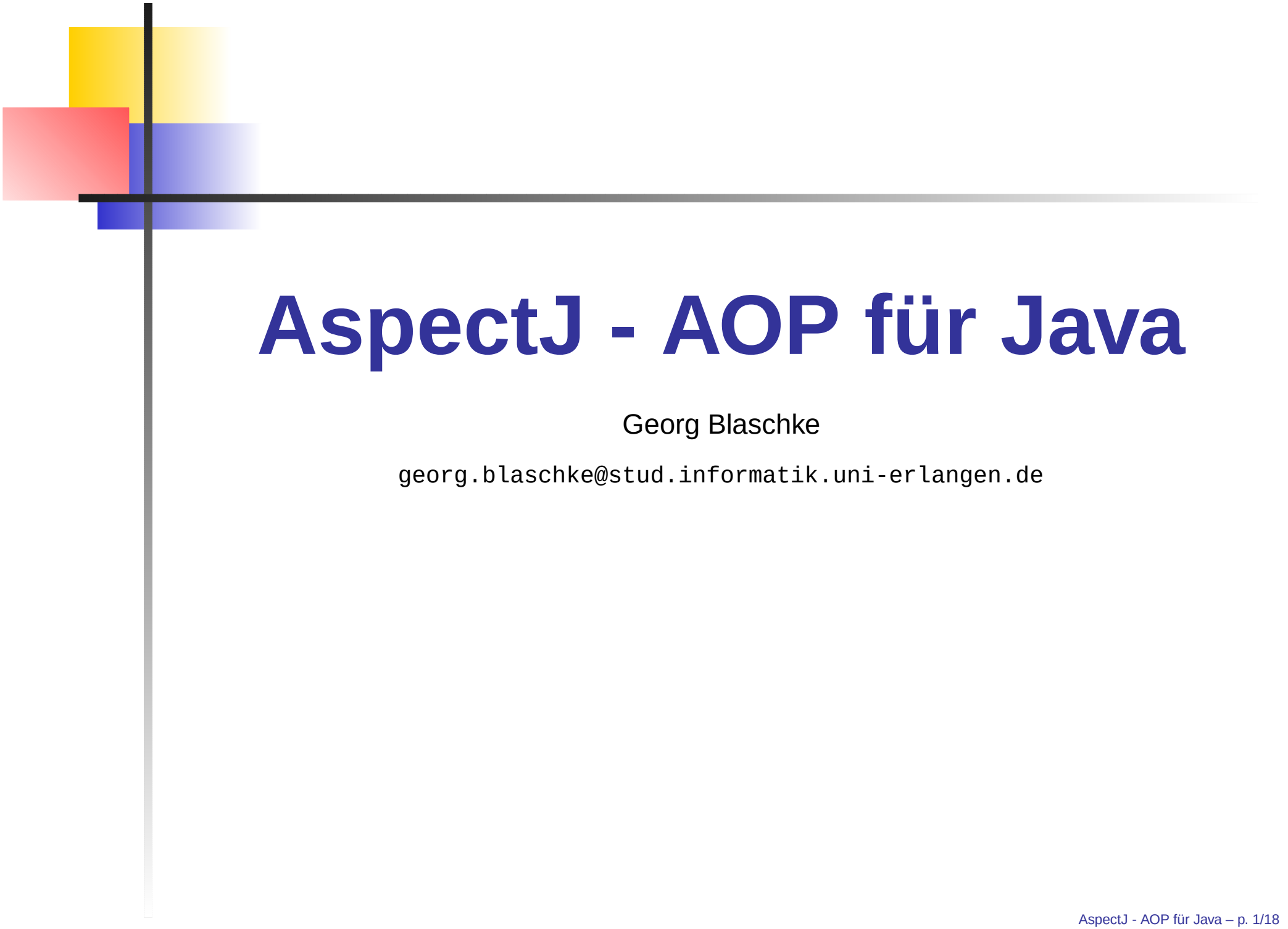
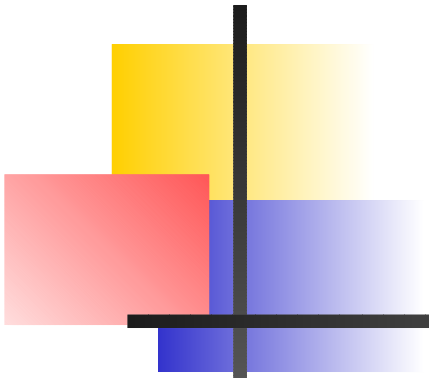


AspectJ - AOP für Java

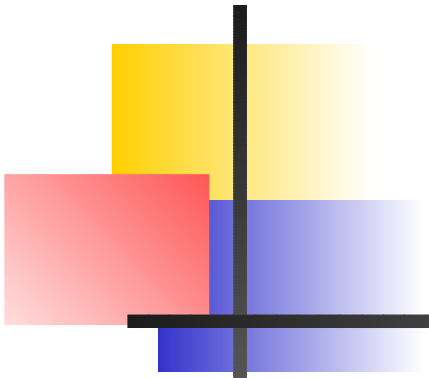
Georg Blaschke

georg.blaschke@stud.informatik.uni-erlangen.de



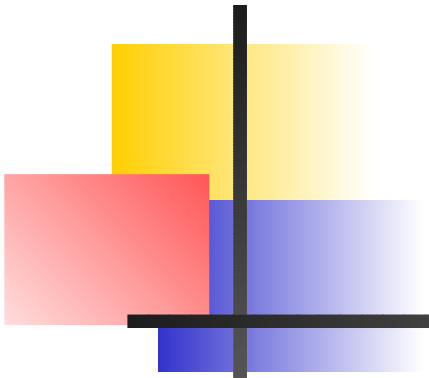
Motivation

- Eigenschaften, die sich durch viele Komponenten eines Systems hindurchziehen
- Modularisierung mit Klassen allein nicht immer ausreichend
- Bessere Lösung: AOP
- Erweiterung von Java zu AspectJ



Übersicht

- zusätzliche Sprachkonstrukte von AspectJ
  - Pointcut
  - Advice
  - Introduction
  - Aspect
- praktischer Einsatz von AspectJ
  - Compiler und Tools
  - Software-Entwicklung
  - Software Produktion
- kurze Demo



Pointcut

- identifiziert einen oder auch mehrere Joint Points
- Beispiel:

```
execution(void ChatClient.send(String))
```

primitive Pointcuts

- execution()
- call()
- args()
- target()
- und viele mehr

nicht-primitive Pointcuts

- sind aus Pointcuts zusammengesetzt
  - ||
  - &&
  - !
- Beispiel:

```
call (void  ChatClient.send(String ))||
call (void  ChatServer.send(String))
```

Deklaration von Pointcuts

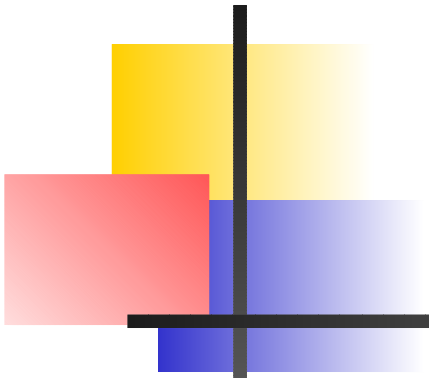
- Schlüsselwort pointcut
- Beispiel:

```
pointcut  send():
 call (void  ChatClient.send(String ))||
 call (void  ChatServer.send(String))
```

Pointcuts und Wildcards

- Form von „eigenschafts-basierten“ Pointcuts
- identifiziert mehrere Joint Point gleichzeitig
- Beispiel:

```
execution(* ChatClient .* (..))
```



Advice

- Before Advice
  - Ausführung vor einem Pointcut
  - before ():
- After Advice
  - Ausführung nach einem Pointcut
  - after ():
- Around Advice
  - Ausführung anstelle eines Pointcuts
  - around():

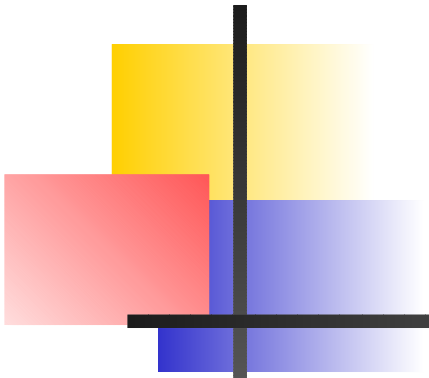
Advice Beispiel

```
pointcut myPointcut(): execution(* ChatClient .* (..));

before(): myPointcut(){
   System.out.println("Eine Methode der Klasse ChatClient
      wird gleich ausgefuehrt");
}

after(): myPointcut(){
   System.out.println("Eine Methode der Klasse ChatClient
      wurde ausgefuehrt");
}
```

Around Advice

- ersetzt identifizierten Code
- braucht Rückgabewert
- Beispiel:

```
void around(String string ):
    execution(void ChatClient.send(String)&&args(string)
{
    System.out.println("Die Zeichenkette: "+string+"
       soll gesendet werden.");
}
```



Introduction

- statische Veränderung von Datentypen
  - Beziehungen zwischen Datentypen
  - zusätzliche Variablen
  - zusätzliche Methoden
- veränderte Typen werden vererbt !

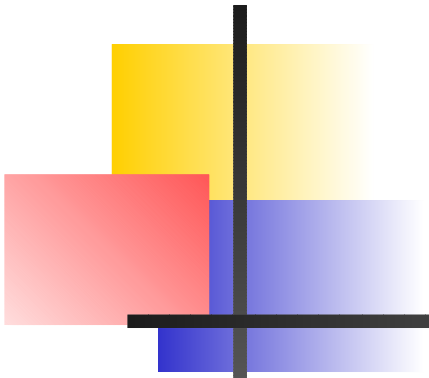
Introduction Beispiele

```
declare parents: ChatClient implements Observer;

private  Subject ChatClient.chatServer;

public  void  ChatClient.update(String s){
        this.display(s);
}
```

Aspect

- Deklaration mit aspect
- Datentyp in AspectJ
- kapselt Aspekte
- Beispiel:

```
public  aspect Tracing{
   after (): execution(* ChatClient .* (..)){
      System.out.println("---> "+
         thisJoinPointStaticPart .getSignature () );
   }
}
```



Compiler

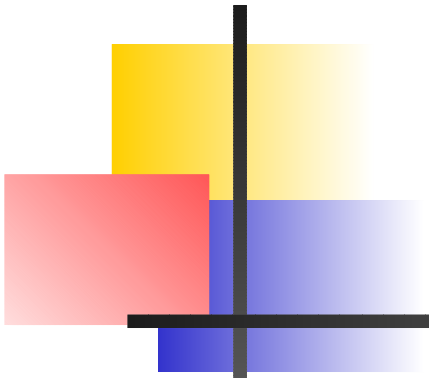
- Java-Compiler versteht Syntax nicht
- eigener Compiler erforderlich
- ajc – der AspectJ-Compiler
- Verwendung
  - Klasse kompilieren:

    ```
    $ ajc Klasse.java
    ```

  - Klasse+Aspekt kompilieren:

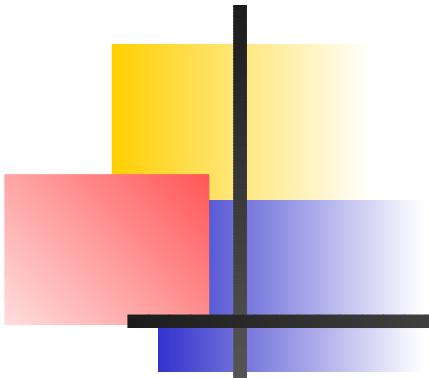
    ```
    $ ajc Klasse.java Aspekt.java
    ```

- Configuration Build File (*.lst) vorteilhaft



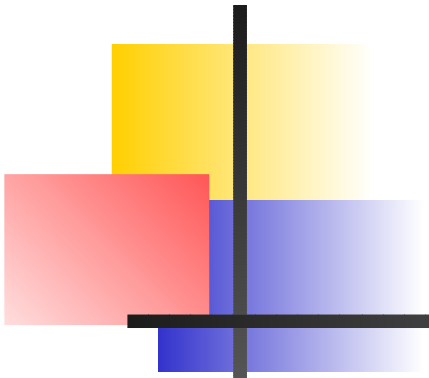
Ajbrowser

- Browser für Aspekt-Code
- visualisiert Einfluss von Aspekt-Code auf Komponenten-Code
- beim Compiler dabei
- Aufruf: `$ ajbrowser <*.lst>`
- später in der Demo



Software Entwicklung mit AspectJ

- Tracing
- Profiling
- Logging



Software Produktion mit AspectJ

- Vorteile durch AOP
  - Erhöhte Wiederverwendbarkeit
  - Bessere Wartbarkeit
  - Leichtere Erweiterbarkeit
- Nachteile
  - Abhängigkeit von Entwickler-Werkzeugen
  - Neue Denkweise erfordlich
  - Übermäßiger Einsatz von Aspekten kann Vorteile zunichte machen