TESTIRANJE SOFTVERA (13S113TS)
LABORATORIJSKA VEŽBA:
ALATI ZA JEDINIČNO TESTIRANJE I POKRIVENOST KODA

Prof. dr Dražen Drašković
Mast. inž. Nikola Stanković

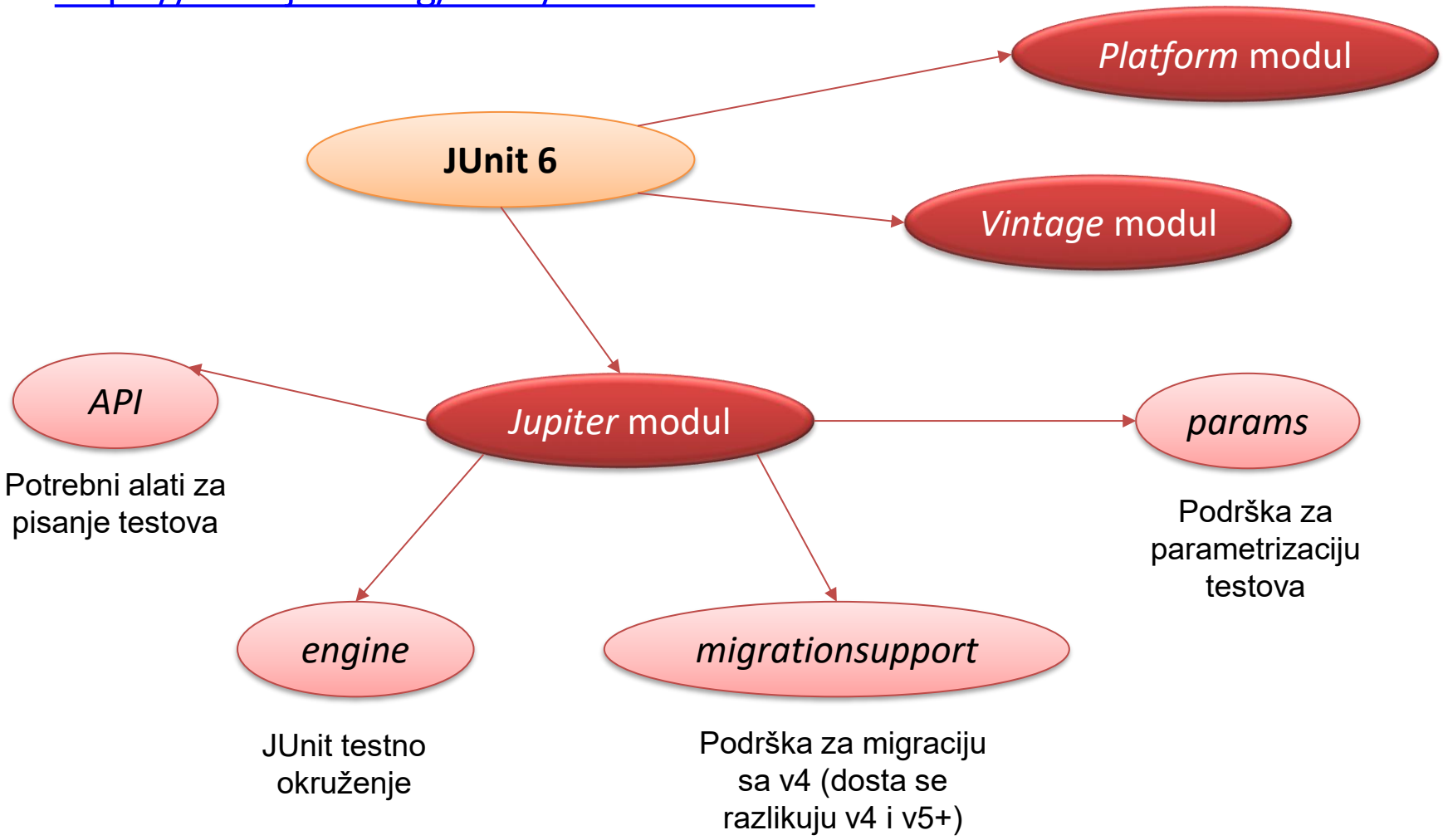
Univerzitet u Beogradu - Elektrotehnički fakultet

TESTIRANJE PROGRAMA UZ POMOĆ JUNIT-A

- JUnit je framework predviđen za jedinično testiranje pojedinačnih Java klasa ili grupe klasa.
- Filozofija koja se promoviše uz JUnit je test-first projektovanje, da sam programer paralelno sa kodiranjem piše i testove.
- JUnit može da se koristi i u klasičnom testiranju za automatizaciju testiranja.

JUNIT BIBLIOTEKA

- JUnit biblioteka – verzija 6 (decembar 2025.): https://docs.junit.org/6.0.1/overview.html



JUNIT BIBLIOTEKA – UVOZ U PROJEKAT

```
<dependencies>
        <!-- JUnit 6 - API -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>6.0.1</version>
            <scope>test</scope>
        </dependency>

        <!-- JUnit 6 - Params -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <version>6.0.1</version>
            <scope>test</scope>
        </dependency>

         <!-- JUnit 6 - Suite -->
        <dependency>
            <groupId>org.junit.platform</groupId>
            <artifactId>junit-platform-suite</artifactId>
            <version>6.0.1</version>
            <scope>test</scope>
        </dependency>
<dependencies>
```

Maven projekat – u pom.xml dodati zavisnosti

PREDNOSTI JUNIT-A

- Odvaja instance test klase za svaki jedinični test.
- Specifična JUnit anotacija koja obezbeđuje inicijalizaciju i redefinisanje metoda: @BeforeEach, @BeforeAll, @AfterEach, @AfterAll.
- Veliki broj različitih metoda za proveru rezultata našeg testa.
- Integracija sa popularnim Java aplikacijama: Eclipse, NetBeans, IntelliJ, Jbuilder.

PISANJE JUNIT SKRIPTE

- Kod za testiranje se piše u Javi.
- Konvencija da testovi idu u *src/test* direktorijum projekta.
- Na početku skripte treba importovati:
  - `import static org.junit.jupiter.api.Assertions.*;`
  - `import org.junit.jupiter.api.*;`
- Statički import služi da se provere (asserts) pišu bez prefiksa klase (to su u stvari statički metodi klase Assert iz JUnit-a).
- Drugi import služi da bi bile vidljive klase JUnit framework-a kao što su Test itd.

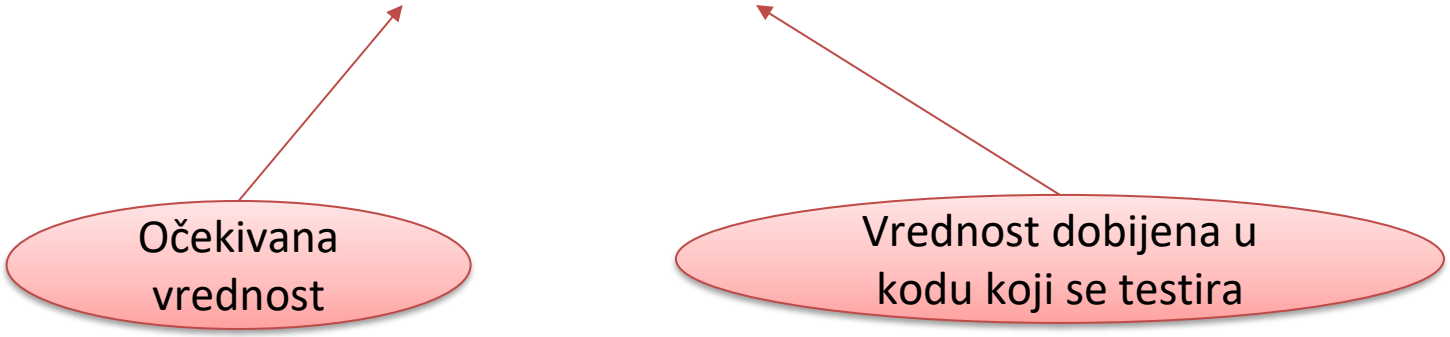
PISANJE TEST PRIMERA

- Svaki poseban test piše se kao bezparametarski javni void metod u test klasi, koji mora da ima @Test prefiks.
- Rezultat izvršavanja koda koji se testira proverava se unutar test metoda pozivom jednog ili više assert metoda definisanih u okviru alata JUnit.

```
@Test
public void add() {
        calculator.add(1);
        calculator.add(1);
        assertEquals(2, calculator.getResult());
}
```

PROVERAVANJE REZULTATA TESTA

- assertEquals(expectedValue, actualValue)



- Ako ne dođe do poklapanja očekivane i dobijene vrednosti, assert baca izuzetak **AssertionFailedError** izveden iz **java.lang.AssertionError** i prekida izvršavanje tekućeg test metoda, ali ne i ostalih.

PROVERAVANJE REZULTATA TESTA

assertEquals(expected, actual, message)

assertEquals(expected, actual, delta)

assertEquals(expected, actual, delta, message)

- Upotrebljava se za double ili float, gde delta predstavlja prihvatljivo odstupanje izmedju očekivane i sračunate vrednosti.

assertFalse(condition)

assertFalse(condition, message)

- Provera da li je uslov false i ukoliko nije, ispisuje poruku message.

assertTrue(condition)

assertTrue(condition, message)

- Provera da li je uslov true i ukoliko nije, ispisuje poruku message.

fail(), fail(message)

- Prekida test i javlja da test nije prošao. Ukoliko je prosleđena poruka message, ispisuje je.

PRIPREMNE I ZAVRŠNE RADNJE (TEST FIXTURE)

- Često imamo dva ili više testova koji koriste isti skup objekata. Ovaj skup objekata naziva se test fixture.
- Upotreba test fixture-a:
  - Kreirati člana klase za svaki objekat iz test fixture-a.
  - Napisati metodu koja postavlja skup radnih objekata (staviti oznaku **@BeforeEach**).
  - Napisati metodu koja čisti skup radnih objekata (staviti oznaku **@AfterEach**).

- Metode obeležene sa @BeforeEach se pozivaju pre svake test metode.
- Metode obeležene sa @AfterEach se pozivaju nakon svake test metode.
- Zašto koristiti? Izbegavaju se bočni efekati između pokretanja test metoda.

PRIPREMNE I ZAVRŠNE RADNJE (TEST FIXTURE)

- Primer:

```
private static Calculator calculator = new Calculator();

@BeforeEach
public void clearCalculator() {
   calculator.clear();
}
```

TESTIRANJE IZUZETAKA

- Za testiranje izuzetaka se koristi metoda assertThrows čiji su parametri klasa očekivanog izuzetka i lambda funkcija gde se izvršava programski kod koji bi trebalo da generiše izuetak.
- Ukoliko se ovaj izuzetak ne generiše ili se pak generiše drugačiji izuzetak, test ne prolazi.

```
@Test
public void divisionByZero() {
        assertThrows(ArithmeticException.class, () ->
        calculator.divide(0));
}
```

OGRANIČENJE VREMENA IZVRŠAVANJA TESTA

- Ukoliko je potrebno definisati maksimalno vreme izvršavanja nekog testa, npr. ukoliko je pogrešno implementirana beskonačna petlja, ono se može definisati, na dva načina.
- Prvi način je pomoću anotacije @Timeout, čiji je nedostatak što određene programske sekvence izvršavanja (poput beskonačne petlje) ne mogu biti prekinute od strane test metode. Primer:

  @Timeout(value = 1, unit = java.util.concurrent.TimeUnit.*SECONDS*)
- Drugi način je korišćenjem assertTimeoutPreemptively metode, kojoj se prosleđuje maksimalno vreme izvršavanja i lambda funkcija sa pozivima koji potencijalno mogu predugo da se izvršavaju. Ona pokreće odvojenu nit, i time garantuje mogućnost prekida izvršavanja originalne niti.

```
@Test
public void sqrtTest() {
    Assertions.assertTimeoutPreemptively(Duration.ofSeconds(1), () -> {
        calculator.squareRoot(16);
        assertEquals(4, calculator.getResult());
    });
}
```

DEAKTIVIRANJE TESTA

- Ako želimo da privremeno preskočimo izvršavanje nekog testa (a da se u finalnom izveštaju to pojavi kao ignorisani test), koristi se oznaka @Disabled.
- U opcionom string parametru, navodi se razlog zašto je test primer deaktiviran.

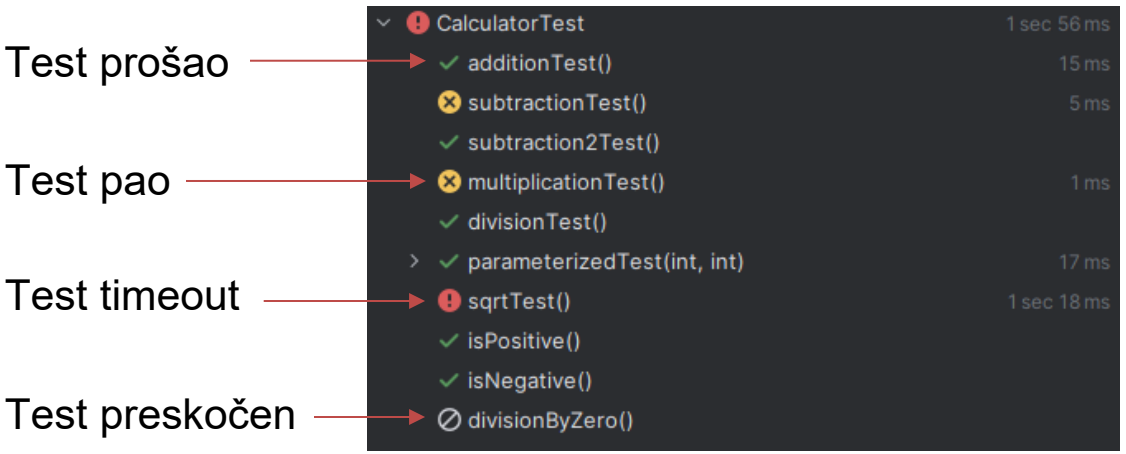
```
@Test
@Disabled("not ready yet")
public void multiply() {
        calculator.add(10);
        calculator.multiply(10);
        assertEquals(100, calculator.getResult());
}
```

IZVRŠAVANJE TESTOVA

- Varijanta iz komandne linije: prevesti kod koji se testira, samu test klasu i pokrenuti test runner klasu JUnit-a.
- Dodati JUnit biblioteku u projekat i izvršiti u okviru programerskog okruženja (IntelliJ, Eclipse, NetBeans, …).

- Primer generisanog izveštaja pri padu testa:

```
org.opentest4j.AssertionFailedError:
Expected :-5
Actual   :-1
```



PARAMETRIZOVANI TESTOVI

- Ako je potrebno da određenu test metodu izvršimo više puta, sa različitim ulaznim podacima, koristimo parametrizovane testove.
- Koristi se @ParameterizedTest anotacija za metodu koja je parametrizovana.
- Parametri se mogu proslediti na više načina:

  1. Kroz @ValueSource anotaciju (ukoliko metoda prihvata samo jedan argument):

     ```
     @ValueSource({„podatak1", „podatak2", „podatak3"})
     ```

  2. Kroz @CsvSource anotaciju (svaki string je jedan set ulaznih podataka, gde su podaci međusobno odvojeni zapetama):

     ```
     @CsvSource({"5,25", "-4,16", "0,0"})
     ```

  3. Kroz @CsvFileSource anotaciju (kojoj se prosleđuju putanja do CSV fajla sa podacima, koja je relativna u odnosu na *test/resources* direktorijum, kao i broj redova fajla koji se preskače):

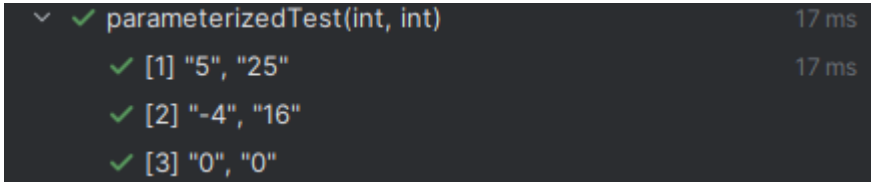
     ```
     @CsvFileSource(resources="/data.csv", numLinesToSkip=1)
     ```

- Metoda koja koristi neku od anotacija za parametrizaciju mora da ima onoliko definisanih parametara koliko joj se prosleđuje kroz kolekciju parametara. Ti podaci se u samoj test metodi koriste na standardan način kroz imena parametara.

PRIMER PARAMETRIZOVANIH TESTOVA

```
@ParameterizedTest
@CsvFileSource(resources = "/data.csv", numLinesToSkip = 1)
public void parameterizedTest(int num, int result) {
    calculator.square(num);
    assertEquals(result, calculator.getResult());
}
```

```
@ParameterizedTest
@CsvSource({"5,25", "-4,16", "0,0"})
public void parameterizedTest(int num, int result) {
    calculator.square(num);
    assertEquals(result, calculator.getResult());
}
```



DEFINISANJE REDOSLEDA IZVRŠAVANJA TESTOVA

- JUnit ne garantuje da će se testovi izvršavati redosledom u kom su metode sa testovima navedene.
- Ukoliko je neophodno definisati redosled izvršavanja, to se radi eksplicitnim definisanjem, korišćenjem @Order anotacije.
- Dodatno je potrebno definisati na nivou čitave klase kakav način određivanja redosleda se koristi korišćenjem @TestMethodOrder anotacije.

```
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class CalculatorTest {
        // ...

        @Test
        @Order(1)
        public void additionTest() {
                calculator.add(5);
                assertEquals(5, calculator.getResult());
        }
        // ...
}
```

DEFINISANJE KOLEKCIJE TESTOVA (TEST SUITE)

- Ako želimo da grupišemo testove iz više klasa u okviru jedne kolekcije testova, potrebno je kreirati novu klasu koja će se ponašati kao kolekcija testova.
- Ova klasa treba da ima @Suite anotaciju, kojom se definiše da je ona kolekcija testova.
- Klase sa testovima se definišu u okviru @SelectClasses anotacije.
- Opciono može da se koristi @SuiteDisplayName anotacija, kojoj se prosleđuje naziv kolekcije testova.

```
@Suite
@SuiteDisplayName("Calculator Suite")
@SelectClasses({
    CalculatorTest.class,
})
public class AllCalculatorTest {
}
```

POKRIVENOST KODA (CODE COVERAGE)

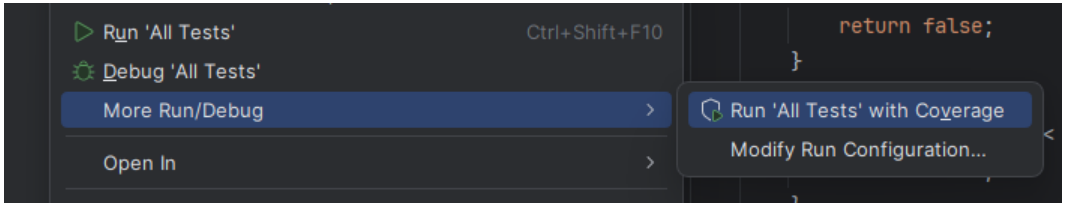
- IntelliJ ima ugrađeni alat za analizu pokrivenosti programskog koda.

- Pokrivenost iskaza - da li je izvršen određeni iskaz u kodu.
- Pokrivenost odluka (grana) - da li je izvršavanje programa prošlo određenom granom.
- Pokrivenost elementarnih uslova - da li je tokom izvršavanja programa u datoj tački ispunjen uslov , kao i njegova negacija.
- Pokrivenost petlji - posmatrana petlja se smatra pokrivenom ako je izvršeno nula, jedna i više od jedne iteracije.
- Pokrivenost klasa - da li je klasa instancirana bar jednom.
- Pokrivenost metoda - da li je metoda pozvana bar jednom.

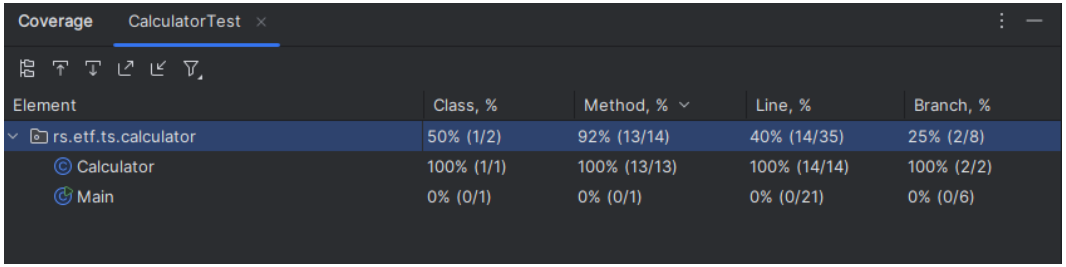
- Alternativa je koristiti drugu biblioteku za analizu pokrivenosti, poput JaCoCo alata, koji može da se uveze kao zavisnost u projekat.

POKRETANJE TESTOVA SA ANALIZOM POKRIVENOSTI

- IntelliJ ima ugrađeni alat za analizu pokrivenosti programskog koda.



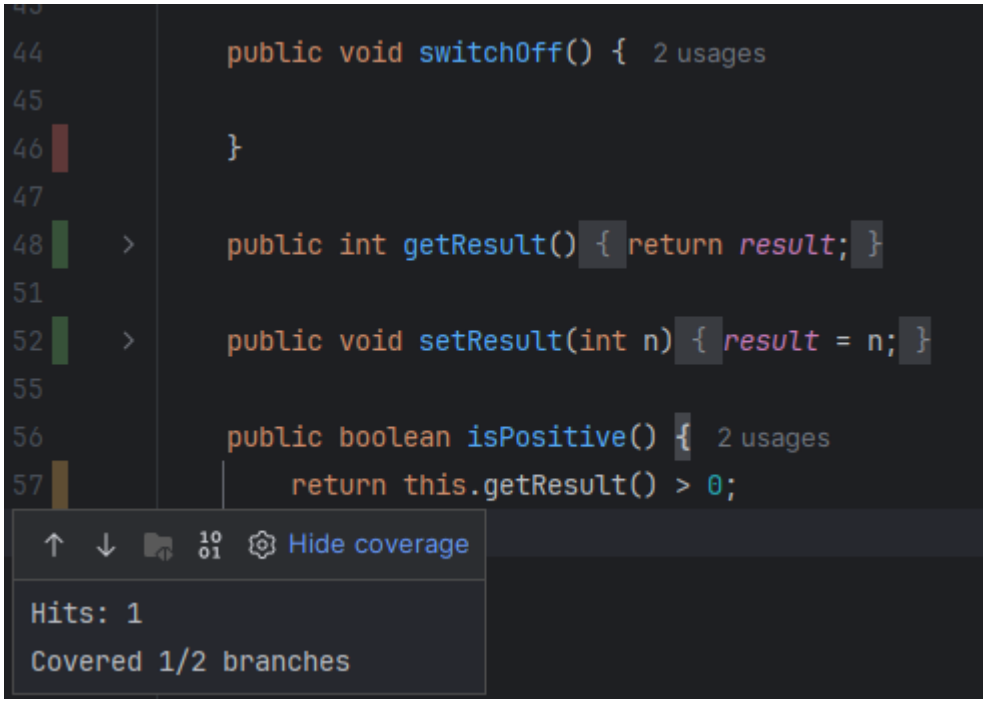
Pokretanje analize



Generisani izveštaj

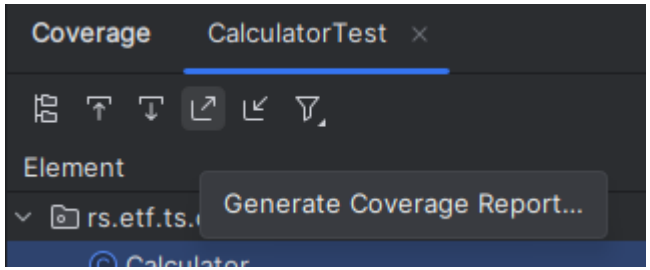
PRIMER IZVEŠTAJA POKRIVENOSTI

- Nakon izvršenih testova, programski kod se markira odgovarajućom bojom u zavisnosti od pokrivenosti:
  - Zelenom, ukoliko je potpuno pokriveno,
  - Žutom, ukoliko je delimično pokriveno (npr. jedna od dve grane) i
  - Crvenom, ukoliko uopšte nije pokriveno.



IZVOZ IZVEŠTAJA O POKRIVENOSTI KODA

- Generisani izveštaj može da se exportuje u vidu .html fajla u preglednijem obliku.



Coverage Summary for Class: Calculator (rs.etf.ts.calculator)

Class
Calculator

```
1  package rs.etf.ts.calculator;
2
3  public class Calculator {
4
5      private static int result;      // Static variable where the result is stored
6
7      public void add(int n) {
8          result = result + n;
9      }
10
11     public void substract(int n) {
12         result = result - 1;         //Bug : should be result = result - n
13     }
14
15     public int substract2(int x, int y) {
16        this.result = x - y;
17        return this.getResult();
18     }
19
20     public void multiply(int n) {
21     }         // not ready yet
22
23
24     public void divide(int n) {
25         result = result / n;
26     }
27
28     public void square(int n) {
29         result = n * n;
30     }
31
32     public void squareRoot(int n) {
33         for (; ;) ;                  //Bug : loops indefinitely
34     }
35
36     public void clear() {           // Cleans the result
37         result = 0;
38     }
39
40     public void switchOn() {
41         result = 0;
42     }
43
44     public void switchOff() {
45
46     }
47
48     public int getResult() {
49         return result;
50     }
51
52     public void setResult(int n) {
53         result = n;
54     }
55
56     public boolean isPositive() {
57         return this.getResult() > 0;
58     }
59 }
```

Current scope: all classes | rs.etf.ts.calculator

Coverage Summary for Package: rs.etf.ts.calculator

Package	Class, %	Method, %	Branch, %	Line, %
rs.etf.ts.calculator	50% (1/2)	81.2% (13/16)	12.5% (1/8)	37.8% (14/37)

Class	Class, %	Method, %	Branch, %	Line, %
Calculator	100% (1/1)	92.9% (13/14)	50% (1/2)	93.3% (14/15)
Main	0% (0/1)	0% (0/2)	0% (0/6)	0% (0/22)

generated on 2025-12-23 17:20

HVALA NA PAŽNJI! ☺

PITANJA ?

Kontakt predavača:
drazen.draskovic@etf.bg.ac.rs
nikolas@etf.bg.ac.rs