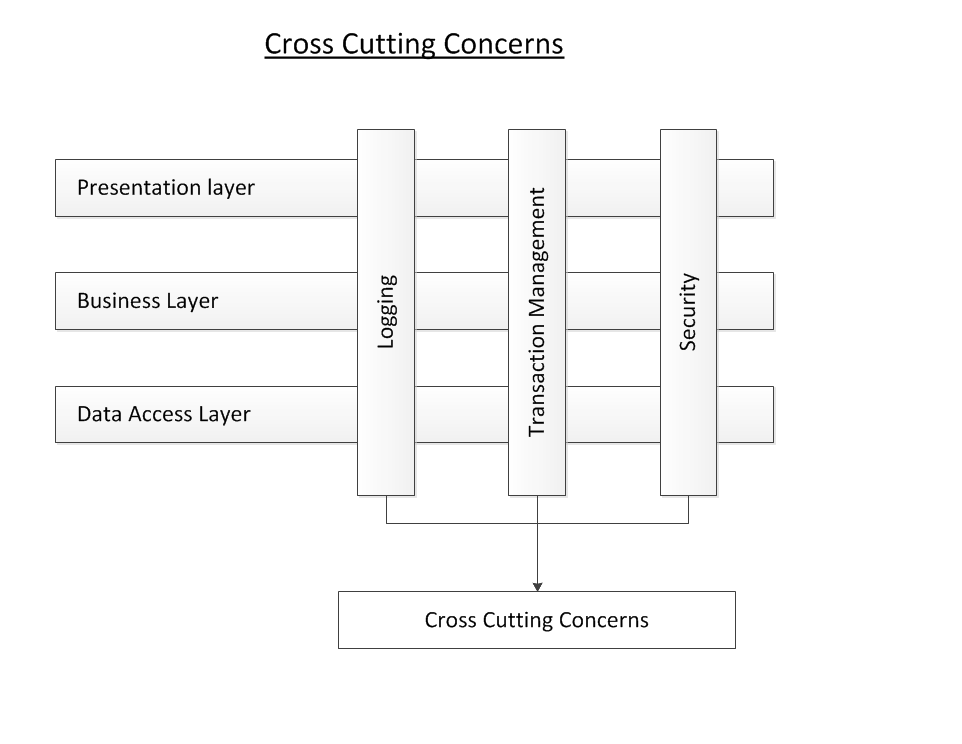
**Cross cutting concerns**

In any enterprise application, there are a number of concerns which need to be taken care of in addition to the main business logic. These concerns are spread across the application and into multiple application layers. Such concerns are logging, transaction handling, performance monitoring, security etc. These concerns are known as *cross cutting concerns* of the application.



Cross Cutting Concerns

AOP help is implementing the cross cutting concerns of the application keeping them separate from the main business logic and thus resulting in *loosely coupled applications*.

**Need of AOP**

Before going in to details of AOP we must understand the need of AOP. Let us take an example of a java class LibraryService which has two methods issueBook and returnBook. The requirement is to log the request parameters and the response value. Here logging is our cross cutting concerns which we want to implement in our application. The code of LibraryService looks as below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | public class LibraryService {      public boolean issueBook(int memberID, int bookID) {          System.out.println("Executing method issueBook("+memberID+","+bookID+") of LibraryService");            boolean status = false;          // Business logic to issue a book from Library            System.out.println("Returning from method issueBook of LibraryService : "+status);          return status;      }      public boolean returnBook(int memberID, int bookID) {          System.out.println("Executing method returnBook("+memberID+","+bookID+") of LibraryService");            boolean status = false;          // Business logic to return the issues book            System.out.println("Returning from method returnBook of LibraryService : "+status);          return status;      }  } |

Now, the requirement comes to add a new book in the Library and thus we need another method addBook in the LibraryService. While writing the method addBook, we need to ensure that we log the request parameters and the response value. Thus the code of addBook method looks as below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public boolean addBook(int bookID) {      System.out.println("Executing method addBook("+bookID+") of LibraryService");        boolean status = false;      // Business logic to return the issues book        System.out.println("Returning from method addBook of LibraryService : "+status);      return status;  } |

Drawbacks of the above conventional approach:

1. Similar logic of logging the request parameters and the response value is spread across multiple methods and thus creating redundant code. This makes maintenance very difficult.
2. Any change in the requirement of application logging will result in changing the code of multiple methods of LibraryService .
3. Adding any new method in the LibraryService will result in rewriting the logging code again in the newly added method. Thus we are not able to reuse the existing logging logic.
4. The main responsibility of LibraryService is to provide various operations of Library rather than logging. Keeping the code of logging in LibraryService is not a good idea.

AOP helps to implement the logging concern (and all other cross cutting concerns) of the application overcoming all of the above drawbacks. AOP keeps the cross cutting concerns separate from the main business logic of the application and weaves them appropriately in the various application object.

Spring AOP is used extensively in Spring’s Transaction Management where the transaction handlers are injected or weaved around the method execution join points. AOP is also used in Spring’s Security module to secure the method call for authenticated and authorised users only. Note that AOP is hidden behind the Spring’s Security namespaces so that the users do not worry about weaving security handlers in application objects - all is done through namespaces