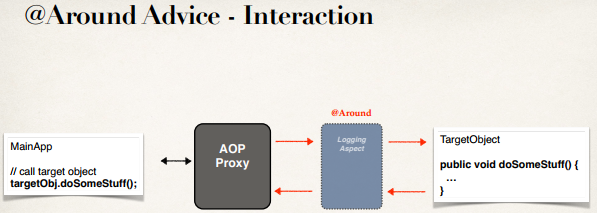
**44.1. AOP @Around Advice Overview**

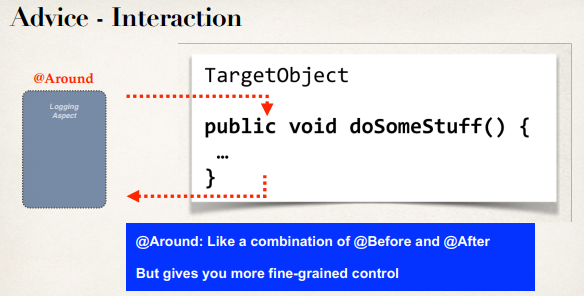
**Advice Types**:

1. **Before advice**: run before the method
2. **After returning advice**: run after the method (success execution)
3. **After throwing advice**: run after method (if exception thrown)
4. **After finally advice**: run after the method (finally)
5. **Around advice**: run before and after method

**@Around Advice – Interaction**:



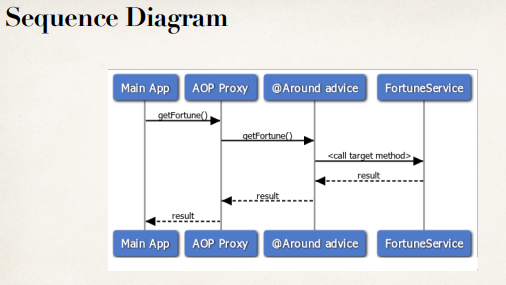
**Advice – Interaction**:



**@Around Advice - Use Cases**:

* Most common: logging, auditing, security
* Pre-processing and post-processing data
* Instrumentation / profiling code
  + How long does it take for a section of code to run?
* Managing exceptions
  + Swallow / handle / stop exceptions

**Sequence Diagram**:

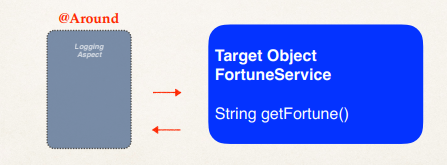


**ProceedingJoinPoint**:

* When using @Around advice
* We will get a reference to a “**ProceedingJoinPoint**”
* This is a handle to the target method
* Our code can use the **ProceedingJoinPoint** to execute target method

**Example**:

* Create an advice for instrumentation / profiling code
* How long does it take for a section of code to run?



**@Around Advice**:

// add a new advice for @Around

@Around("execution(\* com.ruhul.aopdemo.service.\*.getFortune(..))")

**public** Object aroundGetFortune(

ProceedingJoinPoint theProceedingJoinPoint) **throws** Throwable {

// print out which method we are advising on

String method = theProceedingJoinPoint.getSignature().toShortString();

System.***out***.println("\n======> Executing @Around on method: " + method);

// get being timestamp

**long** being = System.*currentTimeMillis*();

// now, lets execute the method

Object result = theProceedingJoinPoint.proceed();

// get end timestamp

**long** end = System.*currentTimeMillis*();

// compute duration and display it

**long** duration = end - being;

System.***out***.println("\n======> Duration: " + duration / 1000.0 + " seconds");

**return** result;

}

4.1. AOP @Around Advice Overview