Zaawansowane metody wytwarzania oprogramowania

Programowanie aspektowe - laboratorium

Temat laboratorium: Telecom

Tomasz Kot
284 993

Konfiguracja środowiska

Autor sprawozdania użył zintegrowanego środowiska programistycznego Eclipse wraz z zainstalowaną wtyczką do programowania aspektowego - AspectJ Development Tool.

Efekty ćwiczeń

Ćwiczenia zostały wykonane zgodnie z wytycznymi przekazanymi w pliku PDF dołączonym do kodu ćwiczenia.

Ćwiczenie 1

Celem pierwszego zadania było znalezienie błędu spowodowanego złym wzorcem punktu przecięcia wprowadzonego przez sfrustrowanego programistę. Wzorzec ten powodował brak możliwości wypisywania na standardowe wyjście. Autorowi udało się go dosyć szybko zlokalizować, a kod poprawionego aspektu został przedstawiony na rysunku numer 1. Widać na nim już poprawioną wersję wzorca, różniącą się od tego złego dodatkowym operandem w koniunkcji ***!within(com.aspects..*)***. Część ta oznacza, że wzorzec ten nie powinien przechwytywać strumieni wyjściowych w klasach z pakietu com.aspects (oraz jego pakietach-dzieciach). Na rysunku numer 2 umieszczono potwierdzenie, iż po tej zmianie projekt działa poprawnie.

```
1  package com.aspects;
2
3  import java.io.PrintStream;
5
6  /*
7   * Aspekt wyswietlajacy w konsoli czas rozpoczecia i zakonczenia nadawania
8   * wiadomosci na konsole.
9   */
10 public aspect MessageTimeDisplay
11 {
12     /* XXX Aspect fields---------------------------------------------- */
13
14     // Pole przechowujace kalendarz sluzacy do uzyskiwania daty i godziny
15     private Calendar cld;
16
17     /* XXX Pointcut and advice ---------------------------------------- */
18
19     // Przekroj przechwytujacy wywolania print i println dla PrintStream
20     pointcut printcut(): call(* PrintStream.print*(..)) && !within(com.aspects..*);
21
22     // Przed wyswietleniem wiadomosci podajemy czas jej nadejscia
23     before(): printcut()
24     {
25         cld = Calendar.getInstance();
26         System.out.print(" TIME[" + cld.getTime() + "]: ");
27     }
28
29     // Po wyswietleniu wiadomosci podajemy czas po przetworzeniu
30     after(): printcut()
31     {
32         cld = Calendar.getInstance();
33         System.out.println(" EOM [" + cld.getTime() + "]");
34     }
35 }
36
```

Rysunek 1. Aspekt z poprawionym wzorcem z zadania numer 1.

```
 TIME[Sun May 23 15:20:20 CEST 2021]: jim calls mik...
 EOM [Sun May 23 15:20:20 CEST 2021]
Insert user:
TomaszKot11
Insert password:
Password1!
 TIME[Sun May 23 15:20:24 CEST 2021]: [new local connection from Jim(650) to Mik(650)]
 EOM [Sun May 23 15:20:24 CEST 2021]
 TIME[Sun May 23 15:20:24 CEST 2021]: call call
 EOM [Sun May 23 15:20:24 CEST 2021]
 TIME[Sun May 23 15:20:24 CEST 2021]: com.telecom.custom.Call@f4f579
 EOM [Sun May 23 15:20:24 CEST 2021]
 TIME[Sun May 23 15:20:24 CEST 2021]: end call call
 EOM [Sun May 23 15:20:24 CEST 2021]
 TIME[Sun May 23 15:20:25 CEST 2021]: mik accepts...
 EOM [Sun May 23 15:20:25 CEST 2021]
 TIME[Sun May 23 15:20:25 CEST 2021]: connection completed
 EOM [Sun May 23 15:20:25 CEST 2021]
 TIME[Sun May 23 15:20:27 CEST 2021]: jim hangs up...
 EOM [Sun May 23 15:20:27 CEST 2021]
 TIME[Sun May 23 15:20:27 CEST 2021]: connection dropped
 EOM [Sun May 23 15:20:27 CEST 2021]
 TIME[Sun May 23 15:20:27 CEST 2021]: mik calls crista...
 EOM [Sun May 23 15:20:27 CEST 2021]
Insert user:
```

Rysunek 2. Aspekt z ćwiczenia 1 działa poprawnie, uwaga: zrzut ekranu zawiera również wyjście aspektu z ćwiczenia 3.

Ćwiczenie 2

Celem ćwiczenia drugiego było zaimplementowanie aspektu śledzącego wywołania metod w projekcie. Aspekt ten miał zapisywać wszystkie “parametry” metod (rodzaj, ilość i typy argumentów wejścia i wyjścia oraz czas ich wykonania). Stworzony aspekt nosi nazwę TrackingCodeExerciseOne. Jego kod został przedstawiony na rysunku numer 3.

Zasadniczym elementem każdego aspektu jest wyrażenie dokonujące cięcia. W tym przypadku zastosowano wyrażenie ***execution(* com.telecom.custom.*.*(..))*** celem przechwycenia wywołań metod o dowolnym typie zwracanym, ilości i typie argumentów wejściowych zawartych w pakiecie (i jego pakietach-dzieciach) ***com.telecom.custom***.

Wartym wspomnienia jest również fakt, iż z racji że niektóre metody zwracają typ ***void*** konieczne stało się zawarcie instrukcji ***proceed*** w bloku try-catch. Dokonano również zmian w kodzie zwracających wartości null na typ ***java.util.Optional***.

Drugi operand koniunkcji - ***target(objectOne)*** przechwytuje obiekt który jest wywoływany. Był on konieczny z racji na chęć wykorzystania go w poleceniu ***proceed*** i “odmrożenia” dalszego wykonania przechwyconej metody.

Zawartość pliku logów została przedstawiona na rysunku numer 4. W czasie wykonania kodu projektu nie widać żadnych oznak działania tego aspektu na standardowym wyjściu. Warto wspomnieć iż dodatkowo, jeśli plik już istnieje, to nowe logi (przy ponownym uruchomieniu) są zbierane po wstawieniu nagłówka w formie aktualnej daty uruchomienia programu.

```java
import java.io.File;

public aspect TrackingCodeExerciseOne {

    private static boolean isFirstRun = true;
    private static File logFile = new File("zmwo_logs.log");

    pointcut methodCallStats(Object objectOne) : execution(* com.telecom.custom.*.*(..)) && target(objectOne);

    Object around(Object objectOne): methodCallStats(objectOne) {
        Timer aspectTimer = new Timer();
        aspectTimer.start();
        Object returnedObject = null;
        try {
            returnedObject = proceed(objectOne);
        } catch(NullPointerException e ) {
            System.err.println("Null pointer exception in method tracking aspect");
        }
        aspectTimer.stop();

        try {
            createLogFileIfNotExists();
            appendTimestampIfFirstEntry();
            StringBuilder stats = prepareMethodCallEntry(thisJoinPoint, returnedObject, aspectTimer);
            appendMethodCallStatsToFile(stats);
        } catch(IOException e) {
            System.out.println("Error occured while creating a log entry");
            System.exit(1);
        }
        return returnedObject;
    }

    private static StringBuilder prepareMethodCallEntry(JoinPoint joinPoint, Object returnedObject, Timer aspectTimer) {
        StringBuilder stats = new StringBuilder();

        stats.append("--------------------");
        stats.append("\n");
        stats.append("Klasa wywołująca: " + joinPoint.getThis().getClass());
        stats.append("\n");
        stats.append("Sygnatura metody: " + joinPoint.getSignature().toString());
        stats.append("\n");
        stats.append("Argumenty metody: " + Arrays.toString(joinPoint.getArgs()));
        stats.append("\n");
        if(returnedObject == null) {
            stats.append("Wynik zwracany przez metode: void");
        } else {
            stats.append("Wynik zwracany przez metode: " + returnedObject.getClass() + " wartosc: " + returnedObject);
        }
        stats.append("\n");
        stats.append("Czas wykonania: " +  aspectTimer.getTime());
        stats.append("\n");
        stats.append("--------------------");
        return stats;
    }

```

Rysunek 3. Fragment kodu aspektu TrackingCodeExerciseOne dokonującego śledzenia wywołania metod w projekcie.

```
Czas wykonania: 5430
-------------------
-------------------
2021/05/23 15:07:49
 -------------------
-------------------
Klasa wywołująca: class com.telecom.custom.Customer
Sygnatura metody: String com.telecom.custom.Customer.toString()
Argumenty metody: []
Wynik zwracany przez metode: class java.lang.String wartosc: Mik(650)
Czas wykonania: 0
------------------------------------
Klasa wywołująca: class com.telecom.custom.Customer
Sygnatura metody: Optional com.telecom.custom.Customer.call(Customer)
Argumenty metody: [Mik(650)]
Wynik zwracany przez metode: class java.util.Optional wartosc: Optional.empty
Czas wykonania: 6418
------------------------------------
Klasa wywołująca: class com.telecom.custom.Customer
Sygnatura metody: String com.telecom.custom.Customer.toString()
Argumenty metody: []
Wynik zwracany przez metode: class java.lang.String wartosc: Crista(415)
Czas wykonania: 0
------------------------------------
Klasa wywołująca: class com.telecom.custom.Customer
Sygnatura metody: Optional com.telecom.custom.Customer.call(Customer)
Argumenty metody: [Crista(415)]
Wynik zwracany przez metode: class java.util.Optional wartosc: Optional.empty
Czas wykonania: 3988
------------------------------------
Klasa wywołująca: class com.telecom.custom.BasicSimulation
Sygnatura metody: void com.telecom.custom.AbstractSimulation.run()
Argumenty metody: []
Wynik zwracany przez metode: void
Czas wykonania: 10443
-------------------
-------------------
2021/05/23 15:11:06
 -------------------
```

Rysunek 4. Fragment plików logów powstałych w czasie działania programu

Ćwiczenie 3

Celem ćwiczenia trzeciego było wprowadzenie systemu uwierzytelniania przy każdym nowym połączeniu. Podana funkcjonalność została uzyskana przy pomocy programowania aspektowego i napisanego aspektu o nazwie AuthorizationAspectExerciseThree. Jego kod został przedstawiony na rysunku numer 5.

Hasła i nazwy użytkowników są przechowywane w pliku user_password_db.txt w formacie przedstawionym na rysunku numer 7, a wczytywane są w bloku static klasy aspektu.

Punkt przecięcia jest zdefiniowany przy pomocy wyrażenia * ***com.teelcom.custom.Customer.call(..)***, które rozumiane jest jako wychwytujące wywołanie metody o nazwie call w klasie Customer o dowolnym rodzaju i ilości argumentów wejściowych, dowolnym typie zwracanym i modyfikatorze dostępu (wzorzec ten mógłby być również bardziej “zawężony”). Przykład poprawnego uwierzytelnienia oraz odrzuconego został przedstawiony na rysunku numer 7.

```java
package com.aspects;

import java.io.BufferedReader;

public aspect AuthorizationAspectExerciseThree {
    static Map<String, String> userPasswordMap = new HashMap<>();

    static {
        // load a file with a password and login
        File userFile = new File("./src/com/aspects/user_password_db.txt");
        FileReader userFileReader = null;
        try {
            userFileReader = new FileReader(userFile);
        } catch(FileNotFoundException e) {
            System.out.println("User and password file not found");
            System.exit(1);
        }
            try(BufferedReader userFileStream = new BufferedReader(userFileReader)) {
                Stream<String> userStream = userFileStream.lines();

                java.util.List<String[]> collectedList = userStream.map(el -> el.split(";")).collect(Collectors.toList());
                for(String[] el : collectedList)
                    userPasswordMap.put(el[0], el[1]);
            } catch (IOException e) {
                System.out.println("Error occured while initializing the list");
                System.exit(1);
            }
    }
    String insertedPassword;
    String insertedUser;

    String foundPassword;
    String foundUser;

    pointcut callMethodCut(Customer customer) : this(customer) && execution(* com.telecom.custom.Customer.call(..));

    before(Customer customer): callMethodCut(customer)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Insert user: ");
        insertedUser = scanner.nextLine();
        System.out.println("Insert password: ");
        insertedPassword = scanner.nextLine();
    }

    Object around(Customer customer) : callMethodCut(customer) {
        foundPassword = userPasswordMap.get(insertedUser);
        if(!(userPasswordMap.containsKey(insertedUser) &&  foundPassword.equals(insertedPassword))) {
            System.out.println("Error occured while inserting the user and password");
            System.exit(1);
        }
        return proceed(customer);
    }
}
```

Rysunek 5. Kod aspektu AuthorizationAspectExerciseThree dokonującego uwierzytelnienia użytkownika.

```
TomaszKot11;Password1!
TomaszKot12;Password1!
TomaszKot13;Haslo123
TomaszKot14;123
KotTomasz98;Password1!
```

Rysunek 6. Format pliku do przechowywania nazw użytkowników oraz haseł aplikacji.

```
 TIME[Sun May 23 16:15:20 CEST 2021]: jim calls mik...
 EOM [Sun May 23 16:15:20 CEST 2021]
Insert user:
TomaszKot11
Insert password:
Password1!
 TIME[Sun May 23 16:15:25 CEST 2021]: [new local connection from Jim(650) to Mik(650)]
 EOM [Sun May 23 16:15:25 CEST 2021]
 TIME[Sun May 23 16:15:25 CEST 2021]: call call
 EOM [Sun May 23 16:15:25 CEST 2021]
 TIME[Sun May 23 16:15:25 CEST 2021]: com.telecom.custom.Call@f4f579
 EOM [Sun May 23 16:15:25 CEST 2021]
 TIME[Sun May 23 16:15:25 CEST 2021]: end call call
 EOM [Sun May 23 16:15:25 CEST 2021]
 TIME[Sun May 23 16:15:26 CEST 2021]: mik accepts...
 EOM [Sun May 23 16:15:26 CEST 2021]
 TIME[Sun May 23 16:15:26 CEST 2021]: connection completed
 EOM [Sun May 23 16:15:26 CEST 2021]
 TIME[Sun May 23 16:15:28 CEST 2021]: jim hangs up...
 EOM [Sun May 23 16:15:28 CEST 2021]
 TIME[Sun May 23 16:15:28 CEST 2021]: connection dropped
 EOM [Sun May 23 16:15:28 CEST 2021]
 TIME[Sun May 23 16:15:28 CEST 2021]: mik calls crista...
 EOM [Sun May 23 16:15:28 CEST 2021]
Insert user:
TomaszKot11
Insert password:
Password1!
 TIME[Sun May 23 16:15:33 CEST 2021]: [new long distance connection from Mik(650) to Crista(415)]
 EOM [Sun May 23 16:15:33 CEST 2021]
 TIME[Sun May 23 16:15:33 CEST 2021]: call call
 EOM [Sun May 23 16:15:33 CEST 2021]
 TIME[Sun May 23 16:15:33 CEST 2021]: com.telecom.custom.Call@180350b
 EOM [Sun May 23 16:15:33 CEST 2021]
 TIME[Sun May 23 16:15:33 CEST 2021]: end call call
 EOM [Sun May 23 16:15:33 CEST 2021]
 TIME[Sun May 23 16:15:33 CEST 2021]: crista accepts...
 EOM [Sun May 23 16:15:33 CEST 2021]
 TIME[Sun May 23 16:15:33 CEST 2021]: connection completed
 EOM [Sun May 23 16:15:33 CEST 2021]
 TIME[Sun May 23 16:15:34 CEST 2021]: crista hangs up...
 EOM [Sun May 23 16:15:34 CEST 2021]
 TIME[Sun May 23 16:15:34 CEST 2021]: connection dropped
 EOM [Sun May 23 16:15:34 CEST 2021]
```

```
 TIME[Sun May 23 16:20:15 CEST 2021]: jim calls mik...
 EOM [Sun May 23 16:20:15 CEST 2021]
Insert user:
TomaszKot11
Insert password:
Niepoprawne
Error occured while inserting the user and password
```

Rysunek 7. Poprawne i błędne uwierzytelnienie użytkownika.

Ćwiczenie 4

Celem ćwiczenia czwartego była symulacja ataku hakerskiego na podstawie znajomości działania aspektu związanego z uwierzytelnianiem oraz obrona przed nim, usuwająca efekt jego działania przy pomocy aspektu “policjanta”. Kod aspektu symulującego hakera ExerciseFourHacker został przedstawiony na rysunku numer 8, natomiast policjanta ExerciseFourPolice na rysunku numer 9.

Aspekt hakera wychwytując wywołanie aspektu AuthorizationAspectExerciseThree dokonuje, przy pomocy mechanizmu refleksji, zamiany wartości w zmiennej przechowującej wartość hasła pobraną od użytkownika na pusty ciąg znaków. Działanie jedynie aspektu hakera zostało przedstawione na rysunku numer 10.

Policjant wykrywa wywołanie aspektu hakera i podczas jego wywołania dokonuje powrotu (przerywa działanie porady hakera), uniemożliwiając zmianę hasła. Efekt działania policjanta w czasie “równoczesnego” działania hakera został przedstawiony na rysunku numer 11.

```
1  package com.aspects;
2
3  import java.lang.reflect.Field;
6
7  public aspect ExerciseFourHacker {
8
9
10     pointcut authorizationHook() : within(com.aspects.AuthorizationAspectExerciseThree) && adviceexecution();
11
12     before() : authorizationHook() {
13         try {
14             Field fieldTwo = thisJoinPoint.getThis().getClass().getDeclaredField("insertedPassword");
15             fieldTwo.setAccessible(true);
16             fieldTwo.set(thisJoinPoint.getThis(), "");
17         } catch(NoSuchFieldException e) {
18             System.out.println("Hacker did not find the field");
19         } catch(IllegalAccessException e) {
20             System.out.println("Hacker did not find the field");
21         }
22     }
23
24 }
25
```

Rysunek 8. Kod aspektu “hakera” dokonującego usunięcia hasła wpisanego przez użytkownika.

```
1  package com.aspects;
2
3  public aspect ExerciseFourPolice {
4
5      pointcut policeCall() : adviceexecution() && within(com.aspects.ExerciseFourHacker);
6
7      void around() : policeCall() {
8          return;
9      }
10 }
11
```

Rysunek 9. Kod aspektu “policjanta” resetujący działania aspektu hakera ExerciseFourHacker

```
 TIME[Sun May 23 18:08:00 CEST 2021]: jim calls mik...
 EOM [Sun May 23 18:08:00 CEST 2021]
Insert user:
TomaszKot11
Insert password:
Password1!
Error occured while inserting the user and password
```

Rysunek 10. Efekt działania aspektu hakera. Z “wyłączonym” aspektem policjanta.

```
 TIME[Sun May 23 18:31:00 CEST 2021]: jim calls mik...
 EOM [Sun May 23 18:31:00 CEST 2021]
Insert user:
TomaszKot11
Insert password:
Password1!
 TIME[Sun May 23 18:31:04 CEST 2021]: [new local connection from Jim(650) to Mik(650)]
 EOM [Sun May 23 18:31:04 CEST 2021]
 TIME[Sun May 23 18:31:04 CEST 2021]: call call
 EOM [Sun May 23 18:31:04 CEST 2021]
 TIME[Sun May 23 18:31:04 CEST 2021]: com.telecom.custom.Call@e285c6
 EOM [Sun May 23 18:31:04 CEST 2021]
 TIME[Sun May 23 18:31:04 CEST 2021]: end call call
 EOM [Sun May 23 18:31:04 CEST 2021]
 TIME[Sun May 23 18:31:05 CEST 2021]: mik accepts...
 EOM [Sun May 23 18:31:05 CEST 2021]
 TIME[Sun May 23 18:31:05 CEST 2021]: connection completed
 EOM [Sun May 23 18:31:05 CEST 2021]
 TIME[Sun May 23 18:31:07 CEST 2021]: jim hangs up...
 EOM [Sun May 23 18:31:07 CEST 2021]
 TIME[Sun May 23 18:31:07 CEST 2021]: connection dropped
 EOM [Sun May 23 18:31:07 CEST 2021]
 TIME[Sun May 23 18:31:07 CEST 2021]: mik calls crista...
 EOM [Sun May 23 18:31:07 CEST 2021]
Insert user:
```

Rysunek 11. Potwierdzenie usunięcia efektu działania przez aspekt policjanta - program działa normalnie.

Podsumowanie

Podsumowując, laboratorium nauczyło autora sprawozdania podstaw programowania aspektowego w języku Java. Mimo, że autor częściej programuje w językach skryptowych to przedstawiony paradygmat jest również możliwy do użycia w nich. Jako przykład można podać bibliotekę dla języka Ruby (tzw. gem) aquarium (https://rubygems.org/gems/aquarium/) bądź artykuł polskiej firmy znanej z szerzenia DDD w języku Ruby dotyczący tego zagadnienia - https://blog.arkency.com/2013/07/ruby-and-aop-decouple-your-code-even-more/ .