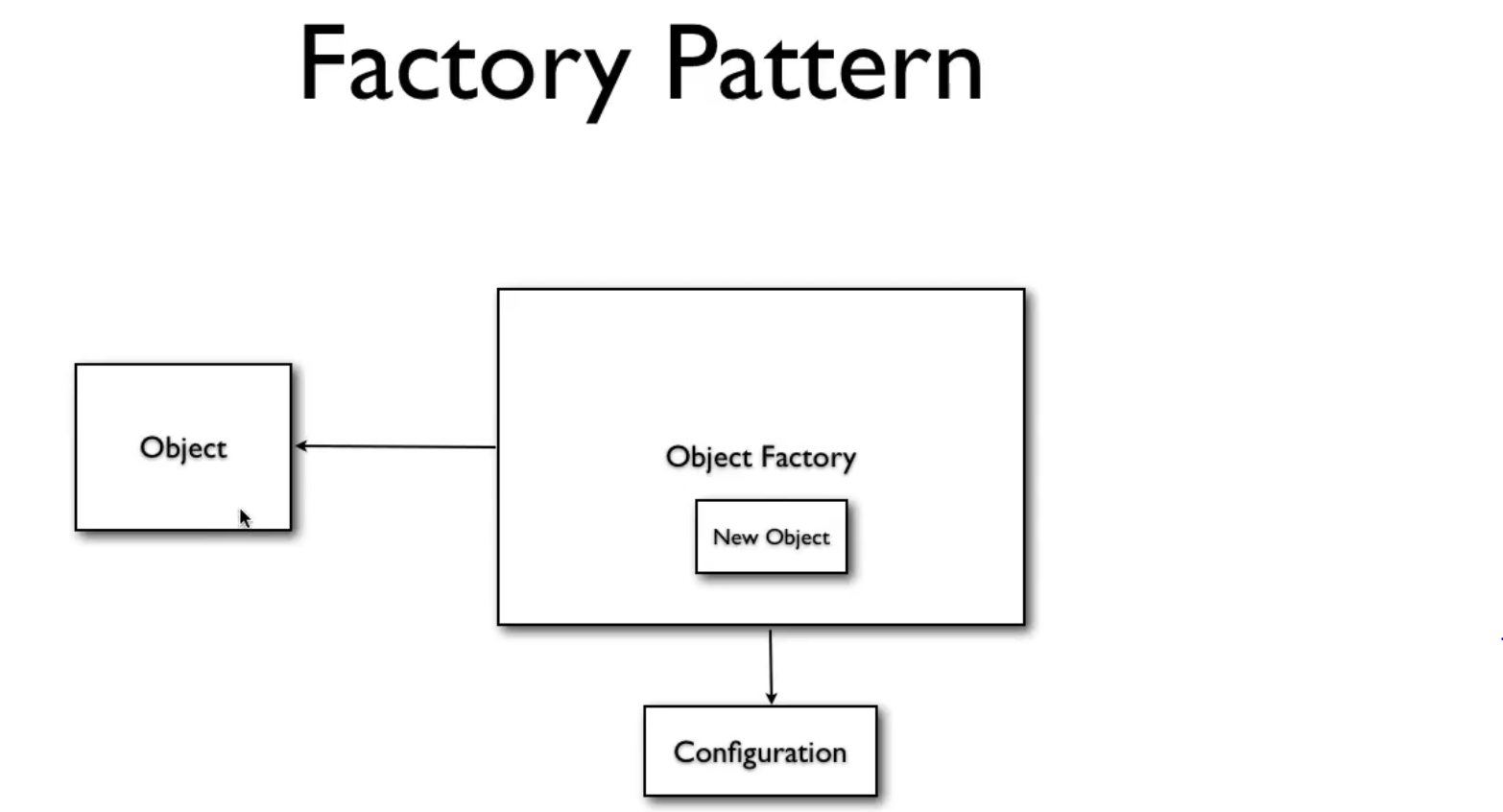
1. **Understanding dependency injection**
2. **Setting up**

Download spring jars from “<https://repo.spring.io/release/org/springframework/spring>”. Create a java project with name ‘SpringDemo’. Right click the project -> properties -> java build path -> libraries -> Add external jars -> browse to the downloaded jars (they are inside libs) and select them all.

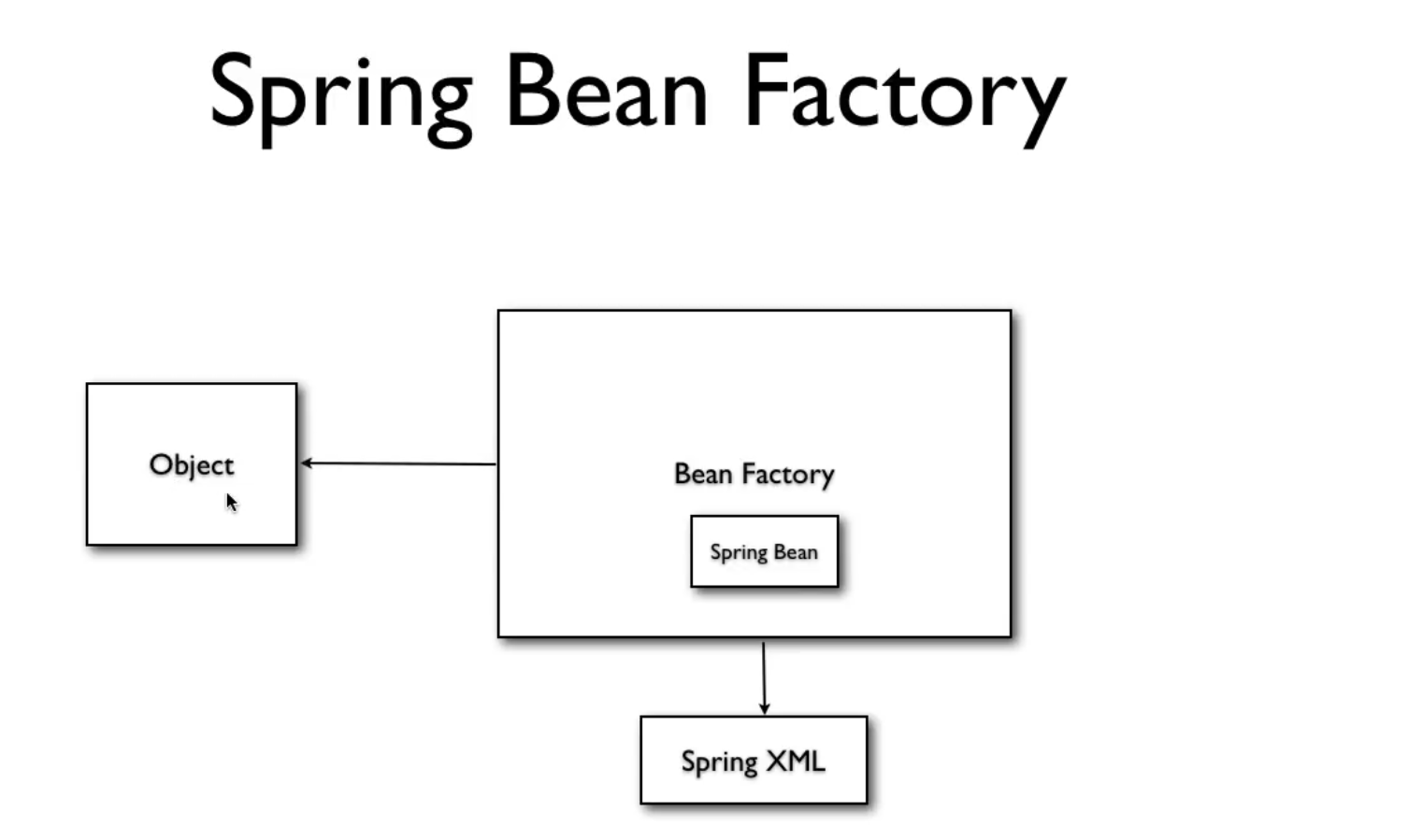
We also need apache common logging jars. Download them from “[https://commons.apache.org/logging/](https://commons.apache.org/logging/download_logging.cgi)” . Set ‘commons-logging-1.2.jar’ also as it was done for previous one.

1. **Understanding spring bean factory**

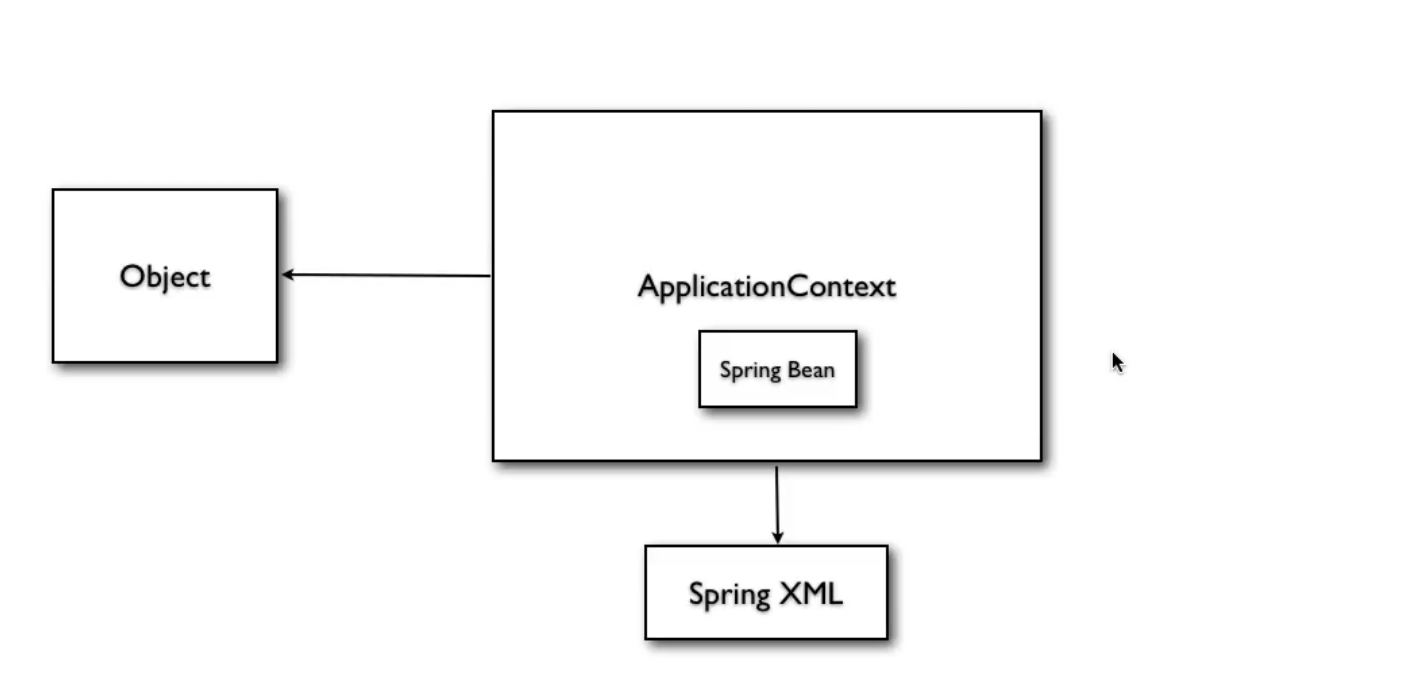
Factory pattern- Objects can be created inside spring container and outside it. If object will be created outside container, then spring will not be able to know about it and hence have no control over it. So we make spring initialize the object and then it supplies the object where it is required.



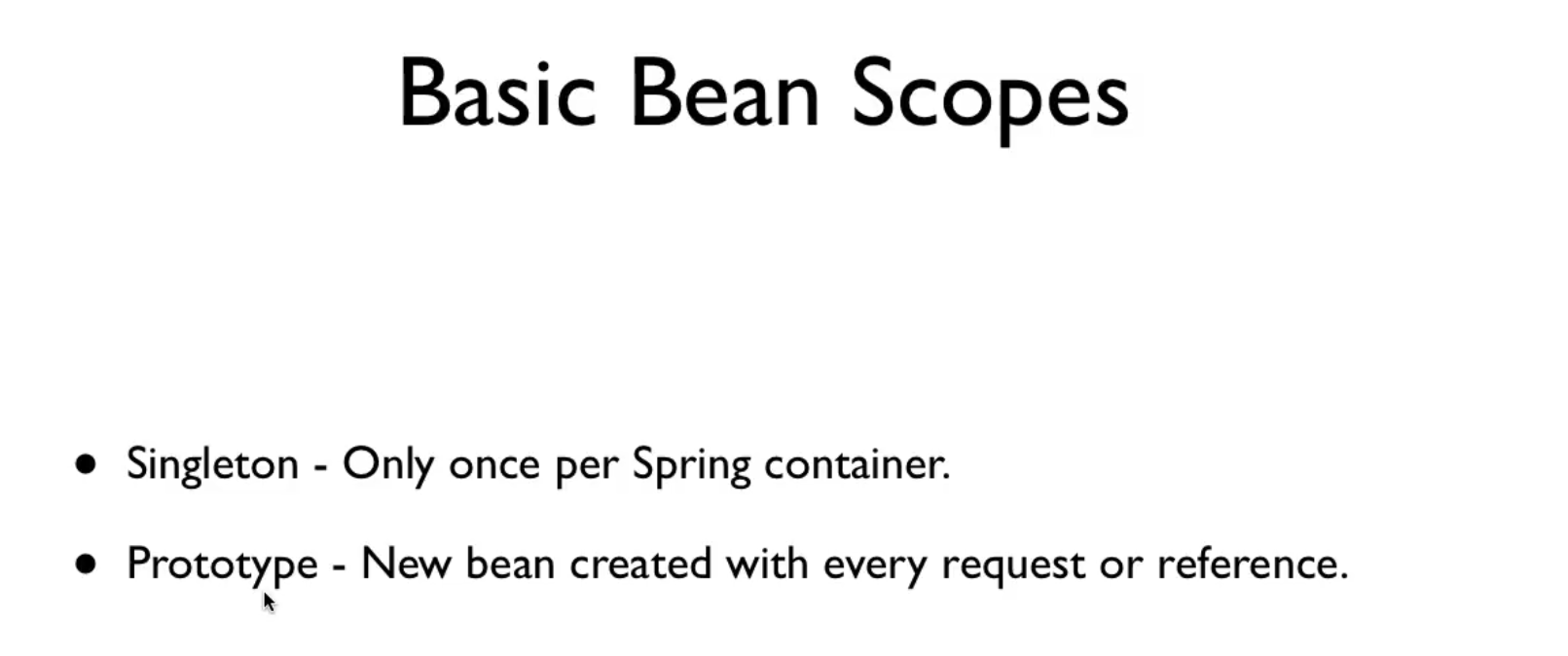
Request for creation of new object came. Object factory checked configuration file which contains blueprint of creation of all object. On the basis of it, it will create new object and supplies this new object to object.



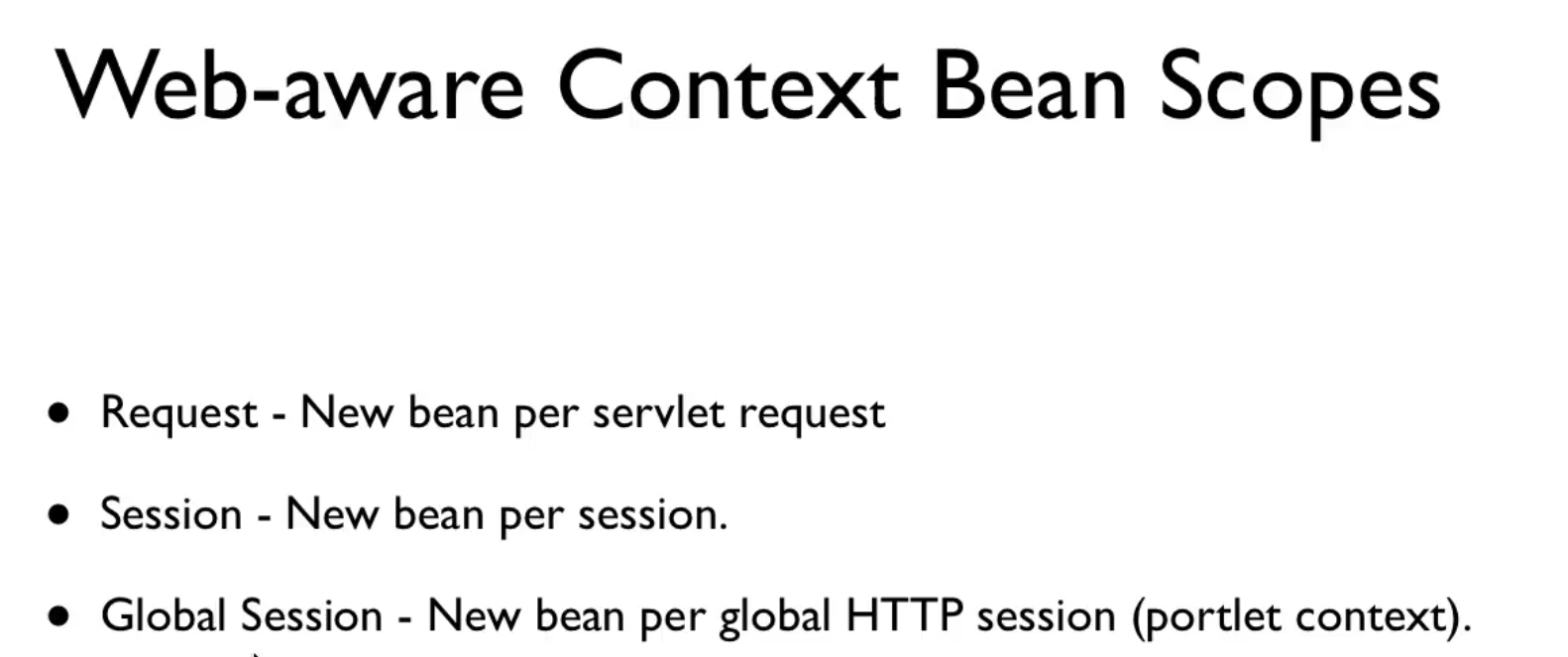
11. **Understanding bean scope**



When ApplicationContext is defined, Applicationcontext creates beans by reading the configuration file. When getBean method is called, then this bean is assigned to the object.

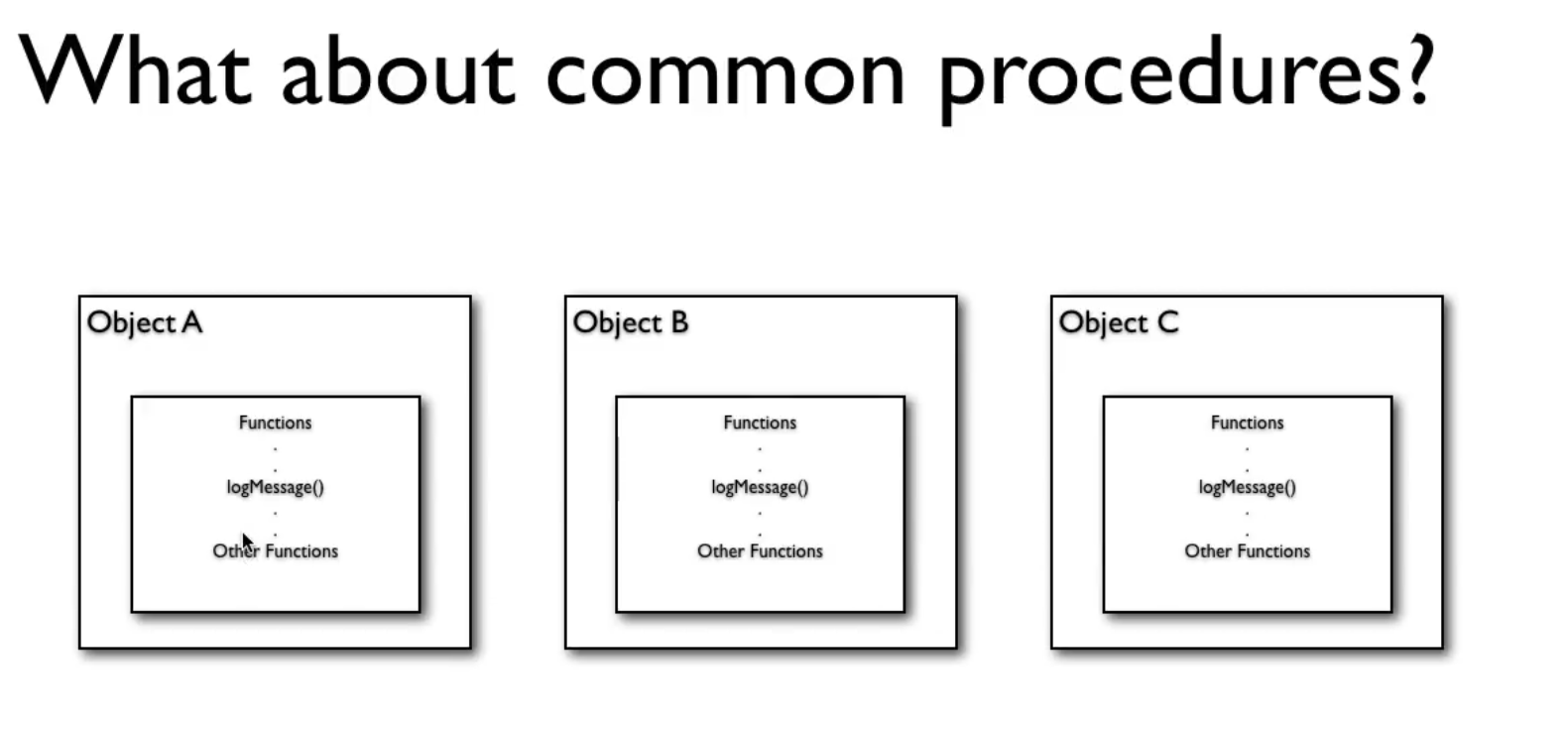


Singleton is by default. In this, beans are created when applicationContext is done and then same beans are used for all the object request. In prototype, beans are created only when request for object arrives and for each request, new beans are created.

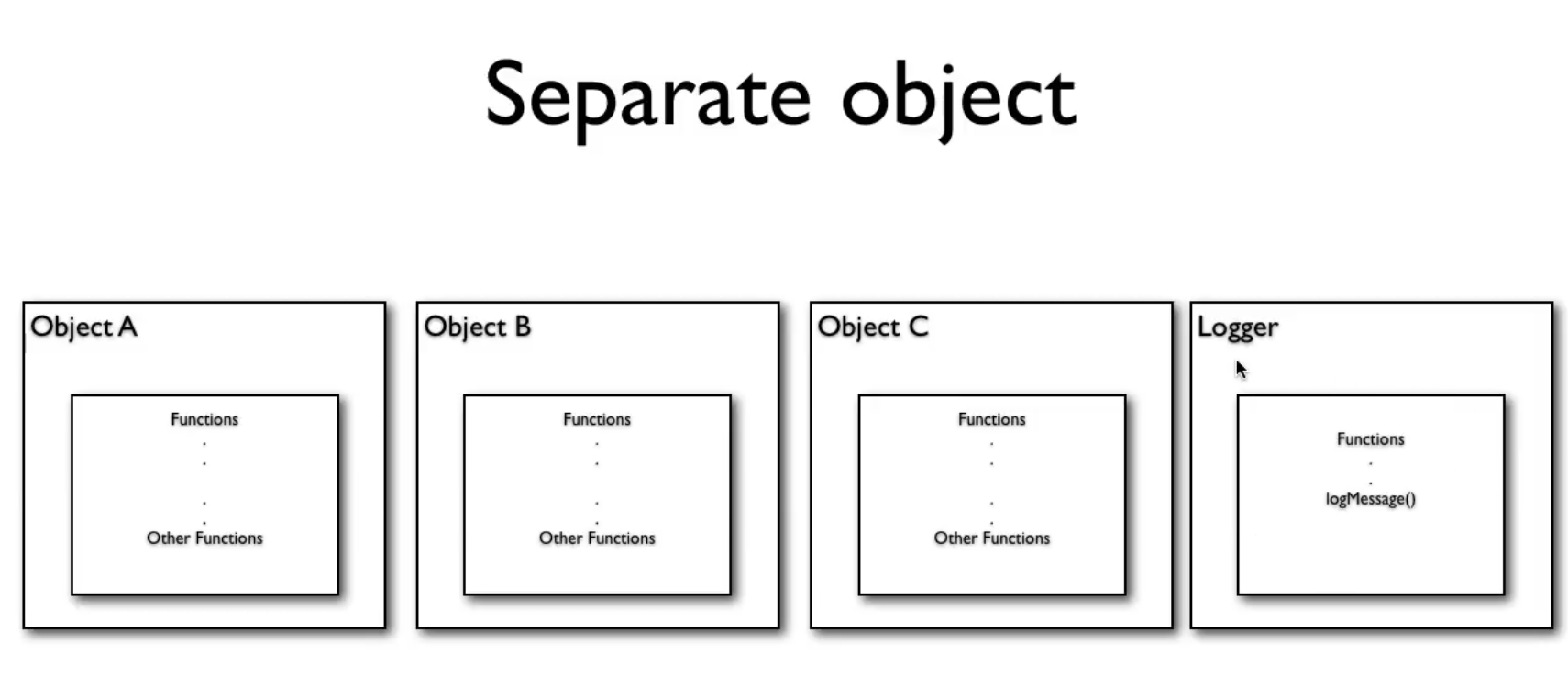


24. **Aspect oriented programming**

AOP is a model of programming. First came functional programming. Then came object oriented programming. But there is one problem with oop.

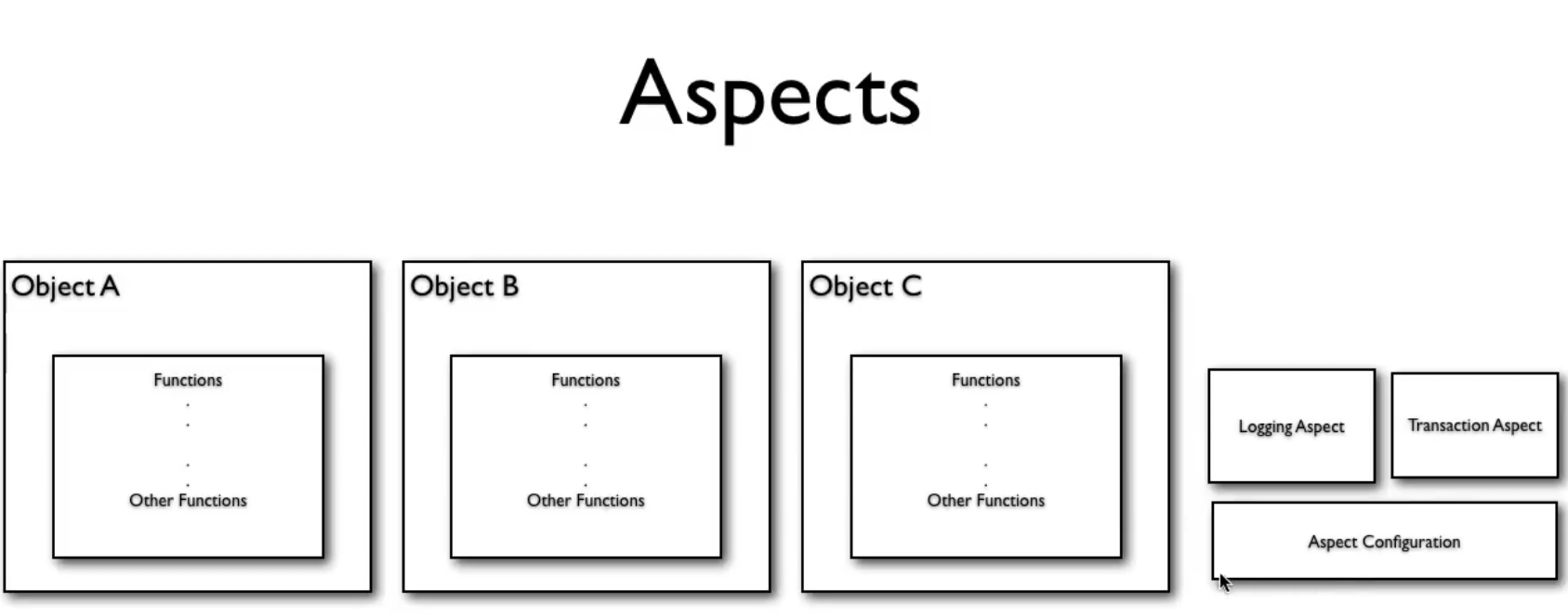


Here log is a common procedure in all classes.



So we have created separate class for logger. But still there is one problem. If we draw diagram of this, then logger will appear out to be the most imp object and not other business objects because all other objects are seem to be dependent on logger. Objects like logger are called crosscutting objects (objects that affects too many objects). Some other cross cutting concerns are- transactions, security etc. Another problem is that we still need to write lot of code inside every object to make a call to logger object.

Solution to above problems is AOP.



What we are doing here is that we are creating aspects (these are classes with some special privileges). Now our classes will not call for logging object. For that we have aspect configuration which tells for which object in which method this aspect applies to.

Spring’s DI helps in this.

