Example 1

```
package com.springframework.domain;
public class Product
 {
	private String description;
	public Product(String description)
	 {
		this.description = description;
	}
	public String getDescription()
	 {
		return description;
	}
	public void setDescription(String description)
	{
		this.description = description;
	}
}
```

```
package com.springframework.services;

public interface ProductService
 {
	List<Product> listProducts();
}
```

```
package com.springframework.services;


@Service
public class ProductServiceImpl implements ProductService
{
	@Override
	public List<Product> listProducts()
	 {
		ArrayList<Product> products = new ArrayList<Product>(2);
		products.add(new Product("Product 1 description"));
		products.add(new Product("Product 2 description"));
	return products;
	}
}
```

```
package com.springframework.controllers;


@Controller
public class MyController
{
	private ProductService productService;
	@Autowired
	public void setProductService(ProductService productService)
	{
		this.productService = productService;
	}
	public List<Product> getProducts()
	{
		return productService.listProducts();
	}
}
```

```
package diexample;

import com.springframework.controllers.MyController;
import com.springframework.domain.Product;

@SpringBootApplication
@ComponentScan("com.springframework")

public class DiExampleApplication
{
	public static void main(String[] args)
	 {
		ApplicationContext ctx =
	SpringApplication.run(DiExampleApplication.class, args);
		MyController controller = (MyController) ctx.getBean("myController");
		List<Product> products = controller.getProducts();
		for(Product product : products)
		{
			System.out.println(product.getDescription());
		}
	}
}
```

@Service

, this tells Spring this class is a Spring Bean to be managed by the Spring Framework. This step is critical, Spring will not detect this class as a Spring Bean without this annotation. Alternatively, you could explicitly define the bean in a Spring configuration file.

On the setter, you see the

@Autowired

annotation. This directs Spring to inject a Spring managed bean into this class. Our controller class is also annotated with the

@Controller

annotation. This marks the class as a Spring Managed bean. Without this annotation, Spring will not bring this class into the context, and will not inject an instance of the service into the class.

@ComponentScan

annotation. By using this annotation Spring will scan the specified package for Spring components (aka Spring managed beans).

In our main method, we get the Spring Context, then request from the context an instance of our controller bean. Spring will give us an instance of the controller. Spring will perform the Dependency Injection for us, and inject the dependent components into the object returned to us.

It is important to remember, the Spring Context is returning to us Spring Managed beans. This means Spring will be managing the dependency injection for us. If for some reason, Spring cannot fulfill a dependency, it will fail to startup. You will see in the stack trace information about the missing dependencies.

Example 2

Example Service

```
@Service
public class MyService {
public String getHello(){
return "Hello";
}
}
```

Private Field Controller

```
@Controller
public class PrivateFieldController {
@Autowired
private MyService myService;
public String saySomething(){
return myService.getHello();
}
}
```

Method Injection

Setter Controller

```
@Controller

public class SetterController {

private MyService myService;

@Autowired

public void setMyService(MyService myService) {

this.myService = myService;

}

public String saySomething(){

return myService.getHello();

}

}
```

Constructor Injection

Constructor Controller

```
@Controller
public class ConstructorController {
private MyService myService;
public ConstructorController(MyService myService) {
this.myService = myService;
}
public String saySomething(){
return myService.getHello();
}
}
```

Best Practice Service Interface

```
public interface BpService {
String getHello();
}
```

Best Practice Service Implementation

```
@Service
public class BpServiceImpl implements BpService {
@Override
public String getHello() {
return "The Best Hello!";
}
}
```

Using Project Lombok

Now, the secret sauce using Project Lombok for best practices in dependency injection is to:

- declare a final property of the interface type
- annotate the class using Project Lombok's required args constructor

Now, Project Lombok will generate a constructor for all properties declared final. And Spring will automatically use the Lombok provided constructor to autowire the clase.

Lombok Controller

```
@RequiredArgsConstructor

@Controller

public class BpFinalConstructorController {

private final BpService bpService;

public String saySomething(){

return bpService.getHello();

}
}
```