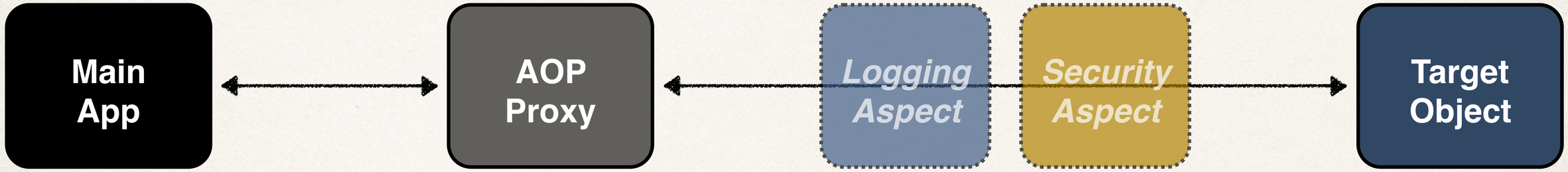
Aspect-Oriented Programming (AOP)
@Before Advice

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

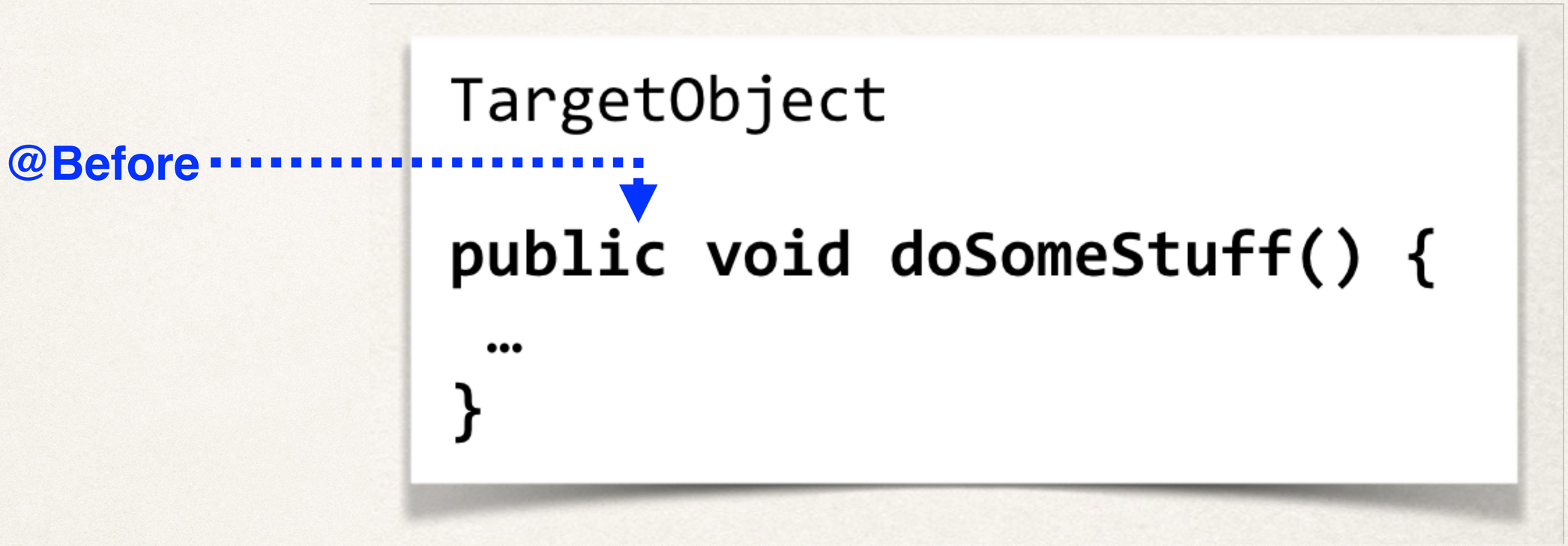
@Before Advice - Interaction



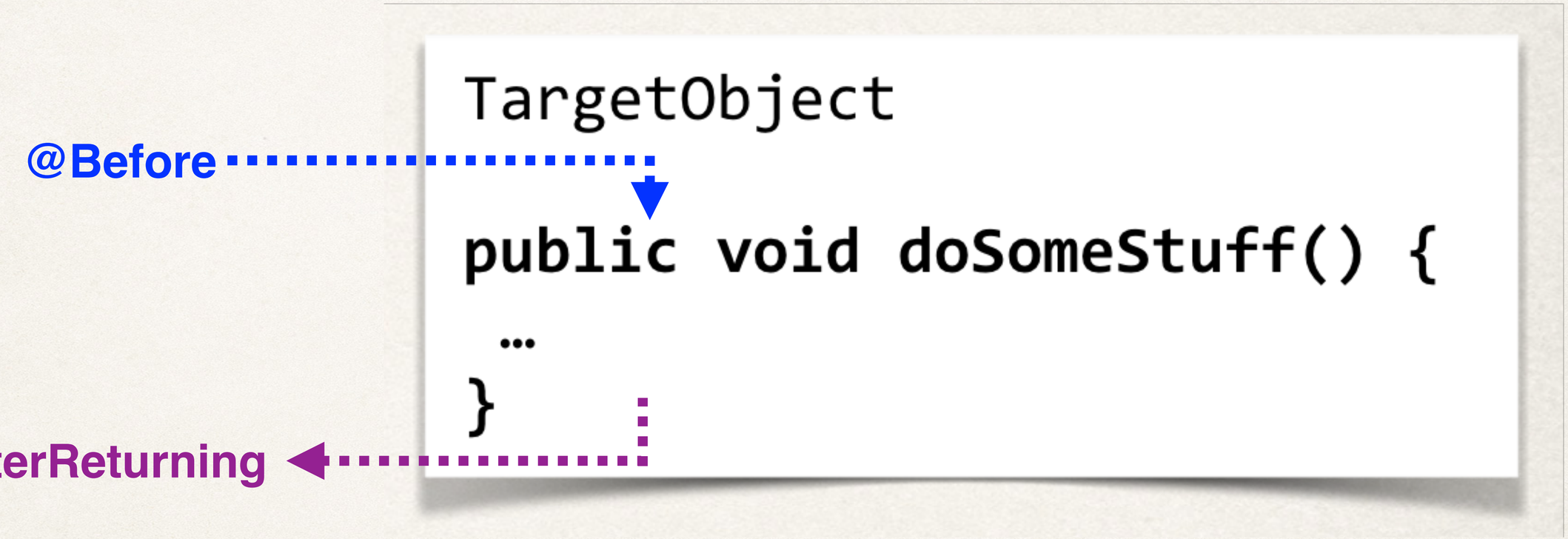

```
MainApp

// call target object
targetObj.doSomeStuff();
```

```
TargetObject

public void doSomeStuff() {
  ...
}
```

Advice - Interaction



Advice - Interaction

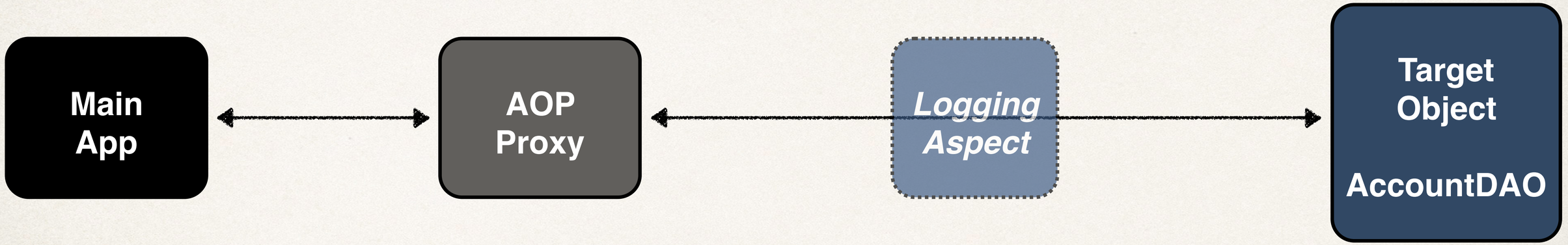


@Before Advice - Use Cases

- **Most common**
  - logging, security, transactions
- **Audit logging**
  - who, what, when, where
- **API Management**
  - how many times has a method been called user
  - analytics: what are peak times? what is average load? who is top user?

AOP Example - Overview

Step-By-Step



MainApp

```
// call target object
theAccountDAO.addAccount();
```

TargetObject - AccountDAO

```
public void addAccount() {
  ...
}
```

Adding AspectJ JAR File

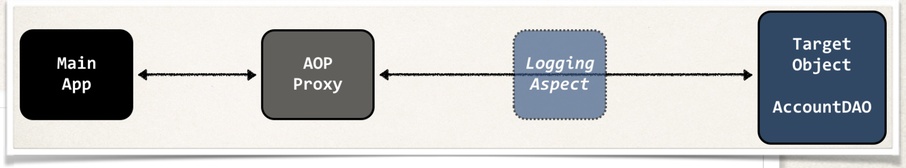
- Need to download AspectJ JAR file
- Even though we are using Spring AOP … still need AspectJ JAR file

- Why?
  - Spring AOP uses some of the AspectJ annotations
  - Spring AOP uses some of the AspectJ classes

Development Process - @Before

1. Create target object: AccountDAO
2. Create Spring Java Config class
3. Create main app
4. Create an Aspect with @Before advice

Step 1: Create Target Object: AccountDAO

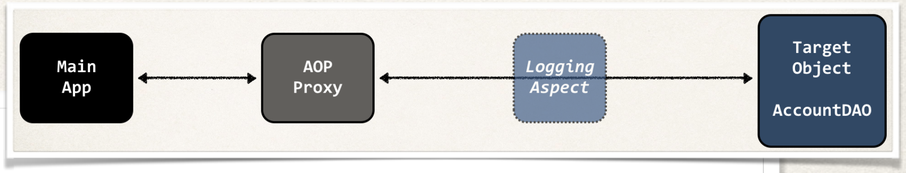



```
@Component
public class AccountDAO {

  public void addAccount() {

    System.out.println("DOING MY DB WORK: ADDING AN ACCOUNT");

  }
}
```

Step 2: Create Spring Java Config class

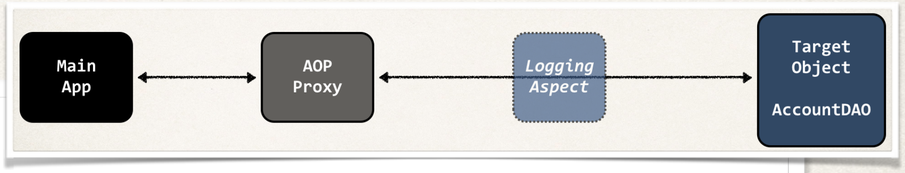
```
@Configuration
@EnableAspectJAutoProxy
@ComponentScan("com.luv2code.aopdemo")
public class DemoConfig {

}
```

Step 3: Create main app



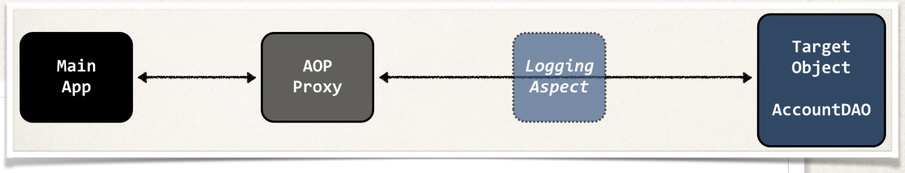

```
public class MainDemoApp {

    public static void main(String[] args) {

        // read spring config java class
        AnnotationConfigApplicationContext context =
                new AnnotationConfigApplicationContext(DemoConfig.class);

        // get the bean from spring container
        AccountDAO theDAO = context.getBean("accountDAO", AccountDAO.class);

        // call the business method
        theDAO.addAccount();

        // close the context
        context.close();
    }

}
```

Step 4: Create an Aspect with @Before advice



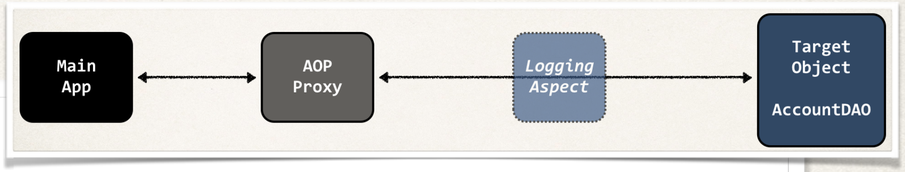

```
@Aspect
@Component
public class MyDemoLoggingAspect {

    ...

}
```

Step 4: Create an Aspect with @Before advice



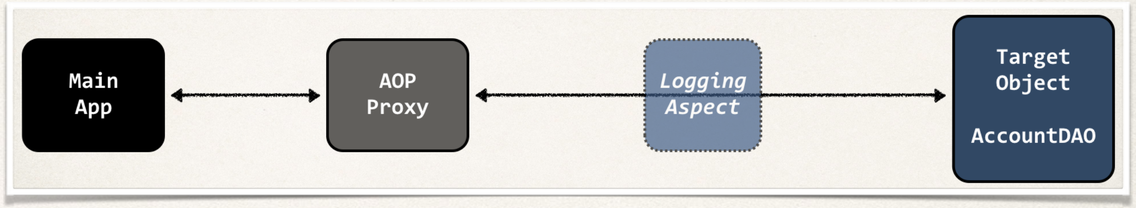

```
@Aspect
@Component
public class MyDemoLoggingAspect {

   @Before("execution(public void addAccount())")
   public void beforeAddAccountAdvice() {

      ...

   }

}
```

Step 4: Create an Aspect with @Before advice




```
@Aspect
@Component
public class MyDemoLoggingAspect {

   @Before("execution(public void addAccount())")
   public void beforeAddAccountAdvice() {

      System.out.println("Executing @Before advice on addAccount()");

   }

}
```

Best Practices: Aspect and Advices



- Keep the code small
- Keep the code fast
- Do not perform any expensive / slow operations
- Get in and out as QUICKLY as possible