09 Section 09 Inversion of Control

Lets consider the traditional way of writing the spring program.

So what we used to do is create the class and then we used to initialize its object by using the new keyword. There was the time when we used to do this traditional coding for practice purpose we used to declare new object every time for different values whenever needed.

```
Car C1 =new Car();
Car C2 =new Car(); and so on...
```

But when we work on real life application this is not the case. We create only once instance of the class and use it wherever necessary.

This means every time we need a new instance, we explicitly create it. However, in real-world applications, manually managing object creation is inefficient and makes the code tightly coupled.

In spring boot we also externalized the supply of these objects and we ask the spring to provide it to us. Now how does Spring do that?

As we are asking this to spring to provide me objects of the class in one way we are asking to invert the control. We say that you only control the object life cycle. Spring boot will be the one to control the life cycle of the object.

Hence here all the objects of all the necessary classes are stored in the container that is called as Inversion of control. This is the container which stores all the objects in it manages them and injects them where ever it is necessary for us in the program.

What are beans?

All objects managed by the **IoC Container** are called **Beans**. A **Bean** is simply an instance of a class that Spring manages.

Application Context is something which is necessary to achieve the inversion of control and this both terms can be used interchangeably.

Going forward, How does the inversion of control knows that which is the class for which I must create beans of?

IOC container scans the packages for the classes. It will create the beans of the classes only which are annotated with the annotation of **@Component.**
When Spring detects this annotation during **Component Scanning**, it registers the `Car` class as a bean inside the **IoC container**.

```
@Component
public class Car{
      //body of class
}
```

IOC container will scan this class and creates its bean in it. Using @Component will make the class to automatically registered as bean/object.