aspectj

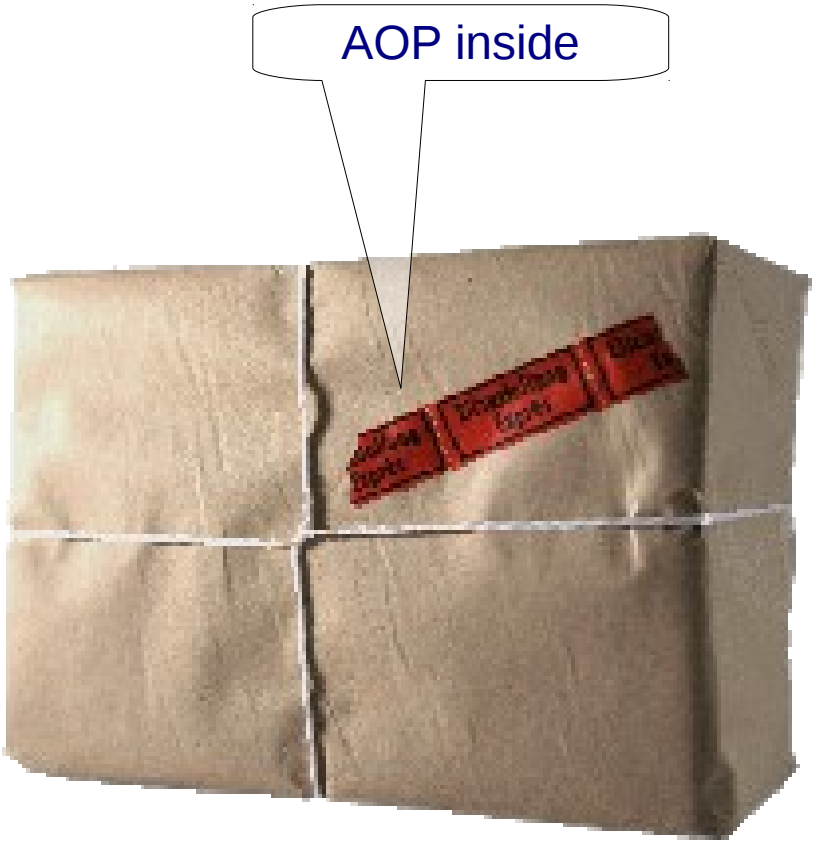
crosscutting objects for better modularity

Margrit Höhme
Software-Entwicklerin Senior
java@mhoehme.de

1. Aspektorientierte Programmierung (AOP)
2. AspectJ-Sprachfeatures
3. Beispiele zum Ausprobieren

Was ist *AOP*?


Java

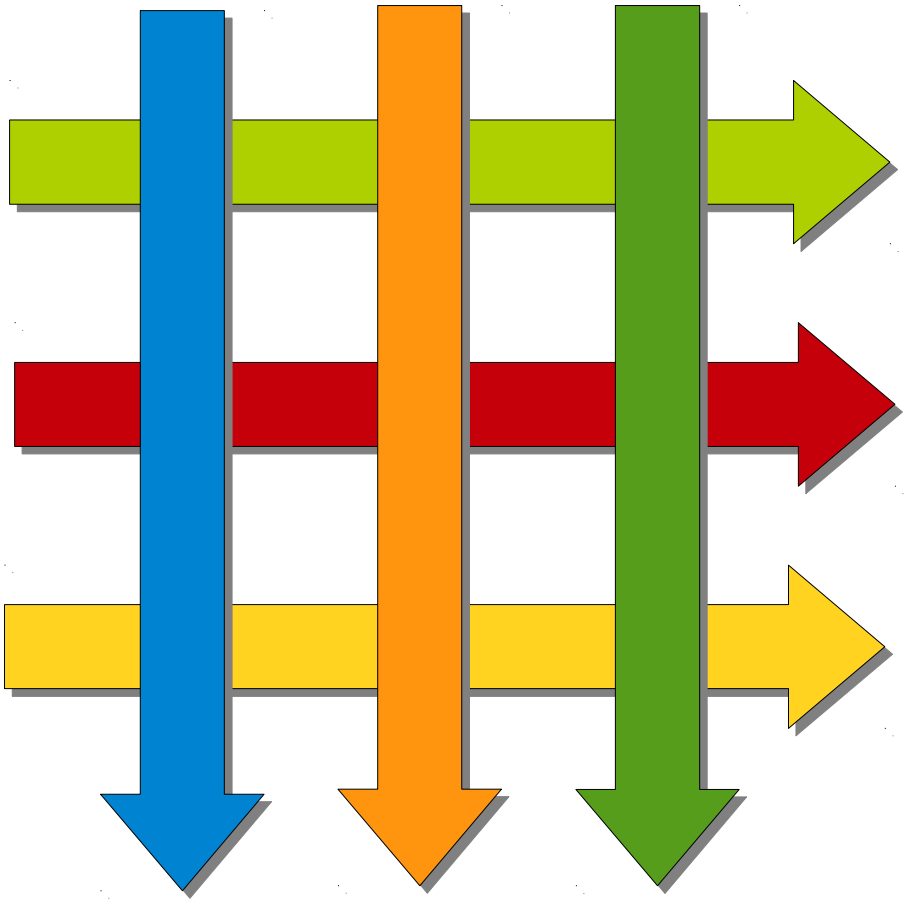


JUGCologne

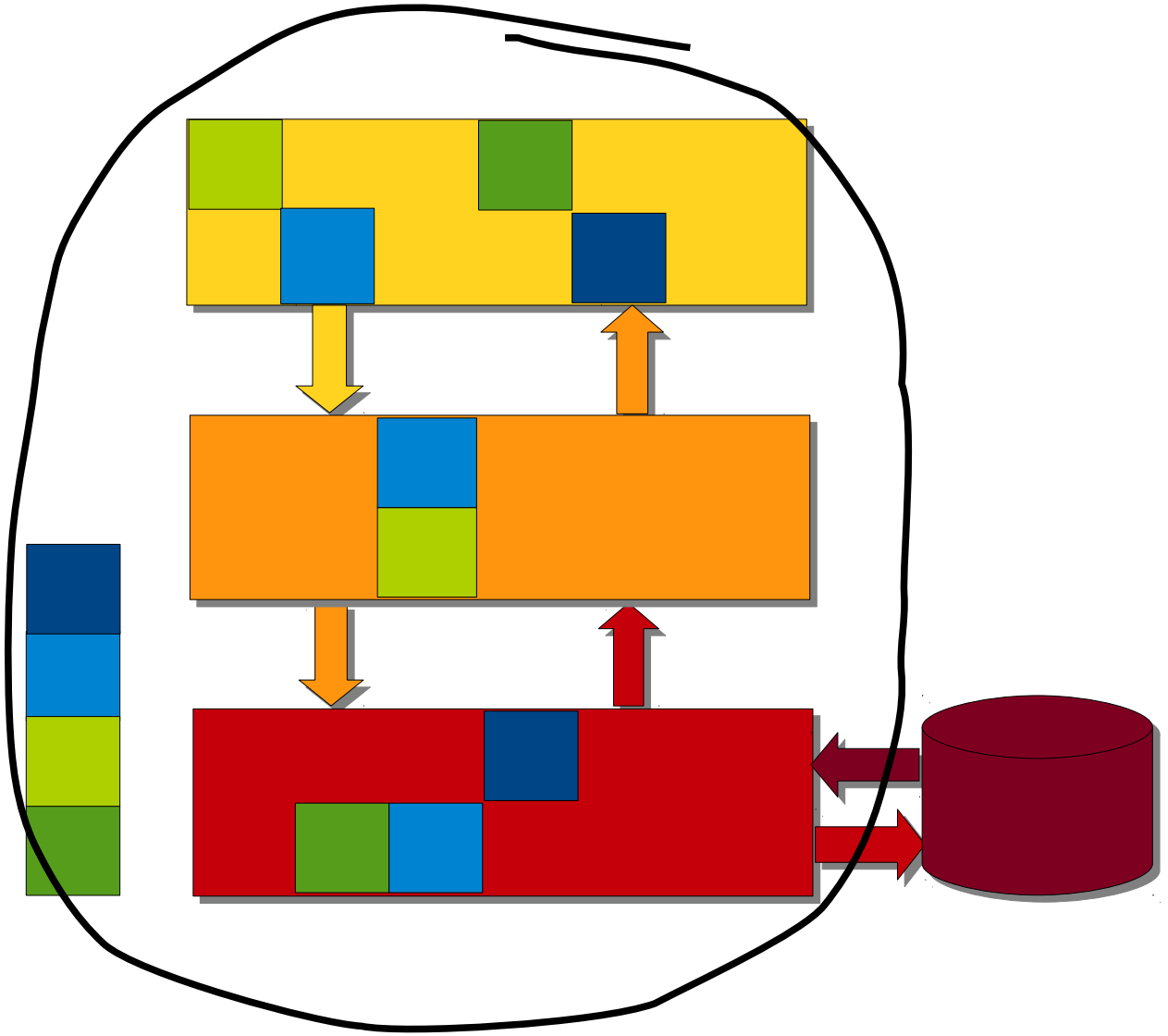
```
package de.mycompany.services;

public class MyService implements Service {

   public void doService(DTO dto) {
      Result r = compute(dto);
      dto.setResult(r);
   }
}
```

```
package de.mycompany.services;

import org.apache.commons.logging.*;

public class MyService implements Service {
  private Log log =
    LogFactory.getLog(MyService.class);

  public void doService(DTO dto) {
    try {
      if (log.isDebug()) {
        log.debug("MyService Enter, INPUT="+dto);
      }

      Result r = compute(dto);
      dto.setResult(r);

      if (log.isDebug()) {
        log.debug("MyService Exit, RESULT="+r);
      }
    } catch (Throwable t) {
      log.error(t.getMessage());
      dto.addErrorCode(t.getMessage());
    }
  }
}
```

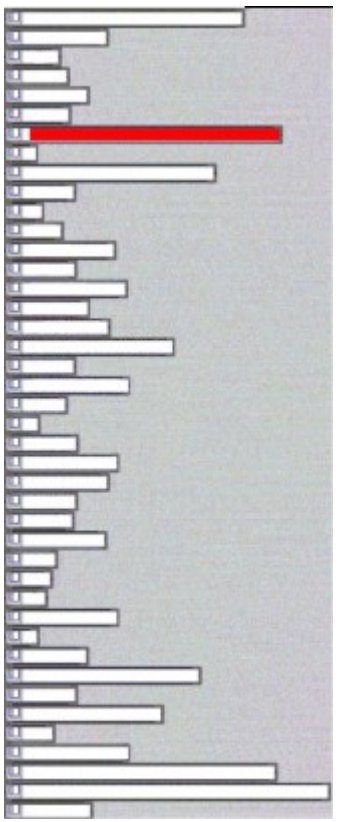
Cross-Cutting Concerns



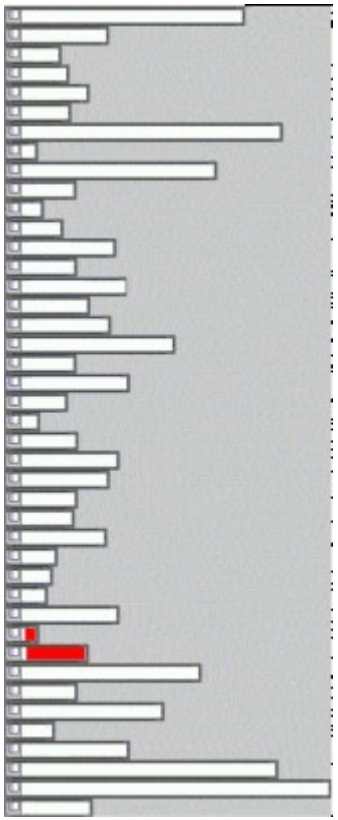
JUGCologne



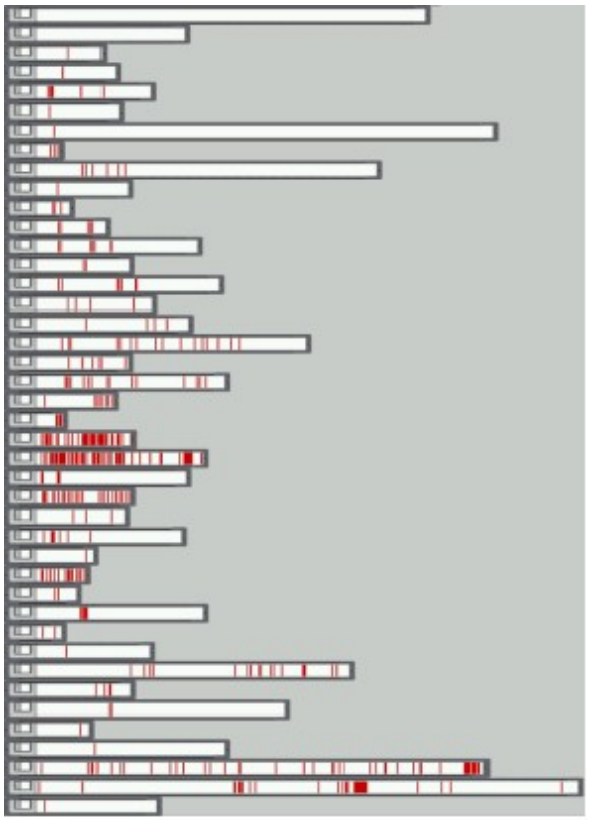
JUGCologne



XML-Parsing



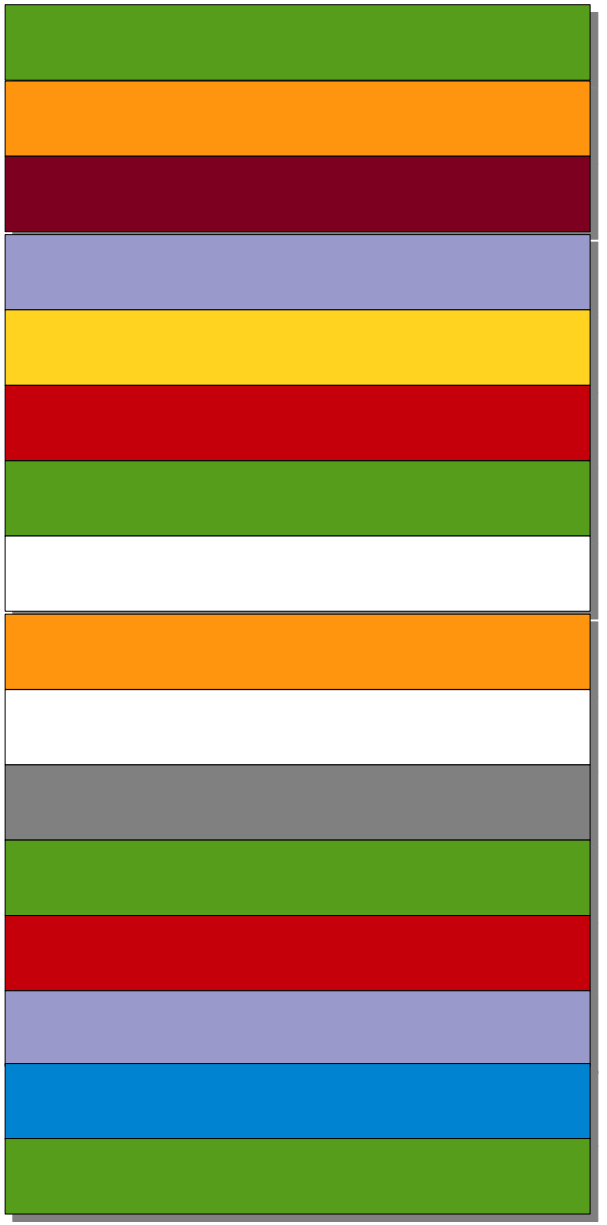
URL-Erkennung



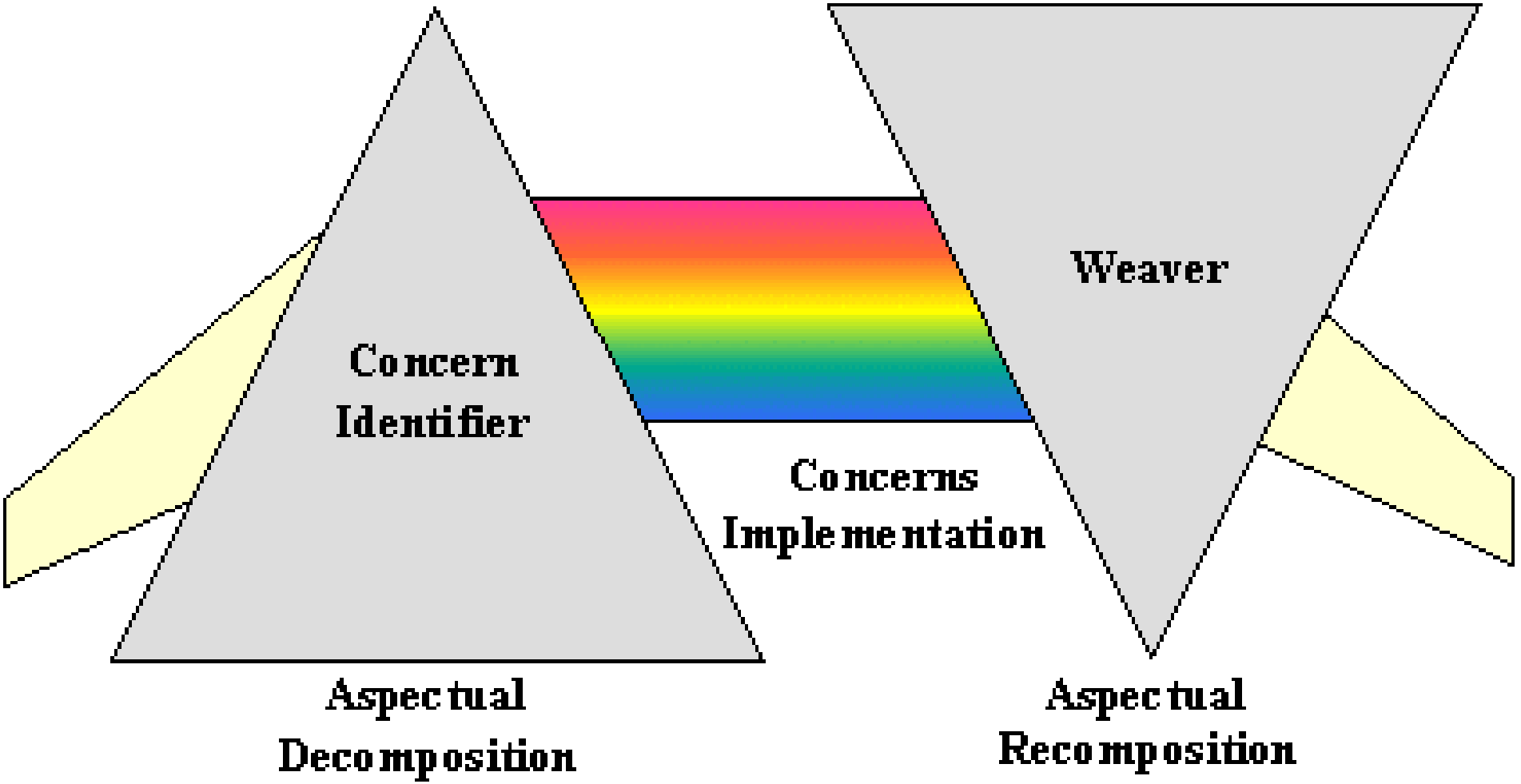
Logging

JUGCologne

```
public class MyService implements Service {
  public void doService(DTO dto) {

    // Debug-Logging("enter")

    // Berechtigungsprüfung

    // Validierung

    // Caching

    // Start Transaktion

    Result r = compute(dto);

    // Ergebnis-Persistierung

    // Ausnahmebehandlung

    // Performance-Logging

    // Commit/Rollback Transaktion

    dto.setResult(r);

    // Debug-Logging("exit")
  }
}
```



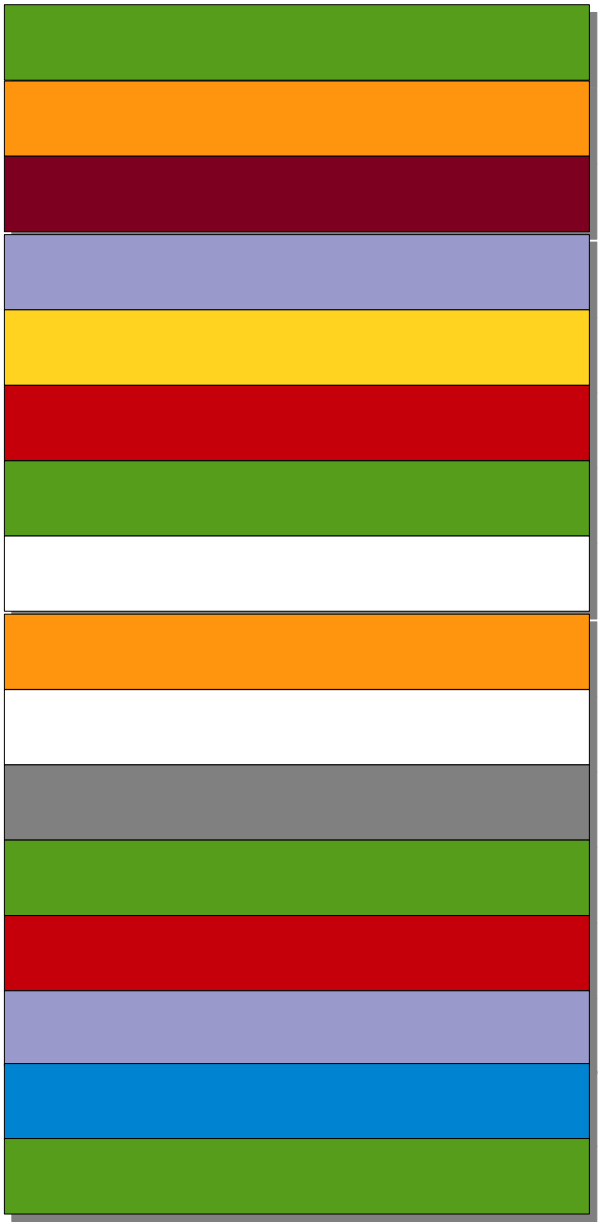
JUGCologne



Ramnivas Laddad
http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html

```
public class MyService implements Service {
  public void doService(DTO dto) {
    try {
      Result r = compute(dto);
      dto.setResult(r);
    } catch(ServiceException ex) {}
  }
}
```

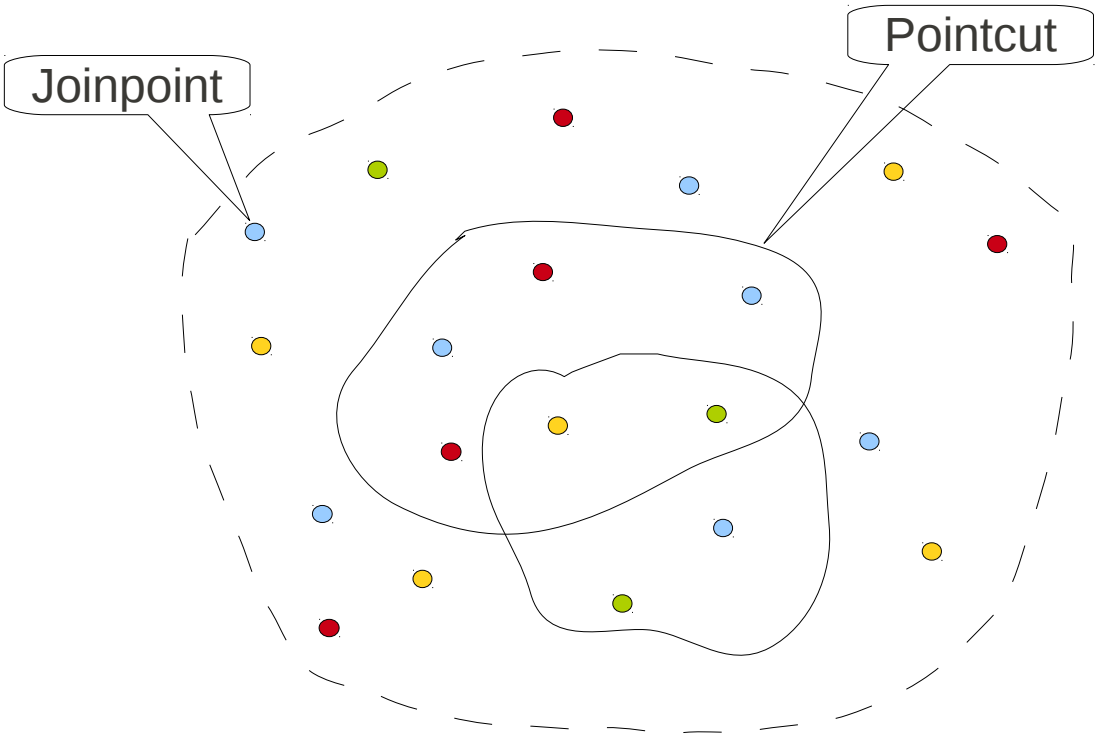
+ DebugLogger.aj
+ Validierung.aj
+ Caching.aj
+ Berechtigungspruefer.aj
+ TransactionHandler.aj
+ PersistenceAspect.aj
+ AusnahmeHandler.aj
+ PerformanceLogger.aj



JUGCologne

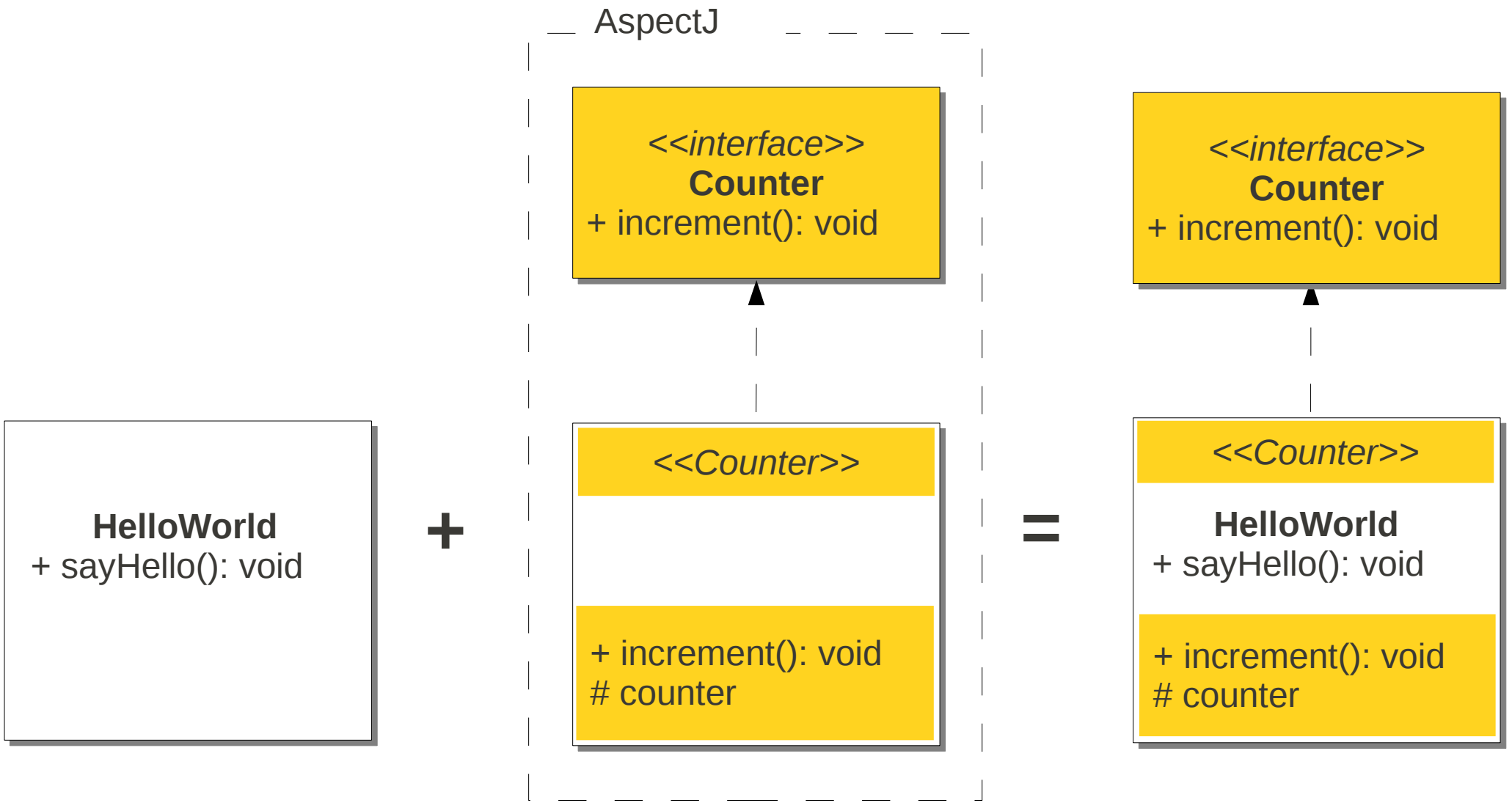
AspectJ Sprachfeatures

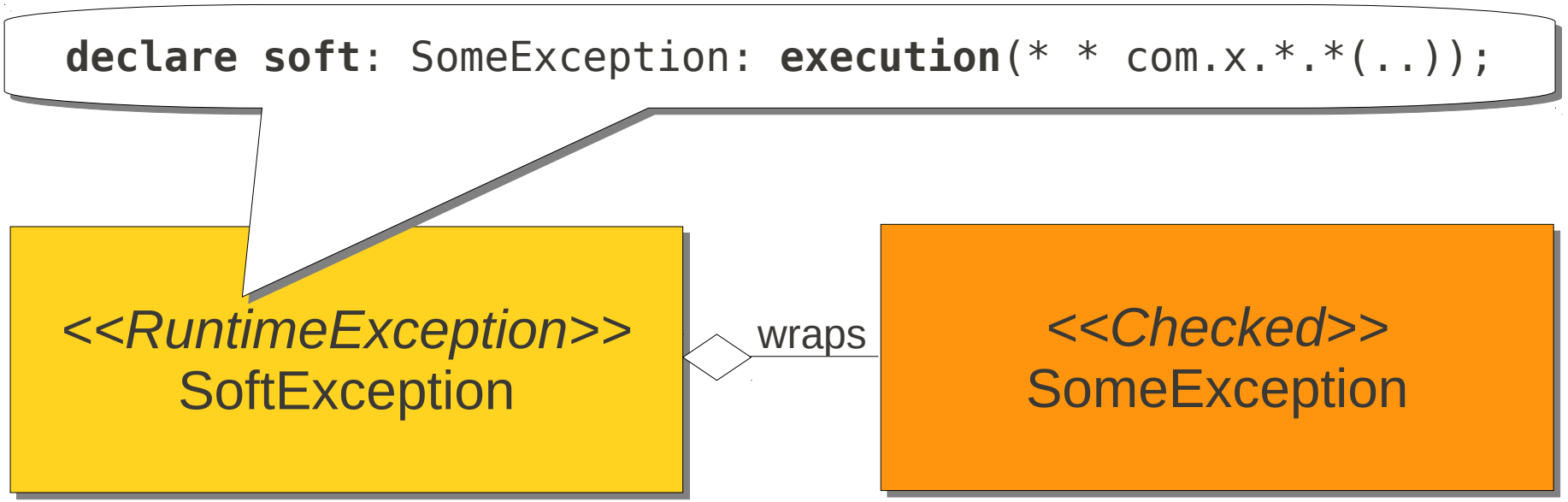
Dynamische Konzepte



Aspekt = Pointcut + Advice

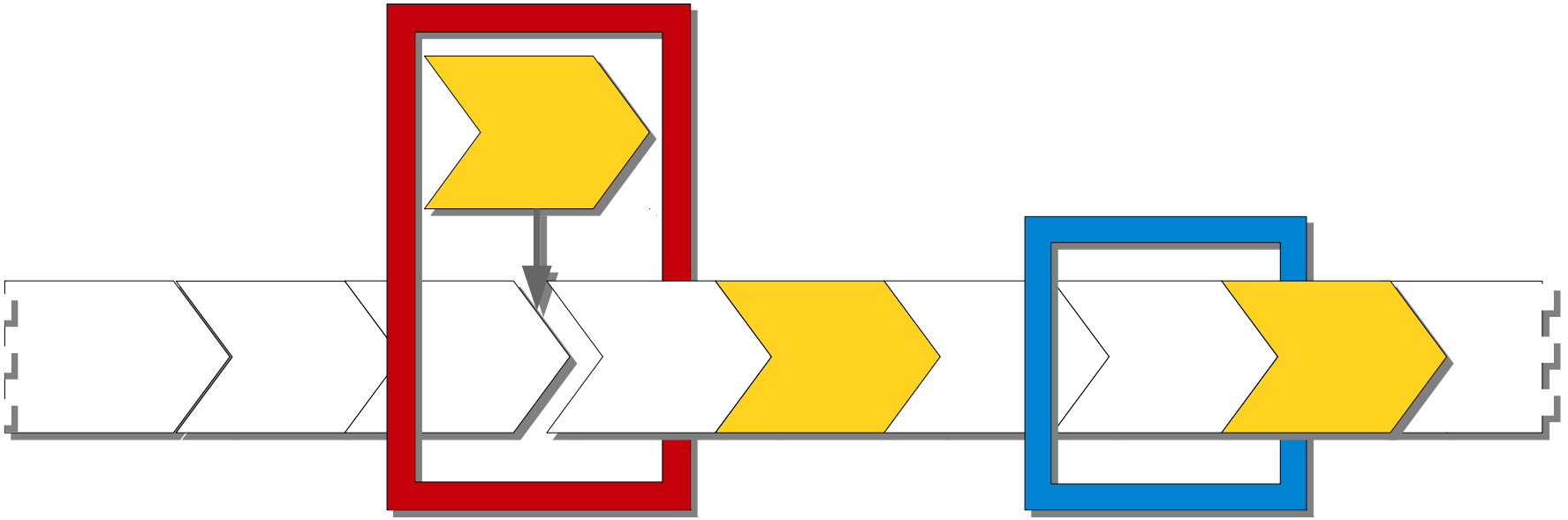
JUGCologne


AspectJ
<<interface>>
Counter
+ increment(): void
HelloWorld
+ sayHello(): void
+
<<Counter>>
+ increment(): void
# counter
=
<<interface>>
Counter
+ increment(): void
<<Counter>>
HelloWorld
+ sayHello(): void
+ increment(): void
# counter


declare soft: SomeException: execution(* * com.x.*.*(..));
<<RuntimeException>>
SoftException
wraps
<<Checked>>
SomeException

Compiler-Warnings und -Errors

```
declare error  : call(* java.sql.*.*(..)): "Use DB Adapter"
declare warning: callBackend() && !within(trusted):
                 "Don't use untrusted code here"
```



- Compile-time
- Post-compile
- Load-time
- **AspectJ:** kein Run-time weaving

Gregor Kiczales *(90er Jahre)*

Adrian Colyer, Ramnivas Laddad, Andy Clement

Alec Vasseur

Mik Kersten, Ron Bodkin

AspectJ-Hoster

Kabir Khan, Bill Bourke, Marc Fleury (*JBoss-AOP)*

Guice AOP (AOP Alliance)

AOSD.11: März 2011 Porto de Galinhas, Brasilien
http://www.visiteportodegalinhas.com, http://www.aosd.net/2011

Hier & heute

Funktionsweise von AspectJ

```
$ $ASPECTJ_HOME/bin/ajc \
    -cp $ASPECTJ_HOME/lib/aspectjrt.jar \
    <Klassen und Aspekte>

$ java -cp $ASPECTJ_HOME/lib/aspectjrt.jar:. Main
```

aspectjrt.jar in Classpath aufnehmen!

- Taskdef
  - ajc-Task

- Maven-aspectj-plugin
  - On-the-fly
  - Permanent

Gradle
a better way to build

- Plugin

JUGCologne

```
//de/mycompany/aop/PerformanceLog.aj
package de.mycompany.aop;
import org.aspectj.lang.JoinPoint;

public aspect PerformanceLog implements Log {
  pointcut p():
    execution(* HelloWorld.*(..)) ||
    execution(HelloWorld.new(..));

  Object around(): p() {
    long t = System.currentTimeMillis();
    try {
      return proceed();
    } finally {
      t = System.currentTimeMillis()-t;
      log(thisJoinPoint, t);
    }
  }

  private void log(JoinPoint jp, long ms) {
    System.out.println(jp + " " + t + " ms");
  }
}
```

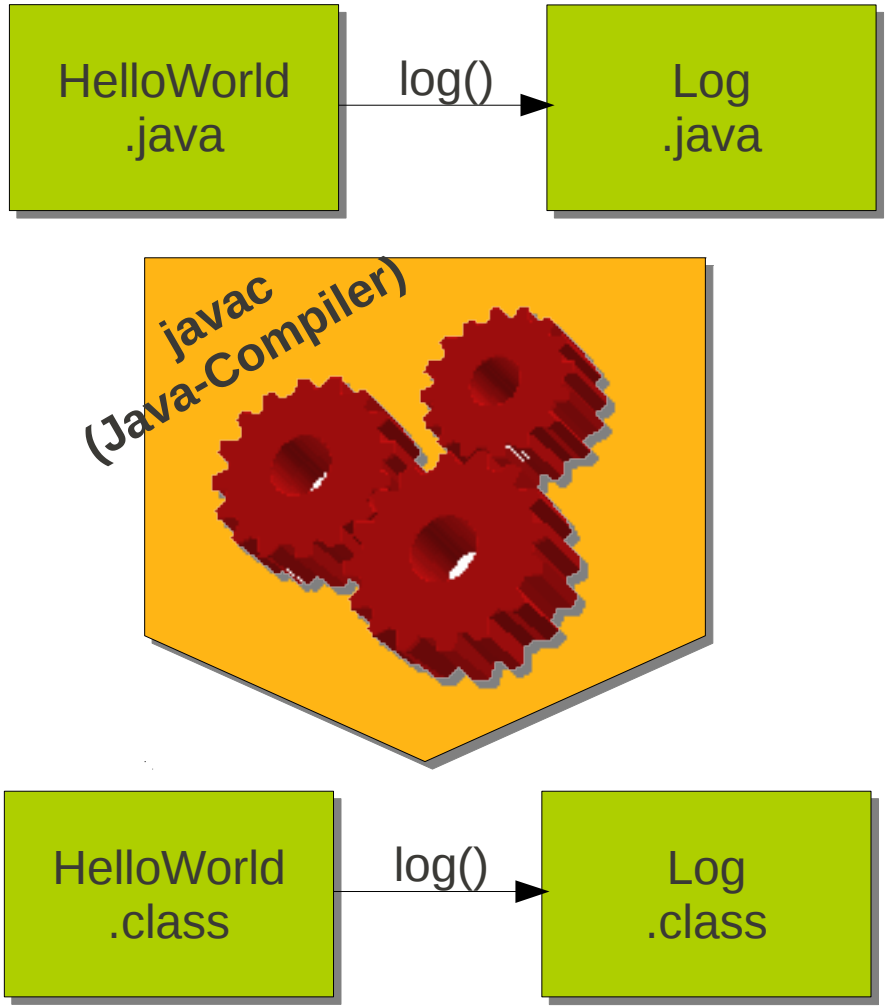
```java
package de.mycompany.aop;

public class HelloWorld {
  private String name;
  public HelloWorld(String name) {
    this.name = name;
  }
  public String getName() {
    return name;
  }
  public void sayHello() {
    System.out.println("Hello, "+
      getName()+"!");
  }
}
```

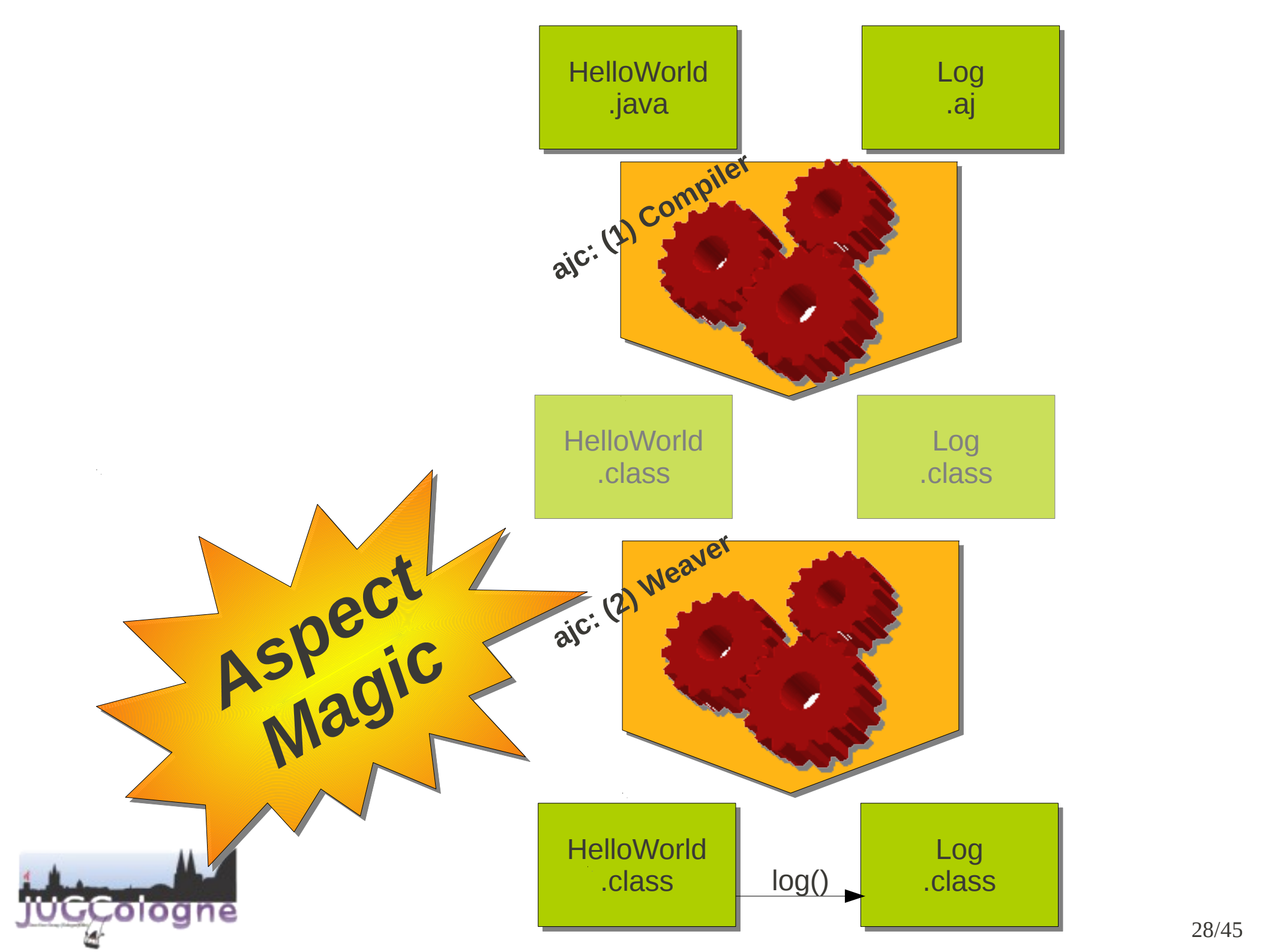
```java
package de.mycompany.aop;

public aspect PerformanceLog implements Log {
  pointcut p():
    execution(* HelloWorld.*(..)) ||
    execution(HelloWorld.new(..));
  public void log(String msg) {
    System.out.println(msg);
  }
  Object around(): p() {
    long t = System.currentTimeMillis();
    try {
      return proceed();
    } finally {
      t = System.currentTimeMillis() - t;
      log(thisJoinPoint + " " + t + " ms");
    }
  }
}
```

HelloWorld.java + PerformanceAspect.aj =

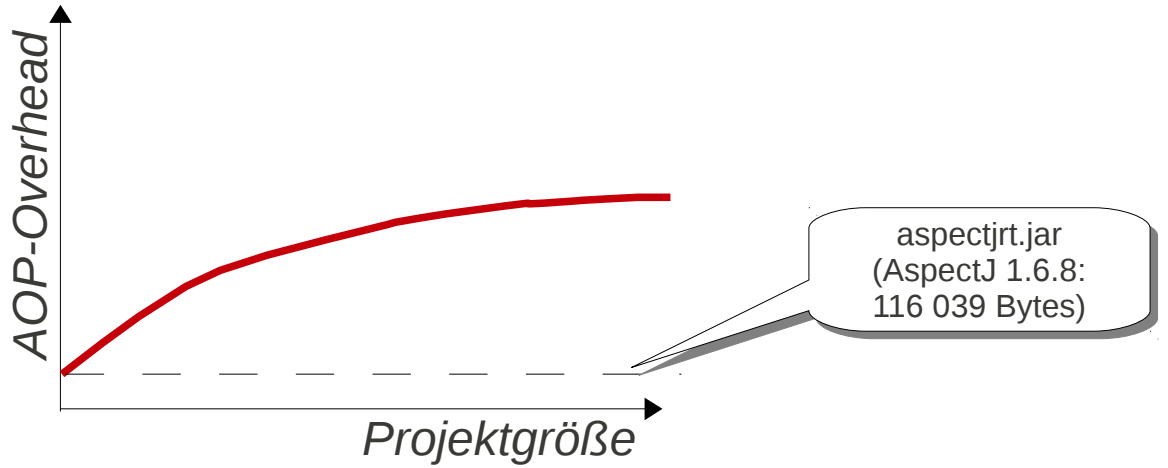
```
execution(de.mycompany.aop.HelloWorld(String)) 1 ms
execution(String de.mycompany.aop.HelloWorld.getName()) 1 ms
Hello, Test!
execution(void de.mycompany.aop.HelloWorld.sayHello()) 2 ms
```

JUGCologne


HelloWorld
.java
log()
Log
.java
javac
(Java-Compiler)
HelloWorld
.class
log()
Log
.class


HelloWorld
.java
Log
.aj
ajc: (1) Compiler
HelloWorld
.class
Log
.class
Aspect
Magic
ajc: (2) Weaver
HelloWorld
.class
log()
Log
.class

- Größe des Bytecodes
  - Runtime-Lib
  - Generischer Code durch Weaving
- Performance
  - Compiletime fast wie handgestrickt
  - Loadtime vernachlässigbar



JUGCologne

Joinpoints

- call, execution
- handler
- within
- withincode
- initialization
  - preinitialization
  - staticinitialization
- get, set
- cflow, cflowbelow
- if
- this
- target
- args

Wildcards

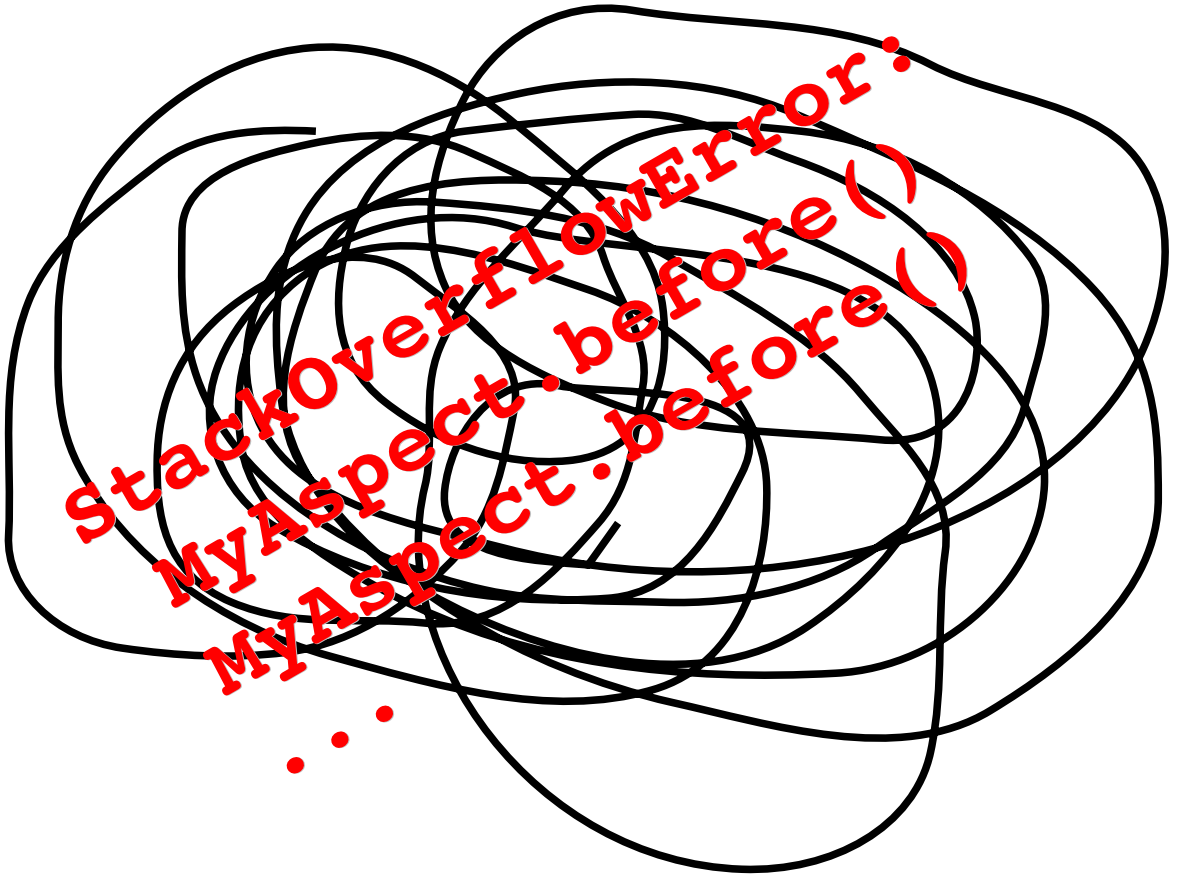
- *, + und ..

Verknüpfungen

- &&, ||, !

JUGCologne

```
public aspect MyAspect {
   before(): call(* *(..)) {
      // ... wünsch dir was
   }
}
```

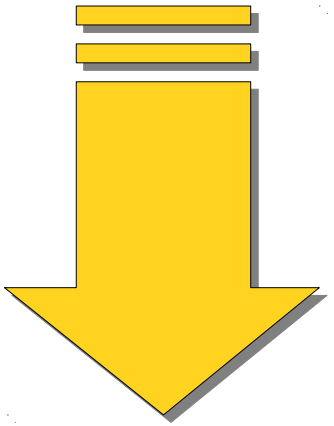


JUGCologne

```
1)  call(public void de.mycompany.aop.HelloWorld.sayHello())
2)  call(void de.mycompany.aop.*.sayHello(..))
3)  call(* Hello*+.say*(..))
4)  call(* *(..) throws !IOException)
5)  call(* * de.mycompany.*.*.*(*))
6)  call(* de.mycompany..*(..))
7)  call(public int *(int))
8)  call(* *Hello(..))
9)  set(static int A.x)
10) within(de.mycompany..*)
11) cflow(call(* sayHello())) && !within(MyAspect)
12) handler(IOException)
```

JUGCologne

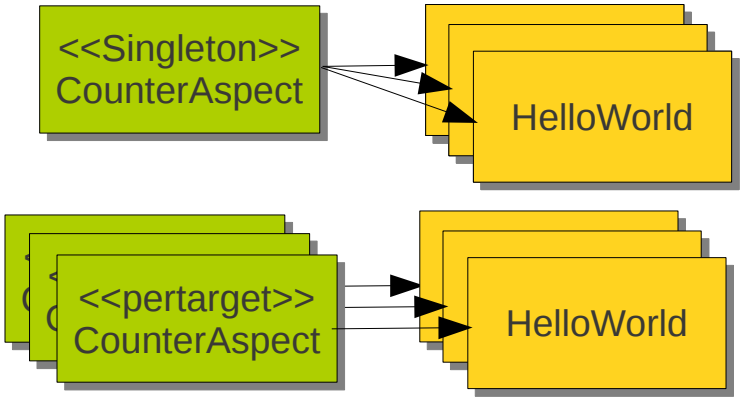
```
public class MyService extends Service {
   @Override
   public void set(int x, int y) {
      ...
   }
}
```



- thisJoinPoint
- this
- target
- args

```
after(Service s, int x, int y) returning:
        call(void Service+.set(int, int))
        && target(s)
        && args(x, y) {
   log(thisJoinPoint+" "+s+"=("+x+"/"+y+")");
}
```

- Vorrangregeln bei mehreren Advices
- Zugriff auf private members: privileged
- Annotations
- Generics
- Singleton vs. pertarget/perthis



3 kurze Beispiele zum Ausprobieren

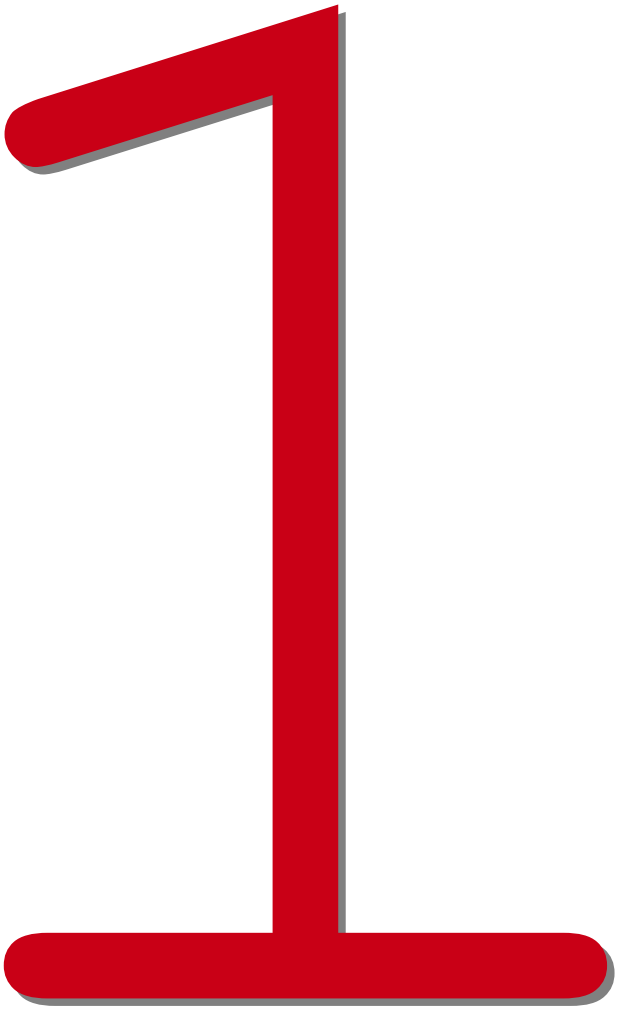
JUGCologne

```
String around(): call(String get*()) {
   return "Dummy";
}
```

JUGCologne

```
--> call(void de.mycompany.aop.HelloWorld.setName(String))
  --> execution(void de.mycompany.aop.HelloWorld.setName(String))
    --> set(String de.mycompany.aop.HelloWorld.name)
    <-- set(String de.mycompany.aop.HelloWorld.name)
  <-- execution(void de.mycompany.aop.HelloWorld.setName(String))
<-- call(void de.mycompany.aop.HelloWorld.setName(String))
--> call(void de.mycompany.aop.HelloWorld.sayHello())
  --> execution(void de.mycompany.aop.HelloWorld.sayHello())
    --> get(PrintStream java.lang.System.out)
    <-- get(PrintStream java.lang.System.out)
    --> call(java.lang.StringBuffer(String))
    <-- call(java.lang.StringBuffer(String))
    --> call(String de.mycompany.aop.HelloWorld.getName())
      --> execution(String de.mycompany.aop.HelloWorld.getName())
        --> get(String de.mycompany.aop.HelloWorld.name)
        <-- get(String de.mycompany.aop.HelloWorld.name)
      <-- execution(String de.mycompany.aop.HelloWorld.getName())
    <-- call(String de.mycompany.aop.HelloWorld.getName())
    --> call(StringBuffer java.lang.StringBuffer.append(String))
    <-- call(StringBuffer java.lang.StringBuffer.append(String))
    --> call(StringBuffer java.lang.StringBuffer.append(String))
    <-- call(StringBuffer java.lang.StringBuffer.append(String))
    --> call(String java.lang.StringBuffer.toString())
    <-- call(String java.lang.StringBuffer.toString())
    --> call(void java.io.PrintStream.println(String))
Hello, Test!
    <-- call(void java.io.PrintStream.println(String))
  <-- execution(void de.mycompany.aop.HelloWorld.sayHello())
<-- call(void de.mycompany.aop.HelloWorld.sayHello())
```

```
public ascpect Tracer {

  final static String B="------------------------------------";

  private int cnt=0;

  private void log(String s) {
    System.out.println (s);
  }

  pointcut p(): cflow(call(* *(..))) && !within(Tracer);

  before(): p() {
    log(B.substring(0,cnt++*2)+"--> "+thisJoinPoint);
  }

  after() returning: p() {
    log(B.substring(0,--cnt*2)+"<-- "+thisJoinPoint);
  }
}
```



```
import org.aspectj.lang.JoinPoint;
public aspect PerformanceLogger {
  private Log logger = LogFactory.getLogger();

  private void log(JoinPoint, long ms) {
     logger.log(thisJoinPoint+" "+ms+" ms");
  }

  Object around(): execution(* doService(..)) {
    long t = System.currentTimeMillis();
    try {
      return proceed();
    } finally {
      log(thisJoinPoint,
          System.currentTimeMillis()-t);
    }
  }
}
```

- Lernkurve
- Standard, aber kein Sprachstandard
- Tool für Entwicklung und Fehlersuche
- Projektrisiken
- Langlebigkeit von AOP-Lösungen
- Design-Entscheidung

→ **Persönliche Entscheidung**

JUGCologne

Vielen Dank.

<java@mhoehme.de>