spring

INDEX

Introduction

Introduction

Java learning is all about 3 things

a. Java language: Core java
b. Java technologies: JDBC, Servlets, JSP, Jndi, JMS, EJB and etc.
c. Frameworks: Spring, Hibernate, Webservices

Jndi: Java Naming and Directory Interface
JDBC: Java DB Connectivity
JSP: Java Server Pages
JMS: Java Messaging Service
EJB: Enterprise Java Beans

a. Programming language:

- It is directly installable software having basic features to develop software application. It defines syntaxes (Rules), semantics (structure) of programming.
- These are base to create software technologies like frameworks, tools, DB software, Operating systems and etc. as well as Tools.
- These are raw materials of software programming.

e.g., C, C++, C#, Java and etc.

b. Software Technology:

- It is a software specification that provides set of rules and guidelines for vendor companies to develop implementation software's by taking support one or another programming language.
- Technologies are not installable. But technologies-based implementation s/w are installable or arrangeable. Working these implementations software is nothing but working with Technologies.
- JDBC is technology giving rules and guidelines to develop JDBC Driver s/w. Working with JDBC driver s/w is nothing but working with JDBC Technology.

e.g.: JDBC, Servlet, JSP, EJB, Jndi and etc...

Up-to java7:

Technology rules are the technology API interfaces.
Technology guidelines are the technology API classes (concrete class).

Note: The technology API abstract classes represent both rules and guidelines.

From Java 8:

Technology API interfaces, abstract classes represent both rules and guidelines.
Technology API concreate classes represent only guidelines. java.sql, javax.sql and etc. called JDBC API packages.
These packages interfaces, abstract classes both represent rules and guidelines these packages classes represent only guidelines.

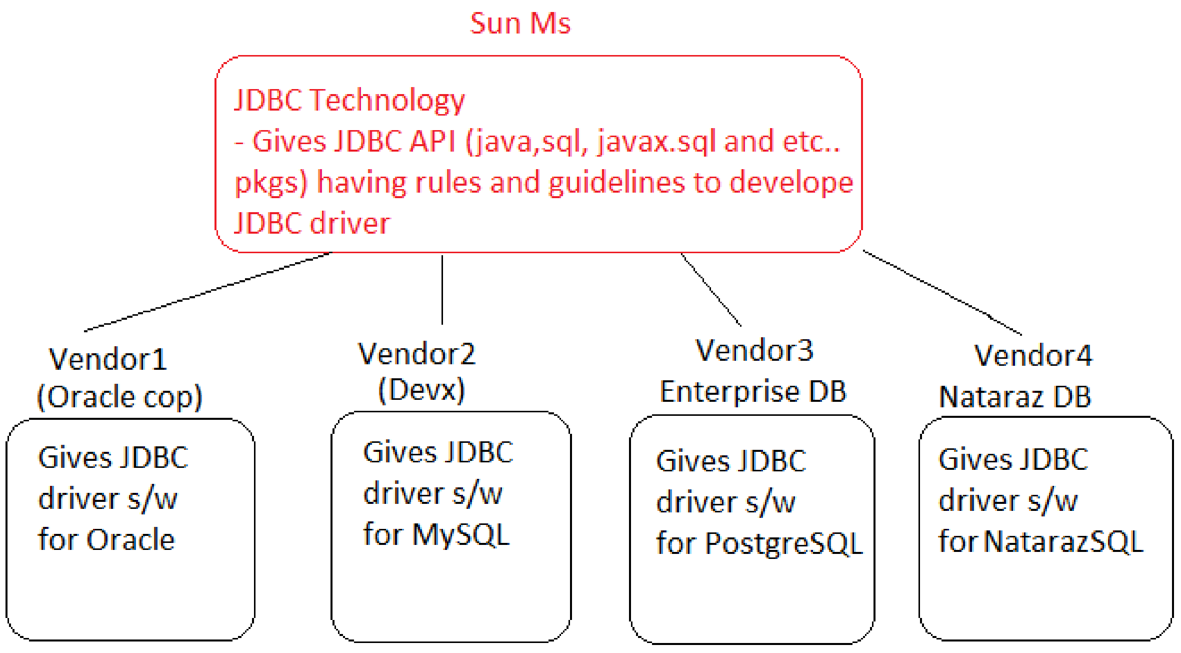
Two types of s/w Technologies

a. Open Technologies:

Here the technology rules and guidelines are open to all the s/w vendors to create implementation softwares.
e.g.: All Java Technologies are open technologies like JDBC, Servlet, EJB, JMS and etc.

b. Proprietary Technologies:

Here the technology rules and guidelines are specific to one vendor and only that vendor is allowed to develop the implementation softwares.
e.g.: All Microsoft technologies like asp.net.



Sun Ms
JDBC Technology
- Gives JDBC API (java,sql, javax.sql and etc.. pkgs) having rules and guidelines to develope JDBC driver
Vendor1 (Oracle cop)
Gives JDBC driver s/w for Oracle
Vendor2 (Devx)
Gives JDBC driver s/w for MySQL
Vendor3 Enterprise DB
Gives JDBC driver s/w for PostgreSQL
Vendor4 Nataraz DB
Gives JDBC driver s/w for NatarazSQL

[Diagram of Technology API to Vendor]

Since all the vendor companies are developing their implementations, software based on common rules and guideline, so the way we work for all the implementation softwares of software technology is more or less same.

Real life example of Framework:

House

Option1

(purchase open land and construct house)

[Here we should take care of both common and specific activities of house construction]

option2

(Go to real estate and purchase readily available house)

[Here we should take care only specific activities of house construction because common activities will be taken care by real estate company/ builder]

Real time example why we need Framework:

- ➢ While working with java technologies to develop the application we should write both common and specific logics, which gives burden to the programmers.
- ➢ While working with frameworks the common logics will be generated dynamically, so programmers should take care of only application specific.

e.g.

Plain JDBC App (Technology App):

a. Load JDBC driver class to register JDBC driver s/w for activating JDBC driver s/w.
b. Establish the connection with DB s/w. (likes road)
c. Create JDBC statement Object. (Like vehicle)
d. Send and execute SQL queries in DB s/w using the support of Statement object. [Application Specific Logic]
e. Gather SQL query results from DB s/w and process them. [ASL]
f. Exception handling.
g. Close JDBC objects like connection, statement and etc.

- While working with software technologies we have boiler plate code problem.

- The code that repeats across the multiple application or in multiple parts of a project either with no changes or with minor changes is called boiler plate code.

  Note: The Common logics JDBC application represents boiler plate code.

  Spring JDBC App (Framework App):

  (Spring JDBC is part of Spring framework which runs on top plain JDBC technology)

  a. Create JDBC template class object (Takes care of common logic).
  b. Send and execute SQL query in DB s/w.
  c. Gather SQL query result from DB s/w and process them.

  Note: While working with Frameworks. we do not write common logic because they will be generated dynamically i.e., there is not boiler code problem.

What is Framework?

Def 1:

Framework is a special installable software that build on top of one or more technologies having ability to generate common logic dynamically, this makes programmers to develop only application specific logics.
It is an installable software that uses existing technologies internally to simplify the application development having ability to generate common logics of application internally and dynamically.

Def 2:

Framework is a special installable software that provides abstraction on one or more technologies and simplifies application development.
It is an installable software that provides abstraction layer on existing technologies to simplify the application development process.

Note: Framework internally uses one or more technologies to generate the common logics but it never makes programmer worry about that aware of that. This is nothing but framework providing abstraction on technologies (hiding implementation/ internals).

E.g., Spring JDBC provides abstraction on plain JDBC technology
Spring MVC provides abstraction on plain Servlet, JSP technologies and etc.
- TV remote provides abstraction TV set circuits.

Advantage of Frameworks based Application Development:

a. Provide abstraction on software technologies, so we do not need have strong knowledge of technologies.
b. Avoids boiler plate code in application development.
c. Gives great productivity.
d. Frameworks give rich API/ Libraries.
e. Framework APIs modeled after real-time use cases, so developing project in company becomes very easy.

To Solve double posting problem
We write 200 lines code of using ServletFilters while developing Servlet, JSP based web application.
To Solve double posting problem in Framework
Just override one pre-define method in Spring MVC. (handleInvalidSubmit(-))

Approaches developing Spring Applications:

a. Using XML driven configurations (4)
b. Using Annotation driven configurations (2)
c. 100% Java code/ Java configure driven configurations (3)
d. Spring Boot driven configurations (Best approach) (1)

Note: The process giving details about any resource like class or interface or file and etc. to underlying serve or container or framework or JVM is called configuration.

Different types of Java Frameworks based on the kind of application we develop

a. Web Application Frameworks
b. ORM Frameworks
c. Application Frameworks
d. Bigdata Frameworks
e. Distributed Application Development Frameworks

Web Application Frameworks:

- These frameworks provide abstraction on Servlet, JSP technologies and simplifies MVC architecture-based web application development.

  M ----> Model
  V ----> View
  C ----> Controller

- MVC is best architecture that defines to use set of components to develop web application.
  e.g.
  struts ----> From Apache (only web application framework, but outdated)
  Spring MVC (part of Spring) ----> From Interface21 (1)
  JSF ----> From Sun MS/ Oracle Corp. (2)
  ADF ----> From Oracle Corp.
  Webwork ----> From open Symphony

  ADF: Application Development Framework
  JSF: Java Server Faces

ORM Frameworks:

- ORM stands for Object Relational Mapping.
- Provides abstraction on plain JDBC/ Java JDBC technology and simplifies to develop objects-based DB software independent persistence logics.
  e.g.
  Hibernate ----> From Soft Tree/ now RedHat (1)
  Toplink ----> From Oracle Corp.
  Eclipse link ----> From Eclipse (3)
  iBatis ----> From Apache (4)
  OJB ----> From Apache
  OpenJPA ----> From Sun MS/ Oracle Corp. (2)

  OJB: Object Java Beans
  JPA: Java Persistence API

.

All these ORM Frameworks are given based on the rules and guidelines of JPA specification

Application Framework:

- It is all-rounder framework that provides abstraction on multiple JSE, JEE technologies and even on some frameworks to develop all kinds of logics and different types of application.

  Spring Framework -----------------------> from Interface21 (Team: Pivotal)

- Using Spring we can develop presentation logics, business logics, persistence logics and etc.
- Using Spring we can develop Standalone Application, Web Application, Distributed Application, Enterprise Application and etc.
- Spring provides abstraction on multiple JSE, JEE technologies like JDBC, Jndi, EJB, Servlet, JSP, JMS and etc. and also provides abstraction on frameworks other frameworks like Hibernate, Toplink, iBatis and etc.
- The application/ components (class with reusable logic) that allows other remote applications accessing its logics is called Distributed Application/ Remoting Application.
  e.g.
  PayPal for E-commerce apps
  GPay/ phone for E-commerce apps
  BSE/ NEST comps for stock trading apps like Share Khan, BigAdda.com, etc.

Note: Spring is not good for developing distributed application, so prefer using Webservices.

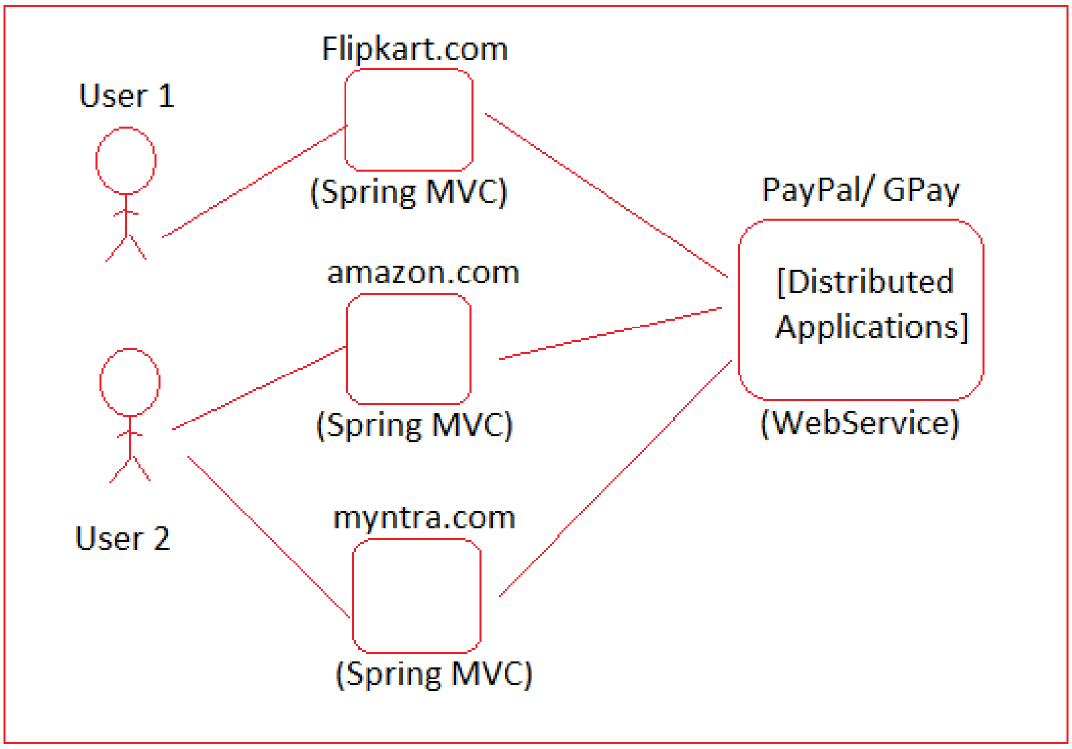
Distributed Application Development Frameworks:

- Simplifies the process of developing distributed application/ remote application.
- We can develop distributed application in two ways,

  - SOAP based webservice development: (outdated)
    JAX-RPC (old/ outdated), JAX-WS, axis and etc.

  - Rest based webservice/ RESTful webservice: (latest)
    Jersey, Rest Easy and etc.

Note: RMI, EJB, Corba and etc., distributed technologies to develop distributed application and they are outdated.



Big Data Frameworks:

- Big Data: The data that is beyond processing capacity is called Big Data. Like PB, TB, XB, YB amounts data.
  e.g.
  YouTube, Facebook generates big amount of data, and this data cannot be stored in regular DB s/w. So Big data frameworks are given to store and process such data by using stack of ordinary computers.

  Hadoop ----> From Apache
  Spark ----> From Apache

Enterprise Application:

- The enterprise can be a standalone app or web application or distributed app or combination of multiple, but it has to deal complex, heavy weight business logics/ operations.
  e.g.
  nareshit.com ----> Web Application
  flipkart.com Web ----> Application (Enterprise Application)
  PayPal ----> Distributed Application (Enterprise Application)
  Flipkart with PayPal/ G-Pay/ Phone-Pay (Enterprise Application)

Q. Why RMI, EJB technologies are outdated?

- If you developed distributed application using RMI, EJB technology no doubt they should developed using Java but it says client should also be there in Java.
- That means, suppose PayPal is developed in java (using EJB. RMI technology), and it will give instruction to all E-Commerce website you should develop your website in Java only if you are trying to develop in PHP, .Net, then you can't communicate with PayPal.
- Because of these restriction RMI, EJB technologies are outdated.

Note: But when it comes to Webservice though distributed application in Java clients can be Java or Non-Java (Like PHP, .net, Python, node.js, etc.)

Type of Frameworks based on mode of application

- Based on mode of application development we do, there are two types of frameworks.
  a. Invasive frameworks
  b. Non-invasive frameworks

Invasive frameworks:

- Here the programmer developed class of framework-based application development or tightly couple with framework API, i.e., these classes must be developed either implementing framework APIs interfaces or extending framework API classes.
- Due to this we cannot out our application/ project classes to other frameworks for execution.
- These frameworks say to programmers come to me, use me and stay with me forever.
  e.g., struts 1.x (outdated)

Note: Our Servlet components class also invasive because it has to implement javax.servlet.Servlet (I) directly or indirectly.

Non-invasive frameworks:

- Here the programmer developed classes of framework-based application/ project are loosely coupled with frameworks APIs, i.e. these classes need not to implement or extend from Framework API interfaces and classes.
- So, we can move these classes to another frameworks, because these classes ordinary classes/ interfaces.

- These frameworks say to programmers come to me, use me and go to other frameworks by taking you are code whenever you want.
  e.g., Spring, Hibernate, Eclipse link, Toplink and etc.

Important terminology:

- Some important terminology before introduce how Spring involved.
    a. Middleware services/ Aspects
    b. Java Beans
    c. Local Client vs Remote Client

Middleware services/ Aspects/ Cross cutting corners

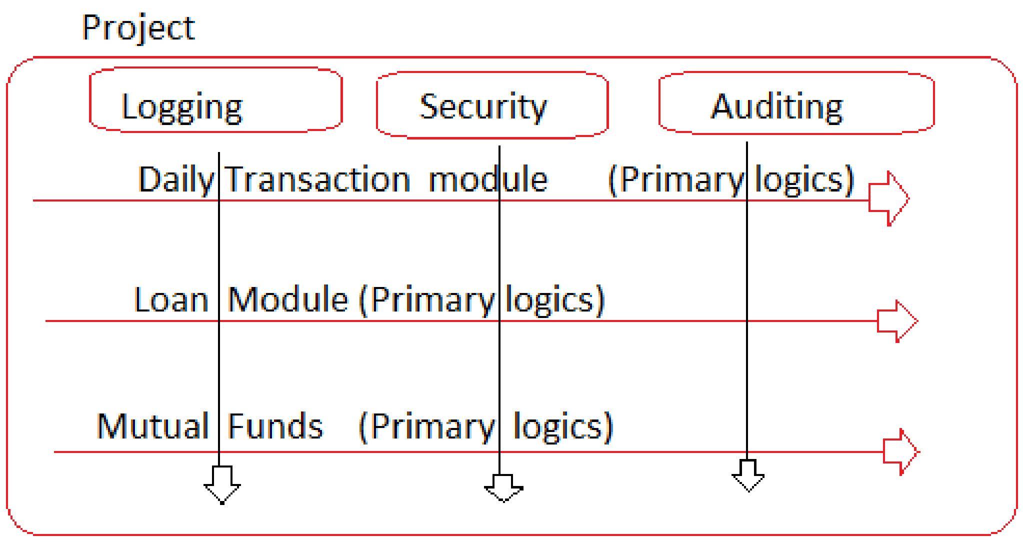
- These are additional, optional and secondary that can enable or disabled on primary logics of the application/ project to make application/ project more perfect and accurate.
- These are not mandatory logics/ minimum logics of application/ project development, these are additional and configurable (enabling/ disabling) logics.
- The minimum and mandatory logics of application development are called primary logics.
  e.g.
    W.R.T: Banking project withdraw, deposit, transfer money, checking balance, opening account and etc. logics.

- The optional and configuration logics of the application development are called secondary logics.
  e.g.
    Logging (keep track of application flow)
    Auditing (keeps track of user activities: user monitoring)
    Security (authentication: checking the identity of user +
              authorization: checking the access permission of the user)

Primary logics: These are horizontal because they will be developed parallel and more ever every primary logic is different from another primary logic.

Secondary logics: These are vertical because they will be developed once and they are trying to apply in multiple Primary logics.

Note: Technical people in the company are called horizontal. HR team, Accounts Team, Network Admin people these are called verticals in the company.


Project
Logging
Security
Auditing
Daily Transaction module (Primary logics)
Loan Module (Primary logics)
Mutual Funds (Primary logics)

Java Bean

- Java bean is a java class that is developed by following some standards.
- Java bean doesn't contain any logic it acts as helper class to carry data from a one class to another class with in the project, or across the multiple projects.
- In order to pass more than 3 values from 1 class to another class or from one project to another project take the support of Java bean.

The standards are:

a. Class must be taken public class [To visible to other class because it is a helper class]
b. Recommended to implement java.io.Serializable (I) [To share the data between to project we have to implement Serializable interface]
c. All member variables (Bean Properties) should private and non-static [Protection and allocated memory in side object for each different object]
d. Every variable (Bean Properties) should have one setter method and one getter method [setter method is used to set value/ modify value of bean property, getter method is used to read data from bean property, these methods also called as Accessor methods]
e. There should be one public zero param constructor directly (created by the programmers) or indirectly (given by the java compiler as default constructor)

Example of Java Bean:

```
public class StudentBean implements java.io.Serializable {
        //Bean properties
        private int sno;
        private String sname;
        private String sadd;

        //Getter and setter methods
        public void setSno(int sno) {
                this.sno=sno;
        }
        public int getSno() {
                return sno;
        }
        public void setSname(String sname) {
                this.sname=sname;
        }
        public String getSname() {
                return sname;
        }
        public void setSadd(String sadd) {
                this.sno=sno;
        }
        public String getSadd() {
                return sadd;
        }
}
```

Here the zero param constructor indirectly add takes care by compiler

Problem without java bean: Design java method having more than 3 params is bas practice

Main class

```
public class StudentRegistration {

        public String register (int sno, String sname, String sadd, int ml, int m2
                                                                int m3) {
                ........
                ........ // logic to insert them in DB table as record
                ........
        }
```

Client app

```
StudentRegistration registration = new StudentRegistration();
String result = registration.register(101, "nimu", "hyd", 55, 66, 88)
```

Limitations of Designing java method having more than 3 params :

- The below problem will happen when we pass more than 3 values While calling method,
    a. You should know the count of argument.
    b. You should know the index of argument.
    c. We cannot ignore to pass one or another argument value which we don't know, we should still pass a dummy value as argument value.

Solution with Java bean: Use Java bean to carry the data

Bean class

```
public class StudentBean implements Serializable {

            private int sno;
            private String name, sadd;
            private int m1, m2, m3;
            //Setters & Getters
            ……
            ……
}
```

Main class

```
Public class StudentRegistration {
            public string register (StudentBean bean) {
                        ……
                        …… // Use getter method to read data from
                        …… // bean object and use them business logics
            }
}
```

Client app

```
// Create Java bean class object having data
StudentBean bean = new StudentBean ();
bean.setNo(110);
bean.setSname("nimu");
bean.setM1(23); bean.setM2(56);
```

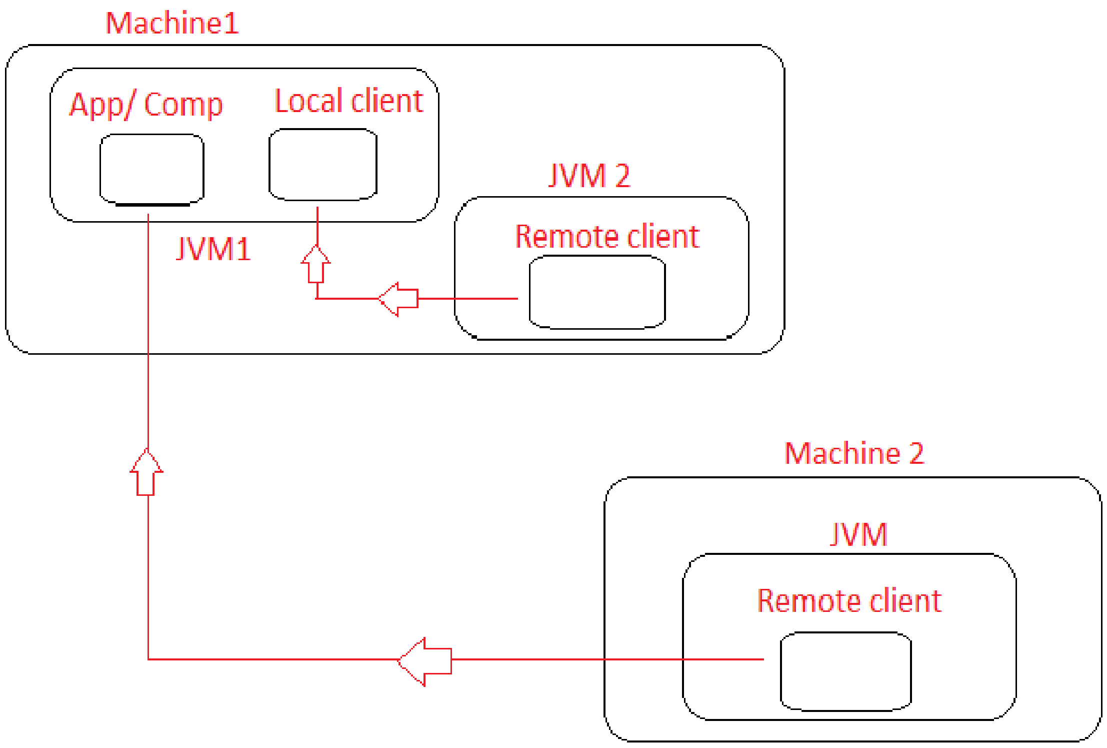
```
//Create main class object
StudentRegistration registration = new StudentRegistration ();
String result = registration.register(bean);
Systen.out.println(result);
```

Advantages of taking Java Bean as method parameter:

- With respect to method call
  a. We need not worry about count of values set to Bean class object.
  b. We need not worry about order/ index of values to set bean class object.
  c. We can ignore to set values; we don't know because bean objects itself takes default values.

Remote client vs Local client

- If the application/ component and its client are staying on the same JVM then that client is called Local client to application/ component.
- If the application/ component and its client are staying on two JVM of same or different computers then that client is called Remote client to application/ component.
- Normal apps can have only local client whereas Distributed application like EJB/ RMI/ Webservice Apps can have both Local and Remote clients.



Note: In one computer we can start one or more JVMs simultaneously. To run each Java application one JVM is required. So, to run multiple Java application simultaneously we need the support multiple JVMs.

How Spring is evolved?

1995 ----> Applet (Good for gamming)

1996 ----> Java bean (started developing by using Java classes + Java beans)

Pros
- Light weight
- Easy to learn and use

Cons
- Does not allow remote client, i.e., allows only local client.
- Not suitable for large scale applications.
- No built-in middleware services, i.e., middleware services must be developed by the programmers manually along with main or primary logic. Thus, improves burden on the programmers.

EJB:

1998 ----> EJB (Enterprise Java Bean) as Distributed technology runs on the top RMI.
EJB is no way related to Java bean better not to compare.
EJB is useful to developed Distributed App/ components.

Pros
- Allows both Remote clients and Local clients
- Gives built-in middleware service
- Suitable for medium scale and large-scale apps development

Cons
- Heavy weight (size wise, learning wise)
- Very complex to learn and use

Transaction Management (Middleware service):

The process of combining related operations into single unit and executing them by applying do everything or nothing principle is called Transaction Management.
e.g., Transfer money operation is combination of money withdraw, deposit operations and these operations must be executed by applying do everything or nothing principle.

Note: 1998 to 2005 the companies have used EJB even in non-distributed applications only to take the advantage of transaction management which was bad practice.

Spring:

2002/ 2003 Spring Framework by Rod Johnson.

Providing abstraction on multiple technologies/ frameworks.

Pros

- All kinds of development are possible.
- All kinds of logics are development is possible.
- Application development using by ordinary Java class giving non-invasive programming.
- Built-in middleware services support on different kinds of applications we can add built-in middleware services.
- Light weight application development.
- Easy to learn and easy to use.

  and etc.

Q. Is Spring alternate to EJB?

Ans. No, not at all

- EJB is distributed technology to develop distribute applications where as Spring is framework to develop all kinds of applications like standalone apps, Web applications, Distributed applications.
- Spring framework provides abstraction on multiple technologies including RMI, EJB but Spring based distributed applications are not good.
- Webservice is really alternate to EJB.

Q. Is Spring alternate for Struts?

Ans. No, not at all

- Struts is useful only to develop web applications, whereas Spring is given to develop all kinds of applications including web applications.
- A module of Spring called Spring MVC is alternate to Struts to develop the Web applications

Q. Is Spring alternate for JEE Technologies?

Ans. No, not at all

- Spring compliments JEE technologies like JDBC, Jndi, Servlet, JSP and etc. i.e., Spring internally uses these technologies to generate common logics dynamically and makes the programmer to just develop application specific logics.
- This is nothing but Spring providing abstraction on technologies

Q. Is Spring alternate for Hibernate?

Ans. No, not at all

- Spring is not having its own ORM framework, Spring ORM module is providing abstraction on multiple ORM frameworks including Hibernate, to simplify Object based O-R mapping persistence logic (persistence logic using no SQL queries)

Note: Spring ORM module is complimenting ORM frameworks including Hibernate.

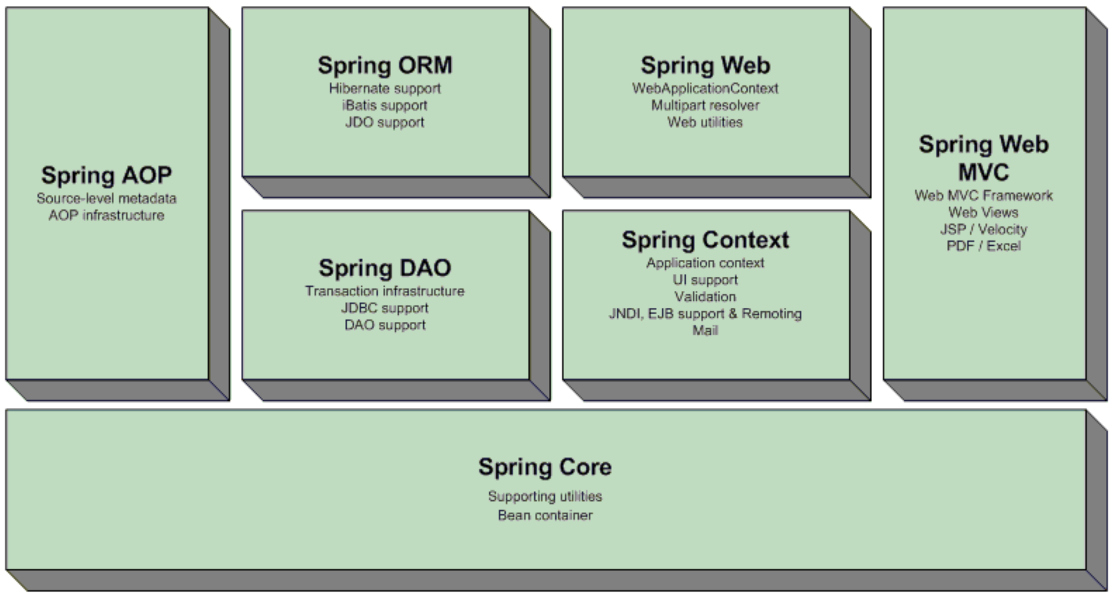
Q. Why the Spring framework is name as Spring?

Ans. The framework needed a name. In the book it was referred to as the "Interface21 framework" (at that point it used com.interface21 package names), but that was not a name to inspire a community. Fortunately, Yann stepped up with a suggestion: "Spring". His reasoning was association with nature (having noticed that I'd trekked to Everest Base Camp in 2000); and the fact that Spring represented a fresh start after the "winter" of traditional J2EE. We recognized the simplicity and elegance of this name, and quickly agreed on it.

Spring 1.x Modules, 2.x, 3.x, 4.x overview diagrams

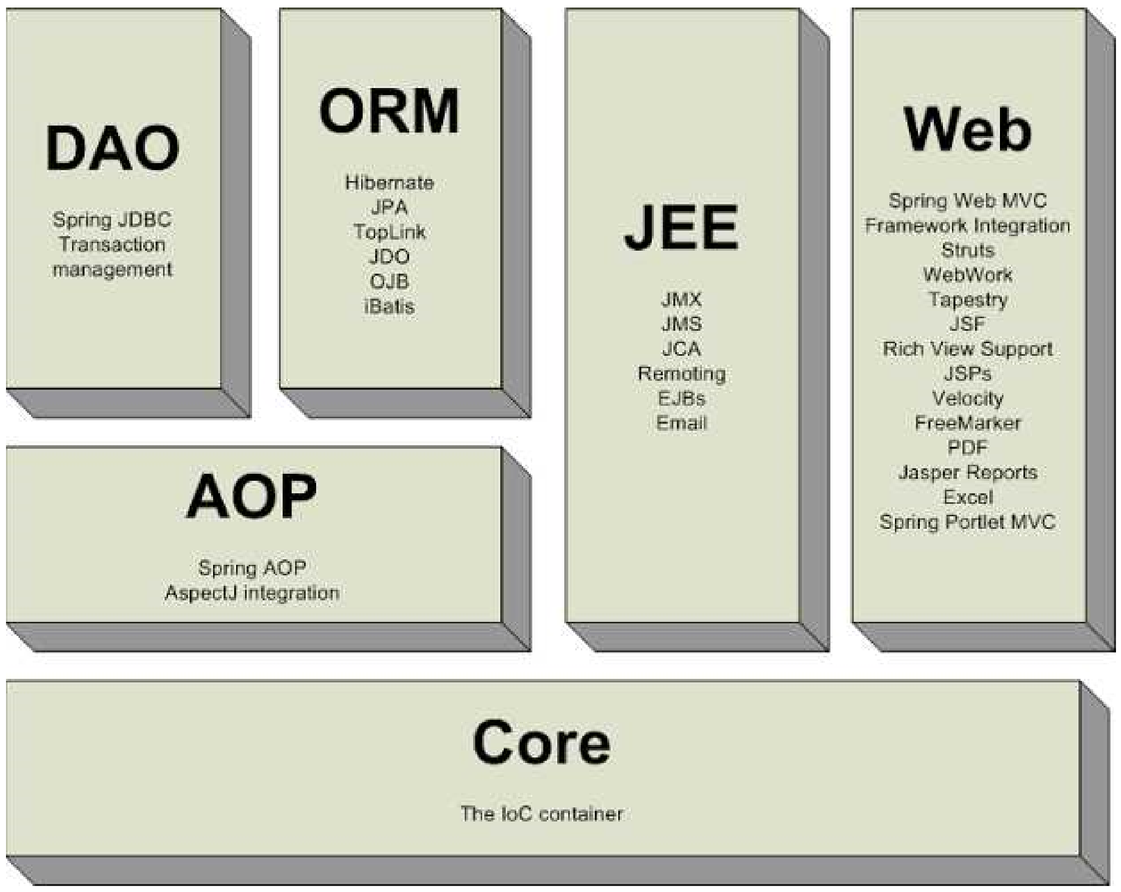
Spring overview diagram [1.x]:

1.x having 7 modules



Spring overview diagram [2.x]:

2.x having 6 modules



Spring 2.x web module = Spring 1.x web + Spring 1.x web MVC +miscellaneous
Spring JEE module = spring 1.x context module + miscellaneous
3.x/ 4.x/ 5.x ---> 20+ modules.

Spring Core:

- It is base module for other modules.
- This module is given to supply Spring containers and perform Dependency management.

Container:

- Container is a software program that manages the whole life cycle of given component class (class with reusable logic) from birth to death (object creation to object destruction).
- Servlet container manages the life cycle of Servlet components.
- JSP container manages the life cycle of JSP components.
- Spring container manages the life cycle of Spring bean.

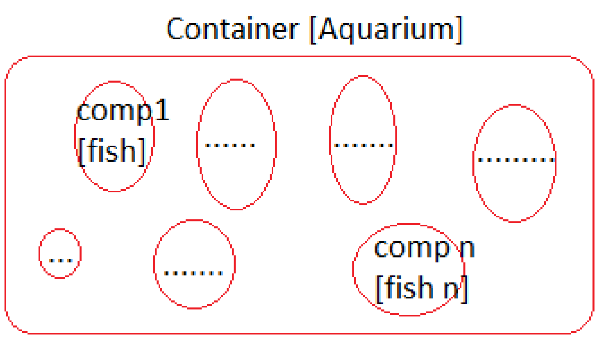
Note:

- ✓ Any Java class (pre-defined or user-defined or third party) whose object is created and managed Spring container is called Spring bean.
- ✓ Even Java bean also can be taken as Spring bean, but Object should be created and managed by Spring container.

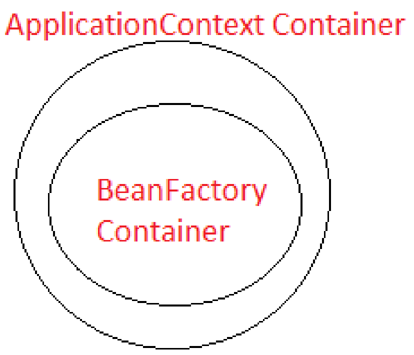
Spring Containers:

- Spring framework gives two built-in spring containers
  a. BeanFactory (Basic)
  b. Application Context (Advanced)

These are no way related to Servlet, JSP containers.



[Diagram]



Dependency management:

- It is the process of assigning Dependent class object to target class object by loading both classes and creating both objects.
- The class that uses other class service is called target/ main class.
- The class that acts as helper class to main/ target class is called as Dependent class.

e.g.

Target class	Dependent class
Flipchart	DTDC
Vehicle	Engine
Student	Course material
Department	Employees
Mobile device	SIM card
and etc.	

- If we give Target class name, Dependent class name to Spring container either through XML file or through Annotations or through Properties file.

- Then Spring container performs
    a. Loading of both classes
    b. Creating both classes' objects
    c. Assigning/ Injecting Dependent class object to target class object.

Q. What is the difference between Servlet container and Spring container?
Ans. Servlet container, JSP container can be used only in web environment to manage the life cycle of only web components (Servlet, JSP components). Whereas Spring containers can be used in all kinds of applications to manage life cycle and dependency management for all kind of classes/ components.

Note: If we use core module alone to develop the applications at maximum, we can develop only standalone applications. Generally, we used other modules Spring for spring applications development which internally uses Spring core module.

Spring DAO:

- DAO stands for Data Access Object.
- Provides abstraction on plain JDBC/ Java JDBC technology and simplifies JDBC style SQL queries-based persistence logic development.
- Gives details exception class hierarchy to display different error message for different database related problems.

Note: Spring DAO also called as Spring JDBC is given to develop JDBC style DB s/w. Dependent persistence logic using SQL queries.

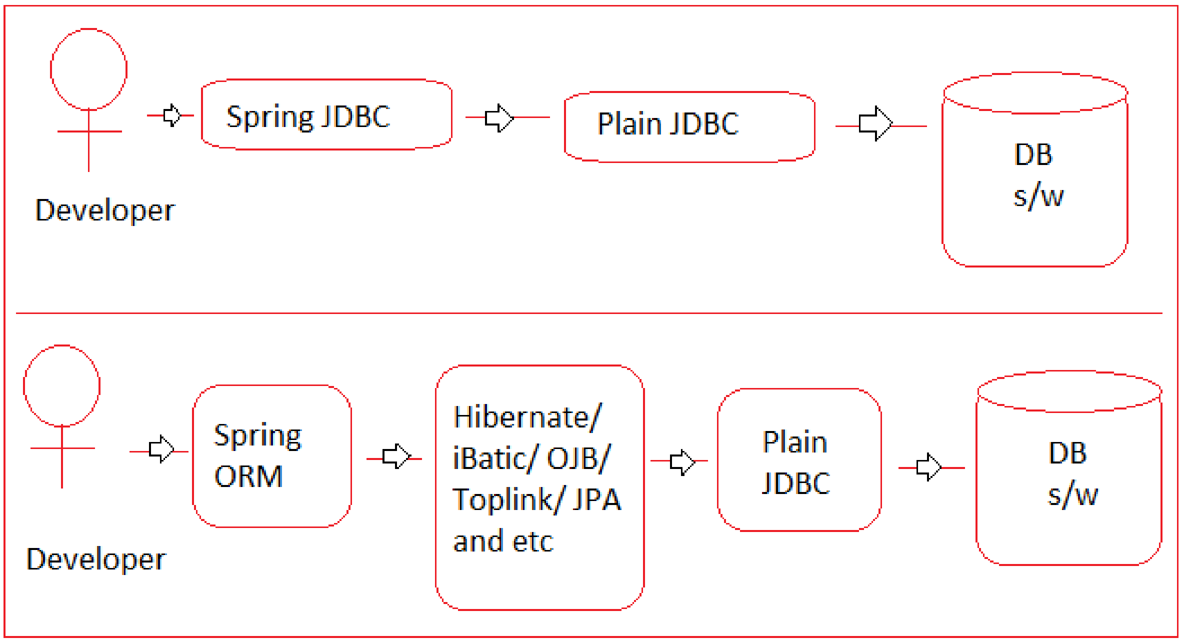
Spring ORM:

- ORM is stands for Object Relational mapping.
- Here the object of java classes represents DB table records, so we can manipulate records through objects.
- Provides abstraction on multiple ORM frameworks like Hibernate, iBatis, Toplink, JDO and etc. and simplifies object-based DB s/w independent persistence logic without SQL queries.
- Spring DAO persistence logic is JDBC style DB s/w dependent persistence logic using SQL queries where Spring ORM persistence logic is DB s/w independent object-based persistence logic without SQL queries.

Note: JDBC is not outdated because hibernate, Spring JDBC and other ORM framework still used JDBC.

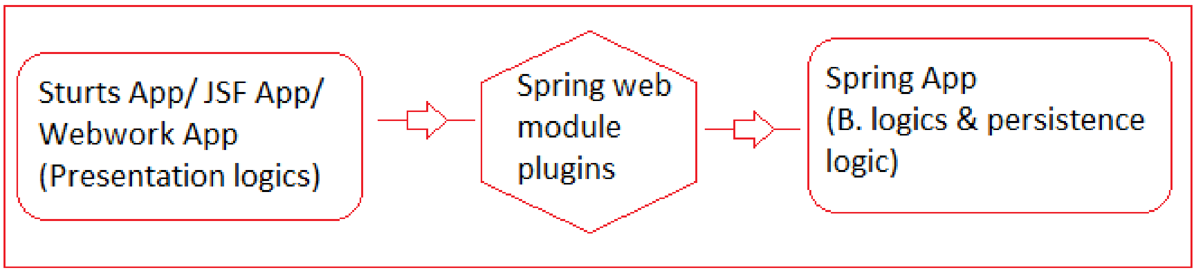
Communication from developer to DB software:

There is different environment to communicate developer to DB s/w like below;



Spring Web:

- This module provides plugins to make our Spring applications interactive or communicative from web framework applications like Struts applications, JSF applications, Webwork applications and etc.
- Plugin is a small patch software or code that provides additional functionalities to existing software or code.


Sturts App/ JSF App/ Webwork App (Presentation logics)
Spring web module plugins
Spring App (B. logics & persistence logic)

Note: It is outdated because Struts, JSF are outdated to develop web applications more ever developers are using Spring Web MVC module in that place.

Spring MVC/ Spring Web MVC:

- It is Spring's direct framework to develop MVC architecture-based Web applications as alternate to JSF, Struts and Webwork frameworks.

- Spring MVC provides better and more features in Web application development compare to Struts, JSF.

Spring JEE/ Spring Context:

- This module provides abstraction on multiple modules to simplifies different types application development,
    a. Plain Jndi and Spring Jndi
    b. Plain RMI, EJB and Spring RMI, EJB
    c. Java Mail and Spring Mail

Plain Jndi and Spring Jndi:

- Jndi: Java Naming and Directory Interface.
- To provide global visibility to Java objects they need to place in Jndi registry, to interact with Jndi registry from our Java applications we take support of Jndi API.

Jndi Registry

Key (Jndi name/ nick name)	Value (object/ obejct reference)
gpay	googlePay object ref
BSE	BSE comp ref
PayPal	PayPal comp ref

- We can keep any object in Jndi registry but we generally keep distributed applications components/ object reference in Jndi registry because they need to have global visibility in order get accessed by remote clients

Java application ----> JDBC API ----> DB s/w (Oracle, MySQL and etc.)
Java application ----> Jndi API ----> Jndi registry (RMI registry, COS registry, Tomcat registry)

Note:

✓ All latest servers like Web logic, Glassfish, Wild fly are having their own Jndi registry.
✓ Spring Jndi provides abstraction on plain Jndi technology and simplifies Jndi code development.

Operation on Jndi registry:

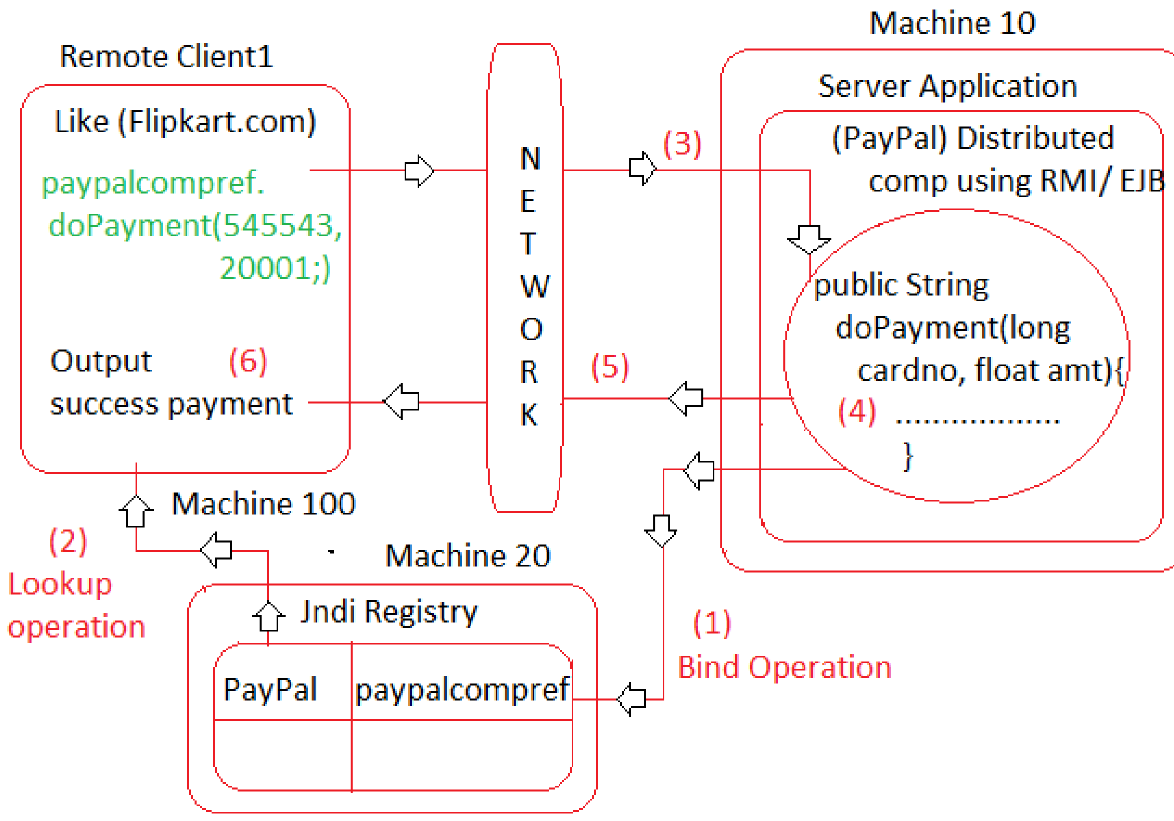
Using plain Jndi or Spring Jndi we perform following operations on Jndi registry.
Bind operation: Keeping object/ object reference in Jndi registry having nick name/ Jndi name.

Rebind operation: Replacing existing object/ object reference with new object/ object reference.
Unbind operation: Removing object/ object reference from Jndi registry.
Lookup operation: searching and getting object/ object reference from Jndi registry based on the given nick name/ Jndi name.

Plain RMI, EJB and Spring RMI, EJB:

- RMI, EJB are distributed technologies to develop applications/ objects/ components distributed applications/ objects/ components.
- The object/ component/ application whose service/ methods can be invoked from remote clients are called distributed objects/ components/ applications.
  e.g., Phone Pay, PayPal, BES stock and etc.

Note: After developing distributed applications/ components/ objects, it will be placed in Jndi registry for global visibility for remote client.



- We cannot call methods of ordinary/ normal class object (RMI, EJB not involved) from remote clients though its object reference is kept in Jndi registry.

- We can call methods of distributed component/ class object (EJB, RMI) from remote clients by keeping its object reference in Jndi registry.

- Spring RMI provides abstraction on plain RMI technology and simplifies distributed application development.
- Spring EJB provides abstraction on plain EJB technology and simplifies EJB application development.

Note:
- ✓ RMI, EJB both are outdated to develop distributed applications, so Spring RMI, EJB are also outdated.
- ✓ Webservice is best option to develop distributed applications.

Java Mail and Spring Mail:

- E-mail accounts and email messages will not be maintained in DB s/w, they will be managed in separate mail servers.
  e.g., James Mail serve, MS exchange server, SMTP one server, and etc.

  Java application ----> JDBC API ----> DB s/w
  Java application ----> Jndi API ----> Jndi registry
  Java application ----> Java Mail API ----> Mail servers

- Java application Performs mail operations like sending mail with or without attachment, reading mail, deleting mail, forwarding mail and etc.
- Spring mail is part of Spring JEE module and simplifies mailing operations by providing abstraction on Java Mail API/ technology.

Spring AOP:

- AOP is stands for Aspect Oriented Programming.
- AOP is not replacement for OOP, in fact it compliments OOP. i.e., built on the top of OOP.
- The logics that are minimum and mandatory to complete the tasks in application are called primary logics.
  e.g., W.R.T to Banking Application
  Withdraw money, deposit money, transfer money, opening account and etc.

- The logics that are additional, optional in application development but useful to make applications more perfect and accurate are called

secondary logics, i.e., our applications can be executed without secondary logics but primary logics are must.
e.g.
Logging (keep track application's flow) [it's no way related to login/ sign-in]
Auditing (keeps track user's flow)
Security (authentication + Authorization)
Transaction management (executes logics by applying do everything or nothing principle), and etc.

Problem without OOP style Programming:

- Here in the methods of classes we mix up both primary and secondary logics.

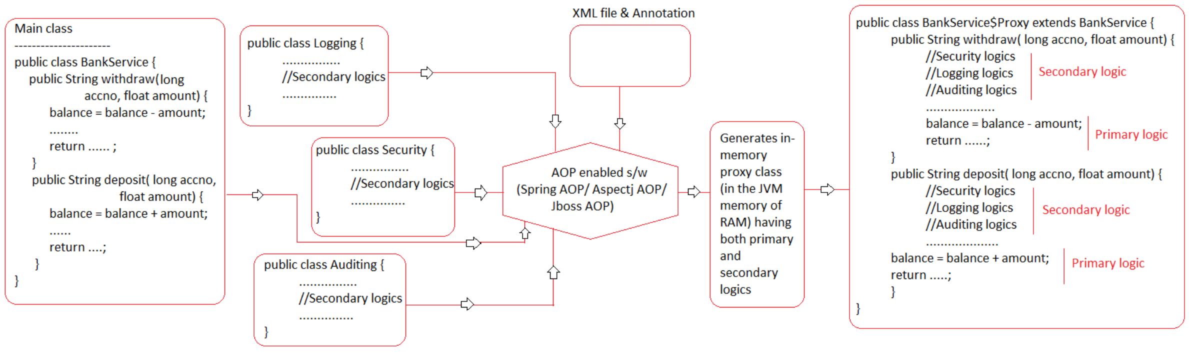
```
public class BankService {
        public String withdraw (long accno, float amount) {
                //Security logics                    |
                ..............                       |
                //Logging logics                     |   Secondary Logic
                ...............                      |
                //Auditing logics                    |
                .............
                balance = balance - amount;          |
                ...............                      |   Primary Logic
                return ......;                       |
        }
        public String deposit (long accno, float amount) {
                //Security logics                    |
                ................                     |
                //Logging logics                     |   Secondary Logic
                ................                     |
                //Auditing logics                    |
                ..................
                balance = balance + amount;          |
                ..................                   |   Primary Logic
                return .......;                      |
        }
}
```

Limitations of OOP style Programming:

a. Primary and secondary logics are mixed up i.e., there is no clean separation between logics.
b. Secondary logics are not reusable across the multiple methods of classes.
c. To enable or disable secondary logics on primary logics we need to disturb the source code, if client organization is not access to source code after release of project, then would be big problem.
d. Overall codes look very clumsy.

Solution using AOP:

a. AOP is a methodology of programming, that says separate secondary logics from primary logics and give them to AOP enabled software Spring AOP, AspectJ AOP, JBoss AOP and etc.
b. To generate dynamic class at runtime having both primary logics and secondary logics mixed up.



```
Main class
----------------------
public class BankService {
    public String withdraw(long
                accno, float amount) {
        balance = balance - amount;
        ........
        return ...... ;
    }
    public String deposit( long accno,
                float amount) {
        balance = balance + amount;
        ......
        return ....;
    }
}
```

```
public class Logging {
        ...............
        //Secondary logics
        ...............
}
```

```
public class Security {
        ...............
        //Secondary logics
        ...............
}
```

```
public class Auditing {
        ...............
        //Secondary logics
        ...............
}
```

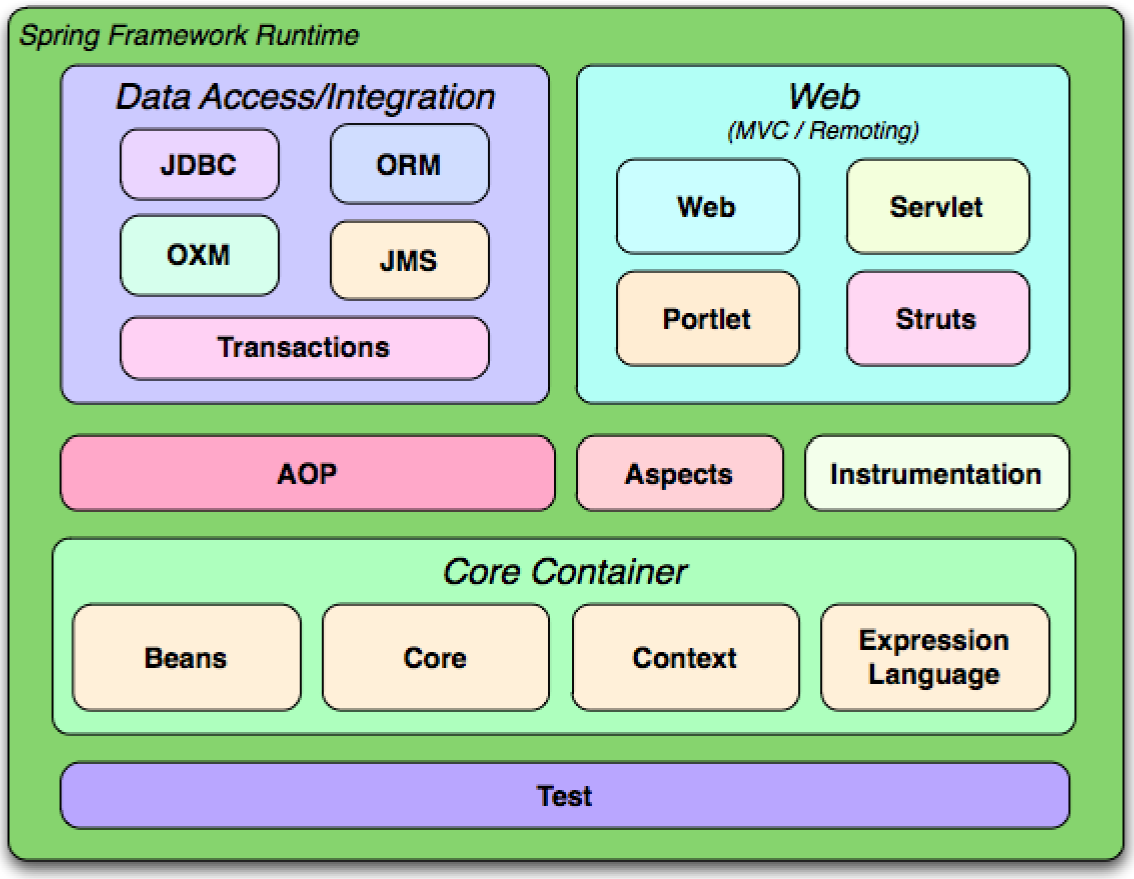
XML file & Annotation
AOP enabled s/w (Spring AOP/ Aspectj AOP/ Jboss AOP)
Generates in-memory proxy class (in the JVM memory of RAM) having both primary and secondary logics

```
public class BankService$Proxy extends BankService {
        public String withdraw( long accno, float amount) {
                //Security logics   |
                //Logging logics    | Secondary logic
                //Auditing logics   |
                ..................
                balance = balance - amount; |
                return ......;              | Primary logic
        }
        public String deposit( long accno, float amount) {
                //Security logics   |
                //Logging logics    | Secondary logic
                //Auditing logics   |
                ..................
        balance = balance + amount; |
        return .....;               | Primary logic
        }
}
```

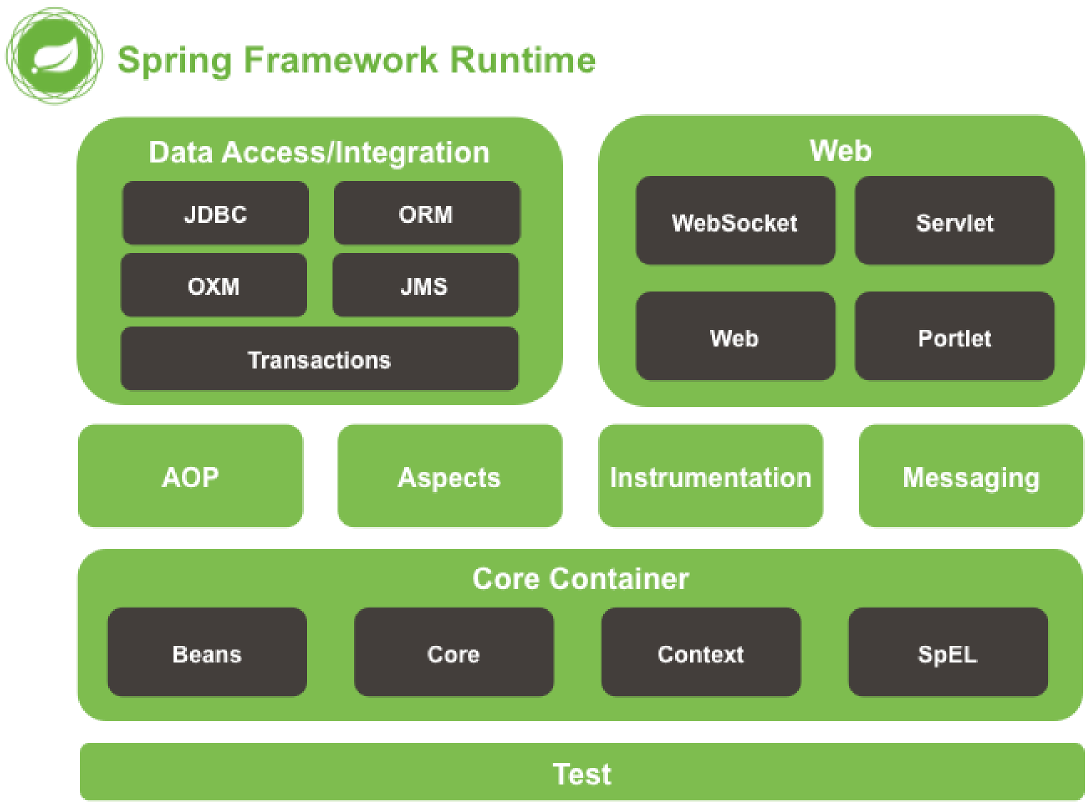
Advantages of AOP style Programming:

a. Code is not clumsy because primary and secondary logics are separated.
b. Secondary logics are reusable across the multiple primary logics of multiple methods in multiple classes.
c. We can enable or disable secondary logics on primary logics without disturbing source code, only by giving instructions to AOP enabled software through XML file or Annotations.
d. We can add new secondary logics on primary logics dynamically.

Spring 3.x Overview Diagram:



Spring 4.x & 5.x overview Diagram: [Both are look like same]



Explain POJI, POJO class, Java Bean class, Bean class/ Component class and Spring Bean class

POJO class (Pain Old Java Object class)

- Normal/ simple/ ordinary java classes without any specialties is called POJO class.
- The Java class that is not extending or implementing Technology/ Framework/ API specific classes or interfaces is called POJO class.
- POJO class is not against of inheriting from class and also not against of implementing interfaces. It is against of extending Technology/ Framework/ API classes and it is also against of implementing Technology/ Framework/ API interfaces.
- Generally, we can compile POJO class by using the support of JDK libraries i.e., no need of any Technology/ Framework/ API libraries (JAR files) in the CLASSPATH.

e.g.

```
class Test {
        ............
        .............
}
"Test" is POJO class
```

```
class Test implements Serializable {
        ...........                |--[Part of JDK APIs]
        ............
}
"Test" is POJO class
```

```
class Test implements java.sql.Connection {
        ..........              |--[Part of JDBC
        .........                 technology API]
}
"Test" is not a POJO class
```

Note : OOPs, Exception handling, Networking API, IO streams, Utility API, Collections, AWT, Swings, Applets, Reflection API, Muti Threading and etc. concepts/ APIs are part of Java language or JDK libraries.

```
class Test implements java.rmi.Remote {
        ...........          |--[Part of RMI technology
        ...........               API]
}
"Test" is not a POJO class
```

```
class Test extends Thread {
        ...........    |--[Part of
        JDK/ Java language
        ........  API]
}
"Test" is a POJO class
```

```
class Test extendss javas.servlet.GenericServlet {
        ...........             |--[Part of Servlet
        ...........                 technology API]
}
"Test" is not a POJO class
```

```
class Test extends Demo {          class Demo {
        ............                        ..........
        .............                       ..........
}                                  }
"Test", "Demo" class are POJO class
```

```
class Test extends Demo {     class Demo implements javax.servlet.GenericServlet {
            ............                ..........
            .............               ..........
}                             }
"Test", "Demo" class are not POJO class
```

```
class Test {
        public void m1() {
                .......
                ....... //JDBC code
        }
}
"Test" is a POJO class, but this class can be
compiled by using JDK libraries
```

```
@Remote --> given by EJB API
class Test {
        ........
        ........
}
"Test" is a POJO class, but this class
can be compiled by using JDK
libraries
```

POJI (Plain Old Java Interface)

- It is an ordinary/ simple/ normal Java Interface without any specialties.
- It is an interface that doesn't extended from Technology/ Framework/ API interfaces.
- Generally, this interface cane be compiled by using JDK libraries, i.e., no need adding third party library to CLASSPATH.

```
interface Demo {
        .......
        .......
}
"Demo" is POJI
```

```
interface Demo extends java.lang.Clonable {
        .......                 |--[part of Java api/ JDK
        .........                        libraries]
}
"Demo" is POJI
```

```
interface Demo extends java.rmi.Remote {
        .......
        .......
}
"Demo" is not a POJI
```

```
@Local -- part of EJB API
interface Demo {
        .......
        .......
}
```

```
interface Demo1 extends Demo {          interface Demo {
        .......                                 .......
        .......                                 .......
}                                       }
Demo1, Demo are POJIs
```

```
interface Demo extends Demo1 {          interface Demo1 extends java.sql.Statement {
        .......                                 .......
        .......                                 .......
}                                       }
Demo1, Demo are not POJIs.
```

- Spring is light weight because of Spring support POJO-POJI model programming.

Note: The target and Dependent class we take as Spring Beans to perform Dependency Management can be taken POJO class.

Java Bean

- Java bean is a java class that is developed by following some standards.
- Java bean doesn't contain any logic it acts as helper class to carry data from a one class to another class with in the project, or across the multiple projects.
- In order to pass more than 3 values from 1 class to another class or from one project to another project take the support of Java bean.

The standards are

a. Class must be taken public class [To visible to other class because it is a helper class].
b. Recommended to implement java.io.Serializable (I) [To share the data between to project we have to implement Serializable interface].
c. All member variables (Bean Properties) should private and non-static [Protection and allocated memory in side object for each different object].
d. Every variable (Bean Properties) should have one setter method and one getter method [setter method is used to set value/ modify value of bean property, getter method is used to read data from bean property, these methods also called as Accessor methods].
e. There should be one public zero param constructor directly (created by

the programmers) or indirectly (given by the java compiler as default constructor).

Bean class/ Component class

- The perfectly designed Java class that is having both state (member variables) and reusable behaviour (methods) and state is also used in behavior (methods) is called Bean class or Component class.

```
public class BankService {
        private String branchName;
        private String ifscCode;
        private String location;
        public String withdraw (long accno, float amount) {
                …………
                //state will be used here
                …………
                return...;
        }
        public String deposit (long accno, float amount) {
                ……………
                //state will be used here
                …………
                return….;
        }
        public String transferMoney (long SsrcAccno, long destAccno
                                        float amount) {
                ………..
                //state will be used here
                ……………
                return ….;
        }
}
```

Note: Bean class/ Component class can be developed as POJO class or non-POJO class based on requirement.

Spring Bean

- The Java class whose object is created and managed by Spring container, i.e. whose life cycle is managed by Spring container is called Spring Bean.
- Spring Bean can be pre-defined class, user-defined class or third party

supplied class.

- We can take POJO class, Java Bean class, Component classes also as Spring Beans.
- To make java class as Spring bean class it must be configured (details must begin specified) in an XML file called Spring ban configuration file or configure using Annotations.
- Any <file name>. xml of any location can be specified as Spring Bean configuration file while creating Spring container in our Spring application, but it is recommended to take applicationContext.xml as Spring bean configuration file.

Note:

- ✓ applicationContext.xml file name is no way related to ApplicationContext Spring container.
- ✓ While working with both BeanFactory and ApplicationContext containers, it is recommended to applicationContext.xml as the Spring Bean configuration file.
- ✓ The process giving details about any resource like class or interface or file and etc. to underlying server or container or framework or JVM is called configuration

applicationContext.xml

```
<beans >
        <bean id-"dt", class="java.util.Date"/> [fully qualified class name as
                Bean ID                                       Spring bean]
        <bean id-"wmg", class="com.nt.beans.WishMessageGenerator"/>
</beans>
```

- With in the Spring application/ Spring container the Spring Beans are not identifies with their fully qualifies class names. They are identified with their bean IDs.

Some F&Qs:

1. Can we develop Spring bean as POJO class? Yes,
2. Every POJO class is Spring bean? No (we should configure in XML file).
3. Every Java bean is POJO class? Yes
4. Component class can be taken Spring bean? Yes
5. Every Spring bean must be a component class? No
6. We can java class as Spring bean without using xml, annotations? No

Spring Framework

Spring Framework Details:

- Type: Java based application framework
- Version: 5.2.7 (Compatible with JDK1.8+
- Vendor: Interface21 (pivotal team)
- Create: Mr. Rod Johnson
  Open-source framework
- To download Software: [Download] dist.
- To install spring: extract zip file

Spring JAR directory:

- <Spring Home>\docs [Gives API docs, ref docs]
- <Spring Home>\libs [Gives Spring libraries of different modules (Jar file)]
- <Spring Home>\schema [Gives XSD file (XML schema definition) as rules
  to develop Spring bean files for different Spring module]

What is Spring? Define Spring

Spring is an open-source, light weight, loosely coupled, aspect oriented, non-invasive, dependency management-based Java application framework given by pivotal team of Interface21 company to develop various of Java, JEE applications in simplified manner by providing abstraction on various Java, JEE technologies and Java Frameworks.

Properties of Spring:

1. Open-source
2. Lightweight
3. Loosely coupled
4. Aspect oriented
5. Non-invasive
6. Dependency Management

Open-source:

- Spring framework is not only free and we get its source code along with installation. [Extracting zip file], for this reason Spring is an open-source framework.
- <Spring home>\ libs folders contain jars
  representing by codes of Spring modules (<name>-<ver>. release.jar)
  docs of Spring modules (<name>-<ver>. release-javadocs.jar)
  source code of Spring modules (<name>-<ver>. release-sources.jar)

Lightweight:

- Spring framework is released as zip having less size.
- Spring framework supplied containers are lightweight containers.
- By just creating object for predefined that implements BeanFactory (I), we can create BeanFactory container.

e.g.

```
//Hold the name add location of Spring bean configuration file
Resource rea = new FileSystemResource("<location>/applicationContext.xml");

//Create BeanFactory container
BeanFactory factory =
            new XmlBeanFactory(res);
```

FileSystemResource is the implementation of Resource (I) that internally uses to java.io.File class to hold the name and location of given Spring Bean configuration file.

Note: Servlet container and JSP containers are heavy weight, because we cannot create/ start programmatically, we need to start the web server/ application server in order to start Servlet container/ JSP container but web Server/ application server is heavy weight to install and to start.

Loosely coupled:

- If the degree of dependency is less between components, then they are called loosely coupled.
  e.g., TV and Remote
- If the degree of dependency is more between components, then they are called tightly coupled.
  e.g., System and keyboard

Spring framework is loosely coupled the reasons are

- Spring framework is given in the form of module having less dependency with each other.
- We can develop Spring applications by using our choice number of modules, by adding libraries of required modules.
- Spring application development supports lots of interface model Programming.
- Spring components can be developed as POJO classes and POJI without having any dependency with Spring APIs.
  and etc.

Aspect oriented:

- AOP style programming is all about separating secondary logics from primary logics during the development but mixing them dynamically at runtime.
- Spring framework gives support for Spring AOP, AspectJ AOP approaches to develop AOP style applications.

Non-invasive:

- The classes of Spring application development can be developed without having any tight coupling Spring API that means we can develop them as ordinary classes/ interfaces (POJOs and POJIs), so that we can move to other frameworks or technologies of java where ever we want. This takes Spring framework as non-invasive.

Dependency Management:

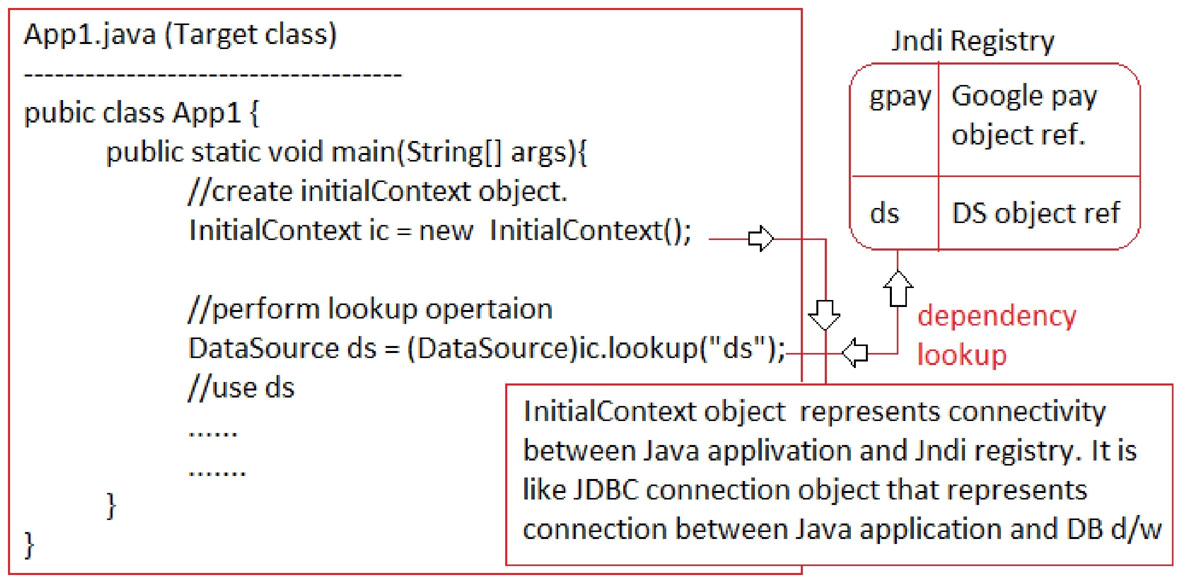
- It is all about arranging and giving dependent class object to target class object, so that target class can use dependent class services.

Two ways of Dependency Management

a. Dependency Lookup
b. Dependency Injection

Dependency Lookup:

- Here target class write some logics and spend some time to search and get dependent class object.
  e.g., Student (target) asking for course material (dependent) and getting from Institute employees.



```
App1.java (Target class)
--------------------------------------
pubic class App1 {
        public static void main(String[] args){
                //create initialContext object.
                InitialContext ic = new  InitialContext();

                //perform lookup opertaion
                DataSource ds = (DataSource)ic.lookup("ds");
                //use ds
                ......
                .......
        }
}
```

InitialContext object represents connectivity between Java applivation and Jndi registry. It is like JDBC connection object that represents connection between Java application and DB d/w

- Our applications getting objects from Jndi registry by writing Jndi Lookup code.

Note: In dependency Lookup target pulls the dependent from different places and keeps it ready for target to use.

Pros and cons of dependency lookup:

Pros:

Target can search and get only required dependent class objects.

Cons:

The target class should write some logics and should spend some to search and get dependent object class objects.

Dependency Injection:

- Here the underlying server/ container/ Framework/ JVM and etc. assigns/ injects dependent class object to target class object dynamically.
- Here dependent vis pushed to target by underlying server/ container/ JVM/ frameworks and etc.
  e.g., Students getting course material the moment he joins for the course.
- Servlet container injects/ assigns ServletConfig object to our servlet class object the moment Servlet class object created.
- JVM calls constructor automatically to assign initial values to object by calling constructor the moment object is created.
- @Resourse (-) annotations gets dependent object from Jndi registry and assigns/ inject to our Servlet class object dynamically.

Pros and cons of dependency injection:

Pros:

Target class need not to write logics or need not spend time to get the dependents.

Cons:

The underlying server/ container/ framework/ JVM and etc. may inject both necessary and unnecessary dependents.

Note: Spring supports both Dependency Lookup and Dependency Injection.

-- The END --