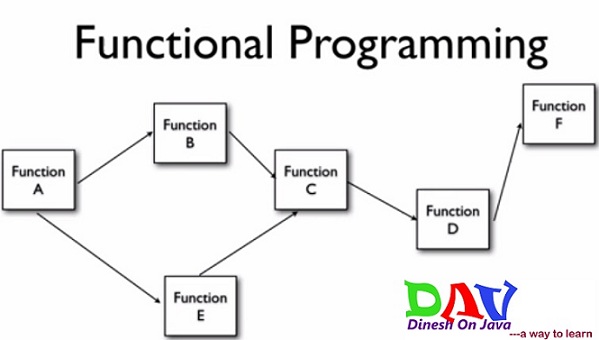
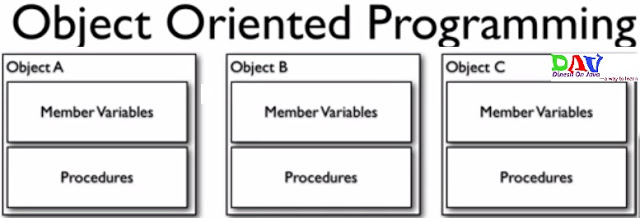
Introduction of AOP ( Aspect Oriented Programming)

I am talking about the **Aspect Oriented Programming** in Short "**AOP**", **Aspect Oriented Programming** is a feature Spring provide in the [**Dependency Injection**](http://www.dineshonjava.com/2012/06/dependency-injection-in-spring.html), Actually **Aspect Oriented Programming** is not only feature but also its different style of programming just like as **Object Oriented Programming.**  
  
Now I am talking about some other programming before Aspect Oriented Programming...  
**1. Functional Programming:** In older programming language like C, we have used functional programming style like below in figure.

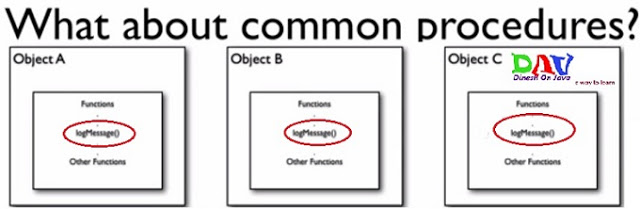


As above see In this style of programming writing code into couple of functions and each function perform unit task and each function call another function as see above and after last function execution then program is completed. But in this style of programming the main problem is complexity, it is very messy style of coding to write big project programming.

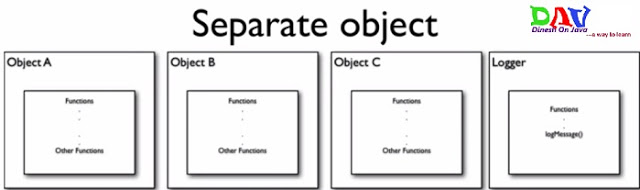
**2. Object Oriented Programming:** In this style of programming we would not think about function when we trying to solve problem by writing code, we would think as individual entities as object when writing the program as below Object A, Object B & Object C...



Here each object contain member variables and methods to perform individual tasks of each individual entity so this is fine but here is also a problem that not all rectify common procedure in all of the objects as **common logging procedure** in all as ***logMessage()*** method in all.



In above see that ***logMessage()*** method in each objects no matter how many objects are there so this is not good design of programming each object has repeating method. **So to solve this type problem we write the separate entity for logger and called in each objects where we want to add read log message as below.**



For using this logger entity in each object we have to make [dependency injection](http://www.dineshonjava.com/2012/06/dependency-injection-in-spring.html) with each beans of business classes or we have use inheritance for accessing logger method of Logger class that is good but couple of problems are also there.

**First Problem is doing the design this type of style there too many dependencies with non business object because  logger object does not have any business logic in the project its using just for logging with each objects in the project.**

**PROBLEMS:**

* **To many relationships with the crosscutting objects**.
* **Code is still required in the all methods**
* **Cannot all be changed at once**

**CROSS CUTTING CONCERNS: Means non business idea or non business logic its not part of our main problem it is related to below...**

* **Security**
* **Logging**
* **Transaction**

**To solve the above problems in the Object Oriented Programming we can using Aspect Oriented Programming.**

**3. Aspects Oriented Programming:** In this style of code we are make Aspects means Aspects are also specific classes which some special methods for particular tasks like logging, security and transactions etc.  
  
***Aspect-Oriented Programming* (AOP)** complements **Object-Oriented Programming (OOP)** by providing another way of thinking about program structure. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the *aspect*. Aspects enable the modularization of concerns such as transaction management that cut across multiple types and objects. (Such concerns are often termed *crosscutting* concerns in AOP literature.)  
  
***Aspect-oriented programming*** entails breaking down program logic into distinct parts (so-called concerns, cohesive areas of functionality). All programming paradigms support some level of grouping and encapsulation of concerns into separate, independent entities by providing abstractions (e.g., procedures, modules, classes, methods) that can be used for implementing, abstracting, and composing these concerns. But some concerns defy these forms of implementation, and are called crosscutting concerns because they "cut across" multiple abstractions in a program.

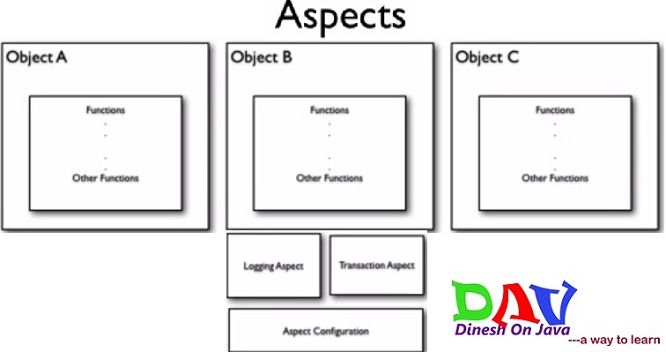
First we make different Aspects...

**a. Logging Aspect**

**b. Transaction Aspect**

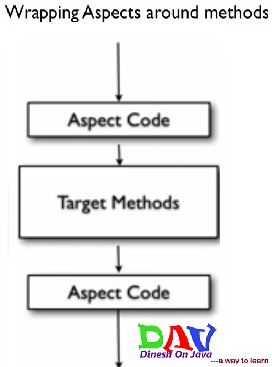
**c. Security Aspect**

 etc... and configure with each of objects as per as our requirement with **Aspect Configuration file** as see below in figure...



**Aspect Configuration file:** Its responsible for configuration for all Aspects with the all object where we want to use. Its configure suppose **Logging Aspect** for a method in **Object A** **Before or After** execution of that method,  
 its just like the.....  
      -**Servlet Filter in Servlet Configuration**   
**-Trigger in Database**  
**-Interceptors in Struts or in Spring MVC.**  
  
**Aspect Configuration tells which aspect apply which method of which class. Aspect Configuration solve our three problems of Object Oriented Programming**

* **To many relationships with the crosscutting objects**- Only single configuration required for every object where we want to use the crosscutting object like Logging Aspect with using Aspect Configuration file
* **Code is still required in the all methods-** No need to code required in all method just put that method on the Aspect Configuration file then code automatically associated with that method and execute Before or After execution of Target Method.
* **Cannot all be changed at once-** We can all be changed at once by using Aspect Configuration file.



There are only two Steps for using Aspects:

1. **Write Aspects**
2. **Configure Aspects where the aspects apply**