**Features of Spring Framework:**

- Spring came against Enterprise Java Beans:

Spring --> Ejb

- In case of Enterprise Java Beans, Application becomes heavy weight and tightly coupled. But, in c

case of Spring, Application becomes light weight and loosly coupled.

- Spring provides services which can be used for our Enterprise Application and we no need to

buy server.

- Spring is lightweigh because it only uses JDK and neither OS dependency nor Application Server

dependency.

-- Spring --> JDK (lighweight)

-- Ejb is heavy weight because it needs application server to run

-- Ejb --> Application Server (heavyweight)

**Dependency Injection and Inversion of control:**

Consider the following code:

class Student{

Car car;

Student(){

car = new MarutiSuzuki();

}

}

The above example code shows the dependency between the Student and Address.

You can say that the Student and Address are tightly coupled.

- Inversion of Control or IoC makes the code loosly coupled. The above example can be used as

follows:

Now the car which is recieved, can be of anytype as long as the third party provides the

implementation of car interface. For eg: car can be of Maruti or Honda.

Here Inversion of Control is achieved as the Student class has not control where the car

object is coming from and hence IoC is acheived.

class Student{

Car car;

Student(Car car){

this.car = car;

}

}

- In Spring Framework, the IoC container used for injecting dependencies uses the XML files

or annotations that you can use to provide metadata to the IoC container.

Spring is based on two methods:

1. Association instead of Inheritence

2. Runtime Polymorphism

Example of “Inheritence instead of Association" :

class A{

// something

}

class B{

// something

}

Now to get the properties of both the classes in a implementation class then you must extend it

class HelloWorld extends A{

// something

}

or

class HelloWorld extends B{

// something

}

Spring instead of using inheritence used association as follows:

Instead of inheriting, we create the object of the class in the implementation class

class A{

//something

}

class B{

//something

}

class HelloWorld{

A a = new A();

B b = new B();

}

2. Runtime Polymorphism:

Suppose you want to implement a class which is having a sayHello method.

There are two classes which implement sayHello method,

Suppose today your implementation class needs to implement sayHello method in English,

and you implemented this by making a associative object of sayHello method in English

But in future your requirement got changed and now you need to implement sayHello method in Japanese

and this will cause problem. So to tackle this we will use runtime polymorphism.

Instead of creating the object directly of the sayHello classes, we use

sayHello object as reference object and pass the classes(which implements sayHello interface)

as new Instance.

interface sayHello{

//something

}

class sayHelloInEnglish implements sayHello{

// something

}

class sayHelloInHindi implements sayHello{

//something

}

class sayHelloimplementation{

sayHelloInEnglish obj = new sayHelloInEnglish();

}

The above code will cause problem, so the new implmentation is:

From this code, if your future requirements got changed then no issue will be

created till the class which is being passed has implemented sayHello interface:

class sayHelloimplementation{

sayHello obj;

public sayHelloimplementation(sayHello obj){

this.obj = obj;

}

}

**Aspect Oriented Programming**:

It is a approach in which the "cross cutting concerns" are kept seperate with the business logic.

The cross cutting concerns can be Security, Logging, Profiling and Transaction Management.

These are cross cutting concerns because these are required across the applications in an enterprise.

Any application that is developed will require these cross cutting concerns.

Why there is need for AOP:

1. Cross Cutting Concerns : We can address these concerns which are common across all application in an enterprise or enterprises.

2. Reuse : Once we develop an aspect, we can use it in our application or across our application.

3. Quick Developement: We can focus on more on the business logic by keeping the aspects seperate.

4. Focus on Aspect: One developer can focus on working on one aspect for example Security and another

can work on Logging.

5. Enabling/Disabling: We can enable and disable the aspects according to our requirements.

Spring framework has the following parts:

1. Spring Core Container: