Abstrak

Pemrograman berorientasi objek sangat popular dalam dunia rekayasa perangkat lunak, karena dapat mengurangi kompleksitas dalam pemrograman. Masalah yang muncul adalah jika dalam suatu objek atau kelas terdapat suatu aspek yang unik, dan tidak dapat dienkapsulasi menjadi satu objek, atau yang biasa dikenal dengan *crosscutting concern*. Hal ini dapat menyebabkan modularitas menjadi tidak bersih. Permasalahan lain yang terjadi adalah tentang desain pemrograman berorientasi objek yang harus menangani permasalahan yang muncul secara berulang-ulang, sehingga perlu dicari solusi umum untuk mengatasi permasalahan tersebut.

Dalam kasus ini, penulis menerapkan implementasi dari *design pattern* pada *aspect-oriented programming* (AOP), yang mana AOP sendiri dibangun berdasarkan *object-oriented programming* (OOP). Namun satu hal yang berbeda bahwa AOP menjanjikan suatu kemampuan dalam hal melokalisir kode (biasa disebut sebagai alat *modularizing crosscutting*), yang tidak terdapat dalam OOP. Sedangkan *design pattern* menawarkan sebuah solusi yang flexibel dalam masalah pengembangan perangkat lunak, yaitu mendukung adanya penggunaan kembali pendekatan dan teknik yang sudah terbukti. Sehingga bila kedua hal ini digabungkan, maka dapat mengurangi kedua permasalahan yang terjadi.

Untuk pengembangan *design pattern* pada AOP ini diimplementasikan dalam AspectJ. AspectJ merupakan perluasan dari *aspect-oriented extension* pada Java, yang berarti bahwa pemrograman pada AspectJ itu sama saja dengan pemrograman dalam Java dalam aspek yang lebih.

Kata kunci : ***aspectj, aspect-oriented extension, aspect-oriented programming, crosscutting concern, design pattern, modularizing crosscutting, object-oriented programming*****.**