Zaawansowane biblioteki programistyczne Java

Mateusz Sapała

Zaawansowane techniki programowania

Wydział Matematyki i Informatyki Uniwersytetu Łódzkiego

Biblioteka programistyczna

- Zbiór funkcji i klas, które służą programiście do ściśle określonego celu.
- Odpowiada za konkretne zagadnienie, bądź kilka zagadnień będących stosunkowo podobnymi.
- Programista używa z biblioteki tylko funkcje, które są mu w danym projekcie potrzebne.
- Po użyciu metody z biblioteki po czym pełna kontrola wykonania kodu wraca do nas.

Apache Commons Lang 3

Biblioteka Apache Commons Lang 3 to populara biblioteka, mający na celu rozszerzenie funkcjonalności platformy Java.

Repertuar biblioteki jest dość bogaty, począwszy od manipulacji stringami, tablicami i liczbami, po implementacje kilku uporządkowanych struktur danych, takich jak ułamki.

StringUtils

StringUtils pozwala nam wykonywać szereg bezpiecznych operacji na ciągach znaków, które uzupełniają/rozszerzają te, które zapewnia klasa String. Przykłady metod:

```
1  StringUtils.isEmpty("");
2  StringUtils.isBlank(" ");
3  StringUtils.isAllLowerCase("abcd");
4  StringUtils.isAllUpperCase("ABCD");
5  StringUtils.isMixedCase("abCD");
6  StringUtils.isAlpha("abcd");
7  StringUtils.isAlphanumeric("abcd1234");
```

NumberUtils

Kolejnym ważnym elementem biblioteki jest klasa NumberUtils, która zapewnia obszerną liczbę metod użytkowych, których celem jest przetwarzanie i manipulowanie typami liczbowymi. Przykłady metod:

```
1  NumberUtils.compare(1234, 1234);
2  NumberUtils.isParsable("1234");
3  NumberUtils.isCreatable("0xff");
4  NumberUtils.createNumber("1234");
5  NumberUtils.max(new int[]{1, 2, 3});
6  NumberUtils.min(new int[]{1, 2, 3});
```

Fraction

Klasa Fraction umożliwia błyskawiczne dodawanie, odejmowanie i mnożenie ułamków zwykłych. Przykłady tworzenie ułamków i dostępnych dla nich metod:

```
1  Fraction fraction1 = Fraction.getFraction(1, 4);
2  Fraction fraction2 = Fraction.getFraction(3, 4);
3  fraction1.add(fraction2);
4  fraction1.subtract(fraction2);
5  fraction1.multiplyBy(fraction2);
6  fraction1.divideBy(fraction2);
7  fraction1.doubleValue();
```

Lombook

Java zadań może wymagać napisania dużej ilości kodu, który często nie wnosi żadnej rzeczywistej wartości biznesowej do danego programu. Project Lombok to biblioteka Java, która automatycznie łączy się ze zintegrowanym środowiskiem programistycznym i ogranicza niepotrzebne linie kodu oraz wzbogaca możliwości Javy.

Lombook podłącza się do procesu kompilacji i automatycznie generuje kod Java do plików .class zgodnie z dodanymi adnotacjami.

```java
public class User {
    private String name;
    private String surname;
    private Date birthDate;
    private String address;

    public UserWithoutBuilder(String name, String surname, Date birthDate, String address) {
        this.name = name;
        this.surname = surname;
        this.birthDate = birthDate;
        this.address = address;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSurname() {
        return surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    public Date getBirthDate() {
        return birthDate;
    }

    public void setBirthDate(Date birthDate) {
        this.birthDate = birthDate;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
}
```

```java
@AllArgsConstructor
@Getter
@Setter
public class UserWithLombook {
    private String name;
    private String surname;
    private Date birthDate;
    private String address;
}
```

```java
public class Main {
    public static void main(String[] args) {
        User user = User.builder()
                .name("name")
                .surname("surname")
                .birthDate(new Date())
                .build();
        System.out.println(user.getName());
    }

    @Builder
    @Getter
    static class User {
        private String name;
        private String surname;
        private Date birthDate;
        @Builder.Default
        private String address = "address";
    }
}
```

Mockito

Mockito to jedna z najpopularniejszych bibliotek Javy. Jej zadaniem jest ułatwienie testowania kodu, w izolacji od jego zależności. Kluczowe funkcjonalności Mockito to:

- Zmiana zachowania obiektów– zwracanie wartości, zgłaszanie wyjątków itp.
- Weryfikacja interakcji – czy wystąpiły interakcje, ile razy i z jakimi parametrami.

Czy jest mock?

Mock (inaczej dubler) to obiekt, którego celem jest zastąpienie oryginalnego obiektu w celu określenia jego zachowania. Całkowicie zastępuje prawdziwy obiekt, ale zachowuje jego publiczne API (metody).

W dużych projektach większy kod dzielony jest na mniejsze fragmenty – klasy/metody. Wtedy większe funkcjonalności komponowane są z mniejszych poprzez chciażby dependency injection (tzw. wstrzykiwanie zależności). Kiedy testujemy fragment kodu, który ma zależność do innego fragmentu kodu (klasa używa innej do realizacji swojej funkcjonalności), wówczas mock rozwiązuje problem posiadania „wstrzykniętej” implementacji.

```java
@AllArgsConstructor
@Getter
@EqualsAndHashCode
public class User {
    private Long id;
    private String name;
    private String surname;
}
```

```java
@RequiredArgsConstructor
public class UserRepository {
    private static final List<User> users = new ArrayList<>();

    public void addUser(User user) {
        users.add(user);
    }

    public List<User> getUsers() {
        return users;
    }
}
```

```java
@RequiredArgsConstructor
public class UserService {
    private final UserRepository userRepository;
    private final AtomicLong id;

    public void addUser(String name, String surname) {
        if (StringUtils.isBlank(name)) {
            throw new IllegalArgumentException("Name have to be provided");
        }
        if (StringUtils.isBlank(surname)) {
            throw new IllegalArgumentException("Surname have to be provided");
        }
        userRepository.addUser(new User(id.incrementAndGet(), name, surname));
    }

    public User getUser(Long userId) {
        return userRepository.getUsers().stream()
                .filter(user -> user.getId().equals(userId))
                .findAny()
                .orElseThrow();
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        UserRepository userRepository = new UserRepository();
        UserService userService = new UserService(userRepository, new AtomicLong(0));
        userService.addUser("Tomasz", "Nowak");
        User user = userService.getUser(1L);
    }
}
```

```java
class UserServiceTest {
    @Test
    void addUser() {
        AtomicLong id = new AtomicLong(0);
        UserRepository userRepositoryMock = Mockito.mock(UserRepository.class);

        UserService userService = new UserService(userRepositoryMock, id);
        userService.addUser("Test", "Test");
        Mockito.verify(userRepositoryMock).addUser(new User(1L, "Test", "Test"));
    }

    @Test
    void getUser() {
        UserRepository userRepositoryMock = Mockito.mock(UserRepository.class);
        User test1 = new User(1L, "Test1", "Test1");
        User test2 = new User(2L, "Test2", "Test2");
        Mockito.when(userRepositoryMock.getUsers()).thenReturn(List.of(test1, test2));

        UserService userService = new UserService(userRepositoryMock, null);
        User actual = userService.getUser(test2.getId());
        assertEquals(test2, actual);
    }
}
```