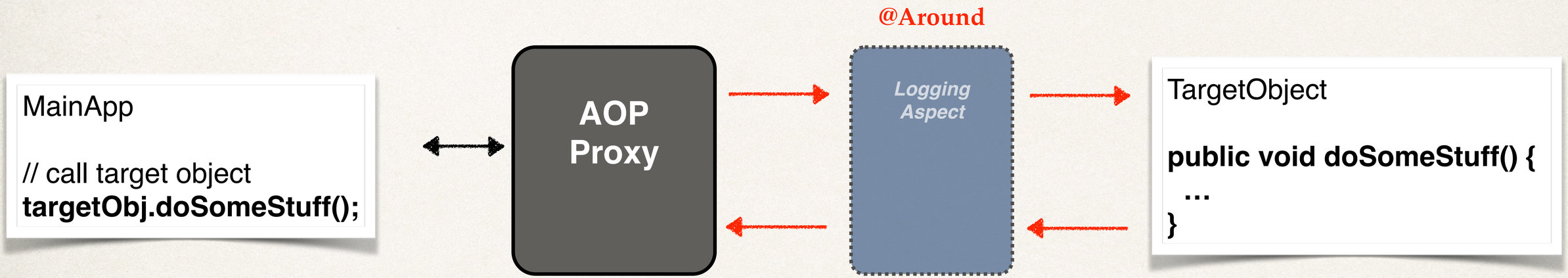
Aspect-Oriented Programming (AOP)
@Around Advice

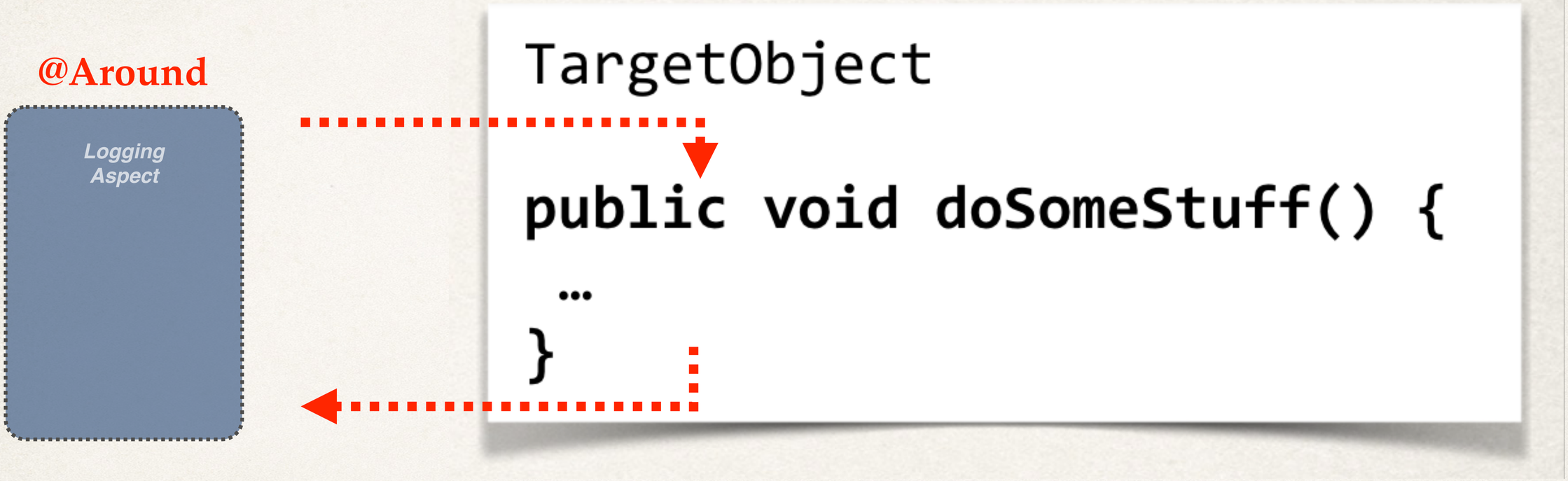
Advice Types

- **Before advice**: run before the method
- **After returning advice**: run after the method (success execution)
- **After throwing advice**: run after method (if exception thrown)
- **After finally advice**: run after the method (finally)
- **Around advice**: run before and after method

@Around Advice - Interaction



Advice - Interaction



@Around: Like a combination of @Before and @After

But gives you more fine-grained control

@Around Advice - Use Cases

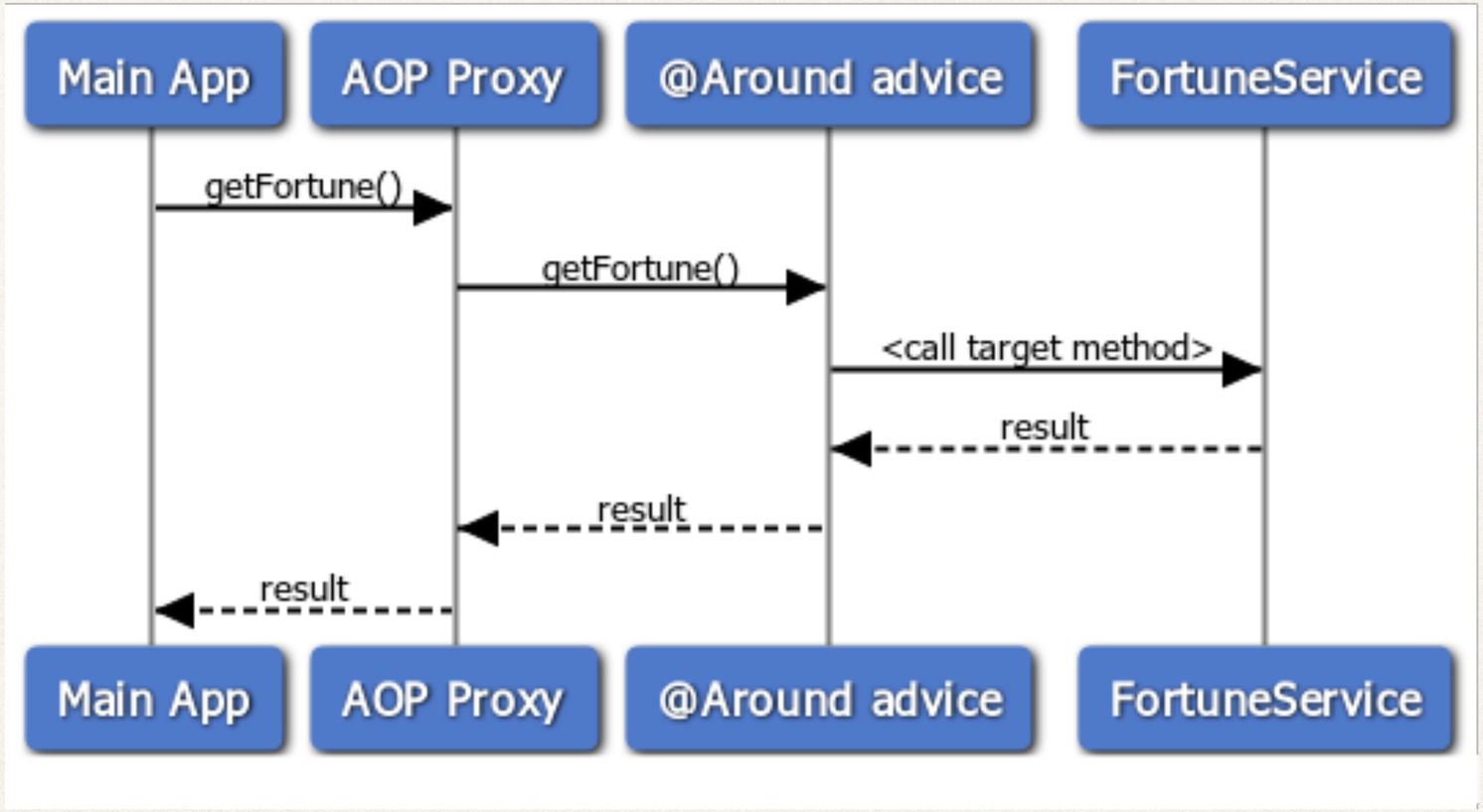
- Most common: logging, auditing, security
- Pre-processing and post-processing data
- Instrumentation / profiling code
  - How long does it take for a section of code to run?
- Managing exceptions
  - Swallow / handle / stop exceptions

Our FortuneService Example - Revisited

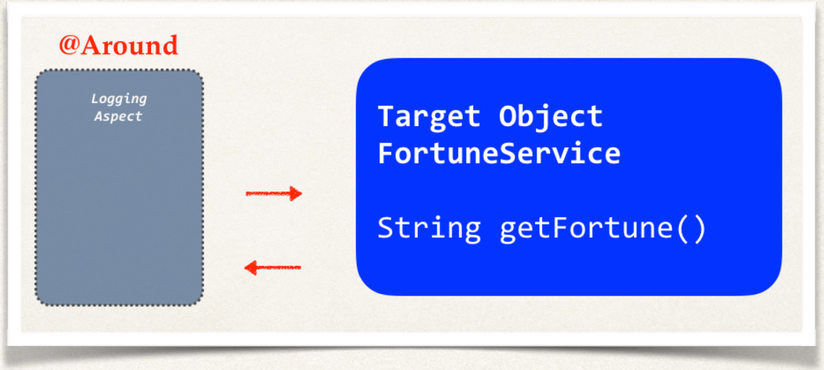
Target Object
FortuneService

String getFortune()

Sequence Diagram



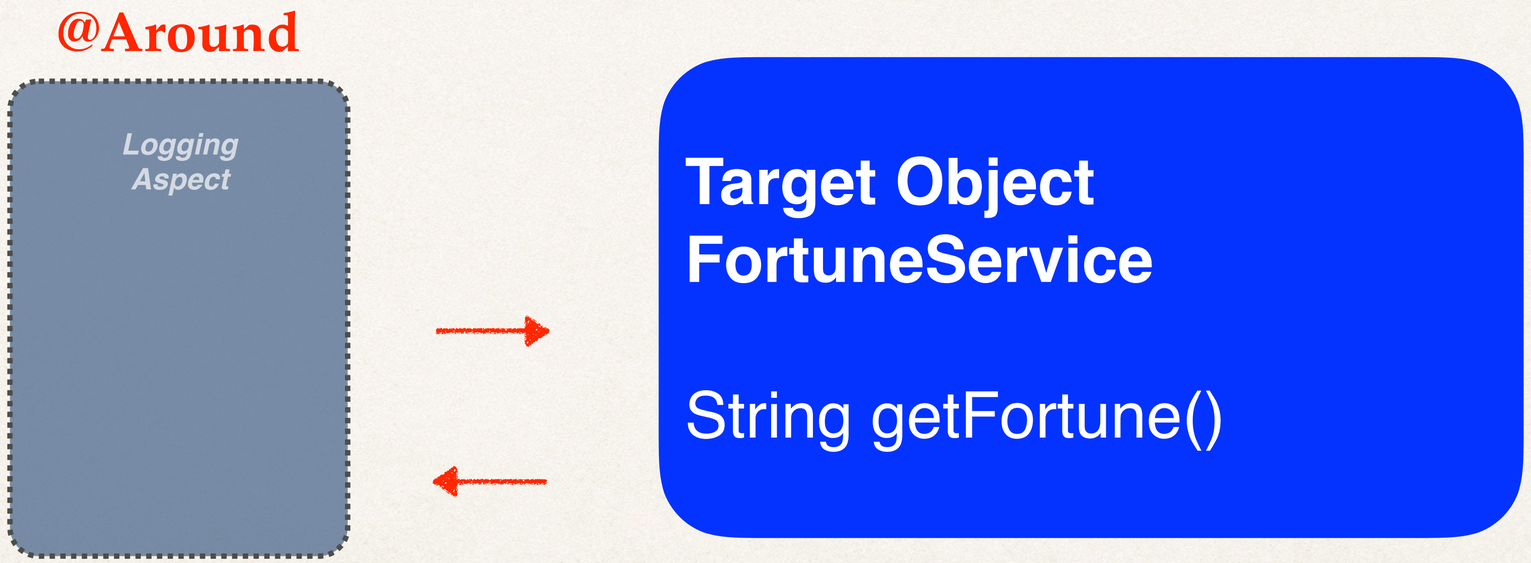
ProceedingJoinPoint



- When using @Around advice
- You will get a reference to a **"proceeding join point"**
- This is a handle to the **target method**
- Your code can use the **proceeding join point** 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

```
@Around("execution(* com.luv2code.aopdemo.service.*.getFortune(..))")
public Object afterGetFortune() throws Throwable {









}
```

@Around Advice

```
@Around("execution(* com.luv2code.aopdemo.service.*.getFortune(..))")
public Object afterGetFortune(
        ProceedingJoinPoint theProceedingJoinPoint) throws Throwable  {









}
```

@Around Advice

```
@Around("execution(* com.luv2code.aopdemo.service.*.getFortune(..))")
public Object afterGetFortune(
        ProceedingJoinPoint theProceedingJoinPoint) throws Throwable  {

    // get begin timestamp
    long begin = System.currentTimeMillis();









}
```

@Around Advice

```
@Around("execution(* com.luv2code.aopdemo.service.*.getFortune(..))")
public Object afterGetFortune(
        ProceedingJoinPoint theProceedingJoinPoint) throws Throwable  {

    // get begin timestamp
    long begin = System.currentTimeMillis();

    // now, let's execute the method
    Object result = theProceedingJoinPoint.proceed();









}
```

@Around Advice

```
@Around("execution(* com.luv2code.aopdemo.service.*.getFortune(..))")
public Object afterGetFortune(
        ProceedingJoinPoint theProceedingJoinPoint) throws Throwable  {

    // get begin timestamp
    long begin = System.currentTimeMillis();

    // now, let's execute the method
    Object result = theProceedingJoinPoint.proceed();

    // get end timestamp
    long end = System.currentTimeMillis();









}
```

@Around Advice

```
@Around("execution(* com.luv2code.aopdemo.service.*.getFortune(..))")
public Object afterGetFortune(
        ProceedingJoinPoint theProceedingJoinPoint) throws Throwable  {

    // get begin timestamp
    long begin = System.currentTimeMillis();

    // now, let's execute the method
    Object result = theProceedingJoinPoint.proceed();

    // get end timestamp
    long end = System.currentTimeMillis();

    // compute duration and display it
    long duration = end - begin;
    System.out.println("\n=====> Duration: " + duration + " milliseconds");


}
```

@Around Advice

```
@Around("execution(* com.luv2code.aopdemo.service.*.getFortune(..))")
public Object afterGetFortune(
        ProceedingJoinPoint theProceedingJoinPoint) throws Throwable  {

    // get begin timestamp
    long begin = System.currentTimeMillis();

    // now, let's execute the method
    Object result = theProceedingJoinPoint.proceed();

    // get end timestamp
    long end = System.currentTimeMillis();

    // compute duration and display it
    long duration = end - begin;
    System.out.println("\n=====> Duration: " + duration + " milliseconds");

    return result;
}
```