AOP

Aspect Oriented Programming

Cross-Cutting Concerns

Cross-Cutting Concern 1

Querschnittliche Belange einer Software, die nicht einfach modularisiert werden können.

Sind meist nicht-funktionale Anforderungen.

Beispiele
Fehlerbehandlung, Logging, Tracing, Caching, Sicherheitsanforderungen...

Cross-Cutting Concern 2

Gehören nicht zu Kernanforderungen des Projektes, sondern stellen Metaanforderungen dar.

Bei vielen Mitarbeitern / Modulen wird es schwierig hier den Überblick über die Implementierungen dieser Anforderungen zu behalten.

Man möchte diese Dinge gerne auch zentral abhandeln können / müssen aber im Code verstreut sein...

Aspect Oriented Programming

Programmierparadigma für
objektorientierte Programmierung (OOP),
um generische Funktionalitäten über
mehrere Klassen hinweg zu verwenden.

Wir wollen loggen

```
1 public void eineMethode() {
2     logger.trace("Betrete \"eineMethode\"");
3
4     // Abarbeitung der Methode
5     m = a + 2;
6
7     logger.trace("Verlasse \"eineMethode\"");
8 }
```

- Darf ich jetzt überall einfügen.
- Im gesamten Code.
- In allen Modulen.
- Viel Arbeit.
- Fehleranfällig.

- Wenn sich der Logger ändert, darf ich überall den Code wieder ändern.

Möglich - Events

```
1 public void eineMethode() {
2     events.methodEnter();
3
4     // Abarbeitung der Methode
5     m = a + 2;
6
7     events.methodExit();
8 }
```

- Besser.
- Zentraler Code!
- Aber ich muss immer noch die Events triggern.
- Dabei können immer noch Fehler passieren / es kann vergessen werden.

AOP to the rescue

```
1  public aspect Tracing {
2      pointcut traceCall():
3          call(* AOPDemo.*(..));
4
5      before(): traceCall() {
6          System.out.println("Betrete \"" + thisJoinPoint + "\"");
7      }
8
9      after(): traceCall() {
10         System.out.println("Verlasse \"" + thisJoinPoint + "\"");
11     }
12 }
```

- Hier mit AspectJ.
- Zentraler Code!
- Wird beim Kompilieren überall hineingeflochten (**code-weaving**).

Beispiel

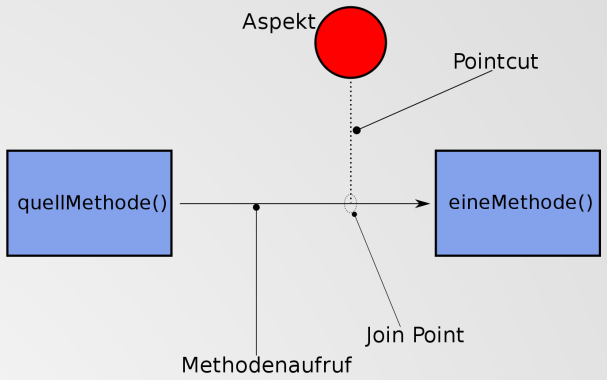
Caching

Caching in Java-Code

- Wir haben ein normales Java-Programm
- Wir wollen bestimmte Methoden cachen
- Wir wollen den Code nicht angreifen (zu viele Methoden)

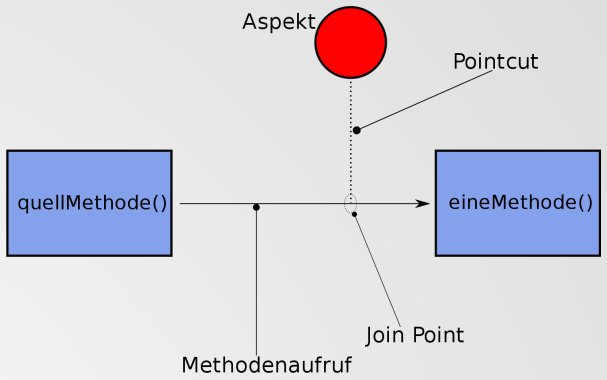
- Dazu ist es nötig jetzt auch zu verändern ob eine Methode überhaupt ausgeführt wird (im cache-hit Fall nämlich nicht).

```aspectj
public aspect Caching {

    pointcut cacheCall():
        call(* AOPDemo.*(..));

    private Map cache = new Map();

    around(): cacheCall(Joinpoint joinPointContext) {

       // Prüfen, ob Rückgabewert für aktuelle Aufruf-Argumente
       // schon im Cache abgelegt wurde
       Object args = joinPointContext.getArguments();
       boolean isCallCached = cache.containsKey(args);

       if (isCallCached) {
             // Umleitung und Austausch des ursprünglichen Methodenaufrufs,
             // gesicherten Rückgabewert aus Cache verwenden
             Object cachedReturnValue = cache.get(args);
             return cachedReturnValue;
       }
       else {
             // Weiterleitung an ursprüngliche Methode und neuen
             // Rückgabewert im Cache sichern
             Object newReturnValue = joinPointContext.proceed();
             cache.put(args, newReturnValue);
             return newReturnValue;
       }
    }
}
```



- **Pointcut** definiert welche Methoden abgefangen werden
- **Advice** definiert wann und den Kontext (before, after, around, ...)

```aspectj
public aspect Caching {

    pointcut cacheCall():
        call(* AOPDemo.*(..));

    private Map cache = new Map();

    around(): cacheCall(Joinpoint joinPointContext) {

       // Prüfen, ob Rückgabewert für aktuelle Aufruf-Argumente
       // schon im Cache abgelegt wurde
       Object args = joinPointContext.getArguments();
       boolean isCallCached = cache.containsKey(args);

       if (isCallCached) {
             // Umleitung und Austausch des ursprünglichen Methodenaufrufs,
             // gesicherten Rückgabewert aus Cache verwenden
             Object cachedReturnValue = cache.get(args);
             return cachedReturnValue;
       }
       else {
             // Weiterleitung an ursprüngliche Methode und neuen
             // Rückgabewert im Cache sichern
             Object newReturnValue = joinPointContext.proceed();
             cache.put(args, newReturnValue);
             return newReturnValue;
       }
    }
}
```



- **Pointcut** definiert welche Methoden abgefangen werden
- **Advice** definiert wann und den Kontext (before, after, around, ...)

Weaving

In Jakarta EE / Microprofile

via CDI

Weaving in Jakarta

- Die meisten modernen AOP Frameworks weaven nicht mehr zur compile-time sondern zur runtime.
- Sie bieten Programmierkonstrukte an, die es dem Programmierer erlauben sich an Events/Methoden 'dranzuhängen'.

- Interceptors
- Decorators

Interceptors

- Werden angeboten für die meisten Lifecycle-Events eines Servers und auch für Methodenaufrufe

```
1  @InterceptorBinding
2  @Retention(RUNTIME)
3  @Target({METHOD,TYPE})
4  public @interface Auditing {
5  }
6
7  @Interceptor
8  @Auditing
9  public class AuditInterceptor implements Serializable {
10     @AroundInvoke
11      public Object auditMethodEntry(InvocationContext ctx) throws Exception {
12            System.out.println("Before entering method:" + ctx.getMethod().getName());
13            return ctx.proceed();
14      }
15 }
```

- Kontext ist ein Methodenaufruf.

Decorators

- Hat als Kontext eine Klasse.

```
1  public interface PlayerItf {
2          public String check();
3          public String getGuess();
4          public void setGuess(String guess);
5  }
6
7  @Decorator
8  public class PlayerDecorator implements PlayerItf,Serializable {
9          @Inject @Delegate PlayerItf player;
10         @Override
11         public String check()  {
12             System.out.println("[Decorator] User check with "+player.getGuess());
13             return player.check();
14         }
15         public String getGuess() {
16                 return player.getGuess();
17         }
18         public void setGuess(String guess) {
19                 player.setGuess(guess.toUpperCase());
20         }
21 }
```

- Ich kann den Zustand der Klasse mit einbeziehen und auch Methoden aufrufen.

References

- https://de.wikipedia.org/wiki/Aspektorientierte_Programmierung
- https://de.wikipedia.org/wiki/Cross-Cutting_Concern
- http://www.mastertheboss.com/jboss-frameworks/cdi/interceptors-and-decorators-tutorial/