**4. Spring Inversion of Control (IOC)**

**What is Inversion of Control**:

Inversion of Control is a principle in software engineering by which the control of objects or portions of a program is transferred to a container or framework. It’s most often used in the context of object-oriented programming.

By contrast with traditional programming, in which our custom code makes calls to a library, IoC enables a framework to take control of the flow of a program and make calls to our custom code. To enable this, frameworks use abstractions with additional behavior built in. If we want to add our own behavior, we need to extend the classes of the framework or plugin our own classes.

**The advantages of this architecture are**:

1. Decoupling the execution of a task from its implementation
2. Making it easier to switch between different implementations
3. Greater modularity of a program
4. Greater ease in testing a program by isolating a component or mocking its dependencies and allowing components to communicate through contracts

Inversion of Control can be achieved through various mechanisms such as: Strategy design pattern, Service Locator pattern, Factory pattern, and Dependency Injection (DI).

**(Wikipedia)**:

In software engineering, inversion of control (IoC) is a programming principle. IoC inverts the flow control as compared to traditional control flow. In IoC, custom-written portions of a computer program receive the flow of control from a generic framework. A software architecture with this design inverts control as compared to traditional procedural programming: in traditional programming, the custom code that expresses the purpose of the program calls into reusable libraries to take care of generic tasks, but with inversion of control, it is the framework that calls into the custom, or task-specific, code.

Inversion of control is used to increase modularity of the program and make it extensible,[1] and has applications in object-oriented programming and other programming paradigms. The term was used by Michael Mattsson.

**Chad Darby**:

The approach of outsourcing the construction and management of object is Inversion of Control.

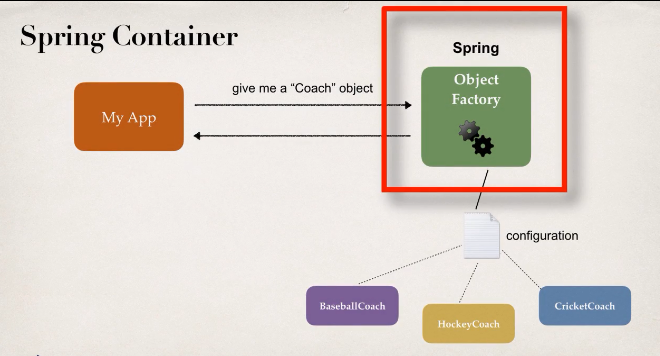
**Spring Container**:

Spring provides an object factory, so we can have our application talk to spring say, hey give me an object based on a configuration file or annotation. Spring will give us the appropriate implementation. So, then our app is configurable.

**Primary functions**:

The primary function of a spring container is

1. Create and manage objects (IOC)
2. Inject object’s dependencies (DI)



Spring Container

**Configuring Spring Container**:

There are three ways of configuring the Spring container.

1. XML configuration file (legacy, but most legacy apps still use this)
2. Java Annotations (modern)
3. Java Source Code (modern)

**1) XML configuration**:

For XML configuration the Spring Development process is

1. Configure our Spring Beans
2. Create a Spring Container
3. Retrieve Beans from Spring Container

**a) Configure our Spring Beans**:

**File: applicationContext.xml**:

<beans ...>

<bean id=*"myCoach"*

class=*"com.lvu2code.springdemo.TrackCoach"*>

</bean>

</beans>

Here the id is the bean id and class is the fully qualified name of the implementation class.

**b) Create a Spring Container**:

Spring container is generally known as **ApplicationContext**

Specialized implementations

1. ClassPathXmlApplicationContext
2. AnnotationConfigApplicationContext
3. GenericWebApplicationContext
4. Other ….

**Create a Spring Container**:

ClassPathXmlApplicationContext context =

**new** ClassPathXmlApplicationContext("applicationContext.xml");

Here we goanna read an XML file that’s on our class path. We use the "ClassPathXmlApplicationContext" and construct this object and pass in the actual name of the configuration file.

**c) Retrieve Beans from Spring Container**:

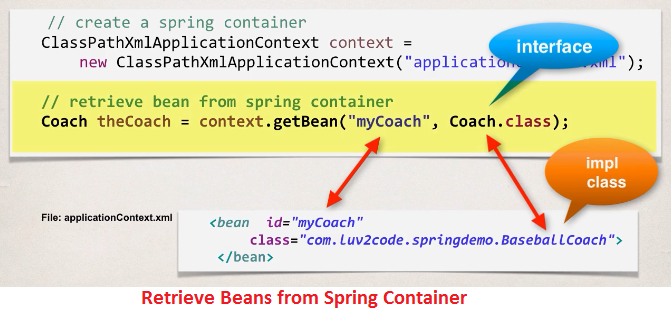
//load the spring configuration file

ClassPathXmlApplicationContext context =

**new** ClassPathXmlApplicationContext("applicationContext.xml");

//retrieve bean from spring container

Coach theCoach = context.getBean("myCoach", Coach.**class**);



4. Spring Inversion of Control