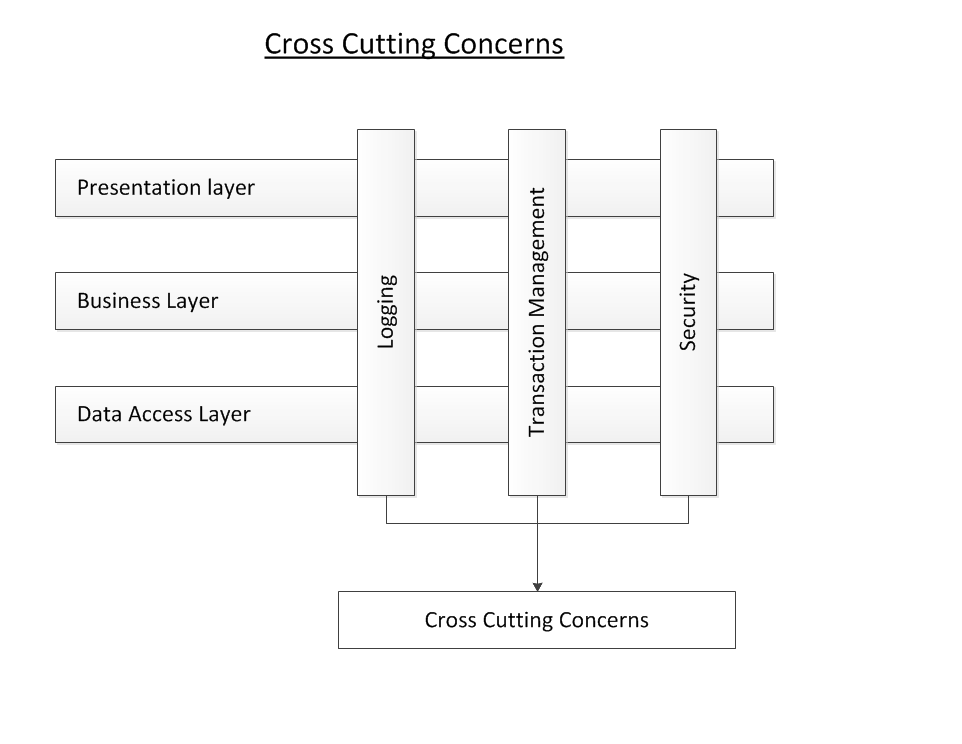
Cross Cutting Concerns Konseptleri



Herhangi bir uygulamada, birçok yerde gerekli olan bazı genel işlevler vardır. Ancak bu işlevsellik uygulamanın iş mantığıyla ilgili değildir. Uygulamanızdaki her iş yönteminden önce rol tabanlı bir güvenlik denetimi gerçekleştirdiğinizi varsayalım. Burada güvenlik çapraz kesme endişesidir. Herhangi bir uygulama için gereklidir, ancak iş açısından gerekli değildir, uygulamada birçok yerde uygulamak zorunda olduğumuz basit bir genel işlevselliktir. Aşağıda, kurumsal uygulama için çapraz kesme endişeleri(cross cutting concerns) ne örnekler verilmiştir.

\*\* Logging and tracing

\*\* Transaction management

\*\* Security

\*\* Caching

\*\* Error handling

\*\* Performance monitoring

\*\* Custom business rules

Kurumsal uygulamaların çoğu, farklı Nesne ve modül türleri için geçerli olan bazı yaygın çapraz kesme endişelerine sahiptir. Yaygın çapraz kesme endişelerinden bazıları günlüğe kaydetme, işlem yönetimi, veri doğrulama vb.

Nesne Yönelimli Programlamada, uygulamanın modülerliği Sınıflar tarafından elde edilirken, Yön Odaklı Programlamada uygulama modülerliği Aspects tarafından elde edilir ve farklı sınıflar arasında kesilecek şekilde yapılandırılır. Spring AOP, normal nesne yönelimli programlama modeliyle elde edemeyeceğimiz sınıflardan çapraz kesme görevlerinin doğrudan bağımlılığını alır. Örneğin, günlüğe kaydetme için ayrı bir sınıfa sahip olabiliriz, ancak yine işlevsel sınıfların uygulama genelinde günlüğe kaydetmeyi başarmak için bu yöntemleri çağırması gerekir.

AOP NEDIR?

AOP, çapraz kesme endişelerinin ayrılmasına izin vererek modülerliği artırmayı amaçlayan bir programlama paradigmasıdır. Spring AOP uygulamasının uygulanmasına dalmadan önce, AOP'nin temel kavramlarını anlamalıyız.

Aspect => Bir yön, birden çok sınıfı kesen bir endişenin modülerleştirilmesidir.

Joinpoint => Bir Joinpoint, bir programın yürütülmesi sırasında bir yöntemin yürütülmesi veya bir özel durumun işlenmesi gibi bir noktadır.

Pointcut => Bir Pointcut, belirli bir JoinPoint'te Bir Aspect tarafından uygulanacak bir Adviceyi eşleştirmeye yardımcı olan bir yüklemdir. Genellikle Adviceyi bir Pointcut ifadesiyle ilişkilendiririz ve Pointcut ile eşleşen herhangi bir Joinpoint'te çalışır.

Advice => Bir Advice, belirli bir Joinpoint'te bir yönüyle yapılan bir eylemdir. Advice türleri arasında "around", "before" ve "after" vardir. Springte, Bir Advice, Joinpoint çevresinde bir önleyici zinciri koruyarak bir önleyici olarak modellenmiştir.

Proxy => Proxy, hedef(target) nesneye tavsiye uygulandıktan sonra çerçeve tarafından oluşturulan nesnedir.

Target => Tearget, tavsiyenin uygulanacağı uygulama nesnesidir.