAccessor extends AbstractAccessor {
        public PropertiesAccessor() {
                super("states");
        }
        public Object getData(){
                Map<String,Object> db=new HashMap()

                 //Logic for Retrive DataBase
                return db;
        }
}
```

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

```java
public class CacheManager {
    private static CacheManager instance;
    private Cache cache;
    private List<IAccessor> accessor;
    private CacheManager() {
        accessor=new ArrayList<IAccessor>();
        cache=Cache.getInstance();
        init();
    }
    public static synchronized CacheManager getCacheManager(){..
    public void init(){
        accessor.add(new PropertiesAccessor());

        /*accessor.add(new DBAccessor);
        accessor.add(new XMLAccessor);
        accessor.add(new FileSystemAccessor());*/

        for (IAccessor iAccessor : accessor) {
            cache.put(iAccessor.getKey(),iAccessor.getData());
        }
    }
}
```

**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 is there means before creating the object of my class (GetLoanDetails) other class should be loaded and populate the data .

➔Because my class using the data (Cache) with getting the help from other class (CacheManager) who populate data.

➔To fulfil such kinds of requirement we can easily use the depends-on additional configuration.

**Ex:**

➔GetLoanDetails class want the data from cache but cache data loaded by CacheManager so before GetLoanDetails object created I want the data available into the cache,

➔Means GetLoanDetails indirectly depends on CacheManager.

➔So before created GetLoanDetails ,CacheManager has to load the data into cache, so GetLoanDetails can easily use the data.

**➔In this case we have to use depends-on and configure GetLoanDetails with depends-on="cacheManager" attribute.**

```
public class Test {
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/don/common/application-context.xml"));
		//ApplicationContext factory=new ClassPathXmlApplicationContext("com/don/common/application-context.xml");
		GetLoanDetails details=factory.getBean("getLoanDetails",GetLoanDetails.class);
		details.getLoanDetails(100000, 30,"home","hyderabad");
	}
}
```

```
public class GetLoanDetails {
	private Cache cache;

public void setCache(Cache cache) {
		this.cache = cache;
	}
public void getLoanDetails(int principle,int tenure,String type,String city){
	Properties props=null;
	if (cache!=null) {
		if (!cache.containsKey("cities")) {
			try {
				throw new Exception("Cities Not Found");
			} catch (Exception e) {}
		}else{
			props=(Properties) cache.get("cities");
		}
		if (props!=null) {
			if (props.containsKey(city)) {
				double interestRate =Double.parseDouble(props.getProperty(city));
				double totalAmt=(principle*interestRate/100)* tenure;
				System.out.println("Interest Rate: "+interestRate+ " and Total Amount in  "+tenure+" months is: "+totalAmt+"");
			}
			else{
				try {
					throw new Exception("city Not found");
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		}
	}
}
```

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

city.properties

```
hyderabad=11.25
chennai=12.35
bangalore=10.19
```

```
public class CacheManager {
        private Cache cache;
        private List<IAccessor> accessor;
        public CacheManager(Cache cache, List<IAccessor> accessor) {
                this.cache = cache;
                this.accessor = accessor;
                init();
        }
        public void init(){
                for (IAccessor iAccessor : accessor) {
                        cache.put(iAccessor.getKey(),iAccessor.getData());
                }
        }
}
```

```
public interface IAccessor {
        public String getKey();
        public Object getData();
}
```

```
public abstract class AbstractAccessor implements IAccessor{
        private String key;
        public AbstractAccessor(String key) {
                this.key=key;
        }
        public String getKey() {
                return key;
        }
        @Override
        public abstract Object getData();
}
```

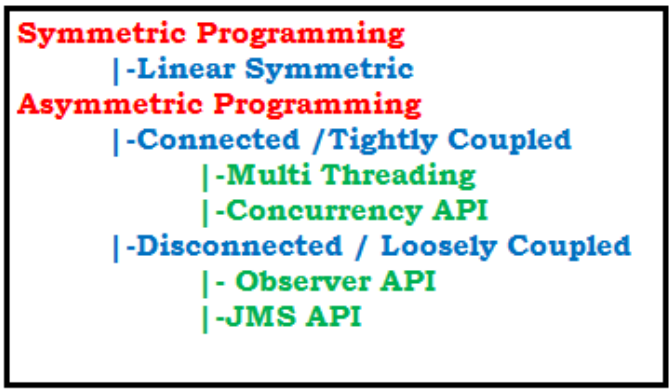
```
public class PropertiesAccessor extends AbstractAccessor {
        public PropertiesAccessor(String key) {
                super(key);
        }
        public Object getData(){
                Properties props=new Properties();
                try {

        props.load(this.getClass().getClassLoader().getResourceAsStream("city.properties"));
                } catch (IOException e) {
                        e.printStackTrace();
                }
                return props;
        }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                   http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="cache" class="com.don.util.Cache" factory-method="getInstance"/>

    <bean id="getLoanDetails" class="com.don.beans.GetLoanDetails" depends-on="cacheManager">
        <property name="cache" ref="cache"/>
    </bean>

    <bean id="cacheManager" class="com.don.util.CacheManager">
        <constructor-arg ref="cache"/>
        <constructor-arg>
            <list value-type="com.don.util.IAccessor">
                <ref bean="propertiesAccessor"/>
            </list>
        </constructor-arg>
    </bean>

    <bean id="propertiesAccessor" class="com.don.util.PropertiesAccessor">
        <constructor-arg value="cities"/>
    </bean>

</beans>
```

# Event Processing

Before we discuss event processing we have to understand how many ways we can process the event processing in the programming world.

**Programming Languages are 2 Types**




**Symmetric**

➔Symmetric driven approach is the linear execution approach one block of execution relay on other block.
➔One will going to call the other and he has to wait until and unless other 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**

➔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.
➔It is time consuming approach.
➔Performance will not be good.
➔In Synchronous one class or method tightly coupled with each other.

**Asymmetric**

➔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 :**

➔One method cannot block other method both are independent in asymettric approach..
➔In Asymmetric classes and method are loosely coupled.
➔Performance will high or throughput will be high.

**Why java has provided multithreading?**

➔**To start executing multiple part of an application that is different pieces of code,**

Example:

➔If I have some piece of code I don't want to get executed one after another one.

➔If I place together manually, then after first part complemented second part is start executing.

➔That's why divided them into two parts provided them into two classes.

➔So that I will provide to the **JVM** and asking the thread Scheduler to execute one after the another.

➔**So the goal of multithreading is achieve the simultaneous execution.**

**Why to use concurrency API?**

➔If we write one logic to perform some operation in all the classes then the code will be duplicated across all the classes.

➔Then what will be computed who will going to use as two parts, so that we can use the second part of the logic by making the first part return the outcome.

➔Here we can make a thread prepare the outcome by using concurrency API.

➔That's why new concurrency comes into the picture. This is the first goal of concurrency. (It is used when unmanageable code will be written across the classes).

**How many ways we can create a thread?**

➔There is only one way we can create the thread that is by extending from the Thread class.

➔But not by implementing from Runnable interface.

➔Because Runnable is an interface by implementing this we can achieve the multithreading but we cannot create thread.

**Why should we go for thread and when for Runnable?**

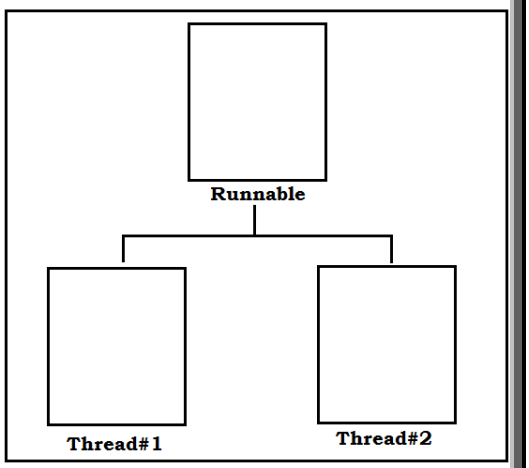
➔We can achieve multithreading by create two implemented class with extending both from the thread.

➔But I have a piece of logic that I want to be the multiple people participating executing and achieving the same result.

➔So avoid creating the class extending from thread because to create multiple threads we have to create multiple object of your class.

➔Instead of that create one object of your class pass multiple threads in using the same object.

➔So the reference of the object is being shared so that we can avoid memory utilization




➔If we have two different different works by using Runnable we can't do it because the logic of doing the work is different .In this case always we should go for Thread.

## For Example

First Name :
Last Name:
Gender : Male Female
Mobile :
Address Line-1 :
Address Line-2 :
City :
State :
Country:
Submit

register.jsp

```
public class ShowRegisterServlet extends HttpServlet {
    protected void service(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException {
        Properties props=null;
        Cache cache=Cache.getInstance();
        if (cache.containsKey("cities")) {
            props=cache.get("cities");
        }
        else{
            //go to the database
            //read the data from database
            //store the data in cache
        }
        //bind the data with request scope & forward to register.jsp page
    }
}
```

First Name :
Last Name:
Gender : Male Female
Mobile :
Address Line-1 :
Address Line-2 :
City :
State :
Country:
Submit

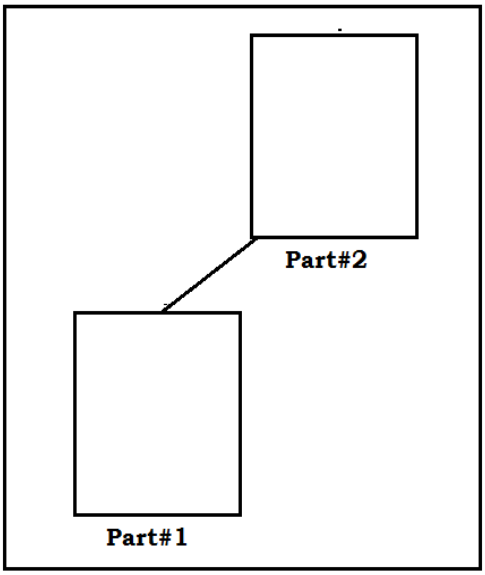
update.jsp

```
public class UpdateServlet extends HttpServlet {
    protected void service(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException {
        //Read the parameter values
        //Create the Data base connection
        //Store the data in Database
        //remove the data from cache

        new Thread(){
            public void run(){
                CacheManager manager=new CacheManager();
            }
        }.start();

        //display the Success Page
        }
    }
}
```

**Why the return type of run() method is void?**

**Symmetric**

➔Suppose I have some part of work.
➔Now I am asking these piece of work to get executed.
➔So whenever I call my other piece of code to execute.
➔So until the pieces of work is being completed I can't continue the process because that will compute the outcome and that outcome is needed for my code to execute,
➔Otherwise I will not execute. I.e. always a pieces of work that we have execute will returns some return value in a linear execution. It is symmetric.



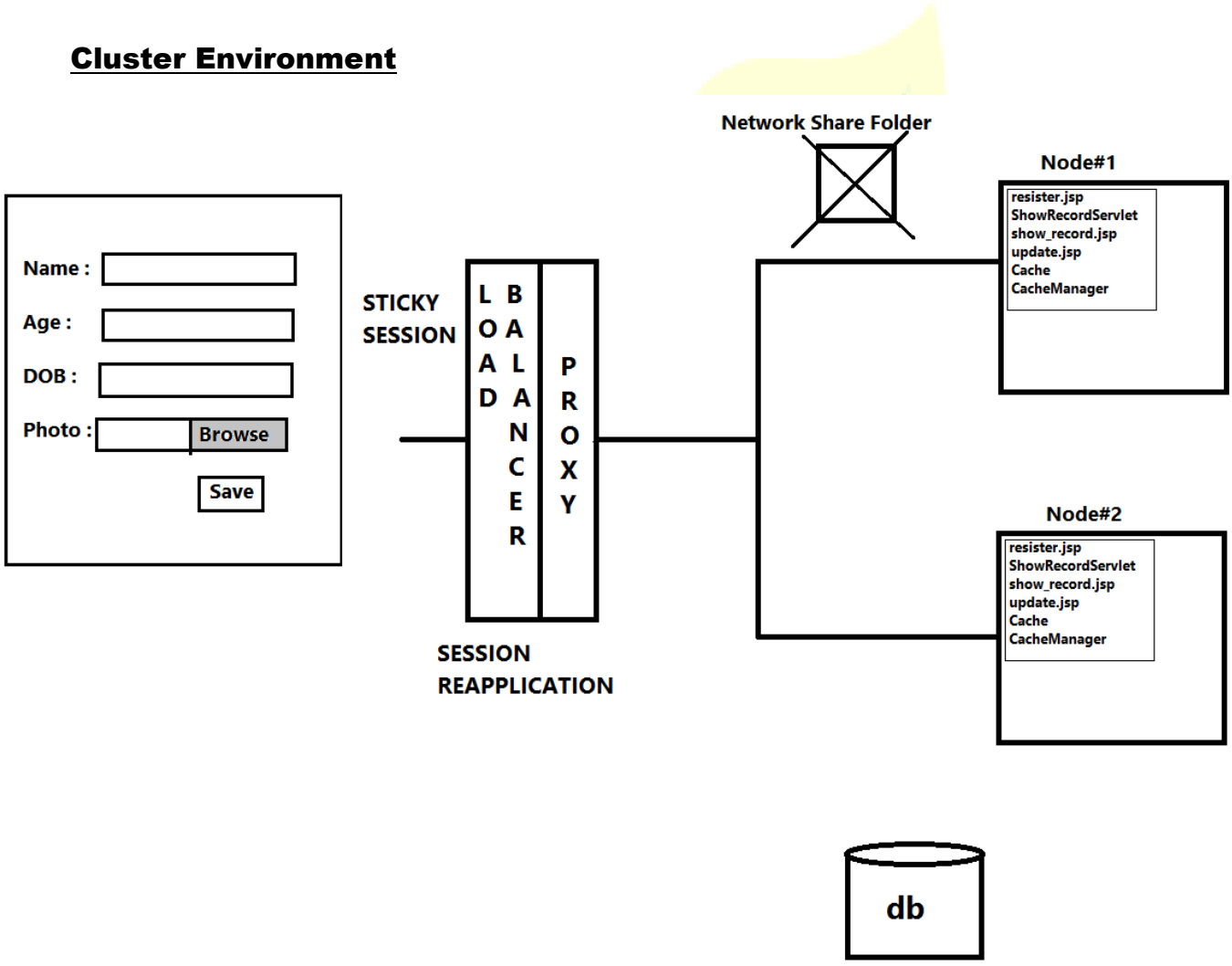
**Asymmetric**

➔But in asymmetric programming there are two ways are there i.e. part-1 and part-2.
➔Here part-1 is calling part-2.
➔No two parts are depend on each other because part-1 is not required the outcome of part-2 to continue the execution,
➔So both can be run simultaneously there is no problem. So the run() method return type is void. Because we don't need to return anything to other one.

**Why concurrency came into picture?**

➔Thread-1 and thread-2 are two different solutions(create the outcome and use the outcome)
➔As these are two different parts we want to create them as two threads. Now tread-1 builds the outcome and use it and thread-2 build the outcome and use it.
➔Both of them has two different run methods so that after completing the first thread the second thread starts execution.

## Cluster Environment



**LoadBalancer /Proxy**

There are 2 types of proxy are there.

➔Hardware Proxy

➔Software Proxy

**Software Proxy:**

➔Assume you will have a computer. In that computer you may have software called proxy server is installed. That will doing load balancing aspects.

**Hardware Proxy:**

➔It is a special box. The hardware that is designed to hold network connections i.e. actual level management will be done.

**Example:**

➔Here is a computer/machines is there. It has network hardware.

➔It will receive the requests, It can receive the request at a certain extends.

➔But to increase the network hardware capabilities there will be hardware proxy will be there.

➔Better performance proxy are those are. Depend on your need hardware or software proxy are there.

➔There is a computer. It doesn't have capabilities to talking 2000 machines.

➔It has only 1 port. So you have to add 1 more network card. It means you will 2 machines can connect to more than that unique switch.

➔Then the computer should be interacting with switch i.e. degradation of performance will increase.

➔That's why you need special machines which act as switch with 2000 loads & in that itself software will running which is called proxy.

➔So both the proxy are connected to the internate. Now you will sending the request.

**For example**

➔We have a servlet here. The name of the servlet is BookTicket.

➔The context root of our application is ReservationAgent.

➔When the user sends a request then the proxy will receive the request.

➔Proxi have to mapped this request to the 1 of the server & has to forward this request to 1 of the application of the application server on which it is running.

➔Then the application server has to identify servlet based on the context root and the servlet url pattern. There are 2 types of url are there.

➔ Friendly url: the url which are not access by the out side world rather it will use in domain

➔ Ugly url: what we are code inside our application or developer friendly url

➔So you have url rewrite at proxy level. There are some aspects of the proxy.

## Sticky Session

Example

➔We send the request. Proxy will receive the request and forward to the node which have less number of loads.

➔Then the servlet in the node receive the request. So in 1st phase I will give all my data to the jsp page then I click on submit.

➔Then the request will come to the proxy and proxy will redirect the request to node-1 which has less number of loads & node-1 servlet has been called.

➔Then the servlet has taken all the data and creates an session object and store the data in the session and display in 2nd phase.

➔Then user will fill the 2nd phase and click on submit. Then aging the request will come to the proxy then the proxy will check which server has less amount of load .

➔Node-2 has now less load then the proxy will redirect the request to node-2 server. After then 2nd servlet has been called.

➔Then the servlet has to take the session but the session is not there. So how can we use the 2nd Phase ?

➔That's why **sticky session** is a property that will be there. Sticky Session means 1st time when the request comes the user who has sends the request has been redirect to node-1.

➔Any consecutive request from the same user of the same session id will not be forwarded to node-2 will be only forwarded to node-1 only which is called sticky session. So now the application will behave properly.

### Disadvantages of Sticky Session

➔There are two machines & these machines are called nodes. So there are 2 nodes and have intranet connection. Means these 2 machines are connected in a same network but will not accessible to the outside world. It will be accessible in between a particular organization only or restricted domain or network only.

➔ Now whenever the user is sending the request the load balancer will receive the request and will verify which side load is less then decide to give the request to node-1 or node-2. Assume node-1 has less load & it send the request to node-1.

➔ After getting the request node-1 creates a session object and it will display as response. Now again the request is send by the same user of the same session id then load balancer will receive the request. Then the load balancer have to send the request to the same node with same session id otherwise user will not get the perfect response. That's why we have to enable sticky session. If we enable the sticky session the request always receive by the same node.

➔To overcome this problem session replication came in to picture, here any node can get the same session id of particular request.

➔There are several Global cache frameworks are available to store data

| **UDP Protocol Framework** |
|---|
| **1. chcache** |
| **2. jcache** |
| **3. oscache** |
| **4. swanecache** |
| **5. dmt cache** |

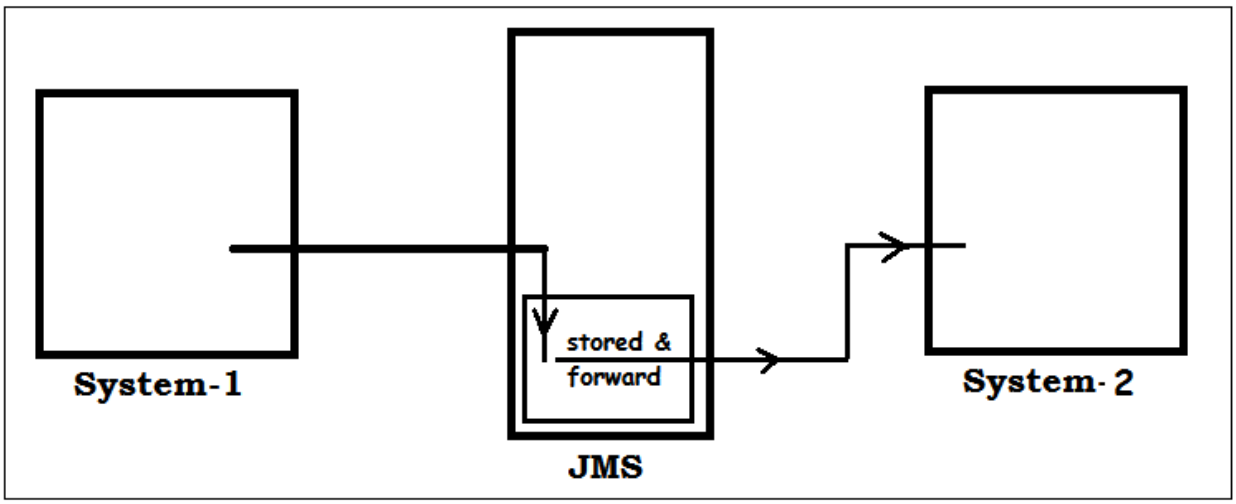
## JMS (java messaging service)

➔JMS is technology which is a message driven communication model. It is also known as disconnected loosely coupled distributed solution.

**How JMS technology does works?**

➔Assume there are 2 people. These 2 people wants to communicate with each other. 2 people means 2 classes assume. How can they talk to each other? If both are available and both are having the presence with each other then only 1 guy can talk with another guy in a normal communication.

**➔But when it comes to JMS technology the 2 systems, system-1 & system-2 this 2 system are may not be connected or may not be seeing each other at all. Without the presence of other person still the system -1 can communicate with system-2 with the help of a design pattern called stored & forward technology.**




➔Stored & forward technology means there will be an infrastructure sitting between these 2 components or systems which wants to exchange the communication. So never the system-1 will directly seeing to the system-2. Rather the system-1 always wants to communicate with the system-2 via this message store.

➔Whatever the data he wants to send the other person will be send through the messaging store. The messaging store will keep holding the data and when the other system has been live has been bring back when the presence of the 2nd system is there at that time the message is store here will be forwarded to the other system so that the other system will start using the data and perform some operation.

➔Again it wants to send the response back to system-1. Now this guy will send back the message to the messaging store. Again the messaging store will forward this message to the 1st system whichever wants to read the data. In this situation these 2 systems are not directly talking to each other even if they never knows the face of each other. Both need not be alive both need not be seeing at each other when 1of the other person dies also still the 1st system will can still availability and continue the communication to the message

store. When the other system has been up then the messages will be forwarded which is called **loosely coupled disconnected distributed solution**.

➔The real time data cannot be exchanged when you use JMS technology.

**Example: we are using this approach** for mobile recharge,4g activate, credit cards blocking all the customer interfacing applications are build on using JMS technology. Because scalability of your application will be there. Any no request will come also it will hold the data because it not handle the data in real time basis. So there is no chance of losing the data.

## How JMS Works?

➔JMS is used for disconnected distributed loosely coupled message oriented system.
➔The JMS basically supports 2 ways of communication model
- ➔Point to Point
- ➔Multi Point Communication

### Point to Point

➔If two Applications communicate with each other then 1st Application has to forward the request to Message provider (Messaging Store).
➔When 1st application sends the request to the Messaging store then only one application can receive the message.
➔Other application can't receive the message.

### Multi Point

➔ When 1st application sends the request to the Messaging store then all application will receive the message at least once who has registered in Messaging Store.
➔It is also called as broadcasting

### Messaging Store

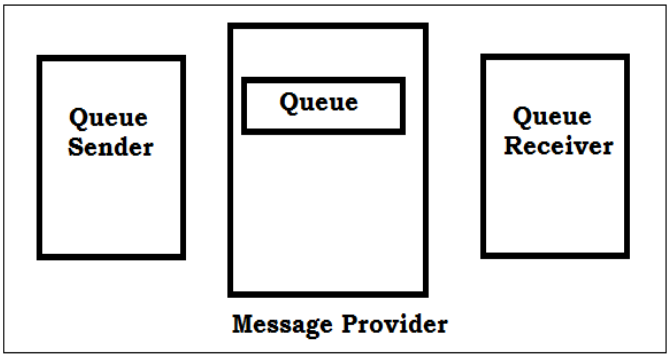
➔One application wants to communicate with each other the Messaging store is required.
➔ In market Messaging Server's or messaging providers are there.
➔They are providing infrastructure for keep hold up the messages and forwarding the message to the other people.
➔ So Infrastructure is required, thst's why there are lots of vendors are there who has provided JMS Infrastructure.
➔The Most popular and best server that is there with in the market is IBM MQ Server ,till now there is no replacement for IBM MQ series Server

### Data Structure

➔Messaging Store needs data structure for holding the messages and forward the messages.
➔So that other application can't see the Message of my application
➔Suppose Application 1 send the message to the other application, and then Application 1 has to give the message to the messaging store.

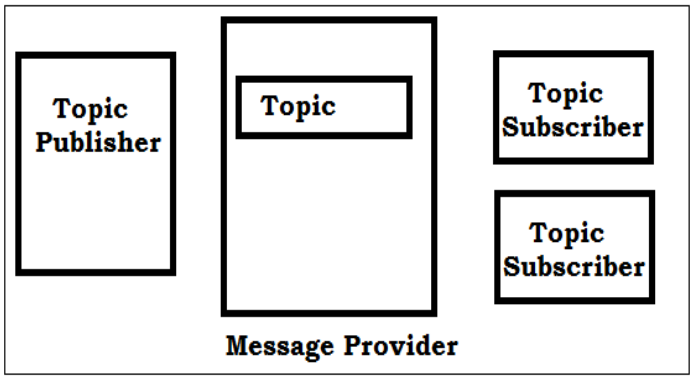
➔ Every application whoever wants to communicate with then first they need to go to the Messaging store and ask for give the space for them.
➔So that what the entire message they give to messaging store, it has to store messages and forward to the people who access to the particular space.
➔So with in the messaging provider we need to create messaging object in which our messages are going to store.
➔How to decide message forward to one application or multiple application? It depends on the communication model that we have chosen.
➔That's why Messaging store has provided 2 types of data structure.
➔Point to Point (Queue Data structure)
➔Multi Point Communication (Topic Data Structure).

**Queue**




➔Queue is a FIRST-IN FIRST-OUT, The message that has been placed in a queue first will be passing as a first place.
➔Queue is used for point to point communication.
➔The Application who sends the message to store in Queue is called Queue Sender.
➔The Application who receives the messages from the Queue is called Queue Receiver.
➔There will be any number of Queue receiver will be there.
➔When one application want to send the messages ,he has to go to the messaging store and gives to the Queue, then the Messaging provider will get activated the it forwards the message to the one of the message receiver.
➔The message in a Queue will send to only one receiver never it send to multiple receivers.
➔So Queue is called Point to Point Communication

**Topic**




➔I n Topic we can publish messages, and there will be multiple subscriber. And the message that is there in the topic will be forwarded to all the subscriber.
➔Who send the message to Messaging store they will call as publisher and who receives the Message they will call as Subscriber.
➔So it is also called as Publisher Subscriber Model/Multi Subscriber/Multitask Model.
➔For a Topic there will be multiple subscribers.
➔So Topic is called as Multi Point Communication.

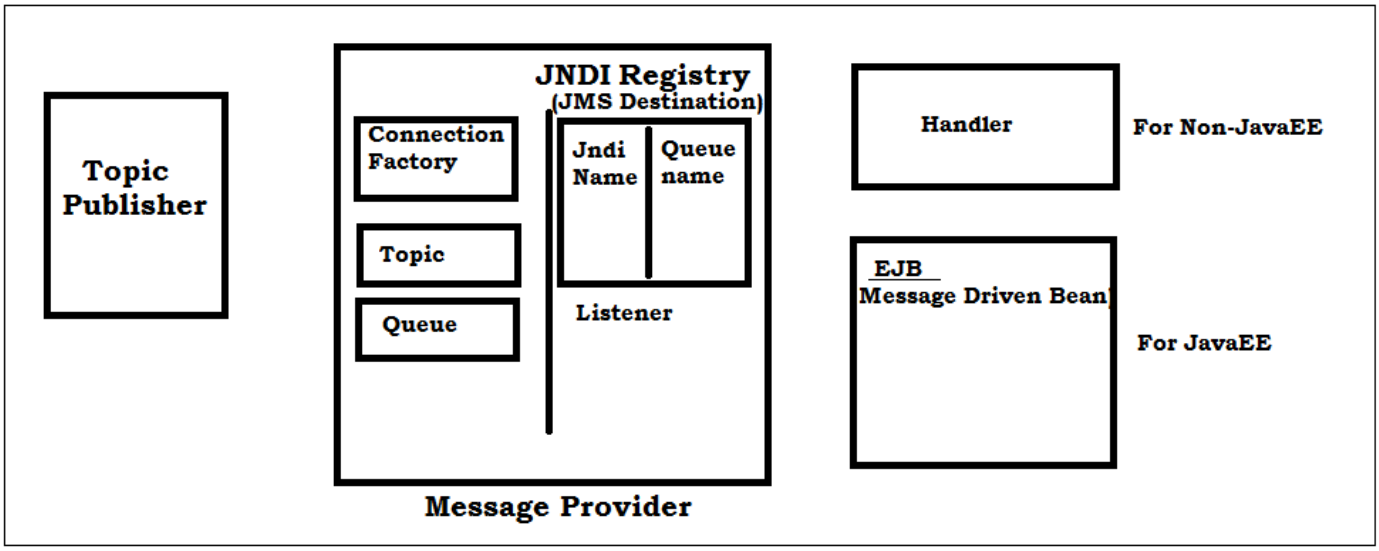
**Q.When we should go for Point-to-Point communication and when we should go for multi point communication?**

## Queue

➔Suppose I am trying to recharge my mobile, I am entering top up voucher and send request to Telephone Provider.
➔In Telephone Provider side there will be a messaging getaway system will be there.
➔In Telephone Provider Server may have lakhs of requests are there so it will take lots of time.
➔Then your request will be in Queue.
➔In this case if provider uses Topic (Multipoint) then a big problem will come at provider side.
➔In case of Topic I am sending 1 request as input to Provider but I will get multiple responses as output means multiple times my mobile number will get recharge.
➔In this case I should go for Queue and I should not go for Update.

## Topic

➔Suppose I am building an application of showing live cricket score.
➔The live Cricket Score information will be managed by ICCI.
➔If I want score information I need to talk to ICCI.
➔In this case not only my application there may be lots of application will send the request for get the live cricket score to ICCI.
➔So if ICCI people will use web services synchronous message Exchanging Pattern then there application will get slow down because of huge traffic, and if they use as synchronous they can't update frequently.
➔Then ICCI people will use JMS with Multipoint Communication which is called Topic.




**Durable Subscription**

1. JNDI Lookup
2. Connection Factory
3. Connection
4. Session
5. Session Create Queue

# Spring Event Processing

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.

## SOURCE

Source is the originator of event, who can raise several types of events. Events could be of multiple types if we consider AWT, we can consider source as a button, which is capable of raising several types of events like button click, mouse move, key pressed etc.

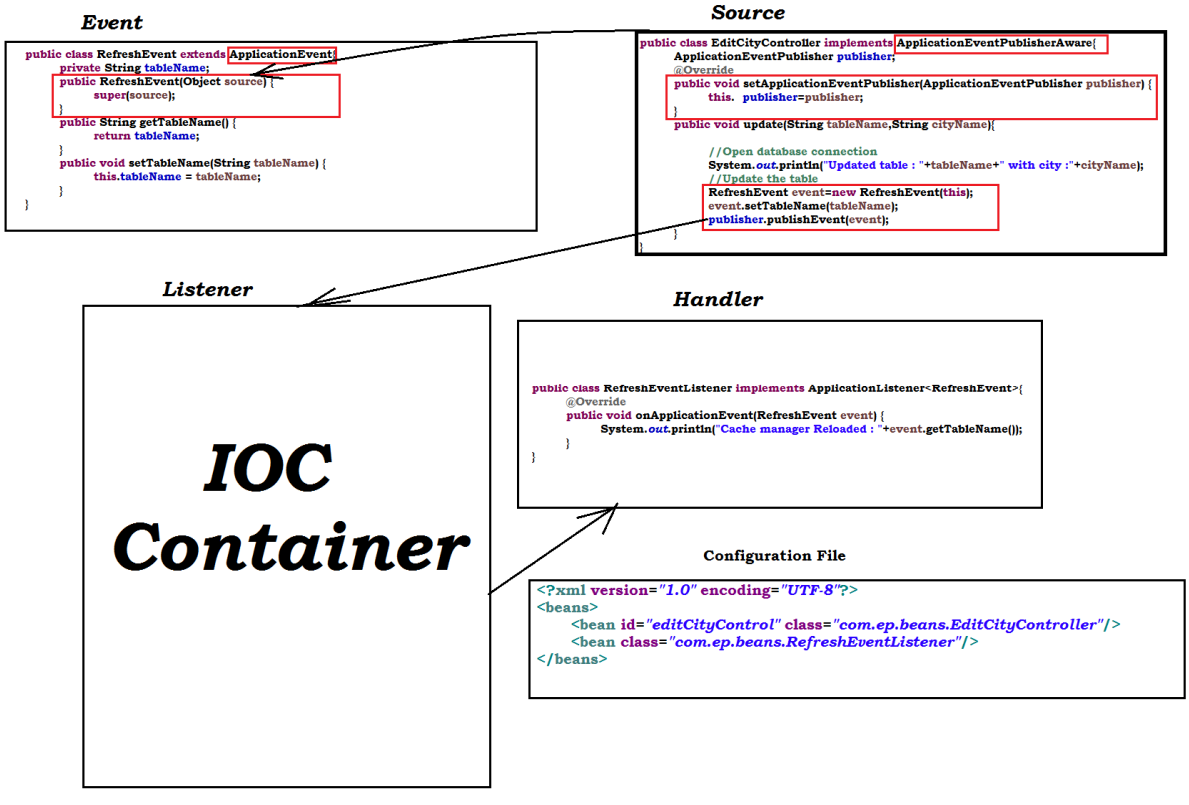
## LISTENER

Listener is the person who will listens for an event from the source. Listener listens for a specific type of event from the source and **triggers an even handling method on the Event handler to process it.**

## EVENT HANDLER

Once a listener captures an event, in order to process it, it calls a method on the event handler class. As there are different types of events, to handle them the event handler contains several methods each target to handle and process a specific type of event.

**Source :(to publish the event it must have ApplicationEventPublisher object )**
**Event :(must extends from ApplicationEvent)**
**Listener :(Here IOC Container act as Listener)**
**Handler :(Must implements from ApplicationListener)**



## Event

```
public class RefreshEvent extends ApplicationEvent{
        private String tableName;
        public RefreshEvent(Object source) {
                super(source);
        }
        public String getTableName() {
                return tableName;
        }
        public void setTableName(String tableName) {
                this.tableName = tableName;
        }
}
```

## Source

```
public class EditCityController implements ApplicationEventPublisherAware{
        ApplicationEventPublisher publisher;
        @Override
        public void setApplicationEventPublisher(ApplicationEventPublisher publisher) {
                this.  publisher=publisher;
        }
        public void update(String tableName,String cityName){

                //Open database connection
                System.out.println("Updated table : "+tableName+" with city :"+cityName);
                //Update the table
                RefreshEvent event=new RefreshEvent(this);
                event.setTableName(tableName);
                publisher.publishEvent(event);
        }
}
```

## Handler

```
public class RefreshEventListener implements ApplicationListener<RefreshEvent>{
        @Override
        public void onApplicationEvent(RefreshEvent event) {
                System.out.println("Cache manager Reloaded : "+event.getTableName());
        }
}
```

### Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
    <bean id="editCityControl" class="com.ep.beans.EditCityController"/>
    <bean class="com.ep.beans.RefreshEventListener"/>
</beans>
```

## BeanFactory vs. Application Context

| BeanFactory | ApplicationContext |
|---|---|
| 1.BeanFactory is lazy initialize, means after creating the Bean Factory it will loads the configuration file and create metadata in In-memory metadata inside JVM but it will not create any bean for the bean definition, when I try to fetch the bean using factory.getbean("...") then only Bean Factory will create the bean | 1. Application Context is eager initialize. Means when I first create the application context with configuration, after loading the configuration, it will immediately create all the singleton beans in the configuration. |
| 2.Bean Factory Doesn't support Internationalization directly | 2. ApplicationContext support internationalization. |
| 3.Beanfactory Doesn't Support Event Processing | 3.ApplicationContext Supports Event Processing |
| 4. BeanFactory will not automatically registers BeanPostProcessor or BeanFactoryPostProcessor, We need to explicitly write the code for registering them. | 4. ApplicationContext up seeing the BeanPostProcessor and BeanFactoryPostProcessor declarations in the configuration will automatically registers them with the container and applies processing. |

## Internationalization Example

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
	<bean id="messageSource" class="ReloadableResourceBundleMessageSource">
		<property name="basenames">
			<list>
				<value>message</value>
				<value>error</value>
			</list>
		</property>
		  <property name="cacheSeconds" value="1"/>
	</bean>
</beans>
```

```
public static void main(String[] args) throws UnsupportedEncodingException {
		ApplicationContext context=new ClassPathXmlApplicationContext("com/i18n/common/application-context.xml");

		System.out.println(context.getMessage("welcomeMessage",null,Locale.getDefault()));
	}
}
```

```
	public static void main(String[] args) {
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/i18n/common/application-context.xml"));
		MessageSource message=factory.getBean("messageSource",ReloadableResourceBundleMessageSource.class);
		System.out.println(message.getMessage("welcomeMessage",null,Locale.getDefault()));
	}
}
```

# BeanFactory Post Processor Example

```
public class ConnectionManager {
	private String driverClassName;
	private String url;
	private String username;
	private String password;
	public void setDriverClassName(String driverClassName) {
		this.driverClassName = driverClassName;
	}
	public void setUrl(String url) {
		this.url = url;
	}
	public void setUsername(String username) {
		this.username = username;
	}
	public void setPassword(String password) {
		this.password = password;
	}
	public Connection getConnection(){
		Connection con=null;
		try {
			Class.forName(driverClassName);
			con=DriverManager.getConnection(url,username,password);
		} catch (SQLException | ClassNotFoundException e) {
			e.printStackTrace();
		}
		return con;
	}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
	<bean id="connectionManager" class="com.bfpp.beans.ConnectionManager">
		<property name="driverClassName" value="${db.driverClassName}"/>
		<property name="url" value="${db.url}"/>
		<property name="username" value="${db.username}"/>
		<property name="password" value="${db.password}"/>
	</bean>

	<bean id="bfpp" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
		<property name="location" value="classpath:db.properties"/>
	</bean>
</beans>
```

file:c:/work/properties/db.properties

**db.properties**

```
db.driverClassName=oracle.jdbc.OracleDriver
db.url=jdbc:oracle:thin:@localhost:1521:xe
db.username=system
db.password=manager
```

Absolute Path

Relative Path

```
public class BFPPTest {
	public static void main(String[] args) throws SQLException {
		//Creating BeanFactory
		BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bfpp/common/application-context.xml"));

		//Applying logic before creating the bean object
		BeanFactoryPostProcessor processor=factory.getBean("bfpp",BeanFactoryPostProcessor.class);
		processor.postProcessBeanFactory((ConfigurableListableBeanFactory)factory);

		//Creating bean object
		ConnectionManager manager=factory.getBean("connectionManager",ConnectionManager.class);
		Connection con=manager.getConnection();
		Statement st=con.createStatement();
		ResultSet rs=st.executeQuery("select * from project");
		while(rs.next()){
			System.out.println("["+rs.getString(1)+","+rs.getString(2)+","+rs.getString(3)+","+rs.getString(4)+","+rs.getString(5)+"]");
		}
	}
}
```

# BeanPostProcessor Example

```java
public class ObjectVO {
        protected Date createDate;
        protected String createBy;
        protected Date lastModifiedDate;
        protected String lastModifiedBy;
                //setters & getters
}
```

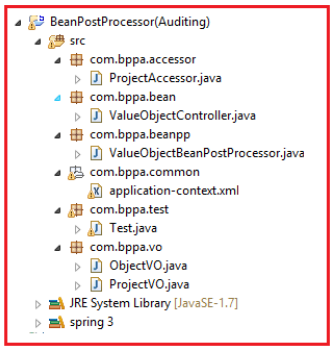
```java
public class ProjectAccessor {
        public static void insert(ProjectVO project){
                //database logic to store in database
                System.out.println(project);
        }
}
```

```java
public class ProjectVO extends ObjectVO{
        private int projectId;
        private String projectName;
        private String title;
        private String status;
        //Setters & Getters
}
```

```xml
<beans>
        <bean id="valueObjectController" class="com.bppa.bean.ValueObjectController">
                <lookup-method name="getProjectVO" bean="projectVO"/>
        </bean>
        <bean id="projectVO" class="com.bppa.vo.ProjectVO" scope="prototype"/>
        <bean id="valueObject" class="com.bppa.beanpp.ValueObjectBeanPostProcessor"/>
</beans>
```

```java
public abstract class ValueObjectController {
        public void getValue(int projectId,String projectName,String title,String status){
                ProjectVO project=getProjectVO();
                project.setProjectId(projectId);
                project.setProjectName(projectName);
                project.setTitle(title);
                project.setStatus(status);

                project.setCreateBy("user1");
                project.setLastModifiedBy("user4");
                ProjectAccessor.insert(project);
        }
        public abstract ProjectVO getProjectVO();
}
```

```java
public class ValueObjectBeanPostProcessor implements BeanPostProcessor {
        @Override
        public Object postProcessAfterInitialization(Object bean, String beanName)throws BeansException {
                if (bean instanceof ObjectVO) {
                        ((ObjectVO)bean).setCreateDate(new Date());
                        ((ObjectVO)bean).setLastModifiedDate(new Date());
                }
                return bean;
        }
        @Override
        public Object postProcessBeforeInitialization(Object bean, String beanName)
                        throws BeansException {
                return bean;
        }
}
```




```java
        public static void main(String[] args) {
                BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bppa/common/application-context.xml"));

                BeanPostProcessor processor=factory.getBean("valueObject",BeanPostProcessor.class);
                ((ConfigurableListableBeanFactory)factory).addBeanPostProcessor(processor);

                ValueObjectController con=factory.getBean("valueObjectController",ValueObjectController.class);
                con.getValue(111, "realspeed", "cipla", "in progress");
        }
}
```

**Q.When We should go for Beanfactory?**

➔In case of developing **small scale application** the clients requirement is very less because less number of user is going to use it.
➔So Event processing is not required in that application.
➔In small scale application there is no required for I18N because the business area/application access area is less.
➔All the objects of my application has not instantiate because all the functionality not required to my application.
➔Memory management is less in small scale application.
➔Computing capacity is less also.
➔So we have to use BeanFactory.

**Q.When we should go for ApplicationContext?**

➔In case of use Enterprise Application these are long term application.
➔Lacks of user going to access it and having high computing capacity and huge memory space required.
➔These kind of application must support I18N and Event processing for improve the performance and scalability of application.
➔For all these things Application Context required because these application required all the objects are present in my application are going to use.

**Q. Why spring removes BeanFactory?**

➔Developing a project in spring is not a easy (everyone can't develop application using spring)
➔For this high skill developers are required and for this client have to pay high resource for this small scale project.
➔For these case small scale application are not showing interest to develop by spring and all the big Enterprise application are build in spring.
➔For this spring removed support for BeanFactory.