18. Spring Framework (АОП)

- Основы аспектно-ориентированного программирования
- Работа с объектами Aspect, создание Advice
- Конфигурация Pointcuts и Joinpoints

Сценарий: внедрить сквозную логику в приложение (для каждого метода бизнес логики будет добавлено ведение логов)

Pom.xml: (используемые библиотеки, также добавлен пакет AspectJWeaver.jar)

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
  <spring.version>5.2.1.RELEASE</spring.version>
</properties>

<dependencies>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.version}</version>
  </dependency>

</dependencies>
```

Класс конфигурации MyConfig.java: (аннотация @EnableAspectJAutoProxy)

```
@Configuration
@ComponentScan("org.example.springAop")
@EnableAspectJAutoProxy
public class MyConfig {
}
```

Класс University: (добавление и получение списка студентов)

```
@Component
public class University {
    private List<Student> students = new ArrayList<>();

    public void addSudents(){
        Student st1 = new Student("Linar Latypov", 5, 9.8);
        Student st2 = new Student("Jane Best", 4, 9.5);
        Student st3 = new Student("Katy Kate", 3, 9.3);

        students.add(st1);
        students.add(st2);
        students.add(st3);

    }

    public List<Student> getStudents(){
        System.out.println("Начало работы метода getStudents");
        //System.out.println(students.get(3));
        System.out.println("Information from method getStudents");
        System.out.println(students);
        return students;
    }
```

Класс UniversityLibrary (получение/возврат/добавление книги, журналов):

```
@Component
public class UniversityLibrary {
    public void getBook() {
        System.out.println("Мы берем книгу из UniversityLibrary ");
        System.out.println("-----------------------------------");
    }
    public String returnBook(){
        //int a = 10/0;
        System.out.println("Мы возвращаем книгу в UniversityLibrary");
        System.out.println("-----------------------------------");
        return "1984";
    }
    public void addBook(String personName, Book book){
        System.out.println("Мы добавляем книгу в UniversityLibrary");
        System.out.println("-----------------------------------------");
    }
    public void addMagazine(){
        System.out.println("Мы добавляем журнал в UniversityLibrary");
        System.out.println("-----------------------------------------");
    }
    public void returnMagazine(){
        System.out.println("Мы возвращаем журнал в UniversityLibrary");
```

```
        System.out.println("----------------------------------------");
    }

    public void getMagazine(){
        System.out.println("Мы берём журнал из UniversityLibrary");
        System.out.println("----------------------------------------");
    }
```

Класс Book (информация о книгах):

```
@Component
public class Book {

    @Value("1984")
    private String name;

    @Value("George Orwell")
    private String author;

    @Value("1949")
    private int yearOfPublication;

    public String getAuthor() {
        return author;
    }
    public int getYearOfPublication() {
        return yearOfPublication;
    }
    public String getName() {
        return name;
    }
}
```

Создание аспектов

Объявление в классе: (аннотация @Aspect)

```
@Component
@Aspect
public class UniversityLogingAspect {
```

Определение аспекта до начала работы основного метода (аннотация @Before, с указанием метода и параметров):

```
@Before("execution(* getStudents())")
public void beforeGetStudentLoggingAdvice(){
    System.out.println("beforeGetStudentLoggingAdvice: логируем получение " +
            "списка студентов перед методом getStudents");
}
```

Определение аспекта после завершения работы основного метода (аннотация @After, с указанием метода и параметров):

```
@After("execution(* getStudents())")
public void afterGetStudentsLoggingAdvice(){
    System.out.println("afterStudentsGetLoggingAdvice: логгируем нормальное
окончание работы метода " +
            "или выброс исключения");
}
```

Определение аспекта после завершения работы основного метода с возвратом какого либо значения (аннотация @AfterReturning, с указанием метода и параметров pointcut, returning, над вернувшимся результатом возможно производить операции):

```
@AfterReturning(pointcut = "execution(* getStudents())", returning =
"students")
public void afterReturningGetStudentLoggingAdvice(List<Student> students){
    Student firstStudent = students.get(0);
    String nameSurname = firstStudent.getNameSurname();
    nameSurname = "Mr. " + nameSurname;
    firstStudent.setNameSurname(nameSurname);
    double avgGrade = firstStudent.getAvgGrade();
    avgGrade = avgGrade -5;
    firstStudent.setAvgGrade(avgGrade);
    System.out.println("afterReturningGetStudentLoggingAdvice: логируем
получение " +
            "списка студентов после работы метода getStudents");
}
```

Определение аспекта после завершения работы основного метода с выбросом исключения (аннотация @AfterThrowing, с указанием метода и параметров pointcut, throwing):

```
@AfterThrowing(pointcut = "execution(* getStudents())", throwing =
"exception")
 public void afterThrowingGetStudentsLoggingAdvice(Throwable exception){
     System.out.println("afterThrowingGetStudentLoggingAdvice: логгируем
выброс исключения" + exception);
 }
```

Определение аспекта с двойным поведением, разделяя поведение на, до и после, выполнения основной логики (аннотация @Around, указание параметра ProcedingJoinPoint, и метода proceed(), для запуска основного метода):

```
@Around("execution(public String returnBook())")
public Object aroundReturnBookLoggingAdvice(ProceedingJoinPoint
proceedingJoinPoint) throws Throwable {
    System.out.println("aroundReturnBookLoggingAdvice: в библиотеку пытаются
вернуть книгу");
    Object targetMethodResult = null;
    try {
        targetMethodResult = proceedingJoinPoint.proceed();
    } catch (Exception e) {
```

```
        System.out.println("aroundReturnBookLoggingAdvice: было поймано
исключение " + e);
        throw e;
    }
    System.out.println("aroundReturnBookLoggingAdvice: в библиотеку успешно
вернули книгу");
    return targetMethodResult;
}
```

Получение сигнатур метода и указание очерёдности выполнения аспекта (аннотация @Order, класс MethodSignature):

```
@Component
@Aspect
@Order(1)
public class LoggingAspect {

    @Before("org.example.springAop.aop.aspects.MyPointcuts.allAddMethods()")
    public void beforeAddLoggingAdvice(JoinPoint joinPoint) {
        MethodSignature methodSignature = (MethodSignature)
joinPoint.getSignature();
        System.out.println("MethodSignature= " + methodSignature);
        System.out.println("MethodSignature.getMethod = " +
methodSignature.getMethod());
        System.out.println("MethodSignature.getReturnType = " +
methodSignature.getReturnType());
        System.out.println("MethodSignature.getName = " +
methodSignature.getName());

        if (methodSignature.getName().equals("addBook")){
            Object[] arguments = joinPoint.getArgs();
            for(Object obj : arguments){
                if(obj instanceof Book){
                    Book myBook = (Book) obj;
                    System.out.println("Информация о книге: название - " +
myBook.getName() + " автор - " + myBook.getAuthor() + " год публикации - "
                     + myBook.getYearOfPublication());
                }
                else if (obj instanceof String){
                    System.out.println("Книгу добавляет " + obj);
                }
            }
        }
        System.out.println("beforeGetLoggingAdvice: попытка получить
книгу/журнал");
        System.out.println("---------------------------------");
    }
```

Класс MyPointcuts: (объединение и комбинирование Pointcuts, аннотация @Pointcut, с ключевым словом execution и указанием сигнатур):

```
public class MyPointcuts {

    @Pointcut("execution(* add*(..))")
    public void allAddMethods(){}

    @Pointcut("execution(*
org.example.springAop.aop.UniversityLibrary.*(..))")
```

```
    private void allMethodsFromUniLibrary() {
    }

    @Pointcut("execution(*
org.example.springAop.aop.UniversityLibrary.get*())")
    private void allGetMethodsFromUniLibrary() {
    }

    @Pointcut("execution(*
org.example.springAop.aop.UniversityLibrary.return*())")
    private void allReturnMethodsFromUniLibrary() {
    }

    @Pointcut("allGetMethodsFromUniLibrary() ||
allReturnMethodsFromUniLibrary()")
    private void allGetAndReturnMethodsFromUniLibrary() {
    }
```

Создание контекста и main() метод:

```
public class Test1 {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(MyConfig.class);
        UniversityLibrary universityLibrary =
context.getBean("universityLibrary", UniversityLibrary.class);
        Book book = context.getBean("book", Book.class);
        universityLibrary.getBook();
        universityLibrary.addBook("Linar", book);
        universityLibrary.returnBook();
        context.close();
    }
```

Запуск методов с выводом логов:

```
Мы берем книгу из UniversityLibrary
-----------------------------------
MethodSignature= void org.example.springAop.aop.UniversityLibrary.addBook(String,Book)
MethodSignature.getMethod = public void org.example.springAop.aop.UniversityLibrary.addBook(java.lang.String,org.example.springAop.aop.Book)
MethodSignature.getReturnType = void
MethodSignature.getName = addBook
Книгу добавляет Linar
Информация о книге: название - 1984 автор - George Orwell год публикации - 1949
beforeGetLoggingAdvice: попытка получить книгу/журнал
-----------------------------------
beforeGetSecurityAdvice: проверка прав на получение книги/журнала
-----------------------------------
beforeGetExceptionHandlingAdvice: ловим/обрабатываем исключение при попытке получить книгу журнал
-----------------------------------
Мы добавляем книгу в UniversityLibrary
---------------------------------------
aroundReturnBookLoggingAdvice: в библиотеку пытаются вернуть книгу
Мы возвращаем книгу в UniversityLibrary
-----------------------------------
aroundReturnBookLoggingAdvice: в библиотеку успешно вернули книгу
```