Aspekt-orijentisano programiranje (AOP)

Uvod

- Određene funkcionalnosti prožimaju većinu drugih funkcionalnosti u aplikaciji
- Delovi programa koji nisu direktno vezani za poslovnu logiku često izgledaju isto za različite poslovne procedure
- Npr. logovanje, upravljanje transakcijama, kontrola pristupa…
- Ako ovaj tip funkcionalnosti implementiramo na klasičan način:
  - isti kod će se ponavljati ili pozivati u svim delovima aplikacije
  - glavna funkcionalnost će biti zaprljana kodom te dodatne funkcionalnosti

Motivacija

- U mnogim delovima sistema se mogu zateći klase i metode kao na ovom primeru
- Kod same metode treba da vrši neku logiku, kao na primer neko izračunavanje. Kod metode nema nikakve veze sa logovanjem ili zaključavanjem - kod je **zaprljan**
- Šta ako se promeni način logovanja? Da li menjati na svim mestima u kodu?

SomeClass.java

```
1  package p;
2
3  public class SomeClass {
4
5      public void someMethod() {
6
7          // code for some authorization goes here ...
8
9          // code for logging start of the operation
10
11         // code for locking objects (thread-safety) ...
12
13
14         // **** method code ****
15
16
17         // code for unlocking objects (thread-safety) ...
18
19         // code for logging end of the operation
20
21
22     }
23
```

Kod metode

Rešenje - šta do sada znamo

- Ne pišemo kod direktno u metodi, već pravimo funkcije koje kasnije pozivamo
- Međutim, kod metode i dalje ostaje **zaprljan** – metoda ne radi samo ono što treba, već poziva neke druge metode koje nemaju nikakve veze sa logikom koju metoda obavlja

SomeClass.java

```
1  package p;
2
3  public class SomeClass {
4
5      public void someMethod() {
6          auth();
7          log();
8          lock();
9
10         // **** method code ****
11
12         unlock();
13         lock();
14     }
15
16     private void auth() {
17         // some code ...
18     }
19
20     private void log() {
21         // some code ...
22     }
23
24     private void lock() {
25         // some code ...
26     }
27
28     private void unlock() {
29         // some code ...
30     }
31
32 }
33
```

Kod metode

Rešenje - AOP

- Pisanjem aspekta za autorizaciju, logovanje i zaključavanje, prethodno pomenuta metoda se može napisati na ovaj način
- Prilikom poziva metode i dalje se vrši provera autorizacije, logovanje početka i kraja izvršavanja operacije, kao i zaključavanje deljenih objekata
- Kod metode sadrži samo poslovnu logiku – kod **nije zaprljan** nekim drugim kodom

SomeClass.java

```
1  package p;
2
3  public class SomeClass {
4
5      public void someMethod() {
6
7          // **** method code ****
8
9      }
10
```

AOP

- Omogućuje organizovanje često korišćenih funkcionalnosti u komponente koje se mogu više puta koristiti
- AOP omogućava primenu ovih komponenti na druge delove aplikacije
  - primena se vrši na **deklarativan** način
  - poslovna logika aplikacije se ne bavi funkcionalnostima koje ove komponente obavljaju
- Glavna metoda je fokusirana na svoju glavnu funkcionalnost, a dodatne funkcije se deklarativno primenjuju, što direktno utiče na to da je kod metode čitljiviji

AOP terminologija

- ***Advice*** - definiše svrhu aspekta (šta aspekt radi) i trenutak kada se kod aspekta izvršava:
  - *Before* – pre poziva metode na koju se aspekt odnosi
  - *After* – nakon metode (bez obzira na ishod metode)
  - *After-returning* – nakon uspešnog završetka metode
  - *After-throwing* – nakon što metoda izazove izuzetak
  - *Around* – omotač oko metode, tako što se deo koda izvršava pre, a deo posle metode
- ***Join Points*** - tačke u aplikaciji na kojima aspekt **može** biti primenjen (npr. poziv metode, pojava izuzetka...). Ovo su samo potencijalna mesta primene
- ***Pointcut*** - konkretno mesto u aplikaciji na kojem je aspekt primenjen

Šta je aspekt?

- Predstavja spoj *Advice i Pointcut*
- Kada *Advice* primenimo nad *Pointcut* dobijamo aspekt. Ovim smo definisli kod koji treba da se pozove na određenom mestu u aplikaciji
- Aspekt je parče koda koji se može vezati za neku metodu tako da se izvrši:
  1. pre poziva metode
  2. posle poziva metode
  3. oko poziva metode (obuhvata poziv)

Spring i AOP

- Jedna od ključnih komponenti Springa je AOP framework

  https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#aop
- Podržava samo metode kao *Join Points* – aspekt se može definisati samo nad metodom
- Aspekt se može definisati kroz:
  - XML - klasična Spring varijanta, prevaziđena
  - Anotacije – oslanja se na *AspectJ* projekat nezavisan od Spring-a

Kreiranje aspekta

- Aspekt je obična klasa anotirana sa *@Aspect*
- Za metode klase se određuje kada će se pozivati putem jedne od anotacija
  - *@Before*
  - *@After*
  - *@AfterReturning*
  - *@AfterThrowing*
  - *@Around*

Definisanje Pointcut

- Za metode aspekta se definiše na koje metode u kodu se primenjuju
- Definiše se kao parametar anotacija sa prethodnog slajda
- Sintaksa *pointcut* izraza: *AspectJ pointcut expression* sintaksa

```
* rs.ac.uns.ftn.informatika.aop.service.SampleService.someMethodReturning(..)
```

Povratni tip metode (u ovom slučaju bilo koji)

Šablon kojim se definiše na koju metodu se Pointcut odnosi

Koje parametre metoda prima (u ovom slučaju bilo koje)

- *Pointcut* je svaka metoda koja se uklapa u definisani šablon

Kako Spring AOP radi?

- Za objekte nad kojima je definisan aspekt, Spring u toku izvršavanja aplikacije (*runtime*) kreira novi *proxy* objekat
- *Proxy* je omotač oko glavnog objekta. Sadrži glavni objekat nad kojim je definisan, ali i metode koje je uveo *Advice* aspekta
- Za svaki poziv metode glavnog objekta Spring će ustvari pozvati metodu *proxy* objekta
- *Proxy* metoda izvršava metodu glavnog objekta, ali I metode aspekta pre ili posle glavne metode