Aspektorientierte Programmierung

- Programmierparadigma
- AspectJ
- *separation of concerns*
- Modularisierung
- Aspekte kapseln Verhalten das mehrere Klassen betrifft

cross cutting concern

- *core concerns* (Kernfunktionalitäten)
- *cross cutting concern* (Querschnittsfunktionalitäten)
- Beispiele *logging*, Authentisierung, *debugging*
- Aspekte eines Programms, die nicht Kernfunktionalität sind, die aber für das Programm notwendig sind

Elemente von AspectJ

- *join point*: Ausführungspunkt in einem Programm
- *pointcut*: wählt Ausführungspunkt mit Kontext
- *advice*: Programmcode der am *join point* ausgeführt werden soll
- *aspect*: Kombination aus *join point*, *pointcut* und *advice*

join point

```
                  public class Test {
Methodenausführung  public static void main(String[] args){
Konstruktoraufruf      Point pt1 = new Point(0,0);
Methodenaufruf         pt1.incrXY(3,6);
                    }
                  }
Klasseninit       public class Point {
Objektinit           private int x, y;

                     public Point(int x, int y) {
Feldzugriff(write)      this.x = x; this.y = y;
                     }
                     public void incrXY(int dx, int dy) {
Feldzugriff(read)        x = this.x + dx; y += dy;
                     }
                  }
```

pointcut

```
public pointcut accountOperation() : call(* Account.*(..))
    Schlüselwort    PCName           PCTyp     Signatur
```

```
execution(Signature)
call(Signature)
get(Signature)
set(Signature)
handler(Signature)
initialization(Signature)
cflow(Pointcut)
within(Signature)
```

advice

```
before() : Pointcut {Programmcode}
around()
after()

before() : call(* Account.*(..)) {Benutzer überprüfen}

pointcut connectionOperation(Connection connection) :
   call(* Connection.*(..) throws SQLException)
     && target(connection);
before(Connection connection) :
   connectionOperation(connection) {
   System.out.println("Operation auf" + connection);
   }
```

```
public aspect JoinPointTraceAspect {
   private int callDepth = -1;

   pointcut tracePoints() : !within(JoinPointTraceAspect);

   before() : tracePoints() {
      callDepth++;
      print("Before", thisJoinPoint);
   }
   after() : tracePoints() {
      callDepth--;
      print("After", thisJoinPoint);
   }
   private void print(String prefix, Object message) {
      for (int i=0, spaces=callDepth*2; i<spaces; i++)
         System.out.print(" ");
      System.out.println(prefix + ": " + message);
   }
}
```

Literatur und Links zu Aspekten

- *AspectJ in Action* von Ramnivas Laddad
- http://eclipse.org/aspectj/

Annotationen

Annotation ist optionaler Parameter, der

- an (fast) beliebige Sprachkonzepte anheftbar ist
- im Java-Code (statisch) gesetzt wird
- von gesamter Werkzeugkette bis zur Laufzeit auslesbar ist

```
@Override
public String toString() { ... }

@BugFix(who="Kaspar", date="1.2.2012", level=3,
        bug="class unnecessary and maybe harmful",
        fix="contents of class body removed")
public class Buggy { }
```

Definition von Annotationen

Syntax von Interfaces adaptiert

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE})
public @interface BugFix {
    String who() default "me"; // author of bug fix
    String date();             // when was bug fixed
    int level() default 1;     // importance level 1-5
    String bug();              // description of bug
    String fix();              // description of fix
}
```

Einschränkungen in @interface

- nur parameterlose Methoden
- als Ergebnistypen nur erlaubt:
  - alle elementaren Typen (`int`, `double`, ...)
  - Aufzählungstypen (`enum`)
  - `String`
  - `Class`
  - andere Annotationen
  - eindimensionale Arrays dieser Typen (als „Mengen“)
- Methodenname `value` → Name in Annotation optional

Annotationen für Annotationsdefinition

```
@Retention(RUNTIME)    // @Retention(value=RUNTIME)
@Target(value=ANNOTATION_TYPE)
public @interface Retention {
    RetentionPolicy value();
}
public enum RetentionPolicy { CLASS, RUNTIME, SOURCE; }

@Retention(value=RUNTIME)
@Target(value=ANNOTATION_TYPE)
public @interface Target {
    ElementType[] value ();
}
public enum ElementType { ANNOTATION_TYPE, CONSTRUCTOR,
 FIELD, LOCAL_VARIABLE, METHOD, PACKAGE, PARAMETER, TYPE; }
```

Annotationen zur Laufzeit

Falls `@Retention(RUNTIME)` wird (echtes) Interface erzeugt:

```
public interface BugFix
                 extends java.lang.annotation.Annotation {
    String who();
    String date();
    int level();
    String bug();
    String fix();
}
```

Zugriff zur Laufzeit

Annotationen über Reflection zur Laufzeit zugreifbar
(falls `@Retention(RUNTIME)`):

```
String s = "";
BugFix a = Buggy.class.getAnnotation(BugFix.class);
if (a != null) {  // null if no such Annotation
    s += a.who() + " fixed a level " + a.level() + " bug";
}
Annotation[] as = Buggy.class.getAnnotations();  // all
Method[] ms = Buggy.class.getMethods();
// verschiedene analoge Methoden auch auf Method, Field
```