CODING SHUTTLE

AOP

Real world use cases

CODING SHUTTLE

Security Aspect

```
@Aspect
@Component
public class SecurityAspect {

    @Before("execution(* com.codingshuttle.aopApp.services.*.*(..)) && @annotation(RequiresAdmin)")
    public void checkAdminAccess() {
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        if (!auth.getAuthorities().contains("ROLE_ADMIN")) {
            throw new SecurityException("Only admins can access this resource.");
        }
    }
}
```

Caching Aspect

```java
@Aspect
@Component
public class CachingAspect {

    private Map<String, Object> cache = new HashMap<>();

    @Around("execution(* com.codingshuttle.aopApp.services.*.*(..))")
    public Object cacheMethodResults(ProceedingJoinPoint joinPoint) throws Throwable {
        String methodName = joinPoint.getSignature().getName();

        if (cache.containsKey(methodName)) {
            return cache.get(methodName);
        }

        Object result = joinPoint.proceed();  // Proceed with the original method call
        cache.put(methodName, result);  // Cache the result

        return result;
    }
}
```

CODING SHUTTLE

Auditing Aspect

```java
@Aspect
@Component
public class AuditAspect {

    @AfterReturning("execution(* com.codingshuttle.aopApp.services.*.get*(..))")
    public void auditAccess() {
        String user = "current_user";  // You'd get this from SecurityContext or similar
        LocalDateTime time = LocalDateTime.now();
        System.out.println("Data accessed by " + user + " at " + time);
    }
}
```

Exception Handling Aspect

```
@Aspect
@Component
@Slf4j
public class ExceptionHandlingAspect {

    @AfterThrowing(pointcut = "execution(* com.codingshuttle.aopApp.services.*.*(..))", throwing = "ex")
    public void handleException(Exception ex) {
        log.error("Exception caught in method: {}", ex.getMessage());
    }
}
```

Do we really need AOP for these?

- Transactional: For database transactions, @Transactional is sufficient, well-optimized, and easier to configure.
- Security: For method-level security, @Secured, @PreAuthorize, and @RolesAllowed are often more straightforward than creating custom security aspects.
- Validation: @Valid annotations provide built-in input validation at the controller and service level.
- Caching: @Cacheable, is powerful and integrate directly with Spring's caching abstraction, so custom caching with AOP is usually unnecessary unless you have a unique requirement.

Where to use AOP?

AOP is often better suited for advanced logging, monitoring, or profiling that applies across multiple layers of the application.

You may also use it for other tasks like Caching, Auditing, Exception Handling, etc. if the inbuilt methods are not sufficient for your use-case.