FAQ: How @Bean works behind the scenes

Question:

During All Java Configuration, how does the @Bean annotation work in the background?

Answer

This is an advanced concept. But I'll walk through the code line-by-line.

For this code:

```
1.   @Bean
2.   public Coach swimCoach() {
3.    SwimCoach mySwimCoach = new SwimCoach();
4.    return mySwimCoach;
5.   }
```

At a high-level, Spring creates a bean component manually. By default the scope is singleton. So any request for a "swimCoach" bean, will get the same instance of the bean since singleton is the default scope.

However, let's break it down line-by-line

```
1. @Bean
```

The @Bean annotation tells Spring that we are creating a bean component manually. We didn't specify a scope so the default scope is singleton.

```
1.  public Coach swimCoach(){
```

This specifies that the bean will bean id of "swimCoach". The method name determines the bean id. The return type is the Coach interface. This is useful for dependency injection. This can help Spring find any dependencies that implement the Coach interface.

The @Bean annotation will intercept any requests for "swimCoach" bean. Since we didn't specify a scope, the bean scope is singleton. As a result, it will give the same instance of the bean for any requests.

```
1.   SwimCoach mySwimCoach = new SwimCoach();
```

This code will create a new instance of the SwimCoach.

```
1.   return mySwimCoach;
```

This code returns an instance of the swimCoach.

Now let's step back and look at the method in it's entirety.

```
1.   @Bean
2.   public Coach swimCoach() {
3.     SwimCoach mySwimCoach = new SwimCoach();
4.     return mySwimCoach;
5.   }
```

It is important to note that this method has the @Bean annotation. The annotation will intercept ALL calls to the method "swimCoach()". Since no scope is specified the @Bean annotation uses singleton scope. Behind the scenes, during the @Bean interception, it will check in memory of the Spring container (applicationContext) and see if this given bean has already been created.

If this is the first time the bean has been created then it will execute the method as normal. It will also register the bean in the application

context. So that is knows that the bean has already been created before. Effectively setting a flag.

The next time this method is called, the @Bean annotation will check in memory of the Spring container (applicationContext) and see if this given bean has already been created. Since the bean has already been created (previous paragraph) then it will immediately return the instance from memory. It will not execute the code inside of the method. Hence this is a singleton bean.

The code for

```
1.   SwimCoach mySwimCoach = new SwimCoach();
2.   return mySwimCoach;
```

is not executed for subsequent requests to the method public Coach swimCoach() . This code is only executed once during the initial bean creation since it is singleton scope.

That explains how @Bean annotation works for the swimCoach example.

====

Now let's take it one step further.

Here's your other question

>> Please explain in detail whats happening behind the scene for this statement.

```
1.  return new SwimCoach(sadFortuneService())
```

The code for this question is slightly different. It is injecting a dependency.

In this example, we are creating a SwimCoach and injecting the sadFortuneService().

```
1.          // define bean for our sad fortune service
2.          @Bean
3.          public FortuneService sadFortuneService() {
4.              return new SadFortuneService();
5.          }
6.
7.          // define bean for our swim coach AND inject dependency
8.          @Bean
9.          public Coach swimCoach() {
10.             SwimCoach mySwimCoach = new SwimCoach(sadFortuneService());
11.
12.             return mySwimCoach;
13.         }
```

Using the same information presented earlier

The code

```
1.          // define bean for our sad fortune service
2.          @Bean
3.          public FortuneService sadFortuneService() {
4.              return new SadFortuneService();
5.          }
```

In the code above, we define a bean for the sad fortune service. Since the bean scope is not specified, it defaults to singleton.

Any calls for sadFortuneService, the @Bean annotation intercepts the call and checks to see if an instance has been created. First time through, no instance is created so the code executes as desired. For subsequent calls, the singleton has been created so @Bean will immediately return with the singleton instance.

Now to the main code based on your question.

```
1.  return new SwimCoach(sadFortuneService())
```

This code creates an instance of SwimCoach. Note the call to the method sadFortuneService(). We are calling the annotated method above. The @Bean will intercept and return a singleton instance of sadFortuneService. The sadFortuneService is then injected into the swim coach instance.

This is effectively dependency injection. It is accomplished using all Java configuration (no xml).

This concludes the line-by-line discussion of the source code. All of the behind the scenes work.

I hope this clears your doubt. :-)