Advanced Topics in Aspect-Oriented Languages:

AspectJ Programming Language

Spring AOP

-complements inversion of control component with AOP middleware
-to provide declarative enterprise services
    -such as declarative transaction management

-uses AspectJ pointcut language

-implemented using run-time proxies

-implemented in pure java
    -no special compilation is necessary

-problem accessing fine grained objects, especially domain objects
    -no field access or method call options,… (need to use AspectJ instead,..)

-requires Spring interface to be used
    -to process anything under the reach of Spring container not more

Architecture based on Aspects

- Architecture is described in files
- Architecture is improved – is in code
- Is expressed also with pointcuts
- Behaviour is added to specific parts of program thanks to aspect orientation

STRENGTHS:

- Architecture is **not in documentation**, but **in code**
- Solutions to problems preventing tangled and scattered code

Steps in defining the Architecture

- **1. Defining the architecture through pointcuts in place of join points**
  - For service -> @Service
  - For repository -> @Repository
  - For komponent -> @Component
- **2. Defining behaviour using advices**
- **3. Assigning advices to particular join points**

Result: Reduction of technical boiler plate code -> cleaner codebase

Define Rules For Services And Repositories

```
public class SystemArchitecture {
	//PRE REPOZITARE
	@Pointcut(
		//HOCIJAKA TRIEDA ANOTOVANA AKO @Repository
		"execution(* (@org.springframework.stereotype.Repository *).*(..))"
	)
	public void Repository() {}

	//PRE SLUZBY
	@Pointcut(
		//HOCIJAKA TRIEDA ANOTOVANA AKO @Service
		"execution(* (@org.springframework.stereotype.Service *).*(..))"
	)
	public void Service() {}

}
```

Advices

-modelled as interceptor

-in chain of such interceptor around particular join point

BEFORE

AFTER

AROUND

AFTER RETURNING

AFTER THROWING

Before Advice in Spring AOP

```
@Component      //SIMILARLY IT MUST BE SPRING BEAN - ASPECT IS SPRING BEAN!!!
@Aspect
public class TracingAspect {
    Logger logger = Logger.getLogger();
    @Before(
        "execution(void doSomething())"
    )
    public void entering(JointPoint joinPoint) {
        logger.trace("entering “
        // FINDING OUT WHICH CLASS AND METHOD WAS CALLED
                    + joinPoint.getStaticPart().getSignature().toString());
    }
}
```

After Advice in Spring AOP

```
@Component      //SIMILARLY IT MUST BE SPRING BEAN - ASPECT IS SPRING BEAN!!!
@Aspect
public class TracingAspect {
    Logger logger = Logger.getLogger();
    @After(
        "execution(void doSomething())"
    )
    public void entering(JointPoint joinPoint) {
        logger.trace("entering “
        // FINDING OUT WHICH CLASS AND METHOD WAS CALLED
                    + joinPoint.getStaticPart().getSignature().toString());
    }
}
```

Around Advice in Spring AOP

```
@Component
@Aspect
public class TracingAspect {
    Logger logger = Logger.getLogger();
    @Around(pointcut =
        "execution(* *(..))",
    )
    public Object traceMethod(ProceedingJoinPoint proceedingJoinPoint)  throws Throwable {
        String methodInformation = proceedingsJP.getStaticPart().getSignature().toString();

        logger.trace("Entering method: " + methodInformation);
        try {
            return proceedingJoinPoint.proceed();
        } catch (Throwable ex) {
            logger.error("Exception occurred in: " + methodInformation, ex);
            throw ex;
        } finally { logger.trace("Exiting method: " + methodInformation); }
    }
}
```

After Throwing Aspect

```
@Component
@Aspect
public class ExceptionLoggingAspect {
    Logger logger = LoggerFactory.getLogger(ExceptionLoggingAspect.class);

    @AfterThrowing(
        "SystemArhitecture.Repository()
            || SystemArchitecture.Service()", throwing = "ex“ )
    public void logException(Exception ex) {
        logger.error("Exception", ex);
    }
}
```

After Throwing Aspect

```
@Component
@Aspect
public class ExceptionLoggingAspect {
    Logger logger = LoggerFactory.getLogger(ExceptionLoggingAspect.class);

    @AfterThrowing(pointcut =
        "SystemArhitecture.Repository()
            || SystemArchitecture.Service()", throwing = "ex“ )
    public void logException(Exception ex) {
        logger.error("Exception", ex);
    }
}
```

After Returning Aspect

-executed only if function successfully returns value

-

```
@Component
@Aspect
public class OnReturnLoggingAspect {
    Logger logger = LoggerFactory.getLogger(ExceptionLoggingAspect.class);

    @AfterReturning(pointcut =
        "SystemArhitecture.Repository()
            || SystemArchitecture.Service()", returning = "result" )
    public void log (Object result) {
        logger.info("Returned „ + result.toString());
    }
}
```

This name has to be the same as parameter of annotated advice

Return value goes here

Pointcut Language

Annotations in Pointcuts

ANNOTATED METHOD

execution(@com.package.Annotation * *(..))

ANNOTATED CLASS

execution(* (@com.package.Annotation *).*(..))

Pointcut examples: Application to Methods

execution(void doSomething())

-refers to metod execution, declaration of method

-return type is void, method named doSomething, no parameters

execution(* *(..))

-uses wildcards

-* are **WILDCARDS** – the main strength of pointcuts

execution(* hello())

-method called hello

-without parameters

-any returning type execution(* hello(int, int))

Pointcut examples: Application to Methods

execution(* hello(*))

-method called hello

-without one parameter of any type

-any returning type

execution(* hello(int, int))

-method called hello

-without two parameter where both are integers

-any returning type

execution(* hello(..))

-method called hello

-any number of parameters of any types

-any returning type

Pointcut examples: Application to Packages and Services

execution(int com.package.other.Service.hello(int))

- defining information about package along with class and method
- class is called Service
- package is called: **com.package.other**
- returning type is integer (**int**)
- method has exactly one parameter with integer type (**int**)

execution(* com.package..*Service.*(..))

- with any returning type
- with any number and type of parameters
- any service name that ends with Service (***Service**)
- any class called Service from a package hierarchy starting with/rooted at **com.package**

  -or in other subpackage from **com.package** --------> TWO DOTS (**..**)

Pointcut examples: Application to Packages and Services

execution(* *.*(..))

- execution of any method (.*)
- any class
- in default package
- with any returning type
- with any number and type of parameters

execution(* *..*.*(..))

-execution of any method (the second .*)

-with any number and type of parameters (the second .., or simply (..))

-with any returning type (the first .)

-in any package (the second *)

-with any class (the third *)

-in any subpackage (the first ..)

Name of Beans as Pointcuts

BEAN THAT ENDS WITH Svc

bean(*Svc)

BEAN PRECONFIGURATION

1. **Java Configuration**
   -bean annotated methods @bean
2. **Annotation parameter**
   -@Component, @Service, @Repository
3. **XML configuration**
   -id / name attribute

Spring AOP Proxies

```
------------------------   calling PROXY    /  ---------------  /
- Other Spring bean -          ------>       /  Original object /
------------------------                     /  ---------------  /
```

1. Getting proxy
(not original object)

2. Code is directed
to original object

3. Advice is called

```
      ||
    \ || /
     \  /
      \/
 -----------
 - Advice -
 -----------
```

```
                    /    PROXY -created by Spring
                    /          - as original object (looks like)
                    /          - injected into other Spring Beans
                    /          - dynamic proxies if interfaces are used
                    /          otherwise (in remaining cases) CGLIB generated subclasses
                    ////////////////////////
```

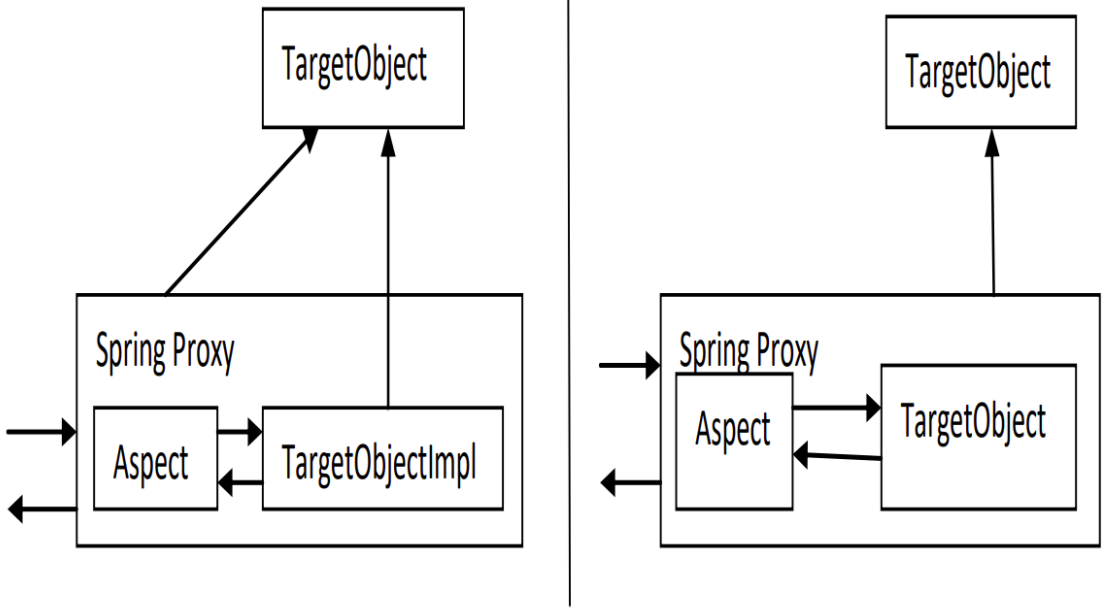


Fig 1 Spring AOP Process

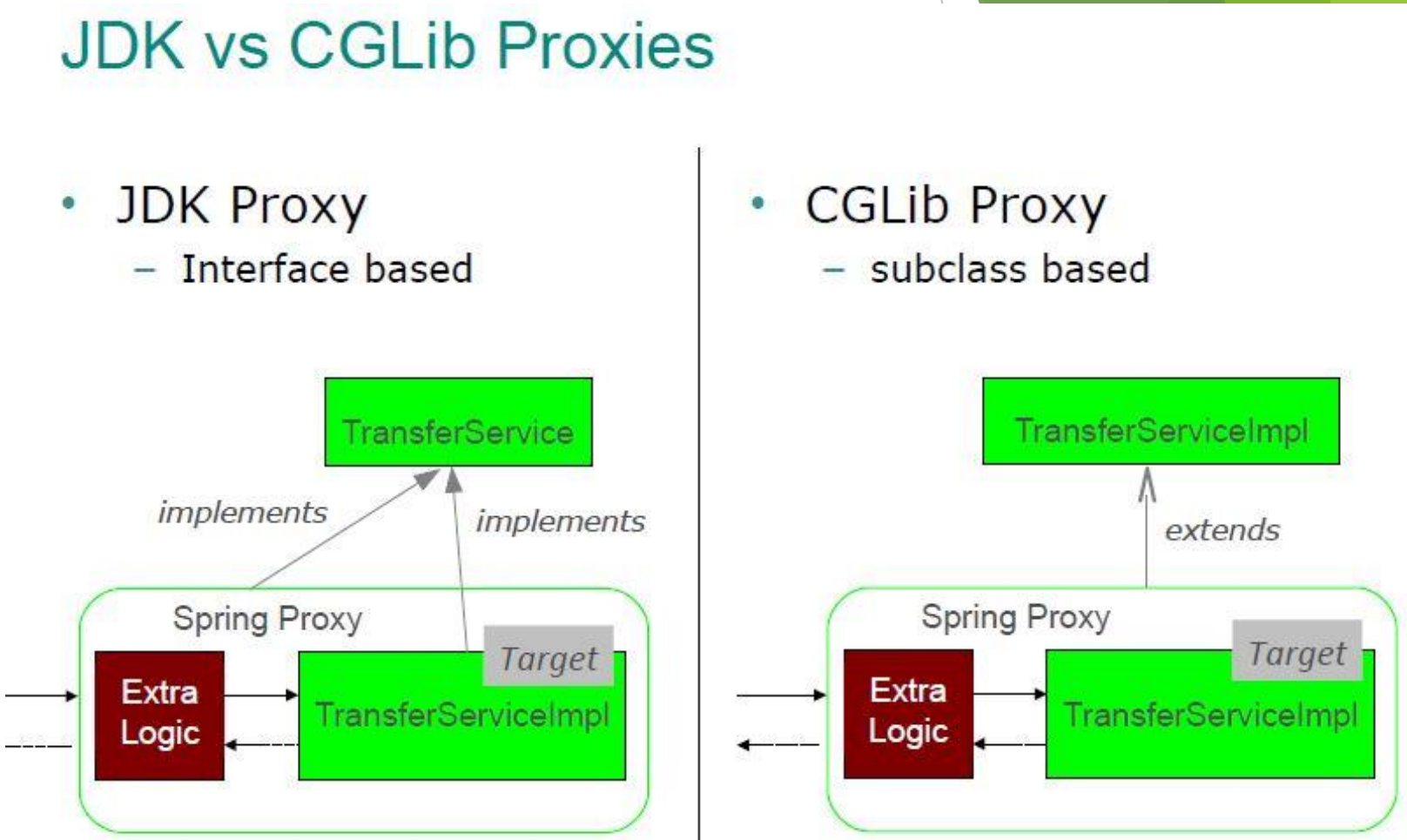
Proxy in Spring AOP: JDK vs CGLIB

Taken from: *https://i.sstatic.net/ZQkEF.jpg*

-Spring AOP is implemented using run-time proxies -> object is always proxy

Weaving at runtime

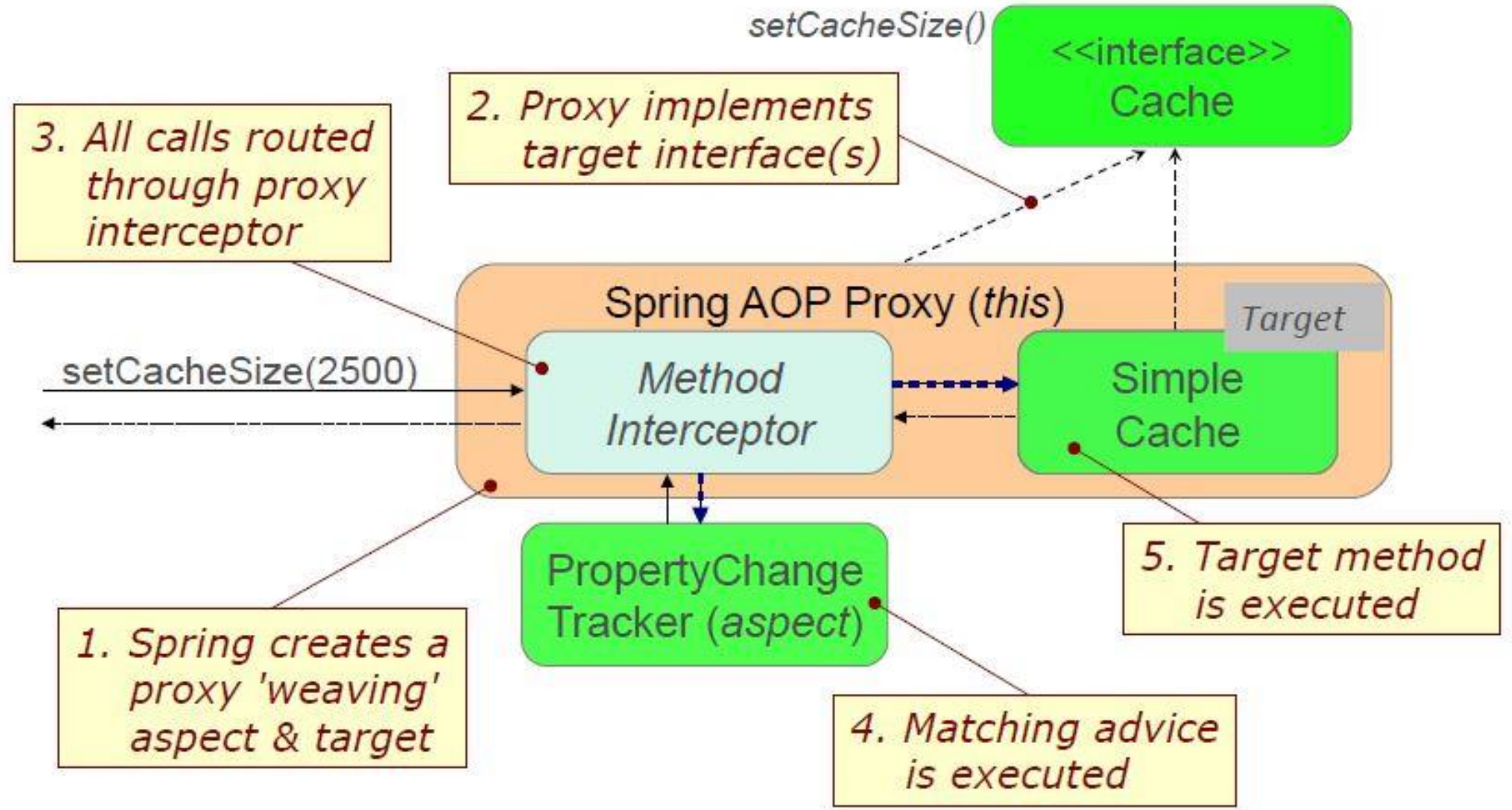
From: R. Laddad, "AspectJ in Action", Enterprise AOP with Spring, Manning Publications, 2010



Default and preferred way
- if interface is implemented by target object

interface is not implemented by target object

How Aspects are Applied



Taken from: *https://i.sstatic.net/bglkX.jpg*

Proxies in Spring AOP

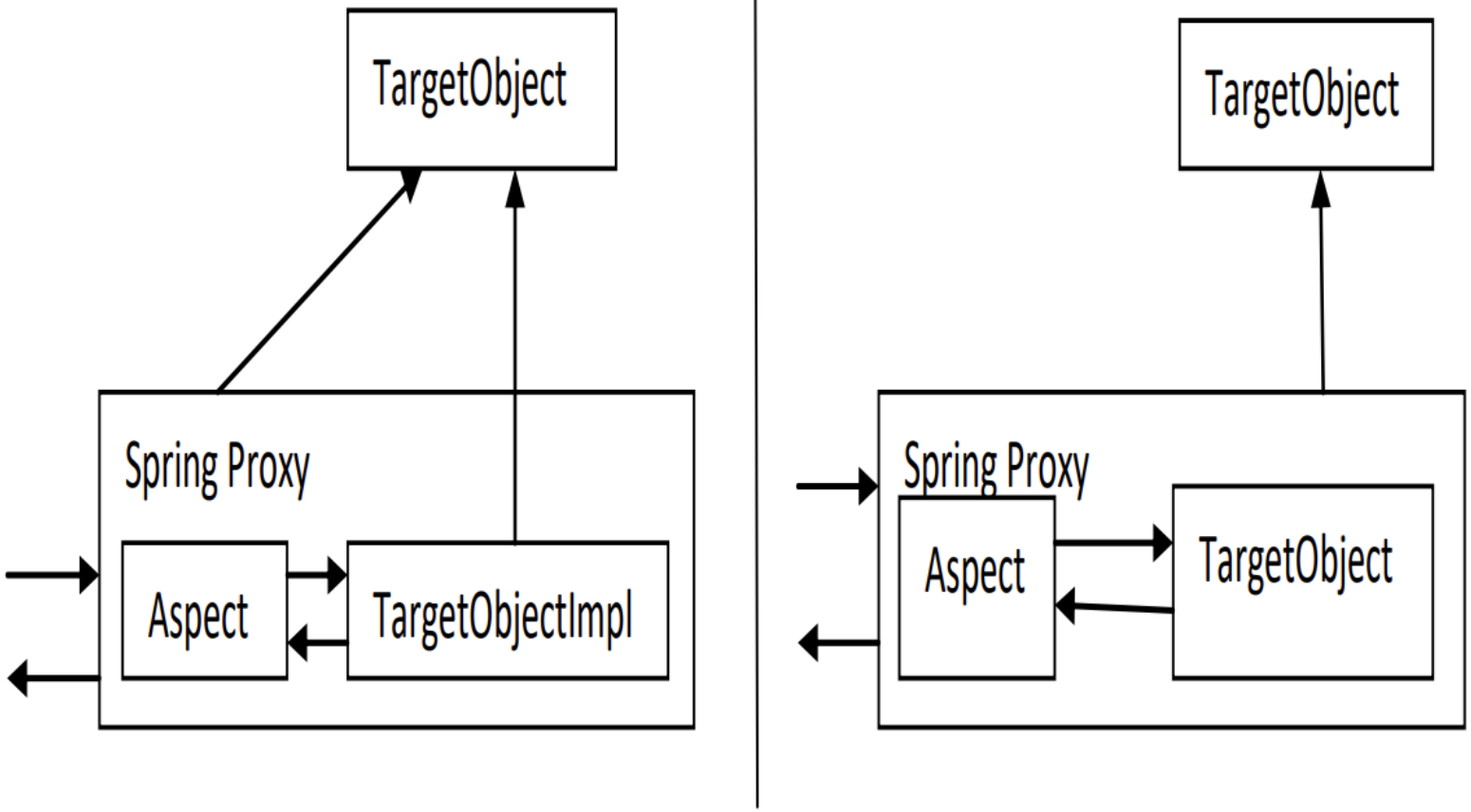


Fig 1 Spring AOP Process

Taken from: *KUMAR, Ravi a Munishwar RAI, 2019. A Comparative Study of AOP Approaches: AspectJ, Spring AOP, JBoss AOP. 2019*

> But what is the exact reason for which all these 3 interface are implemented?

If the proxy didn't implement all of those interfaces, the bean couldn't be wired into other beans that use that interface (you'd get a ClassCastException). For example, autowiring all of the beans of that interface into a bean. Additionally, things like getBeanNamesForType wouldn't work if the proxy didn't implement the interface.

> So it seems to me that exist 2 kinds of proxies (correct me if I am saying wrong assertions)

Yes that's correct. See ScopedProxyMode. By default, Spring won't create a proxy. It only creates a proxy if it needs to wrap the bean to add additional behavior (AOP). Note that there's also a special case of the CGLIB based proxy that uses Objenesis to deal with subclassing targets that don't have a default constructor.

> CGLIB Proxy: in wich, it seems to me that, the proxy extend the implementation of the wrapped object adding to it the extra logic features

When you use CGLIB based proxies, the constructor for your bean gets called twice: once when the dynamically generated subclass is instantiated (to create the proxy) and a second time when the actual bean is created (the target).

> I think that in AOP the extra logic is represented by the implementation of the method interceptor (is it true?)

The proxy is essentially just invoking the chain of advice needs to be applied. That advice isn't implemented in the proxy itself. For example, the advice for `@Transactional` lives in TransactionAspectSupport. Take a look at the source to JdkDynamicAopProxy.

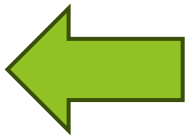
> and the standard logic is represented by the implementation of the method defined into the interfaces.

Assuming that you're programming against interfaces and using JDK proxies that's correct.

> But, if the previous reasoning are correct, my doubts is: why I need to define these interface and do that the object wrapped by the object implement these interfaces? (I can't understand if the proxy itself implement these interfaces).

If you want to use interface based proxies you need to use interfaces. Just make sure all of your beans implement interfaces, all of your advised methods are defined by those interfaces, and that when one bean depends on another bean, that dependency is specified using an interface. Spring will take care of constructing the proxy and making sure it implements all of the interfaces.

In your diagram, you have "Spring AOP Proxy (this)". You have to be really careful with using `this` when you're using any type of proxying.



Using this you refer to proxy

1. Calls within the same class won't have advice applied because those calls won't pass through the proxy.
2. If in one of your beans you pass `this` to some outside code, you're passing the target of the AOP advice. If some other code uses that reference, the calls won't have AOP advice applied (again, you're bypassing the proxy).

Pointcut Annotation

@Pointcut

```
public class PointcutsForCheckingOrders {
    @Pointcut("execution(@annotation.Check * *(..))")
    public void checkOrder() {
    }
}
```

USAGE

```
@Around("PointcutsForCheckingOrders.checkOrder()")
public void check(ProceedingJoinPoint proceedingJP) throws Throwable {
}
```

Variant restricted to particular package:

```
@Around("packageName.PointcutsForCheckingOrders.checkOrder()")
public void check(ProceedingJoinPoint proceedingJP) throws Throwable {
}
```

Allowing Aspects In Configuration of Spring Framework

Option 1

```
<beans>
    // KAZDA BEANA S ASPEKT ANOTACIAMI SA STANE ASPEKTOM
    <aop:aspectj-autoproxy />
    // SCAN PRE SPRING BEANY
    <context:component-scan base-package="simplepackagename" />
</beans>
```

Option 2

```
@Configuration
@EnableAspectJAutoProxy        //ENABLES @Aspect ANNOTATION
@ComponentScan(basePackages="simplePackageName")
public class SimpleAspectConfiguration {


}
```

AspectJ vs Spring AOP vs JBoss

Only these method join points are supported in Spring AOP

-> ***rather aiming integration with inversion of control component in Spring***

Taken from: *KUMAR, Ravi a Munishwar RAI, 2019. A Comparative Study of AOP Approaches: AspectJ, Spring AOP, JBoss AOP. 2019*

Table I
Summary of Compatible Join points

Join points	Spring AOP Compatibility	AspectJ Compatibility	JBoss AOP Compatibility
Calling Method	×	✓	✓
Execution of Method	✓	✓	✓
Calling Constructor	×	✓	✓
Execution of Constructor	×	✓	×
Execution of Static initialization	×	✓	✓
Initialization of Object	×	✓	✓
Field reference	×	✓	✓
Field assignment	×	✓	✓
Execution of Handler	×	✓	✓
Execution of Advice	×	✓	✓

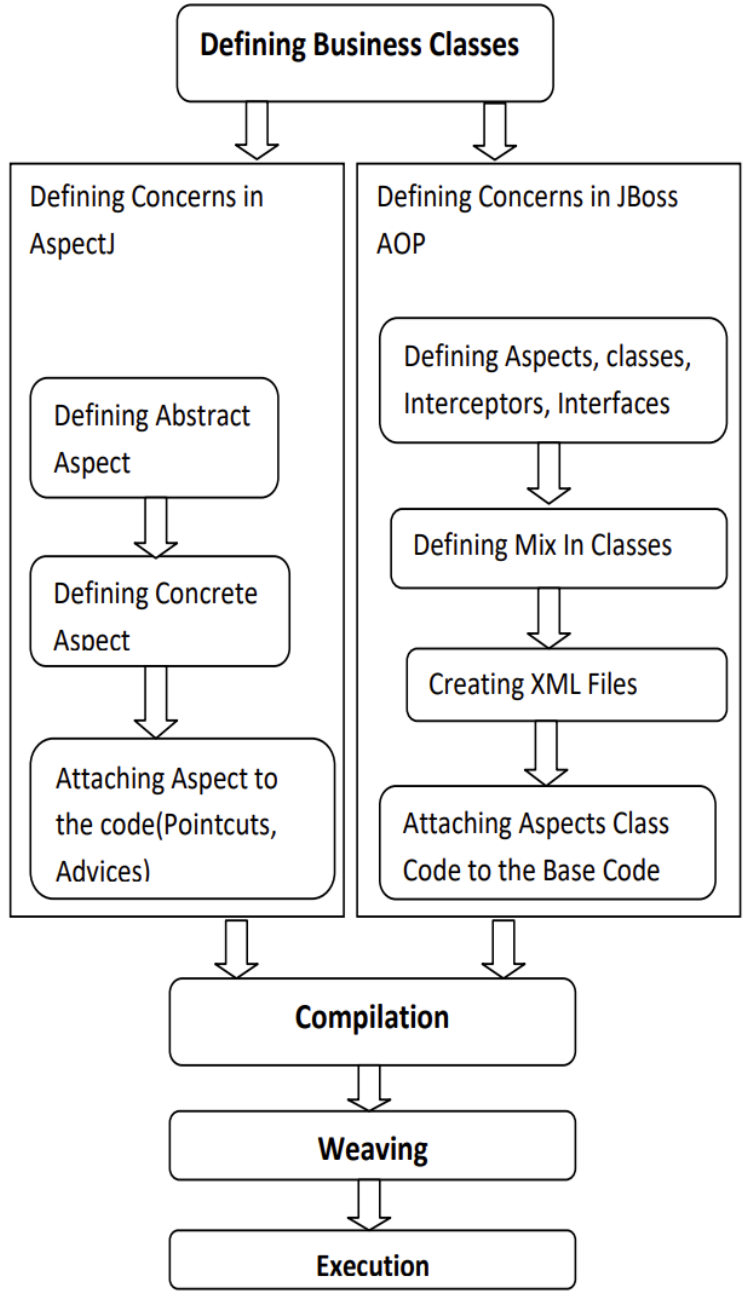


Fig 2 JBoss AOP and AspectJ programming process

Runtime Performance

The results were obtained with 2 million iterations. L atest run: Dec 20, 2004

In **ns** - *nanoseconds*

AWBench (ns/invocation)	aspectwerkz	awproxy	aspectwerkz_1_0	aspectj	jboss	spring	dynaop	cglib	ext:aopalliance	ext:spring	ext:aspectj
before, args() target()	**10**	**25**	**606**	**10**	**220**	**355**	**390**	**145**	**-**	**220**	**-**
around x 2, args() target()	**80**	**85**	**651**	**50**	**290**	**436**	**455**	**155**	**465**	**476**	**-**
before	15	20	520	15	145	275	320	70	-	40	10
before, static info access	30	30	501	25	175	275	330	70	-	35	-
before, rtti info access	50	55	535	50	175	275	335	75	-	35	-
after returning	10	20	541	10	135	285	315	85	-	45	15
after throwing	3540	3870	6103	3009	5032	-	6709	8127	-	-	3460
before + after	20	30	511	20	160	445	345	80	-	35	20
before, args() primitives	10	20	555	10	195	350	375	145	-	210	-
before, args() objects	5	25	546	10	185	325	345	115	-	200	-
around	60	95	470	10	-	225	315	75	-	-	90
around, rtti info access	70	70	520	50	140	250	340	80	70	70	-
around, static info access	80	90	486	25	135	245	330	75	80	80	-

Taken from: https://web.archive.org/web/20150520175004/https://docs.codehaus.org/display/AW/AOP+Benchmark

Runtime Performance

The results were obtained with 2 million iterations. L atest run: Dec 20, 2004

AWBench (relative %)	aspectwerkz	awproxy	aspectwerkz_1_0	aspectj	jboss	spring	dynaop	cglib	ext:aopalliance	ext:spring	ext:aspectj
before, args() target()	**1 x**	**2.5 x**	**60.6 x**	**1 x**	**22 x**	**35.5 x**	**39 x**	**14.5 x**	**-**	**22 x**	**-**
around x 2, args() target()	**1 x**	**1 x**	**8.1 x**	**0.6 x**	**3.6 x**	**5.4 x**	**5.6 x**	**1.9 x**	**5.8 x**	**5.9 x**	**-**
before	1 x	1.3 x	34.6 x	1 x	9.6 x	18.3 x	21.3 x	4.6 x	-	2.6 x	0.6 x
before, static info access	1 x	1 x	16.7 x	0.8 x	5.8 x	9.1 x	11 x	2.3 x	-	1.1 x	-
before, rtti info access	1 x	1.1 x	10.7 x	1 x	3.5 x	5.5 x	6.7 x	1.5 x	-	0.7 x	-
after returning	1 x	2 x	54.1 x	1 x	13.5 x	28.5 x	31.5 x	8.5 x	-	4.5 x	1.5 x
after throwing	1 x	1 x	1.7 x	0.8 x	1.4 x	-	1.8 x	2.2 x	-	-	0.9 x
before + after	1 x	1.5 x	25.5 x	1 x	8 x	22.2 x	17.2 x	4 x	-	1.7 x	1 x
before, args() primitives	1 x	2 x	55.5 x	1 x	19.5 x	35 x	37.5 x	14.5 x	-	21 x	-
before, args() objects	1 x	5 x	109.2 x	2 x	37 x	65 x	69 x	23 x	-	40 x	-
around	1 x	1.5 x	7.8 x	0.1 x	-	3.7 x	5.2 x	1.2 x	-	-	1.5 x
around, rtti info access	1 x	1 x	7.4 x	0.7 x	2 x	3.5 x	4.8 x	1.1 x	1 x	1 x	-
around, static info access	1 x	1.1 x	6 x	0.3 x	1.6 x	3 x	4.1 x	0.9 x	1 x	1 x	-

Taken from: https://web.archive.org/web/20150520175004/https://docs.codehaus.org/display/AW/AOP+Benchmark

AspectJ

-the most developed aspect-oriented language

-much faster compile-time weaving is used as default to weave aspects

-no additional run-time overhead

-own pointcut language

-all types ow weaving are supported

Pointcuts - different kinds

- Annotation based pointcuts
- Method/constructor call
- Method/constructor execution
- Execution object pointcuts
- Control flow based pointcuts
- Argument pointcuts
- Initialization
- Field Access
- Exception Handling
- Advice execution
- Conditional pointcut
- Pointcut based on Lexical structure

Example: Annotation based join points

```
1  var newVariable = "Hallo";
2  function a(param1, param2) {
3      @wholeClass({ "outsideGallery": "true" })
4      class GG {
5          callMe() { /* ... callMe function logic ...
                */ }
6      }
7  }
8  /* ... other content ... */
```

Automated pointcut extraction with semantic information

```
1  var markerVP1 = { "global": {}, "inner": { "p": 699 } }
2  @AnnotationVP1.variableVP() var newVariable = "Hallo";
3  var markerVP2 = { /*...*/ };
4  @AnnotationVP2.functionVP() function a(param1, param2) {
5      var markerVP3 = { /*...*/ };
6      @AnnotationVP3.classVP()
7      class GG {
8          @AnnotationVP10.classVariableVP()
9          markerVP6 = { /*...*/ };
10         @AnnotationVP4.classFunctionVP()
11         callMe() {
12             var markerVP4 = { /*...*/ };
13         }
14         @AnnotationVP11.classVariableVP()
15         markerVP7 = { /*...*/ };
16     }
17 }
18 var markerVP15 = { /*...*/ };
19 /* ... other content ... */
```

```
var markerVP1 = { "global": {}, "inner": { "p": 0 } };
@AnnotationVP1.variableVP() var newVariable: string = "Hallo";
var markerVP2 = { "global": { "globalVariables": [{ "name": "newVariabl
};
@AnnotationVP2.functionVP() function a(param1: number, param2: number)
    var markerVP3 = { "allAvailableCalls": ["a([%[param1: number]%], [%
                }] }, "inner": { "p": 737, "allAvailableCalls": ["a([%[
    @wholeClass({ "outsideGallery": "true" })
    class GG {
        @AnnotationVP18.classVariableVP()
        markerVP6 = { "allAvailableCalls": ["a([%[param1: number]%], [%
            ], "global": { "globalVariables": [{ "name": "newVariable",
        @wholeClassMethod({ "outsideGallery": "true" })
        callMe() {
            var markerVP4 = { "allAvailableCalls": ["a([%[param1: numbe
                        }] }, "inner": { "p": 351, "allAvailableCalls":
            console.log("Called");
            var markerVP5 = { "allAvailableCalls": ["a([%[param1: numbe
                        }] }, "inner": { "p": 351, "allAvailableCalls":
        }
        @AnnotationVP19.classVariableVP()
        markerVP7 = { "allAvailableCalls": ["a([%[param1: number]%], [%
            ], "global": { "globalVariables": [{ "name": "newVariable",
    }
    var markerVP8 = { "allAvailableCalls": ["a([%[param1: number]%], [%
                }] }, "inner": { "p": 737, "allAvailableCalls": ["a([%[
    new GG().callMe();
    var markerVP9 = { "allAvailableCalls": ["a([%[param1: number]%], [%
                }] }, "inner": { "p": 737, "allAvailableCalls": ["a([%[
    @AnnotationVP25.variableVP() let local = 5;
    var markerVP10 = { "allAvailableCalls": ["a([%[param1: number]%], [
    @AnnotationVP35.variableVP() var globalOne = 6;
    var markerVP11 = { "allAvailableCalls": ["a([%[param1: number]%], [
    @AnnotationVP45.variableVP() let local2 = 5;
    var markerVP12 = { "allAvailableCalls": ["a([%[param1: number]%], [
    @AnnotationVP55.variableVP() var globalOne2 = 6;
    var markerVP13 = { "allAvailableCalls": ["a([%[param1: number]%], [
    return self;
    var markerVP14 = { "allAvailableCalls": ["a([%[param1: number]%], [
}
var markerVP15 = { "allAvailableCalls": ["a([%[param1: number]%], [%[pa
console.log(a(null, null).local);
var markerVP16 = { "allAvailableCalls": ["a([%[param1: number]%], [%[pa
@wholeClass({ "outsideGallery": "true" })
class BB {
    @AnnotationVP144.classVariableVP()
    markerVP31 = { "allAvailableCalls": ["a([%[param1: number]%], [%[pa
    @variableDeclarationClass({ "outsideGallery": "true" })
```

```
var newVariable = "Hallo";

function a(param1, param2) {
    class GG {
        callMe() {
            console.log("Called");
        }
    }
    new GG().callMe();
    let local = 5;
    var globalOne = 6;
    let local2 = 5;
    var globalOne2 = 6;
    return self;
}
console.log(a(null, null).local);

class BB {
    nnnnn = 4;
    constructor(ccc: number, ddd: number) {

    }

    funnnc(eee): string {

    }

    inner(): void {
        let wwwww = 4;

        function cfunc(pr, pr2) {
            console.log(pr + pr2);
            console.log("wwwww");
            console.log(wwwww);

            function sssss() {
                console.log("Done");
            }
        }
        sssss();
        cfunc(wwwww, this.nnnnn);
    }

    rrrr = 4;

}

new BB().inner();
```

Pointcut

-selects sets of join points and collects context in their place

Example

To capture constructor call, when Player instance is going to be created

```
call(Player.new(..))
```

To check condition (for example for variability handling)

To capture constructor call, when Player instance is going to be created, but setNames from Configuration has to be true

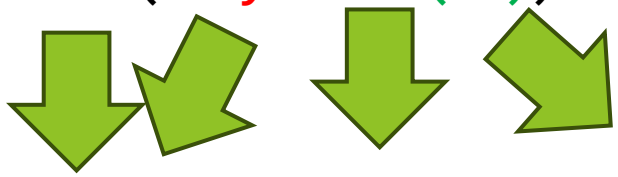
- `call(Player.new(..)) && if(Configuration.setNames);`

Complex examples with whole declaration (within aspect):

```
pointcut manageDifficultyDuringInstantiationOfPlayerOpponent(
        String opponentID, int[] playerShips, BoardManager boardManager):
    call(AbstractPlayer Battleship.instantiateOpponent(String, int[], BoardManager))
            && args(opponentID, playerShips, boardManager)  && !within(DifficultyManagement);

pointcut manageDifficultyDuringInstantiationOfPlayerOpponent2(
        Battleship battleship, String opponentID, BoardManager boardManager):
    call(AbstractPlayer Battleship.instantiateOpponent(String, BoardManager))
            && args(opponentID, boardManager) && this(battleship);
```

Pointcut language

```
call(Player.new(..))
```

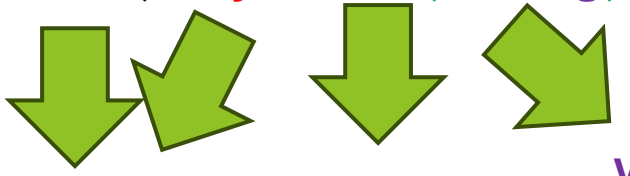


Player class is called

To capture constructor

Any number, name, and type of arguments

```
execution(Player.new(String))
```



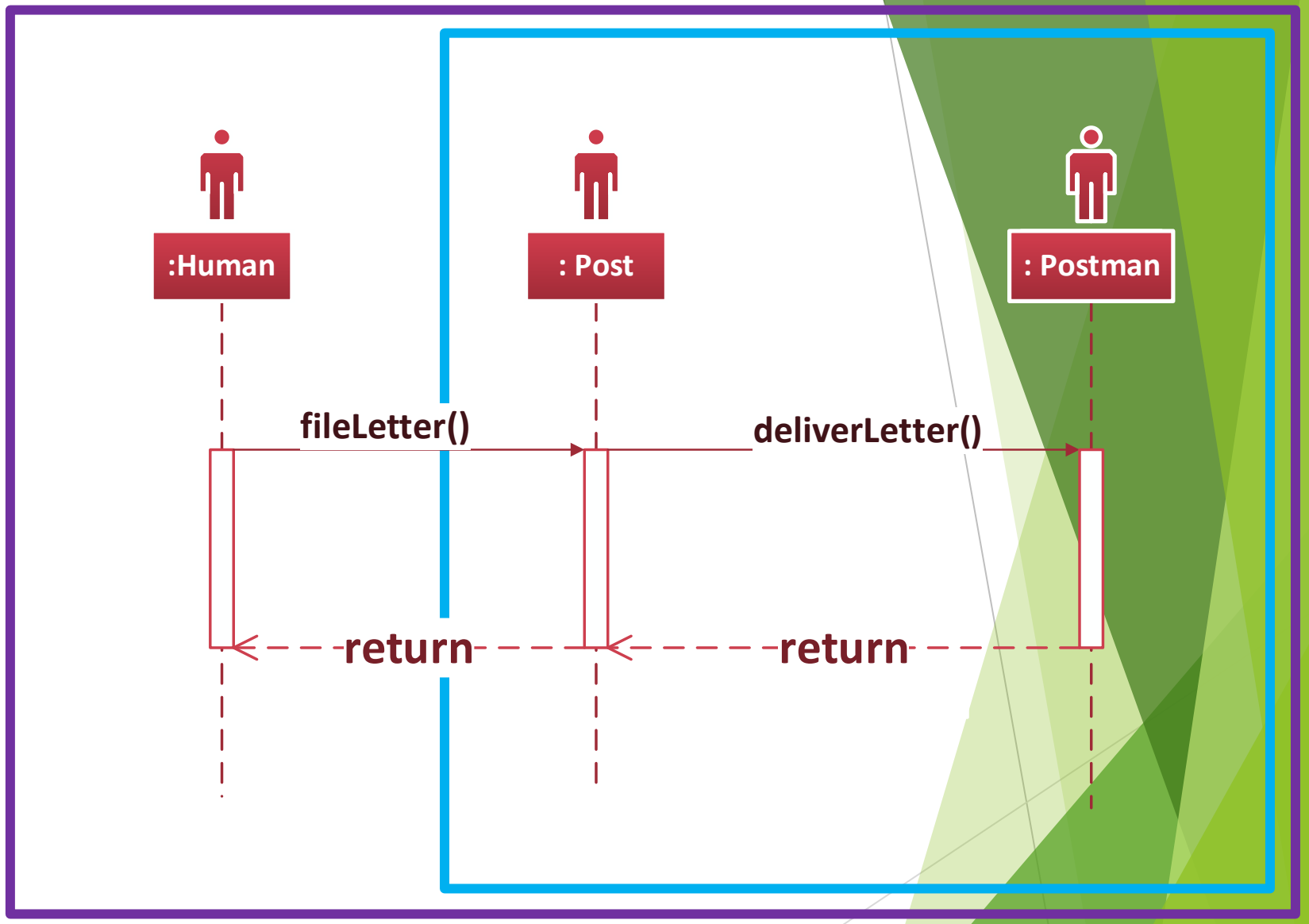
Player class is executed

To capture constructor

With one argument from which the first one has type String

Control flows

Capturing join points in particular control flows



Any return type

cflow(call(* Post.fileLetter()))

cflowbelow(call(* Post.fileLetter()))

Other example: cflowbelow(execution(* Post.fileLetter()))

Pointcuts Based on Lexical Structure

Inside its lexical structure

Type of something
- in form of pattern

within(Post)

-join points contained **within** particular entity – aspect or class

Example within(Post)
- all join points within Post class

Signature of something
- in form of pattern

withincode(* Post.fileLetter())

-join points contained **within** particular code – constructor or method

Example within(* Post.fileLetter())
- all join points within fileLettter() method of Post class

Execution Object Pointcuts

Treats types of objects during the execution

Type or Object

this(Post)

-this is instance of particular Type

Example this(Post)
- all join points that are instances of Post class

No wildcards!!!

Not matching static methods!!!

Type or Object

target(Post)

-target is instance of particular type

Example target(Post)
- object on which methods are called is type of Post - all such join points

Arguments Pointcuts

In accordance with type and number of arguments

TypePattern or Object

args(double, String)

-arguments matching with type and number
- all join points where the first argument is double and second is String

args(double, ..)

-first arguments matching required types and other are optional and not restricted in their number and type
- all join points where the first argument is double and other are optional

Conditional Checkpoints

Conditional checking

Boolean Expression

```
if(true)
```

-checks if Boolean expression matches
-all join points where condition is true

Idiom/programming tip:

Nullify the advice:

```
if(false)
```

```
if(Timer.time() > 10)
```

-checks if time is higher then 10
- all join points where time is higher than 10

Introduction

-a static crosscutting instruction
-to introduce relation such as specialization

Example [USED FOR EXAMPLE JUST TO MARK SOME CLASSES FOR FURTHER PROCESSING]:

declare parents: Human **implements** TaxDodger;

declare parents: Human **extends** RichTaxDodger;

Exception Softening

-softens exception: allows to treat checked exception as unchecked one

Checked Exception – caller must catch it or declare to throw it

Example:

```
declare soft: IllegalStateException: call(* changeState(..));
```

Precedence

-changing a precedence of aspects

Example: changing precedence of aspects from

MyAspect1 >> MyAspect2 >> MyAspect3

to

MyAspect3 >> MyAspect1 >> MyAspect2

declare precedence: MyAspect3, MyAspect1, MyAspect2;

Precedence

Use of wildcards causes errors in case of circular dependence!!!

-changing a precedence of aspects

-useful to handle overloading during use of intertype declaration where the variables are the same

Example: changing precedence of aspects from

MyAspect1 >> MyAspect2 >> MyAspect3 >> **MyExclusiveAspect1**

to

MyAspect3 >> MyAspect1 >> **MyExclusiveAspect1** >> MyAspect2

wildcard

declare precedence: MyAspect3, MyAspect1, **MyExclusiveAspect***, MyAspect2;

Difficulty configuration

Prepare configuration (with difficulty settings) before creating player's specific instance

1. PREPARATION

```
5  public aspect PlayersPrecedence {
6      declare precedence: DifficultyManagement, ComputerInstantiator;
7  }
```

2. POINTCUTS

The same pointcuts
(with other names)

"Hook" functions

```
pointcut manageDifficultyDuringInstantiationOfPlayerPlayer2(
        Battleship battleship, String playerID, BoardManager boardManager);
    call(AbstractPlayer Battleship.instantiatePlayer(String, BoardManager))
        && args(playerID, boardManager) && this(battleship);
```

```
pointcut manageDifficultyDuringInstantiationOfPlayerOpponent(
        String opponentID, int[] playerShips, BoardManager boardManager);
    call(AbstractPlayer Battleship.instantiateOpponent(String, int[], BoardManager))
        && args(opponentID, playerShips, boardManager)  && !within(DifficultyManagement);

pointcut manageDifficultyDuringInstantiationOfPlayerOpponent2(
        Battleship battleship, String opponentID, BoardManager boardManager);
    call(AbstractPlayer Battleship.instantiateOpponent(String, BoardManager))
        && args(opponentID, boardManager) && this(battleship);
```

```
pointcut manageDifficultyDuringInstantiationOfPlayerPlayer(
        String playerID, int[] playerShips, BoardManager boardManager);
    call(AbstractPlayer Battleship.instantiatePlayer(String, int[], BoardManager))
        && args(playerID, playerShips, boardManager)  && !within(DifficultyManagement);
```

Compile-time declaration

-a static crosscutting instruction
-to introduce compile time warnings and errors to warn or prevent some unwanted development practices

Example [TO PREVENT AND CHECK UNWANTED DEVELOPING PRACTICES]:

-a compiler warns when any of System.print or System.println method is used

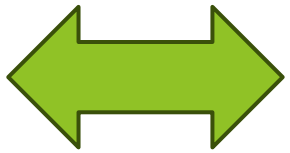
```
declare warning: call(void System.out.print*(..)) :
"Do not output directly to the console. Use logger instead!";
```

Example throwing an error

```
declare error: call(void System.out.print*(..)) :
"Do not output directly to the console. Use logger instead!";
```

Policy Enforcement

-Improving reusability
-Untangling policy code with source code
-Preventing scattering of policy code
-Easy application of policies

Compile-time  **Run-time**

```
declare warning: call(void System.out.print*(..)) :
"Do not output directly to the console. Use logger instead!";

declare error: call(void System.out.print*(..)) :
"Do not output directly to the console. Use logger instead!";
```

Example: Intertype declaration in AspectJ

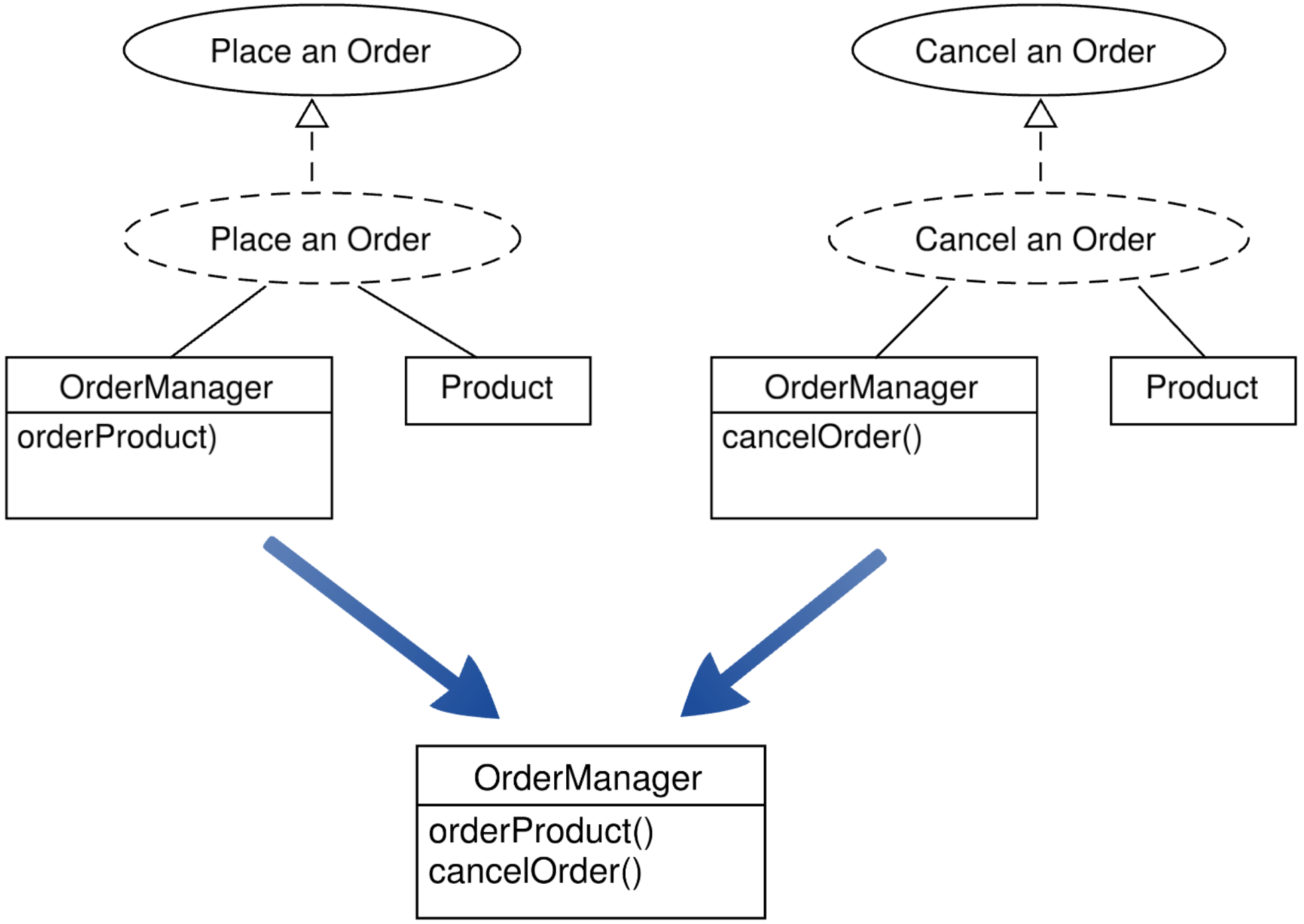
```
public aspect PlayerName {

    private String Player.name;

    private void Player.setName(String playerName) {
        this.name = playerName;
    }

    private String Player.getName() {
        return this.name;
    }
}
```

Extends Player class with ***variable*** called name with String type

Extends Player class with **function** setName(String playerName)

Extends Player class with **function** getName()

Possibly to set up functionality additively and incrementally, only empty classes that will be extended are required.


Place an Order
Cancel an Order
Place an Order
Cancel an Order
OrderManager
orderProduct)
Product
OrderManager
cancelOrder()
Product
OrderManager
orderProduct()
cancelOrder()

Example: Aspect in AspectJ

```
public aspect PlayerName {

    Player around(): call(Player.new(..)) && if(Configuration.setNames){
        Scanner reader = InputReader.getReader();
        System.out.println("Set player name:");
        String playerNameLine = reader.nextLine().replace("\n", "");

        Player createdPlayer = proceed();
        createdPlayer.setName(playerNameLine);

        System.out.println(createdPlayer.getName());

        return createdPlayer;
    }

}
```

A Complex Pointcut With Advice

```
pointcut manageOpponentTurn(Battleship battleship, AbstractPlayer player1, AbstractPlayer player2):
    call(* Battleship.opponentTurn(AbstractPlayer, AbstractPlayer))
            && args(player1, player2) && this(battleship);
```

```
void around(Battleship battleship, AbstractPlayer player1, AbstractPlayer player2):
    manageOpponentTurn(battleship, player1, player2) {
    if (Configuration.computerOpponent) {
        // battleship.compMakeGuess(player1, player2); FOR OBJECT ORIENTATED WAY OPTION IN FUTURE
        Battleship.compMakeGuess(player1, player2);
    } else {
        proceed(battleship, player1, player2);
    }
}
```

Advice

- The advice is the action and decision-making part which it allows for the expression of intersecting action at the joining point (Laddad 2003). Point connection is captured by the corresponding pointcut. It is implemented as construction in the form of a method called before (before), behind (after) or wrapping (around) the specified join point.
- To express modification of original code

```
Player around(): myPointcut {
    Scanner reader = InputReader.getReader();
    System.out.println("Set player name:");
    String playerNameLine = reader.nextLine().replace("\n", "");

    Player createdPlayer = proceed();
    createdPlayer.setName(playerNameLine);

    System.out.println(createdPlayer.getName());

    return createdPlayer;
}
```

Advice Structure

Advice type

Pointcut reference or specification

```
boolean around(Player processedPlayer): hasShipPointcut(processedPlayer) {
    String playerId = processedPlayer.getId() + StatisticVariableNames.HITS;
    boolean result = proceed(processedPlayer);

    if(result) {
        IntegerObject playerHit = (IntegerObject) statisticManager.getVariable(playerId);
        if (playerHit == null) {
            playerHit = new IntegerObject(0);
            playerHit.increaseValue();
        } else {
            playerHit.increaseValue();
        }
        statisticManager.addVariable(playerId, playerHit);
    }
    System.out.println("Player: " + playerId +
            " Unsuccessful hits: " + Integer.toString(getUnsuccessfulHits(processedPlayer)));
    return result;
}
```

Advice body

Around Advice

```
Player around(): myPointcut {
    Scanner reader = InputReader.getReader();
    System.out.println("Set player name:");
    String playerNameLine = reader.nextLine().replace("\n", "");

    Player createdPlayer = proceed();
    createdPlayer.setName(playerNameLine);

    System.out.println(createdPlayer.getName());

    return createdPlayer;
}
```

Before advice

```
before(Player processedPlayer, Player otherPlayer): playerMove(processedPlayer, otherPlayer) {
    String playerId = processedPlayer.getId() + StatisticVariableNames.MOVES;
    IntegerObject playerMove = (IntegerObject) statisticManager.getVariable(playerId);
    if (playerMove == null) {
        playerMove = new IntegerObject(0);
        playerMove.increaseValue();
    } else {
        playerMove.increaseValue();
    }
    statisticManager.addVariable(playerId, playerMove);

    IntegerObject playerMove1 = (IntegerObject) statisticManager.getVariable(playerId);
    System.out.println(playerMove1.getValue());
}
```

Reflective API – AspectJ

-information about join point

thisJoinPoint

-dynamic information about join point:
- -method arguments
- -information about target object
- -information about execution object

- static information can be reached by calling getStaticPart()

thisJoinPointStaticPart

-static information about join point:
- -source location
- -information about kind of method execution/call, field-set,…
- -join point signature

- provides structural context information about join point

thisEnclosingJoinPointStaticPart

-information about enclosing join point (enclosing context)
-depends on join point [enclosing – surrounding]:
call of method – enclosing context is **execution of caller method**

Association of Aspects in AspectJ

Per Virtual Machine

-default association

Per Object

-association with object

Per Control Flow

-association with control flow

-to manage states/data in particular control flow

Per Object Aspect Association

perthis(myPointcut())

-separate aspect instance is associated with this object for all join points matching pointcut myPointcut())

pertarget(myPointcut())

-separate aspect instance is associated with target object for all join points matching pointcut myPointcut())

-to define specifics later

-association with object

```
public abstract aspect MyAspect perthis(myPointcut()) {
    abstract pointcut myPointcut();
}
```

-the specific pointcut should be provided in inherited aspect

Per Control Flow Aspect Association

`percflow(myPointcut())`

-separate aspect instance is associated with the control flow at join points matching pointcut `myPointcut()`

`percflowbelow(myPointcut())`

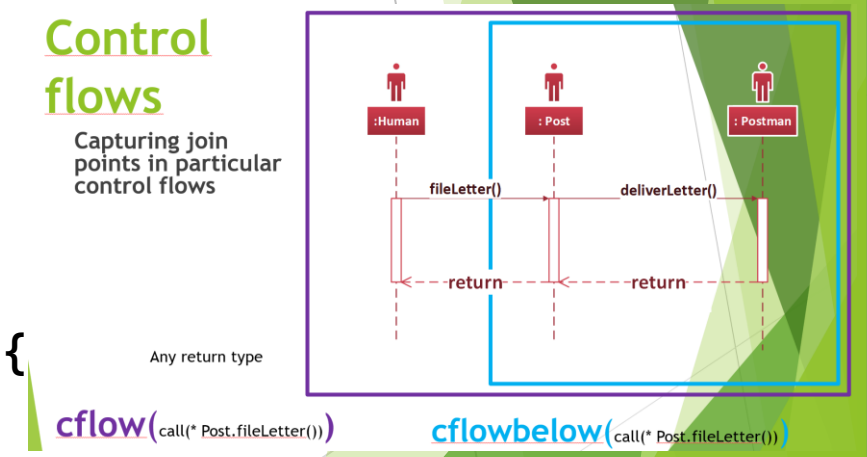
-separate aspect instance is associated with control flow below at join points matching pointcut `myPointcut()`

-to define specifics later

-association with object

```
public abstract aspect MyAspect percflow(myPointcut()) {
    abstract pointcut myPointcut();
}
```

-the specific pointcut should be provided in inherited aspect



Privileged Aspect - AspectJ

-to access private variable or method which is declared In particular class

```
privileged public aspect PlayerName {

    private String Player.name;


    private void Player.setName(OtherPlayer oPlayer) {
        this.name = oPlayer.privatePlayerName;
    }


    private String Player.getName() {
        return this.name;
    }
}
```

Spring AOP vs AspectJ

Table II
Difference between Spring AOP and AspectJ

Spring AOP	AspectJ
Pure Java implementation	Using extensions of Java implementation
Compilation process is not separated	AspectJ compiler (ajc) needed unless Load Time Weaving is set up
Available runtime weaving	Not available runtime weaving. Compile-time, post-compile, and load-time Weaving supported
Method level weaving supported so less robust	Use final class/methods, weave fields, constructors, static initialization, methods, etc. so most robust
Implemented on beans managed by Spring container	Implemented on all domain objects
Method execution point cuts supported	All point cuts supported
Proxies are created of targeted objects, and aspects are applied on these proxies	Aspects are weaved directly into code before application is executed (before runtime)
Much slower than AspectJ	Performance is better than Spring AOP
Easy to learn and implement	More complicated than Spring AOP

Taken from: *KUMAR, Ravi a Munishwar RAI, 2019. A Comparative Study of AOP Approaches: AspectJ, Spring AOP, JBoss AOP. 2019*

LARA – language independent aspect-oriented programming

-example of DSL (domain specific language)
-generic source code transformation and analysis

JackDaw: https://specs.fe.up.pt/tools/jackdaw/

Independence on programming language:

thanks to separation of the LARA language from the language specification of the target language, which consists of a specification of relevant points of interest, along with their attributes and actions

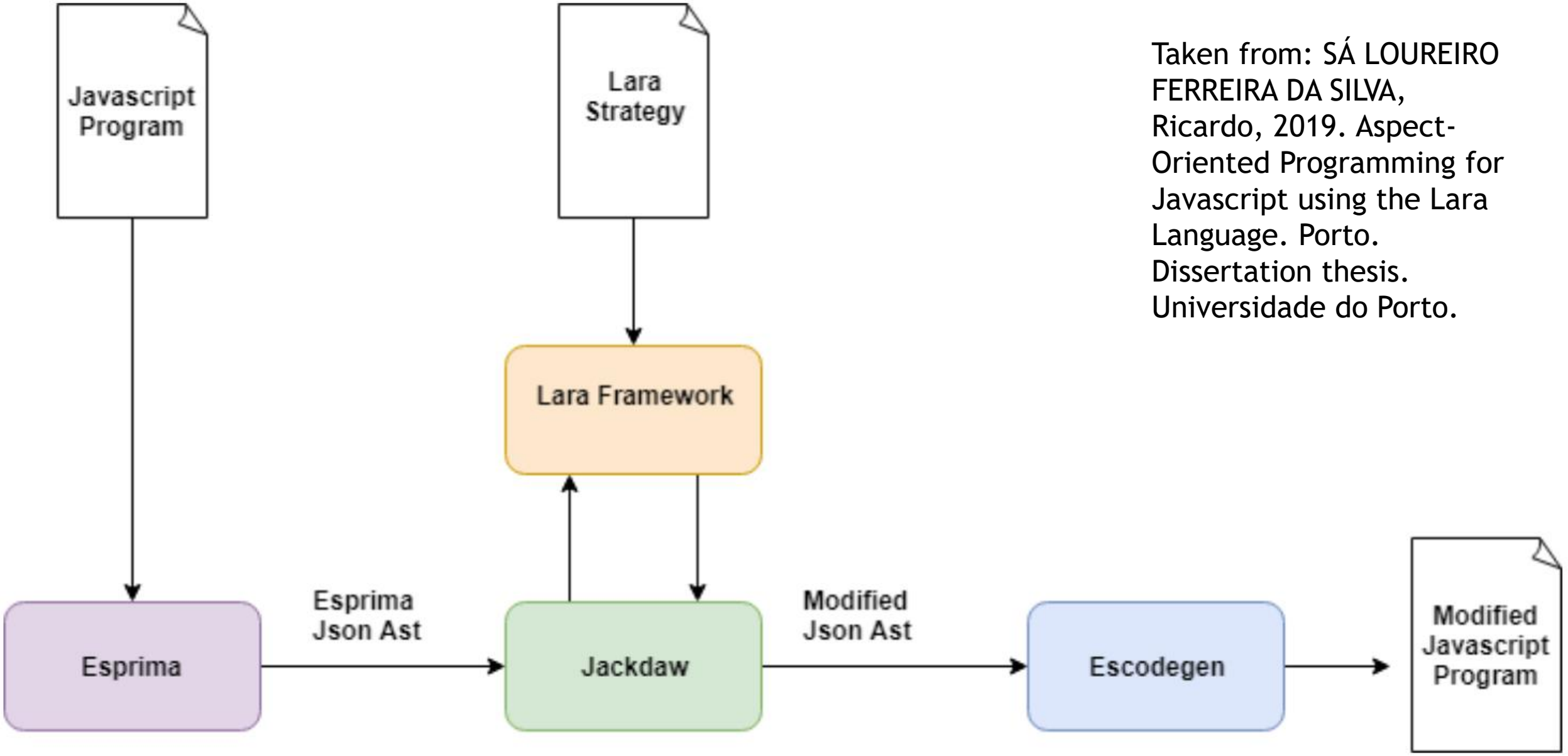
Source-to-source transformations

Taken from: SÁ LOUREIRO FERREIRA DA SILVA, Ricardo, 2019. Aspect-Oriented Programming for Javascript using the Lara Language. Porto. Dissertation thesis. Universidade do Porto.

On Figure 2.1 we present a simple example of LARA code. This code declares a LARA aspect which selects all the function calls inside functions and proceeds to insert a new comment in the line prior to each call.

```
1 aspectdef LaraAspect
2     select function.call end
3     apply
4         $call.insert before "// Call" + $call.name + " at line: " + $call.line;
5     end
6 end
```

Figure 4.1: Jackdaw flowchart



Taken from: SÁ LOUREIRO FERREIRA DA SILVA, Ricardo, 2019. Aspect-Oriented Programming for Javascript using the Lara Language. Porto. Dissertation thesis. Universidade do Porto.

Approach - flow

- **1. Parsing of JavaScript code into AST -** Esprima (high performance, standard-compliant ECMAScript parser written in ECMAScript)
  - *Tokenizing and parsing JavaScript into AST - JSON object (all used structures and hierarchies optionally with comments, location of nodes,…)*
  - Lack of JavaScript parsers in JavaScript – some exists like ***Rhino*** or ***Java 8 Nashorn Javascript engine***
- ***2. Creation of JackDaw join points***
  - JavaScript language structures and statements present in the source code:
    - declarations, while statements, if statements, try-catch statements, etc
  - Operable/supported join points used as (see following examples):
    - project, file, functionDeclaration, classDeclaration, declaration, forStatement, whileStatement, switchStatement, ifStatement, tryStatement and expressionStatement.
  - Each is mapped to Java class where node is queried internally
    - Results in performed actions and returned respective attributes (from the weaver language specification)

Approach - flow

- ***3. Executing LARA code (*LARA selector language)**
  - ***to query for a particular type of join point***
    - Respective object in AST must be searched and applied (wrapped) to particular type of join point
    - Generic join point:
      - In case of no such type of join point – its wrapped around generic join point (with methods common to every type of such join points)
      - For join point attributes that return another join point (to wrap JSON nodes to already undefined join points)
  - *Querying and capturing the JackDaw join points*
- ***3. Weaving aspects into AST using Jackdaw***
  - *interacting and applying actions to AST ordered by LARA framework*
  - ***Using* Jackdaw Query Engine *for searche operations***
    - getting the parent of a join point, getting all child nodes, descendant nodes, nodes of particular type, etc.
- **4. Transformation of updated AST back to JavaScript code -**
  Escodegen library(ECMAScript code generator from Mozilla's Parser API AST)
  - Allows to customize formatting of the transformed/generated code
  - Performed by JackDaw

Tests: Obfuscation

Variable renaming

```
aspectdef RenameDeclarations

  var i = 0;
  select declarator end
  apply
    i++;
    var newName = "new_var_"+i;
    $declarator.refactor(newName);
  end

end
```

Taken from: SÁ LOUREIRO FERREIRA DA SILVA, Ricardo, 2019. Aspect-Oriented Programming for Javascript using the Lara Language. Porto. Dissertation thesis. Universidade do Porto.

Figure 4.2: LARA aspect that renames all declarations.

```
var a = 1
while(a < 5){
  a++;
}
var a = 3
if(a >= 5){
  console.log(a);
}
```

Figure 4.3: Input JavaScript code.

```
var new_var_1 = 1;
while (new_var_1 < 5) {
    new_var_1++;
}
var new_var_2 = 3;
if (new_var_2 >= 5) {
    console.log(new_var_2);
}
```

Figure 4.4: Generated JavaScript code.

Features

```
1 aspectdef insertDocumentation
2
3   select function end
4   apply
5     var text = "/* Function: " + $function.name + " */";
6     $function.insert("before", text);
7   end
8 end
```

Figure 4.8: LARA aspect for inserting a comment.

```
1 aspectdef addPrefix
2
3   select function.blockStatement end
4   apply
5     if($function.name == "fun1"){
6       $blockStatement.children[0].insert("before","console.log('prefix code.');");
7     }
8   end
9 end
```

Taken from: SÁ LOUREIRO FERREIRA DA SILVA, Ricardo, 2019. Aspect-Oriented Programming for Javascript using the Lara Language. Porto. Dissertation thesis. Universidade do Porto.

Figure 4.11: LARA aspect for inserting a prefix function.

```
1
2 import obfuscation.Obfuscator;
3 aspectdef obfuscateFile
4
5   select file end
6
7   apply
8     var configuration = {predicates:{min:3,max:6}, functions:[{name:"fun1",algorithms:["
          opaque","flat"]}],excluded:[]};
9     applyCustomObfuscation($file,configuration);
10  end
11 end
```

Figure 5.1: LARA aspect uses a custom configuration in order to apply obfuscation to the functions of a file.

```
1
2 import obfuscation.Obfuscator;
3 aspectdef getObfuscableFunctionsAndExecute
4
5   select file end
6
7   apply
8     var config = discoverObfuscableFunctions($file);
9     applyCustomObfuscation($file,config);
10  end
11 end
```

Figure 5.2: LARA aspect that uses the discovery method in order to create a working obfuscation configuration for a file.

Taken from: SÁ LOUREIRO FERREIRA DA SILVA, Ricardo, 2019. Aspect-Oriented Programming for Javascript using the Lara Language. Porto. Dissertation thesis. Universidade do Porto.

Sample JavaScript/TypeScript AST

Abstract syntax tree (AST)

Available as part of NodeJS - Express JS API:

https://github.com/jperdek/variabilityMgmtCodeConstructsComplexity/tree/master/astConverterAndComplexityEvaluator

Unpack it!!!

cd pathTo/astConverterAndComplexityEvaluator

npm install

npm start

```
E:\aspects\typescriptConverter>npm start

> typescript_parser@1.0.0 start
> node ./typescriptParser.js

Server up and running on port 5001
```

```
1
2   const express = require('express');
3   const app = express();
4   const cors = require('cors');
5   const escomplex = require('typhonjs-escomplex');
6   const escomplex2 = require('escomplex');
7   const { Complexity } = require('eslintcc');
8
9
10  const bodyParser = require('body-parser');
11  //app.use(bodyParser.urlencoded({ extended: true }));
12  //app.use(bodyParser.json());
13  app.use(bodyParser.text({limit: '50mb'}));
14
15  const ts = require('typescript');
16  const fs = require('fs');
17
18  app.post('/convert', function (request, response) {
19      const ast = ts.createSourceFile("x.ts", request.body, ts.ScriptTarget.Latest);
20      const printer = ts.createPrinter({ newLine: ts.NewLineKind.LineFeed });
21      const code = printer.printNode(ts.EmitHint.Unspecified, JSON.parse(JSON.stringify(ast)), JSON.parse(JSON.stringify(ast)));
22      response.json({ast});
23      response.setHeader('Content-Type', 'application/json');
24      response.status(200);
25  });
26
27  app.post('/convertBack', function (request, response) {
28      console.log(request.body);
29      const ast = JSON.parse(request.body);
30      const printer = ts.createPrinter({ newLine: ts.NewLineKind.LineFeed });
31      const code = printer.printNode(ts.EmitHint.Unspecified, ast, ast);
32      response.set('Content-Type', 'text/html');
33      response.send(code);
34      response.status(200);
35  });
36
```

Convert JavaScript into AST

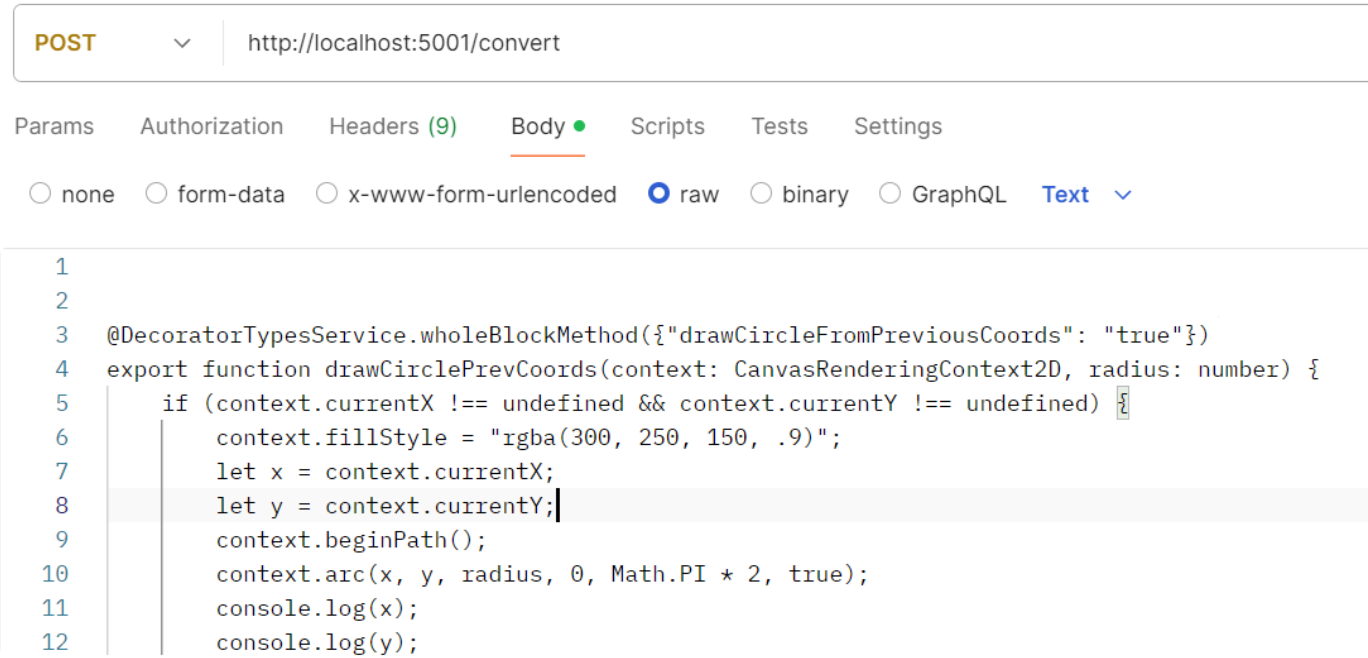
POST http://localhost:5001/convert

```
function popData() {
}
class AAA {
    myFunc(variable : number) {

    }
}
```

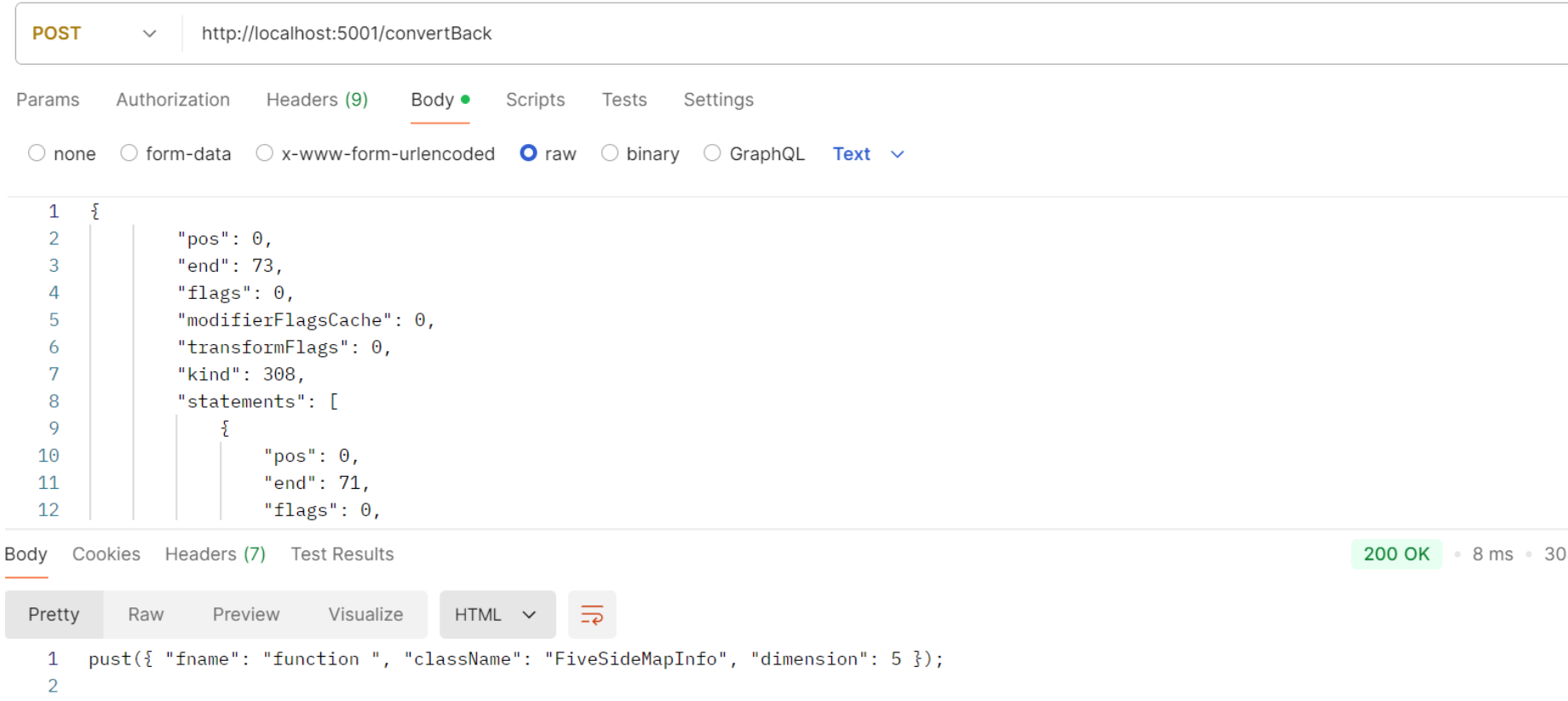
Convert TypeScript into AST

POST http://localhost:5001/convert

```
@DecoratorTypesService.wholeBlockMethod({"drawCircleFromPreviousCoords": "true"})
export function drawCirclePrevCoords(context: CanvasRenderingContext2D, radius: number) {
    if (context.currentX !== undefined && context.currentY !== undefined) {
        context.fillStyle = "rgba(300, 250, 150, .9)";
            let x = context.currentX;
            let y =    context.currentY;
        context.beginPath();
        context.arc(x, y, radius, 0, Math.PI * 2, true);
        console.log(x);
        console.log(y);
        context.closePath();
        context.fill();
    }
}
```



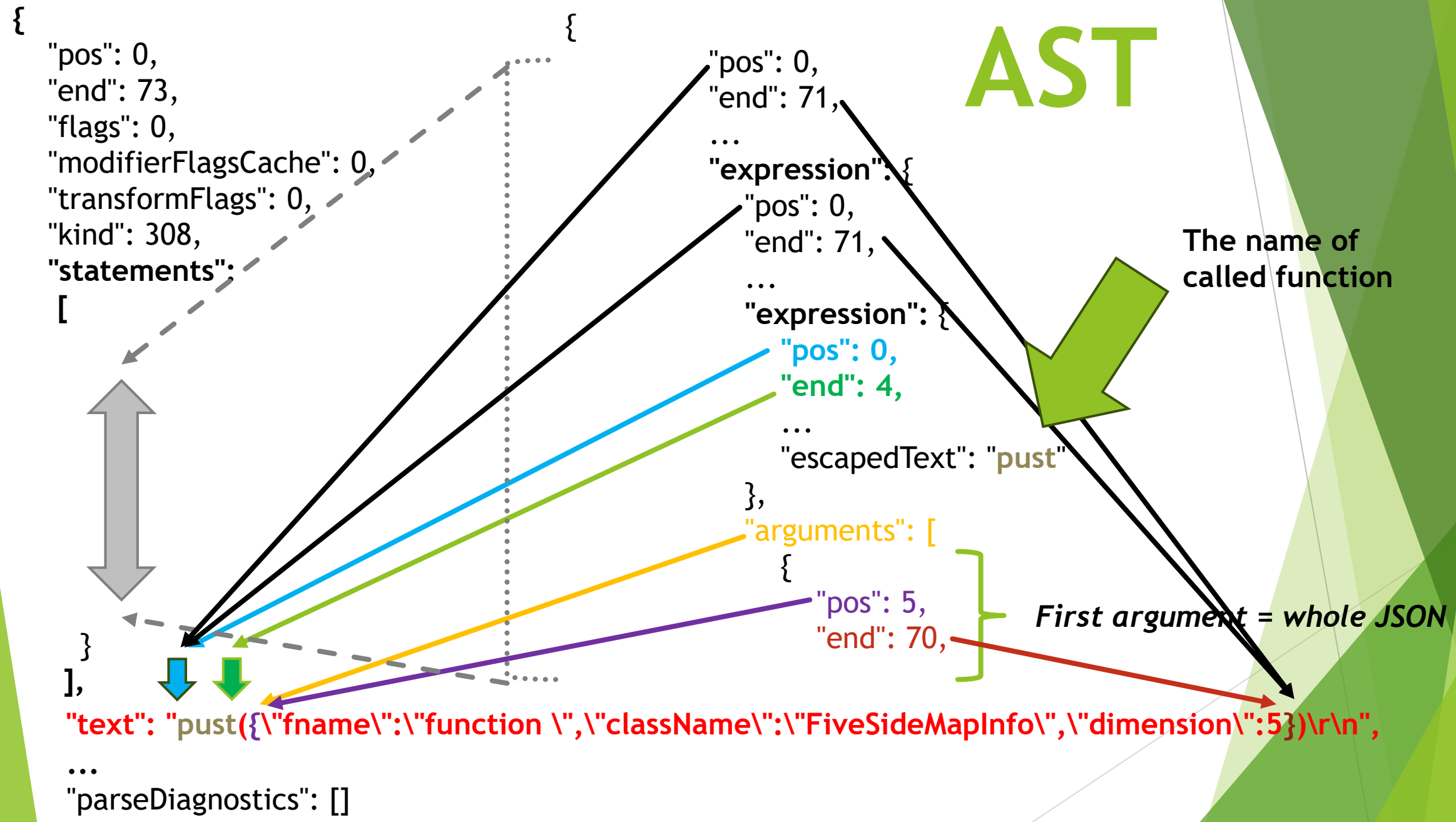
Convert From AST Back Into Code



POST http://localhost:5001/convertBack

{"pos":0,"end":73,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":308,"statements":[{"pos":0,"end":71,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":241,"expression":{"pos":0,"end":71,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":210,"expression":{"pos":0,"end":4,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":79,"escapedText":"pust"},"arguments":[{"pos":5,"end":70,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":207,"properties":[{"pos":6,"end":25,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":299,"name":{"pos":6,"end":13,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":10,"text":"fname","hasExtendedUnicodeEscape":false},"initializer":{"pos":14,"end":25,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":10,"text":"function ","hasExtendedUnicodeEscape":false}},{"pos":26,"end":55,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":299,"name":{"pos":26,"end":37,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":10,"text":"className","hasExtendedUnicodeEscape":false},"initializer":{"pos":38,"end":55,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":10,"text":"FiveSideMapInfo","hasExtendedUnicodeEscape":false}},{"pos":56,"end":69,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":299,"name":{"pos":56,"end":67,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":10,"text":"dimension","hasExtendedUnicodeEscape":false},"initializer":{"pos":68,"end":69,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":8,"text":"5","numericLiteralFlags":0}}],"multiLine":false}]}}],"endOfFileToken":{"pos":71,"end":73,"flags":0,"modifierFlagsCache":0,"transformFlags":0,"kind":1},"fileName":"x.ts","text":"pust({\"fname\":\"function\",\"className\":\"FiveSideMapInfo\",\"dimension\":5})\r\n","languageVersion":99,"languageVariant":0,"scriptKind":3,"isDeclarationFile":false,"hasNoDefaultLib":false,"bindDiagnostics":[],"pragmas":{},"referencedFiles":[],"typeReferenceDirectives":[],"libReferenceDirectives":[],"amdDependencies":[],"nodeCount":15,"identifierCount":1,"identifiers":{},"parseDiagnostics":[]}

AST


{
"pos": 0,
"end": 73,
"flags": 0,
"modifierFlagsCache": 0,
"transformFlags": 0,
"kind": 308,
"statements":
[
}
],
"parseDiagnostics": []
}
{
"pos": 0,
"end": 71,
...
"expression": {
"pos": 0,
"end": 71,
...
"expression": {
"pos": 0,
"end": 4,
...
"escapedText": "pust"
},
"arguments": [
{
"pos": 5,
"end": 70,
"text": "pust({\"fname\":\"function \",\"className\":\"FiveSideMapInfo\",\"dimension\":5})\r\n",
...
The name of called function
First argument = whole JSON

Restrictions of using aspects in typescript

BRIEFNESS

How easy, how exactly, and without complications it is possible to use given tool

INVASIVENESS

how well aspects are separated from the rest of the code

MATURITY

All abilities and possibilities of the whole functionality provided by a given library

Komponent / Nástroj	AspectScript	AOJS	AspectJS
Invazívnosť (invasiveness)	-	+	-
Stručnosť (briefness)	++	+	++
Vyspelosť (maturity)	++	-	-

The comparison of AOP tools (Huang et al. 2015)

Wenhao Huang, Chengwan He, and Zheng Li. 2015. A Comparison of Implementations for Aspect-Oriented JavaScript:. Zhengzhou, China. https://doi.org/10.2991/csic-15.2015.9

Ricardo Sá Loureiro Ferreira da Silva. 2019. Aspect-Oriented Programming for Javascript using the Lara Language. Dissertation thesis. Universidade do Porto, Porto.

TABLE I. Table aggregated summary of comparison

Framework	Comparison Criteria		
	Invasiveness	*Briefness*	*Maturity*
AspectJS	-	++	-
AOJS	+	+	-
AspectScript	-	++	++

Wenhao Huang, Chengwan He, and Zheng Li. 2015.
A Comparison of Implementations for Aspect-Oriented
JavaScript:. Zhengzhou, China.
https://doi.org/10.2991/csic-15.2015.9

Table 4.1: Table showing comparison between tools.

Component / Tool	AspectScript	AOJS	AspectJS	aspect.js	aspectjs	**Jackdaw**
Invasiveness	-	+	-	-	-	++
Briefness	+	+	+	-	++	++
Maturity	+	-	-	+	-	–

Taken from: SÁ LOUREIRO FERREIRA DA SILVA, Ricardo, 2019. Aspect-Oriented Programming for Javascript using the Lara Language. Porto. Dissertation thesis. Universidade do Porto.

Aspect C++

Guide for Windows, especially Windows 11

For IDE try: https://acdt.aspectc.org/

Based on old Eclipse Neon – not worked for me

AspectC++ Quick Reference

Concepts

Aspects are modular implementations of crosscutting concerns. They can affect *join points* in the component code, e.g. class definitions, or in the dynamic control flow, e.g. function calls, by *advice*. A set of related join points is called *pointcut* and defined by a *pointcut expression*.

Aspects

Aspects extend the concept of C++ classes. They may define ordinary class members as well as *advice*.

aspect *A* **:** *public B* **{ ... };**
defines the aspect *A*, which inherits from class or aspect *B*

Slices

A slice is a fragment of a C++ element like a class. It may be used by introduction *advice* to implemented static extensions of the program.

slice class ***ASlice*** **{ ... void** ***f*****(); ... };**
defines a class slice called *ASlice*
slice void ***ASlice::f*****() { ... }**
defines a non-inline member function f() of slice ASlice

Advice

An advice declaration specifies *how* an aspect affects a set of join points.

advice *pointcut* **: around**(...) {...}
the advice code is executed in place of the join points in the pointcut
advice *pointcut* **: before/after**(...) {...}
the advice code is executed before/after the join points in the pointcut
advice *pointcut* **: order**(*high*, ...*low*)**;**
high and *low* are pointcuts, which describe sets of aspects. Aspects on the left side of the argument list always have a higher precedence than aspects on the right hand side at the join points, where the order declaration is applied.
advice *pointcut* **: slice class : public** *Base* {...}
introduces a new base class *Base* and members into the target classes matched by *pointcut*.
advice *pointcut* **: slice** *ASlice* **;**
introduces the slice *ASlice* into the target classes matched by *pointcut*.

Match Expressions

Match expressions are primitive pointcut expressions. They filter program entities based on their signature.

Type Matching

`"int"`
matches the C++ built-in scalar type `int`
`"% *"`
matches any pointer type

Namespace and Class Matching

`"Chain"`
matches the class, struct or union *Chain*
`"Memory%"`
matches any class, struct or union whose name starts with "Memory"

Function Matching

`"void reset()"`
matches the function *reset* having no parameters and returning `void`
`"% printf(...)"`
matches the function *printf* having any number of parameters and returning any type
`"% ...::%(...)"`
matches any function, operator function, or type conversion function (in any class or namespace)
`"% ...::Service::%(...) const"`
matches any const member-function of the class `Service` defined in any scope
`"% ...::operator %(...)"`
matches any type conversion function
`"virtual %C::%(...)"`
matches any virtual member function of `C`
`"static % ...::%(...)"`
matches any static member or non-member function

Variable Matching

`"int counter"`
matches the variable *counter* of type `int`
`"% guard"`
matches the global variable *guard* of any type
`"% ...::%"`
matches any variable (in any class or namespace)
`"static % ...::%"`
matches any static member or non-member variable

Template Matching†

`"std::set<...>"`
matches all template instances of the class *std::set*
`"std::set<int>"`
matches only the template instance *std::set<int>*
`"% ...::%<...>::%(...)"`
matches any member function from any template class instance in any scope

Predefined Pointcut Functions

Predefined pointcut functions are used to filter, map, join, or intersect pointcuts.

Functions / Variables

call(*pointcut*) N→C_C‡‡
provides all join points where a named and user provided entity in the *pointcut* is called.
builtin(*pointcut*)‡ N→C_B
provides all join points where a named built-in operator in the *pointcut* is called.
execution(*pointcut*) N→C_E
provides all join points referring to the implementation of a named entity in the *pointcut*.
construction(*pointcut*) N→C_{Cons}
all join points where an instance of the given class(es) is constructed.
destruction(*pointcut*) N→C_{Des}
all join points where an instance of the given class(es) is destructed.
get(*pointcut*) N→C_G
provides all join points where a global variable or data member in the *pointcut* is read.
set(*pointcut*) N→C_S
provides all join points where a global variable or data member in the *pointcut* is written.
ref(*pointcut*) N→C_R
provides all join points where a reference (reference type or pointer) to a global variable or data member in the *pointcut* is created.

pointcut may contain function, variable, namespace or class names. A namespace or class name is equivalent to the names of all functions and variables defined within its scope combined with the || operator (see below).

Control Flow

cflow(*pointcut*) C→C
captures join points occuring in the dynamic execution context of join points in the *pointcut*. The argument *pointcut* is forbidden to contain context variables or join points with runtime conditions (currently cflow, that, or target).

Types

base(*pointcut*) N→$N_{C,F}$
returns all base classes resp. redefined functions of classes in the *pointcut*
derived(*pointcut*) N→$N_{C,F}$
returns all classes in the *pointcut* and all classes derived from them resp. all redefined functions of derived classes

Scope

within(*pointcut*) N→C
filters all join points that are within the functions or classes in the *pointcut*
member(*pointcut*) N→N
maps the scopes given in *pointcut* to any contained named entities. Thus a class name for example is mapped to all contained member functions, variables and nested types.

Source: https://www.aspectc.org/doc/ac-quickref.pdf

Context

that(*type pattern*) N→C
returns all join points where the current C++ `this` pointer refers to an object which is an instance of a type that is compatible to the type described by the *type pattern*

target(*type pattern*) N→C
returns all join points where the target object of a call or other access is an instance of a type that is compatible to the type described by the *type pattern*

result(*type pattern*) N→C
returns all join points where the result object of a call/execution or other access join point is an instance of a type described by the *type pattern*

args(*type pattern*, ...) (N,...)→C
a list of *type patterns* is used to provide all joinpoints with matching argument signatures

Instead of the *type pattern* it is possible here to pass the name of a **context variable** to which the context information is bound. In this case the type of the variable is used for the type matching.

Algebraic Operators

pointcut **&&** *pointcut* (N,N)→N, (C,C)→C
intersection of the join points in the *pointcuts*

pointcut **||** *pointcut* (N,N)→N, (C,C)→C
union of the join points in the *pointcuts*

! *pointcut* N→N, C→C
exclusion of the join points in the *pointcut*

Named Pointcuts and Attributes

Pointcut expressions can also refer to user-defined pointcuts.

class *[[myns::myattr]]* C {...}
annotates class C with the attribute *myattr* from the namespace *myns*.

pointcut *mypct*() = "C";
defines a "named pointcut" *my*pct(), which represents the class "C"

attribute *myattr*(); // in *myns*
declares a user-defined attribute *myattr*(), which also represents "C"

JoinPoint-API for Advice Code

The JoinPoint-API is provided within every advice code body by the built-in object **tjp** of class **JoinPoint**.

Compile-time Types and Constants

That [type]
object type (object initiating a call or entity access)

Target [type]
target object type (target object of a call or entity access)

Entity [type]
type of the primary referenced entity (function or variable)

MemberPtr [type]
type of the member pointer for entity or "void *" for nonmembers.

Result [type]
type of the object, used to *store* the result of the join point

Res::Type, Res::ReferredType [type]
result type of the affected function or entity access

Arg<i>::Type, Arg<i>::ReferredType [type]
type of the i^{th} argument of the affected join point (with $0 \leq i < ARGS$)

ARGS [const]
number of arguments

Array [type]
type of an accessed array

Dim<i>::Idx, Dim<i>::Size [type], [const]
type of used index and size of the i^{th} dimension (with $0 \leq i < DIMS$)

DIMS [const]
number of dimensions of an accessed array or 0 otherwise

JPID [const]
unique numeric identifier for this join point

JPTYPE [const]
numeric identifier describing the type of this join point (*AC::CALL*, *AC::BUILTIN*, *AC::EXECUTION*, *AC*::CONSTRUCTION, *AC*::DESTRUCTION, *AC::GET*, *AC::SET* or *AC::REF*)

Runtime Functions and State

static const char ***signature**()
gives a textual description of the join point (type + name)

static const char ***filename**()
returns the name of the file in which the joinpoint shadow is located

static int **line**()
the source code line number in which the joinpoint shadow is located

That ***that**()
returns a pointer to the object initiating a call or 0 if it is a static method or a global function

Target ***target**()
returns a pointer to the object that is the target of a call or 0 if it is a static method or a global function

Entity ***entity**()
returns a pointer to the accessed entity (function or variable) or 0 for member functions or builtin operators

MemberPtr **memberptr**()
returns a member pointer to entity or 0 for nonmembers

Result ***result**()
returns a typed pointer to the result value or 0 if there is none

Arg<i>::ReferredType ***arg**<*i*>()
returns a typed pointer to the i^{th} argument value (with $0 \leq i < ARGS$)

void ***arg**(*int* i)
returns a pointer to the i^{th} argument memory location ($0 \leq i < ARGS$)

void **proceed**()
executes the original code in an around advice (should be called at most once in around advice)

AC::Action **&action**()
returns the runtime action object containing the execution environment to execute (*trigger()*) the original code encapsulated by an around advice

Array ***array**()
returns a typed pointer to the accessed array

Dim<i>::Idx **idx**<*i*>()
returns the value of the i^{th} used index

Runtime Type Information

static AC::Type **resulttype**()
static AC::Type **argtype**(*int* i)
return a C++ ABI V3†† conforming string representation of the result type / argument type of the affected function

JoinPoint-API for Slices

The JoinPoint-API is provided within introduced slices by the built-in class **JoinPoint** (state of target class *before* introduction).

static const char ***signature**()
returns the target class name as a string

That [type]
The (incomplete) target type of the introduction

BASECLASSES [const]
number of baseclasses of the target class

BaseClass<I>::Type [type]
type of the I^{th} baseclass

BaseClass<I>::prot, BaseClass<I>::spec [const]
Protection level (AC::PROT_NONE /PRIVATE /PROTECTED /PUBLIC) and additional specifiers (AC::SPEC_NONE /VIRTUAL) of the I^{th} baseclass

MEMBERS [const]
number of member variables of the target class

Member<I>::Type, Member<I>::ReferredType [type]
type of the I^{th} member variable of the target class

Member<I>::prot, Member<I>::spec [const]
Protection level (see BaseClass<I>::prot) and additional member variable specifiers (AC::SPEC_NONE /STATIC /MUTABLE)

*static ReferredType *Member<I>*::**pointer**(*T *obj=0*)
returns a typed pointer to the I^{th} member variable (obj is needed for non-static members)

*static const char *Member<I>*::**name**()
returns the name of the I^{th} member variable

Example (simple tracing aspect)

```
aspect Tracing {
  advice execution("% Business::%(...)") : before() {
    cout << "before " << JoinPoint::signature() << endl;
} };
```

Reference sheet corresponding to AspectC++ 2.3, July 12, 2022. For more information visit `http://www.aspectc.org`.

† support for template instance matching is an experimental feature

‡ This feature has limitations. Please see the AspectC++ Language Reference.

†† `https://mentorembedded.github.io/cxx-abi/abi.html#mangling`

‡‡ C, C_C, C_B, C_E, C_{Cons}, C_{Des}, C_G, C_S, C_R: Code (any, only *Call*, only *Builtin*, only *Execution*, only object *Construction*, only object *Destruction*, only *Get*, only *Set*, only *Ref*)

N, N_N, N_C, N_F, N_V, N_T: Names (any, only *Namespace*, only *Class*, only *Function*, only *Variables*, only *Type*)

Source: https://www.aspectc.org/doc/ac-quickref.pdf

Preparation for Windows

Some steps will differ on Unix based systems

1. Download AspectC
2. Download and install G++ compiler
3. Prepare C/C++ source files and headers
4. Prepare AspectC headers
5. Compile and weave them together
6. Launch executable

C++ aspect-oriented Weaving tool - AspectC

- 1. Download from: https://www.aspectc.org/Download.php
- 2. Extract to chosen source directory

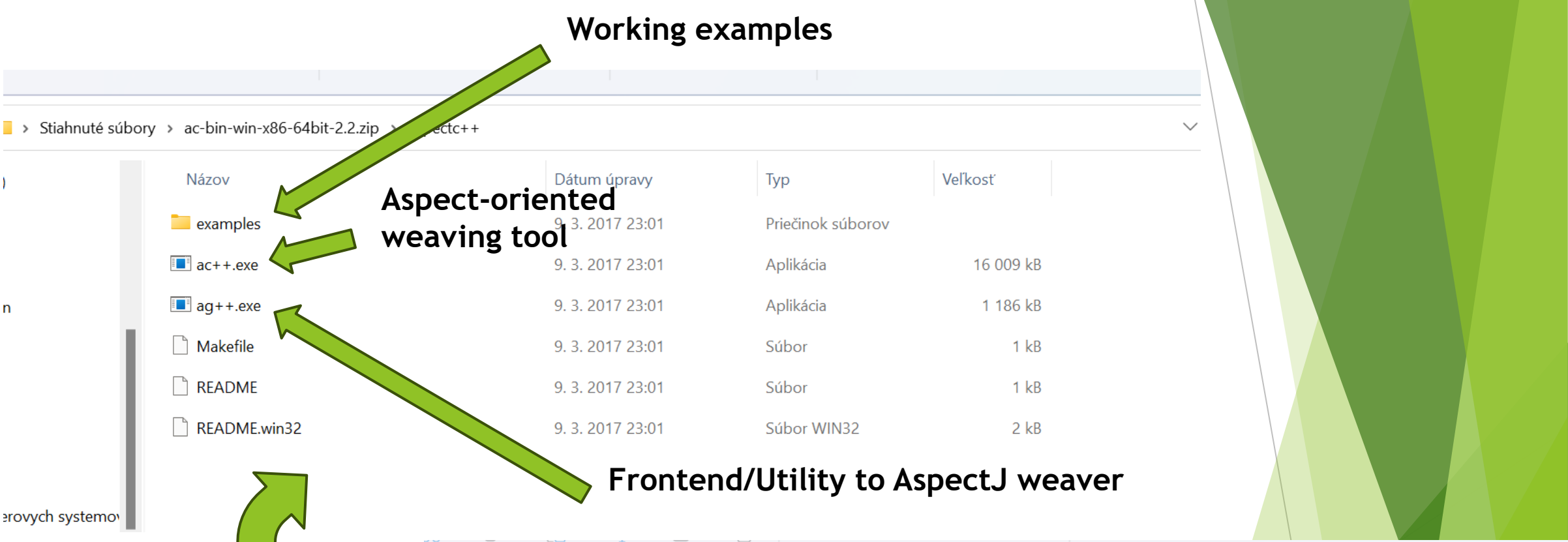
s already included. The AspectC++ Addin for Visual Studio .NET is currently not available and looking for a new maintainer.

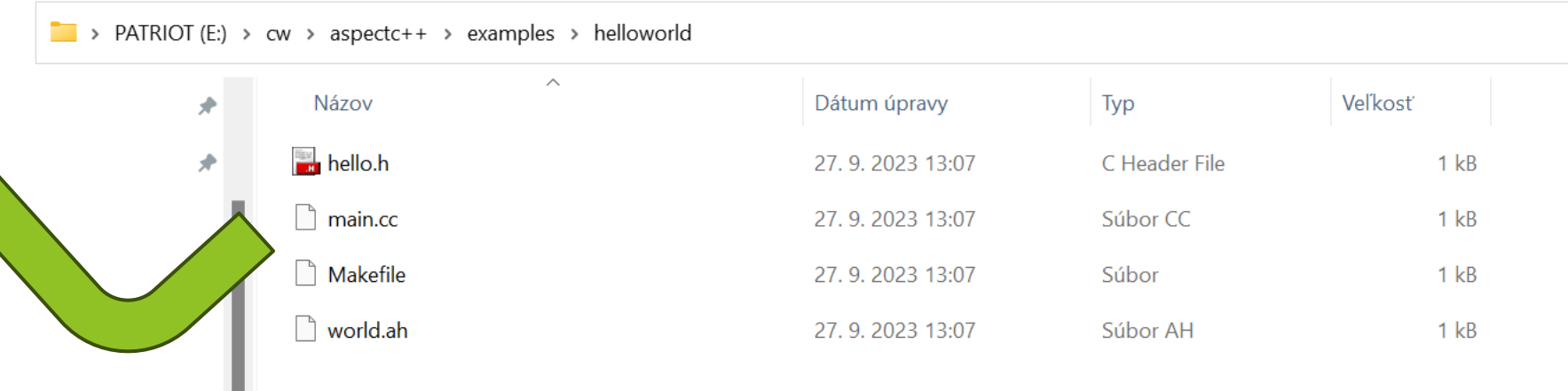
untu Linux systems available. They can be installed with "apt-get install aspectc++" or similar commands or package management tool
e or the AspectC++ Ubuntu page. Thanks a lot **Reinhard Tartler** for maintaining these packages!

good reason)

Version	Release-Date	Download	Install notes	ChangeLog
2.2	10.03.2017	Linux/x86 Linux/x86_64 MacOSX/x86_64 Windows/x86_64 Sources	README README.win	2.2
2.1	10.07.2016	Linux/x86 Linux/x86_64 MacOSX/x86_64 Windows/x86_64	README README.win	2.1

3. Copy some working examples here





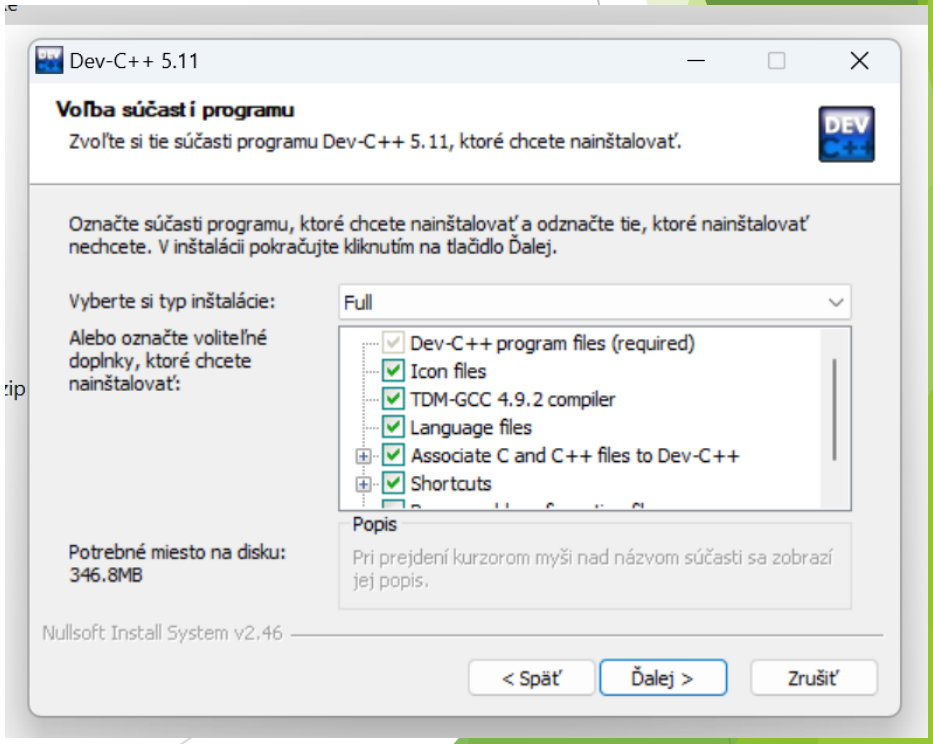
G++ compiler

- I recommend installing DevC++ Orwell update – works on Windows 11
  - G++ is included
- Available at: http://orwelldevcpp.blogspot.com
  - With Additional information

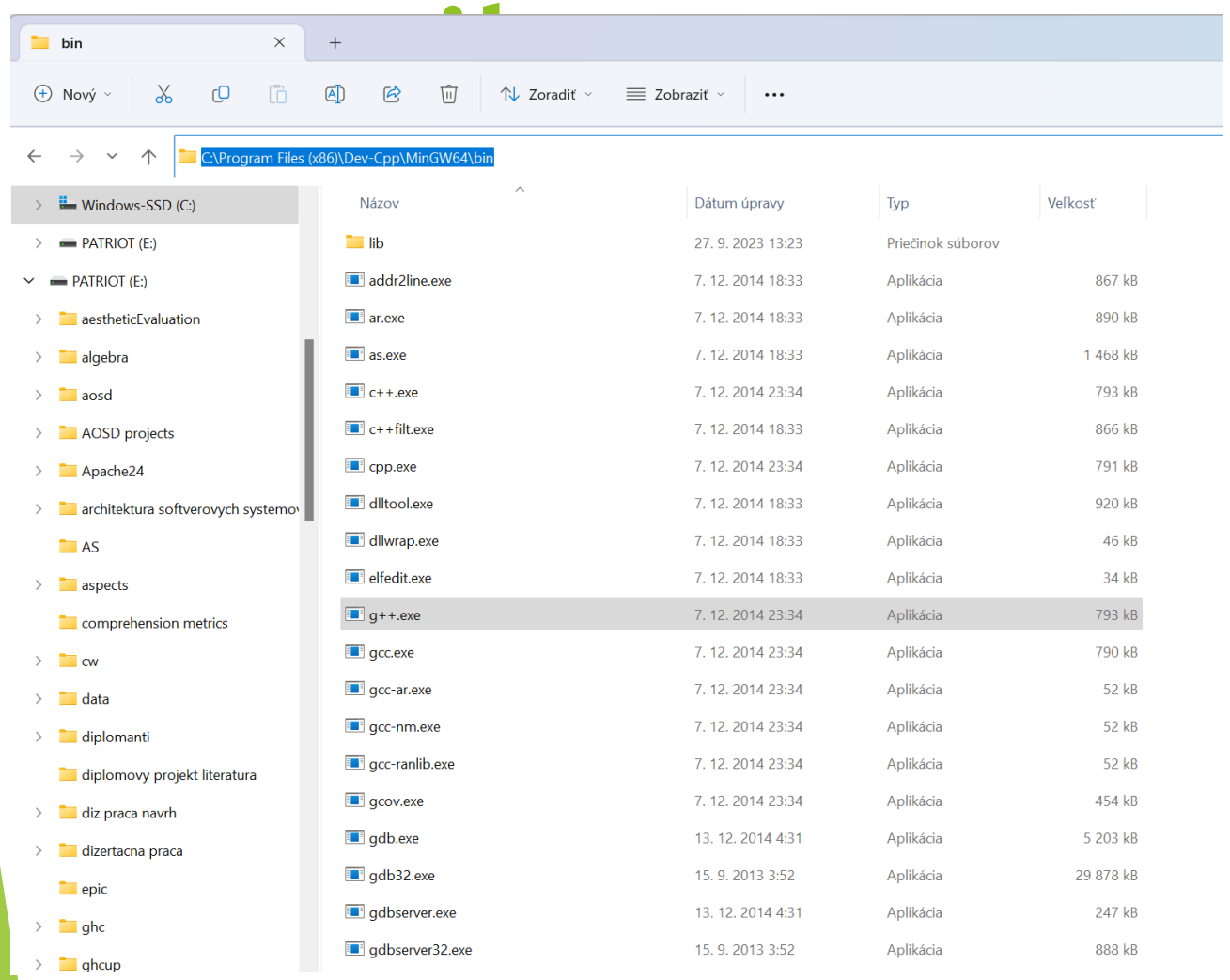
Download and install:

https://sourceforge.net/projects/orwelldevcpp/

Click full installation



Find path to your G++



Optionally you can add it to you path/ system path

C:\Program Files (x86)\Dev-Cpp\MinGW64\bin

Working example

- **Header file:** hello.h
- **C/C++ source code:** main.cc
- **AspectC++ header:** world.ah

```
E:\cw\aspectc++>"ag++.exe" -o result.o hello.h main.cc --Xweaver -a world.ah --c_compiler "C:\Program Files (x86)\Dev-Cp
p\MinGW64\bin\g++.exe" --ac_compiler "ac++.exe"

E:\cw\aspectc++>result.exe
Hello
World

E:\cw\aspectc++>
```

```
#ifndef __HELLO_H__
#define__HELLO_H__
#include <iostream>
void hello(){
	std::cout << "Hello" << std::endl;
}
#endif
```

Header file: hello.h

```
#include "hello.h"
int main(){
	hello(); //print "Hello"
	return 0;
}
```

C/C++ source code

main.cc

```
#ifndef __WORLD_AH__
#define __WORLD_AH__
#include <iostream>
	aspect World {
		advice execution("void hello()") : after() {
			//print "World" after execution of the 'hello()‘ function
			std::cout << "World" << std::endl;
		}
	};
#endif
```

AspectC++ header: world.ah

Compilling and Weaving the program

More information in: https://www.aspectc.org/doc/ag-man.pdf

- "ag++.exe" -o result.exe hello.h main.cc --Xweaver -a world.ah --c_compiler "C:\Program Files (x86)\Dev-Cpp\MinGW64\bin\g++.exe" --ac_compiler "ac++.exe"

```
E:\cw\aspectc++>"ag++.exe" -o result.o hello.h main.cc --Xweaver -a world.ah --c_compiler "C:\Program Files (x86)\Dev-Cp
p\MinGW64\bin\g++.exe" --ac_compiler "ac++.exe"

E:\cw\aspectc++>result.exe
Hello
World

E:\cw\aspectc++>
```

Aspecs have been applied - Successfully done!

Analysis of AspectC utility

Program name

C/C++ source files
– including headers

Resulting executable

- "ag++.exe"
- -o result.exe hello.h main.cc
- --Xweaver -a world.ah
- --c_compiler "C:\Program Files (x86)\Dev-Cpp\MinGW64\bin\g++.exe"
- --ac_compiler "ac++.exe"

AspectC headers

Path to g++
compiler

Path to AspectC compiler

Related literature

Also for embedded systems,...

- https://www.aspectc.org/Publications.php
- https://www.aspectc.org/doc/aosd-2005-demo.pdf

- Search also for:
  - software product lines
  - conditional compilation
  - weaving
  - Aspects, aspect-oriented tools

Decorators in Python

application to asynchronous functions

```
cosmos_db = CosmosDb("")
SECS_DISABLE_CACHED = 5

async def main(request: func.HttpRequest) -> func.HttpResponse:
    data = request.route_params.get('data')
    data_values = await cosmos_db.get(data)

    if data == 'first':
        result = await do_something(data, data_values, ['aaa', 'bbb', 'ccc'])
    elif report == 'second':
        result = await do_something(data, data_values, ['11', '22', '33'])
    else:
        result = await do_something(data, values, ['x', 'd'])
    return func.HttpResponse(result)
```

Decorator applied on async function: Use in FAAS

Azure functions

Package in Python: azure-functions

```
def use_cache_decorator(function_to_do):
    async def use_cache(report, data_values, value_array):
        if data_values is not None and data_values['time'] + SECS_DISABLE_CACHED > time.time():
            return data_values['value']
        else:
            result_dict = dict()
            result_dict['id'] = report
            result_dict['value'] = await function_to_do(value_array)
            result_dict['time'] = time.time()
            response = await cosmos_db.upsert(result_dict)
            return response['value']
    return use_cache
```

```
@use_cache_decorator
async def do_something(item_ids: 'list[str]') -> str:
    items = await MyAssets.get(item_ids)
    result = calculate(items)
    return f'result for [{", ".join(item_ids)}] = {result}'
```

References

Capabilities, Basics, Advanced topics and Application of AspectJ: The AspectJ in Action *Laddad, Ramnivas, 2003. AspectJ in action: practical aspect-oriented programming. Greenwich, CT: Manning. ISBN 978-1-930110-93-9.*

Dynamic Aspect-Oriented Programming in Java: The HotWave Experience: https://link.springer.com/chapter/10.1007/978-3-642-35551-6_3

CARDOSO, João M.P., Tiago CARVALHO, José G.F. COUTINHO, Wayne LUK, Ricardo NOBRE, Pedro DINIZ a Zlatko PETROV, 2012. LARA: an aspect-oriented programming language for embedded systems. V: Proceedings of the 11th annual international conference on Aspect-oriented Software Development - AOSD '12 [online]. Potsdam, Germany: ACM Press, s. 179. ISBN 978-1-4503- 1092-5. Dostupné na: doi:10.1145/2162049.2162071

RASHID, Awais, Ana MOREIRA a João ARAÚJO, 2003. Modularisation and composition of aspectual requirements. V: the 2nd international conference: Proceedings of the 2nd international conference on Aspect-oriented software development - AOSD '03 [online]. Boston, Massachusetts: ACM Press, s. 11–20. ISBN 978-1-58113-660-9. Dostupné na: doi:10.1145/643603.643605

SÁ LOUREIRO FERREIRA DA SILVA, Ricardo. Aspect-Oriented Programming for Javascript using the Lara Language. Porto. Dissertation thesis. Universidade do Porto, 2019

WASHIZAKI, Hironori, Yoichi NAGAI, Rieko YAMAMOTO, Atsuto KUBO, Tomohiko MIZUMACHI, Kazuki EGUCHI, Yoshiaki FUKAZAWA, Nobukazu YOSHIOKA, Hideyuki KANUKA, Toshihiro KODAKA a Nobuhide SUGIMOTO, 2009. AOJS: aspect-oriented javascript programming framework for web development. V: Proceedings of the 8th workshop on Aspects, components, and patterns for infrastructure software - ACP4IS '09 [online]. Charlottesville, Virginia, USA: ACM Press, s. 31. ISBN 978-1-60558-450-8. Dostupné na: doi:10.1145/1509276.1509285

TOLEDO, Rodolfo, Paul LEGER a Éric TANTER, 2010. AspectScript: expressive aspects for the web. V: Proceedings of the Eighth International Conference on AspectOriented Software Development - AOSD '10 [online]. Rennes and Saint-Malo, France: ACM Press, s. 13 [cit. 18.5.2022]. ISBN 978-1-60558-958-9. Dostupné na: doi:10.1145/1739230.1739233

SOARES, Sérgio, Paulo BORBA a Eduardo LAUREANO, Distribution and persistence as aspects. Software: Practice and Experience [online]. 2006, roč. 36, č. 7, s. 711–759. ISSN 0038-0644, 1097-024X. Dostupné na: doi:10.1002/spe.715

SOARES, Sergio, Eduardo LAUREANO a Paulo BORBA. Implementing distribution and persistence aspects with aspectJ. ACM SIGPLAN Notices [online]. 2002, roč. 37, č. 11, s. 174–190. ISSN 0362-1340, 1558-1160. Dostupné na: doi:10.1145/583854.582437

HUANG, Wenhao, Chengwan HE a Zheng LI. A Comparison of Implementations for Aspect-Oriented JavaScript. 2015, Dostupné na: doi:10.2991/csic-15.2015.9

Flanagan, D.: canto-js: An improved api for the html canvas tag (2010)

Aspect-oriended recreation of design patterns, application of patterns: *R. Miles, AspectJ cookbook, 1st ed. Sebastopol, CA; Farnham: O'Reilly Media, 2004.*

Q1. Which approach is best appropriate with my existing or new application?

Q2. Which AOP approach is most suitable for implementation?

Q3. How fast will it merge with my application?

Q4. What is the performance elevated?

Taken from: *KUMAR, Ravi a Munishwar RAI, 2019. A Comparative Study of AOP Approaches: AspectJ, Spring AOP, JBoss AOP. 2019*