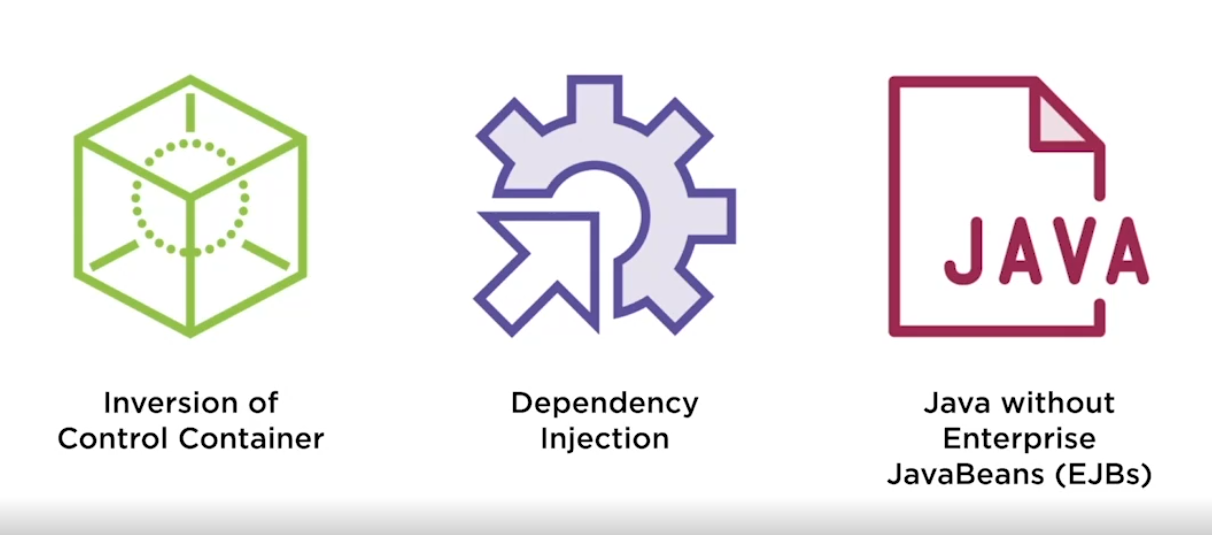
**Spring Framework and Fundamentals**

What is Spring?

Spring framework is started out as an **inversion of control container** Often refer as **Dependency injection**.

It was used to replace some of the complex configuration of earlier Java enterprise edition. How we can do same things without using EJBs.



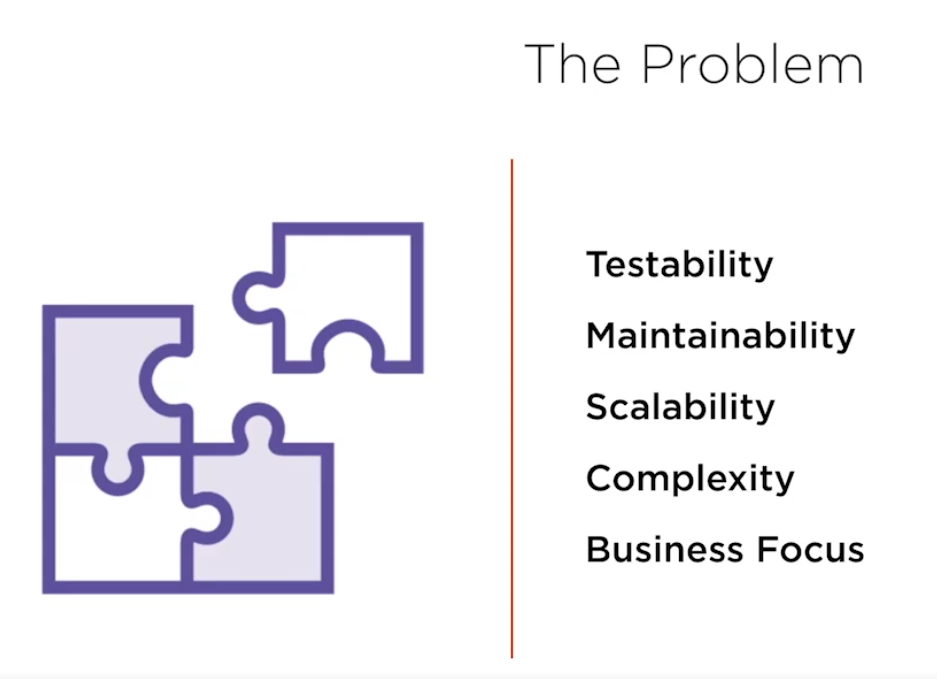
* It was started to reduce complexity in enterprise java development and later also providing enterprise development without EJBs.
* Spring can be used with or without java beans primarily now is used without EJBs.
* Spring allowed us to do enterprise development without **application server**. Tomcat that we use is isn’t an application server rather it is a web server.
* It is complex to do development with application server like WebSphere and with tomcat as it is light weight can be used easily.

So finally, what is spring?



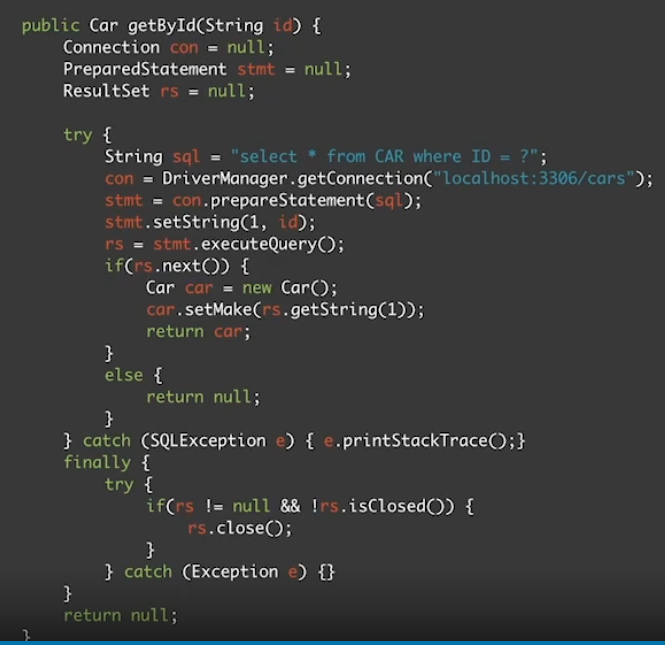
* JEE
* **POJO** – plain old java object
* **Unobtrusive**- As spring was developed to reduce complexity of enterprise development so it will not come in ay of development.
* **AOP/Proxies** – Spring also use AOP/Proxies to apply things like transactions to code to get those cross-cutting concerns out of our application so our code is smaller and more light weight.
* **Best Practices** – Spring is built around best practices

**Advantages of Spring:**



* Increases testability and maintainability. Helps in scaling our code.
* It decouples things and helps in add caching layer
* Reduce code complexity
* Focus on business, business doesn’t care about what framework we use it cares about things done and spring helps us do complex things faster.

**PROBLEM**



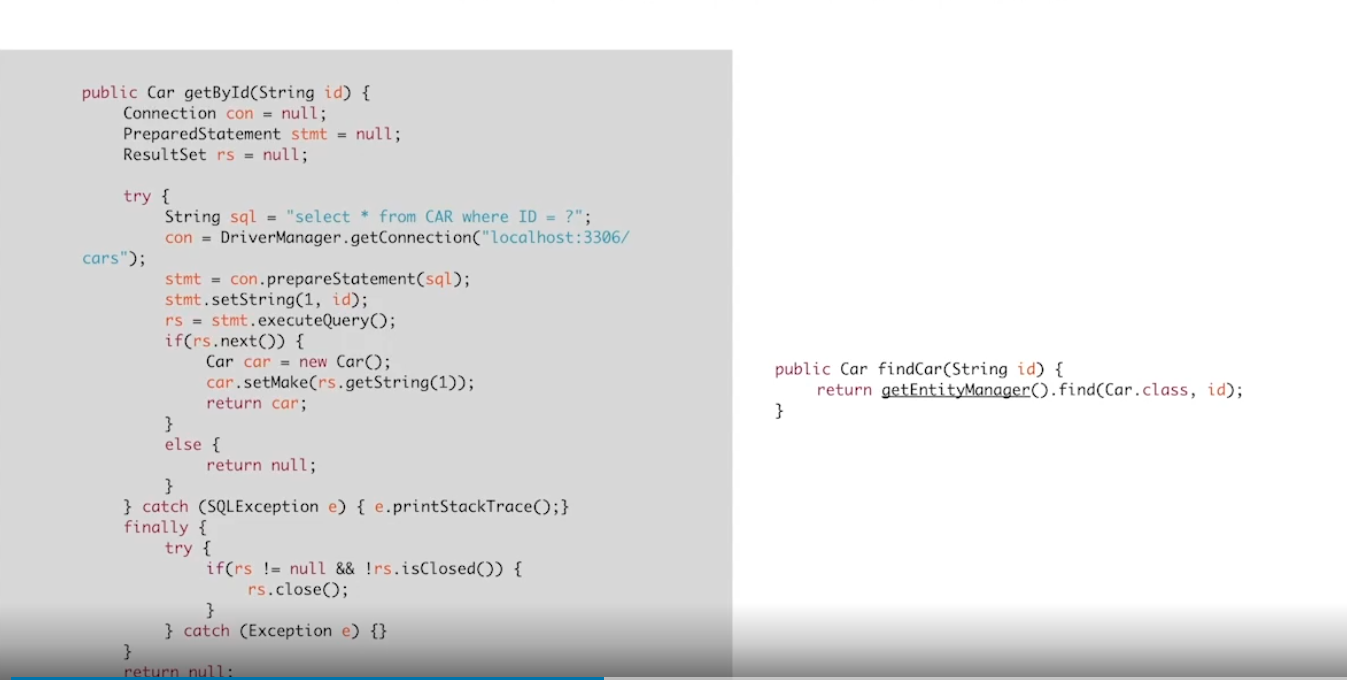
The only thing that business care about in the above code is selecting the car object and setting it. All these initialization, try/catch , big finally block etc etc just makes our code complex. Lets see how Spring resolves it.

THE SOLUTION

The solution we get from spring or using spring framework is:

* We can remove **Configuration** code or lookup code
* Developers can **Focus** upon business needs. Business doesn’t care about try/catch , exceptions etc
* **Increase TESTING**
* **Annotation/XML** based development – Rather than having big configuration code we can have annotation-based development.
* To achieve all of this spring encourage us for **interface-based development.**

**BUSINESS FOCUS REVISTED**



This library used on right is basically Spring-JDBC template code. It removes all configuration, we don’t need to care about all these. By using **TEMPLATE-Method pattern** we have achieved it.

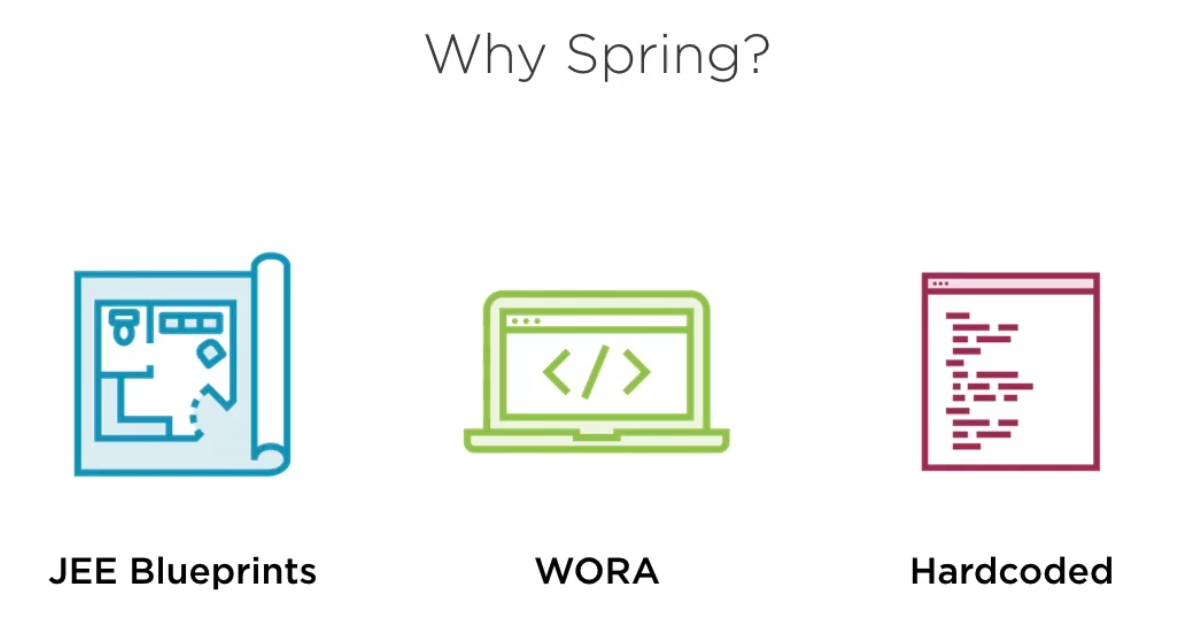
**So it basically helps us to achieve Business goals faster.**

**HOW SPRING HELPS US IN BUSINESS FOCUS?**

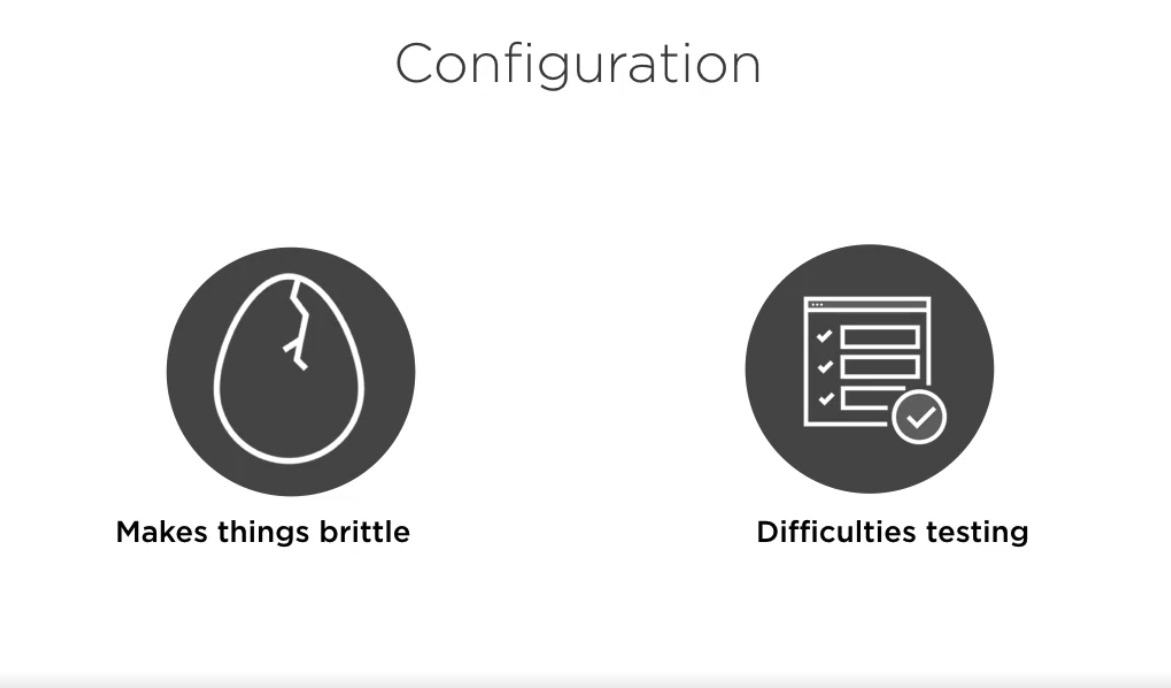
* **POJOs**
* **Hash Map**- Spring is thought of as a glorified hash map, it calls the Application context. Application context is configured spring container with all of the dependency wired up.
* **Registry**

**WHY SPRING WAS DEVELOPED?**

* Spring was developed to make existing task easier.
* Before spring we used some design patterns from **JEE blueprints** to help establish better code and repeatable process. This was helpful but often **made code brittle and untestable.**
* **WORA(Write once run anywhere)-** Another problem was if we have to recompile code into other environment or need to change URL etc.
* Actual implementation should not be **hardcoded inside application.**







We should always get rid of configuration code like

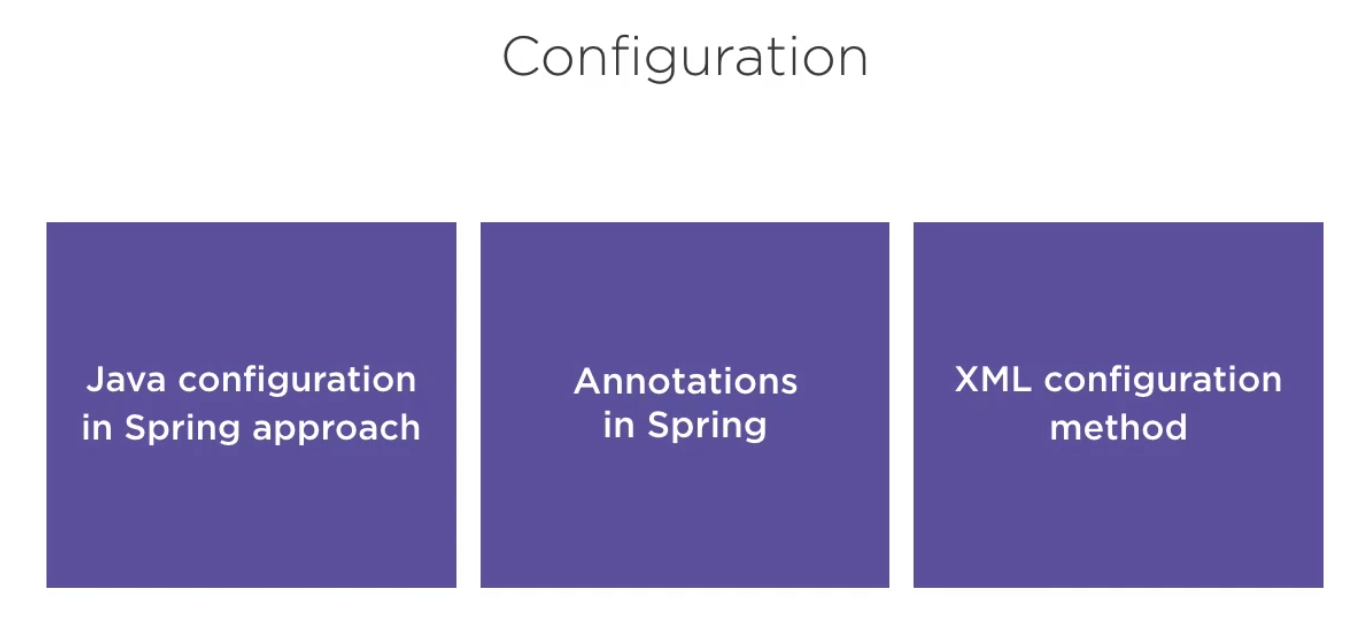
**public class** SpeakerServiceImpl **implements** SpeakerService {  
  
 *//This is the painpoint where we have to manually create a object of repo class. We should reduce Configuration code from our application  
 // cause configuration code is brittle (hard to move to diff env)* **private** SpeakerRepository **speakerRepository** = **new** HibernateSpeakerRepositoryImpl();  
  
 **public** List<Speaker> findAll(){  
 **return speakerRepository**.findAll();  
 }  
}

In the above code we have manually create the instance with new and we have hardcoded the reference of HibernateSpeakerRepositoryImpl object , if we make any changes we need to rebuild the entire application and such type of configuration we don’t want because.

* These are brittle, hard to move to different server env.
* Hard to test.

**CONFIGURATIONAL code is something that we don’t require as a business logic. Hence, we should minimize that.**

**In Spring we can deal with configuration using below:**



**In order to use Spring in our project we need to download Spring libraries from Maven using POM file.**

**Once we do this**

<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-context</**artifactId**>  
 <**version**>5.3.9</**version**>  
 </**dependency**>  
</**dependencies**>

**Then it will download all external libraries**

