**Introduction:**

In Java application development platform, interceptors are used to enable the develops to call the interceptors class which varies in terms of methods on attached target class through lifecycle event. Intercepts are known to use the metadata annotations in conjunction with the target class or as an individual interceptor class. These are required to answer the concerns of the developers over the design flow and to make sure application meets the specification provided. These concerns cannot be separated from the system flow nor they can be treated in conjunction with the system flow. It actually belongs to a system outcast which proves to be crucial to incorporate with the system parts. This defines to be an object-oriented form to handle the application system environment.

Java platform possesses the interceptors as component of the system which have to implement the cross-cutting concerns. Interceptors can be defined inside the target class as a method to intercept or as an interceptor class. These class comprises the methods that are called within the lifecycle events of the target class. These interceptor metadata annotations are summarized below.

|  |  |
| --- | --- |
| Interceptor Metadata Annotation | Details |
| javax.interceptor.AroundInvoke | It assigns the interceptor method. |
| javax.annotation.PreDestroy | It assigns the method before destroying the lifecycle events |
| javax.interceptor.AroundTimeout | It assigns the interceptor method to timeout and impose it as enterprise bean method. |
| javax.annotation.PostConstruct | It assigns the method to intercept after construct lifecycle events. |

Table 1. Metadata Annotations

**Method Interceptors:**

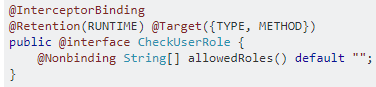
Interceptors are usually used to invoke target class over the lifecycle event within a business method to either develop or remove timeout. It is used to implement in application like logging, auditing and other similar forms. For example, when it is required to fill customer details into the system. It will index each individual customer details through logging in history of the customer that are changed in the database. It can also index the permission or authorization that the customer has while executing any transaction on the system. These are the basic cross cutting concerns of a simple customer database application which have to be considered while creating the system.

An enforced attempt to inject these concerns into the system may interfere with the system which may result in tangling and scattering but this disruption can be avoided by use of correct interceptors for the system. Interceptors are the best method while initializing the callback when the developers wants to call a specific interceptor method after the beans have been initiated but before the transaction towards the client. This initiation method can look over any information to configure state of the model and can also visualize the resources being used in JNDI environment.

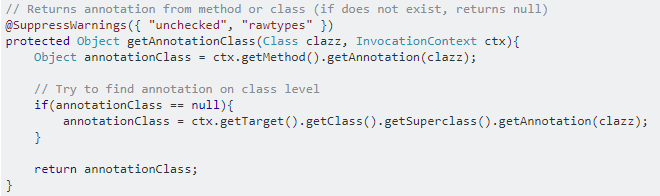
Interceptor lifecycle defines the life of the interceptor classes when used in conjunction with the target class. When the target class is created, it also declares the class instances in the target class. While if the target class is using multiple classes, it also creates multiple instances within the target class instances. These instances are completely configured before @postconstruct metadata is invoked. The other metadata @Predestroy is invoked only when the target class has not been called in the system and after the class interceptors have been destroyed.

Java Interceptors class are ideal when cross cutting concerns are treated in isolation while calling them in the form of separate class which enables the developers to access these classes again in a different bean cell environment. Initially interceptors were only assigned the EJBs. With the increased complexity in the environment and cross cutting concerns the specification of the interceptor class were improved through enabling the system to introduce new interceptor method after it has been called before. This also binds the annotation with execution of the interception in JAVA run-time environment

While using data access field in Java environment using interceptors following is the presented example according to metadata annotations.







The interceptor is initially defined to check the user permissions and define the interceptor as a public class which runs through the database to check whether the user information exist in the database or not and if it does not the return value is null otherwise it follows through the program to present the available information at class level.

**References:**

1. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., & Stefani, J.-B. (2006). The FRACTAL component model and its support in Java. Software Practice and Experience, 1257-1284.

2) Daniel, L., Navarro, B., Sudholt, M., Vanderperren, W., Fraine, B. D., & Suv´ee, D. (2006). Explicitly distributed AOP using AWED. System Communicants, 33.

3) Ghag, G. (2007, January 5). Implement crosscutting concerns using Spring 2.0 AOP. Retrieved from JavaWorld: <https://www.javaworld.com/article/2077666/web-app-frameworks/implement-crosscutting-concerns-using-spring-2-0-aop.html>

4) Juneau, J. (2013). Introducing Java EE 7. Berkeley, Calif.: Apress.

5) Noble, J., Schmidmeier, A., Pearce, D. J., & Black, A. P. (2007). Patterns of Aspect-Oriented Design. Euro PLOP (p. 27). Hillside Europe.

6) Oracle. (2018). 54.2 Using Interceptors. Oracle.

7) Oracle. (2018). Lesson 8: Object-Oriented Programming. Oracle.

8) Oracle. (2018). The Java EE 6 Tutorial. Oracle.

9) Oracle. (2018). Using Interceptors in CDI Applications. Oracle.

10) Sabot-Durand, A. (2014, June 10). You think you know everything about CDI events… Think again! Retrieved from Next Presso: <http://www.next-presso.com/2014/06/you-think-you-know-everything-about-cdi-events-think-again/>

11) Spring. (2018). 4.11 Annotation-based configuration. Spring.