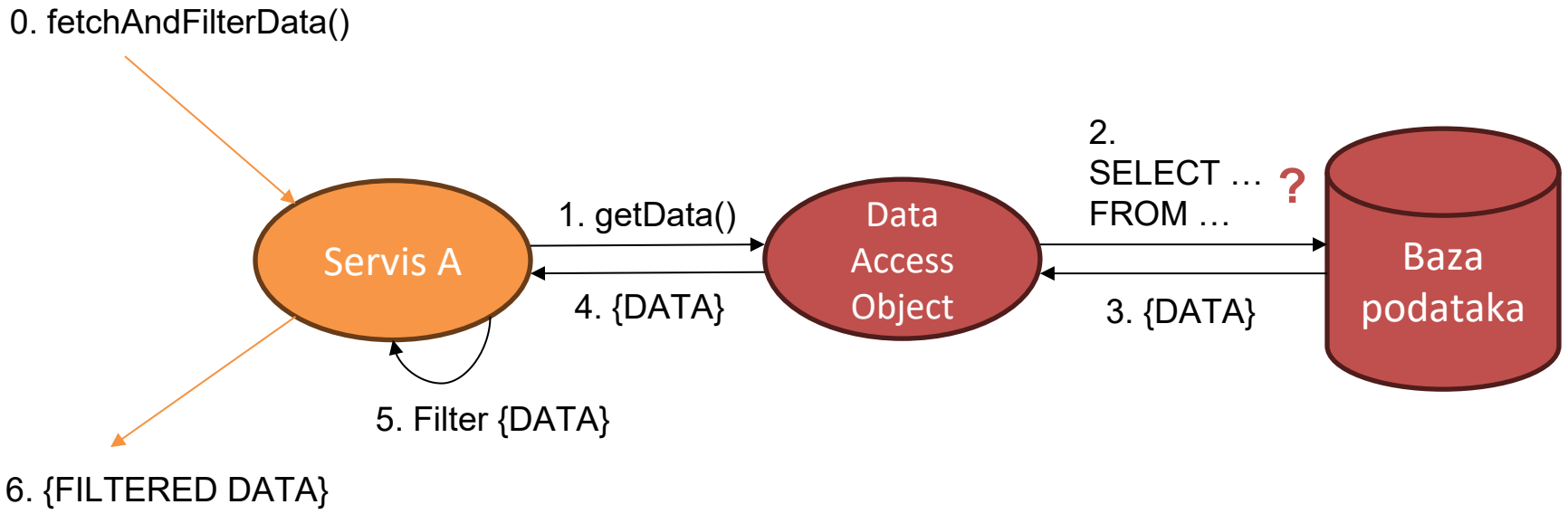
TESTIRANJE SOFTVERA (13S113TS)
LABORATORIJSKA VEŽBA:
ALATI ZA MOCK I MUTACIONO TESTIRANJE

Prof. dr Dražen Drašković
Mast. inž. Nikola Stanković

Univerzitet u Beogradu - Elektrotehnički fakultet

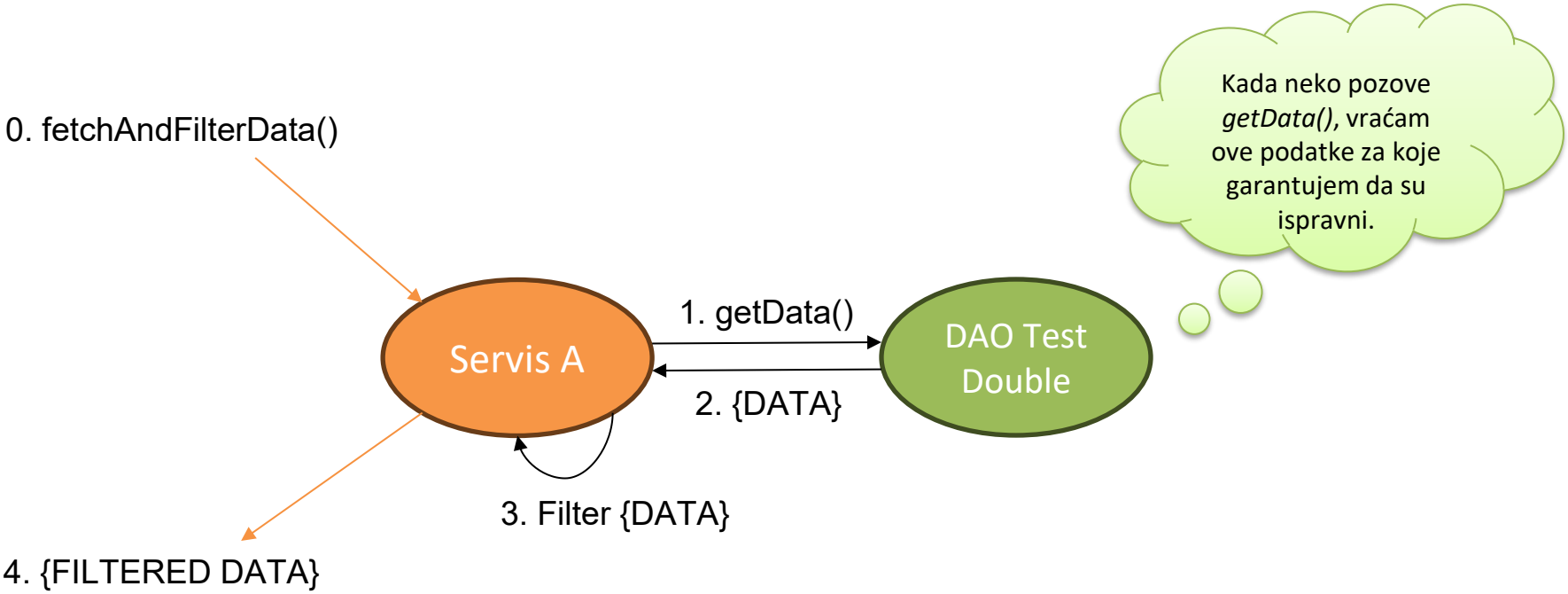
TEST DUBLERI (1)

- Želimo da testiramo metodu *fetchAndFilterData* servisa Servis A.
- Ova metoda poziva metodu *getData* objekta za pristup bazi podataka (*Data Access Object*) kako bi dobila podatke koje dalje treba da obradi i filtrira. Servis A zbog toga zavisi od DAO objekta.



TEST DUBLERI (2)

- Ako želimo jedinično da testiramo Servis A (da izolujemo zavisnosti), kako da budemo sigurni da DAO objekat uopšte koristi dobar upit za dohvatanje podataka iz baze?
- Moramo se osigurati da to bude slučaj – koristimo test dublere.
- Eksplicitno definišemo ponašanje test dublera – ako se desi A, uradi B.



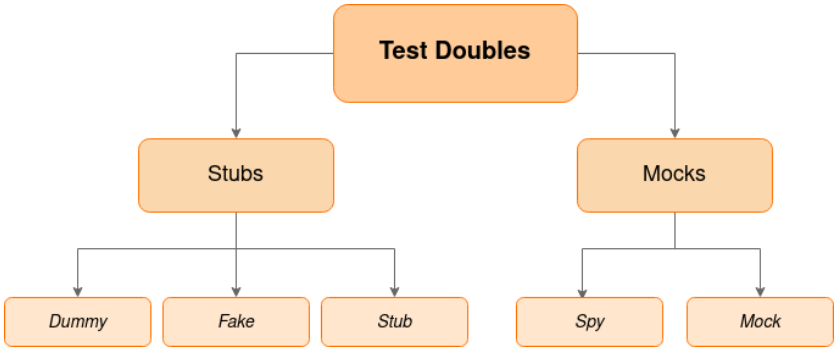
MOCKITO FRAMEWORK

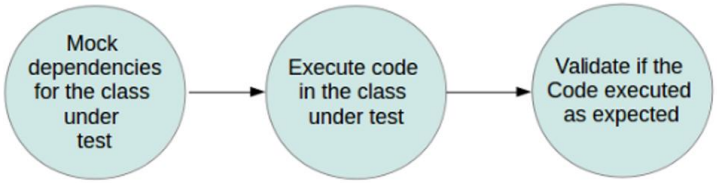
- Mockito je radni okvir dizajniran za Javu, koji omogućava kreiranje i upravljanje test dublerima pri jediničnom testiranju.
- U Mockito-u su mogući sledeći test dubleri:

  - **Stubs** (stab) - objekat sa predefinisanom povratnom vrednošću koju metod treba da vrati tokom testa

  - **Spies** (špijuni) - objekti slični stabovima, ali dodatno se beleži kako su izvršeni (trace)

  - **Mocks** (mokovi) - objekti koji imaju povratne vrednosti izvršenih metoda tokom testa i beleže kako su izvršeni, a takođe mogu izbaciti i izuzetak ako prime neočekivani poziv. Možemo mock-ovati i cele interfejse i klase.





RAD SA MOCK OBJEKTIMA (1)

- Za kreiranje mock objekta se koristi statička metoda *mock* iz biblioteke *Mockito*. Metodi se prosleđuje klasa čiji objekat želi da se mock-uje.
- Klasa koja se testira (*Calculator*) je parametrizovana objektom *ResultStorage*. Zapravo se inject-uje objekat *Calculator* klase ovim mock objektom.

```
@Test
void add_shouldStoreSum() {
    ResultStorage storage = mock(ResultStorage.class);
    when(storage.get()).thenReturn(10);

    Calculator calculator = new Calculator(storage);
    calculator.add(5);

    verify(storage).get();
    verify(storage).set(15);
}
```

```
<!-- Mockito -->
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>5.21.0</version>
    <scope>test</scope>
</dependency>
```

Ne zaboraviti zavisnost u pom.xml!

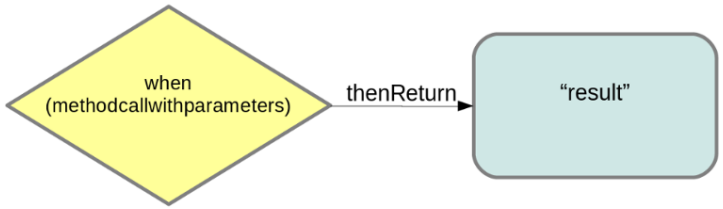
RAD SA MOCK OBJEKTIMA (2)

- Mock objektu eksplicitno definišemo ponašanje metodama *when* i *thenReturn*.
- Kao argument ***when*** metode se prosleđuje poziv koji treba da izazove akciju mock objekta (šta taj mock objekat treba da vrati po pozivu metode). Podatak koji mock objekat vraća se definiše kao argument metode ***thenReturn***.

```
@Test
void add_shouldStoreSum() {
    ResultStorage storage = mock(ResultStorage.class);
    when(storage.get()).thenReturn(10);

    Calculator calculator = new Calculator(storage);
    calculator.add(5);

    verify(storage).get();
    verify(storage).set(15);
}
```



RAD SA MOCK OBJEKTIMA (3)

- Treba proveriti da li je metoda koja se testira interagovala ispravno sa mock objektom, odnosno da li je pozvala sve metode mock objekta koje je trebalo da pozove.

- To proveravamo pozivom metode ***verify*** sa argumentom koji je mock objekat, nad kojom se poziva konkretna metoda za koju ispitujemo da li se pozvala za mock.

```
@Test
void add_shouldStoreSum() {
    ResultStorage storage = mock(ResultStorage.class);
    when(storage.get()).thenReturn(10);

    Calculator calculator = new Calculator(storage);
    calculator.add(5);

    verify(storage).get();
    verify(storage).set(15);
}
```

```
public void add(int n) {
    int result = storage.get();
    result = result + n;
    storage.set(result);
}
```

- Ovim testom smo definisali da mock objekat treba da vrati 10 kada se pozove njegova metoda *get*, a nakon poziva metode *calculator.add* koju testiramo, proverili smo da li su *get* i *set* metode mock objekta pozvane u *add* metodi.

RAD SA MOCK OBJEKTIMA (4)

- Može se eksplicitno definisati da metoda mock objekta ne sme nijednom da se pozove:

```
verify(storage, never()).get();
```

- Može se eksplicitno definisati da metoda mock objekta mora da se pozove tačno određeni broj puta:

```
verify(storage, times(3)).get();
```

- Može se eksplicitno definisati da metoda mock objekta mora da se pozove bar jedanput:

```
verify(storage, atLeastOnce()).get();
```

- Argument koji se prosleđuje mock objektu prilikom poziva ne mora da bude konkretna vrednost, već može da bude čitav opseg, npr. bilo koji ceo broj:

```
verify(storage).set(anyInt());
```

MUTACIONO TESTIRANJE

- Mutaciono testiranje određuje kvalitet skupa napisanih testova, tako što određuje koliko su testovi dobri u otkrivanju mutanata originalnog programskog koda.
- Potrebno je napraviti mutante originalnog programa, mutiranjem određenih iskaza i naredbi.
- Potom se za svaki test primer proverava da li je ponašanje testa identično za originalni program i za mutanta. Ukoliko jeste, mutant je preživeo, i samim tim test nije visokog kvaliteta. Ukoliko se izvršavanje razlikuje, test primer je otkrio mutanta i zbog toga se smatra dobrim test primerom.
- Proverom svih test primera za svakog mutanta, određuje se mutacioni skor – metrika koja na skali od 0 do 1 određuje kvalitet skupa test primera. Što je mutacioni skor veći, to je skup test primera bolji.

$$MS(T) = \frac{|D|}{|L| + |D|}$$

PIT FRAMEWORK (1)

- PIT je Java framework koji omogućava mutaciono testiranje.
- Pre upotrebe, potrebno je definisati zavisnosti u *pom.xml* fajlu.
- U izvornom obliku, PIT framework nema podršku za JUnit 5 (i 6) biblioteku, tako da je potrebno definisati i zavisnost ka plugin-u za JUnit 5 (ujednno i za 6).

```
<!-- PIT -->
<dependency>
        <groupId>org.pitest</groupId>
        <artifactId>pitest</artifactId>
        <version>1.22.0</version>
</dependency>

<!-- PIT JUnit 5 Plugin -->
<dependency>
        <groupId>org.pitest</groupId>
        <artifactId>pitest-junit5-plugin</artifactId>
        <version>1.2.3</version>
        <scope>test</scope>
</dependency>
```

PIT FRAMEWORK (2)

- U *pom.xml* fajlu treba definisati koja je to klasa čiji se mutanti prave (*targetClasses*) i koji su to testovi za datu klasu koje treba izvršiti (*targetTests*).

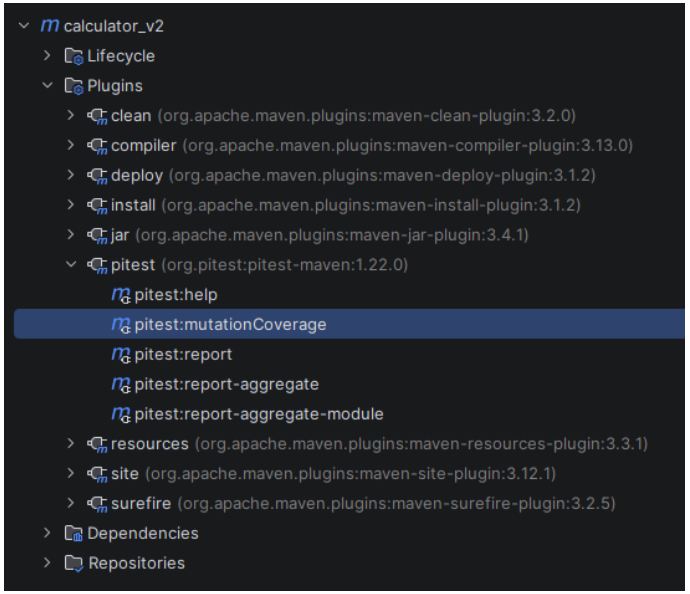
```
<build>
    <plugins>
        <plugin>
            <groupId>org.pitest</groupId>
            <artifactId>pitest-maven</artifactId>
            <version>1.22.0</version>
            <configuration>
                <targetClasses>
                    <param>rs.etf.ts.calculator.Calculator</param>
                </targetClasses>
                <targetTests>
                    <param>rs.etf.ts.calculator.CalculatorTests</param>
                </targetTests>
            </configuration>
        </plugin>
    </plugins>
</build>
```

MUTACIONO TESTIRANJE SA PIT-OM (1)

- Pokretanje analize:

1. Odabrati ***View/Tool Windows/Maven*** opciju u meniju i proširiti tekući Maven projekat.
2. Proširiti ***Plugins*** sekciju, pa isto tako i ***pitest*** sekciju.
3. Duplim klikom na ***pitest:mutationCoverage*** se pokreće plugin za analizu mutacionog skora i generisani izveštaj se smešta u ***target/pit-reports*** direktorijum u okviru projekta.

- Ovo će izvršiti mutaciono testiranje nad klasom ili klasama testova koje su navedene u *targetTests* tagu *pom.xml*-a.



MUTACIONO TESTIRANJE SA PIT-OM (2)

- Preduslovi za pokretanje:

1. Testovi u klasama testova se prvo moraju odvojeno pokrenuti, pre pokretanja plugin-a za analizu.
2. Svi testovi moraju da uspešno prolaze (green tests), kako bi uopšte mogla da se sprovede komparacija izvršavanja nad originalnim programom i mutantima.

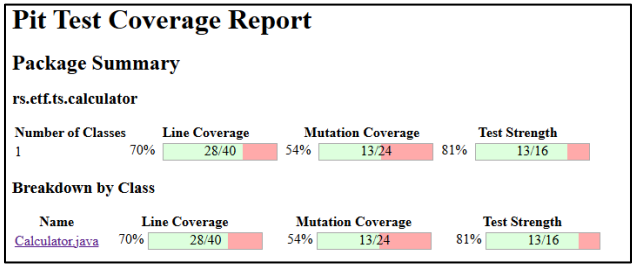
- Primer ispisa u konzoli nakon izvršavanja:

```
============================================================
- Statistics
============================================================
>> Line Coverage (for mutated classes only): 28/40 (70%)
>> 1 tests examined
>> Generated 24 mutations Killed 13 (54%)
>> Mutations with no coverage 8. Test strength 81%
>> Ran 28 tests (1.17 tests per mutation)
```

$$\frac{\text{13 (ubijenih mutanata)}}{\text{24 (ukupno)} - \text{8 (bez pokrivenosti)}} = 0.81$$

MUTACIONO TESTIRANJE SA PIT-OM (3)

- Izveštaju analize se pristupa preko *index.html* stranice u generisanom *pit-report* direktorijumu:



Calculator.java

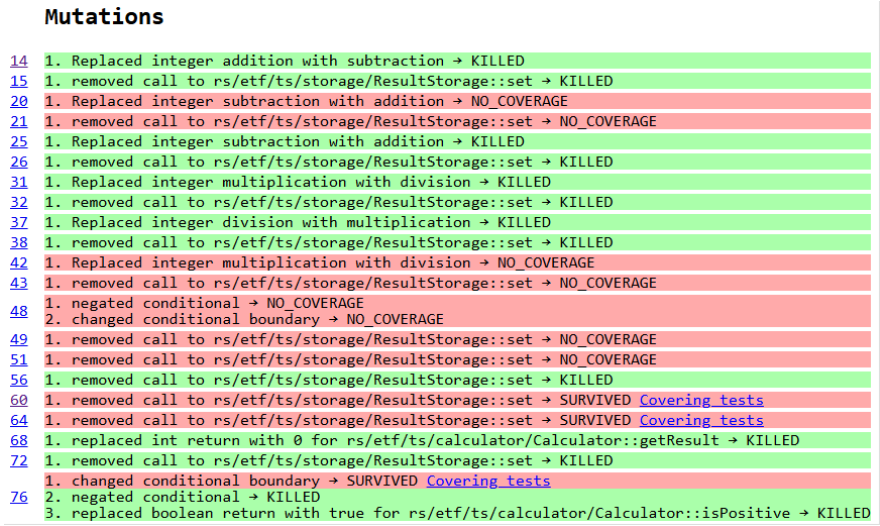
```
1   package rs.etf.ts.calculator;
2   import rs.etf.ts.storage.*;
3
4   public class Calculator {
5
6       private final ResultStorage storage;
7
8       public Calculator(ResultStorage storage) {
9           this.storage = storage;
10      }
11
12      public void add(int n) {
13          1. add : Replaced integer addition with subtraction → KILLED
14  1
15  1       storage.set(result);
16      }
17
18      public void substract(int n) {
19          int result = storage.get();
20  1       result = result - 1;        // BUG: should subtract n
21  1       storage.set(result);
22      }
```

- Broj označen plavom bojom pored odgovarajuće linije predstavlja broj mutanata koji je generisan na osnovu te linije.
- Hover preko datog broja prikazuje koja mutacija je izvršena kao i status mutanta na kraju analize.

MUTACIONO TESTIRANJE SA PIT-OM (4)

- U izveštaju se u sekciji za mutacije može videti status svih generisanih mutanata.
- Mogući statusi mutanata:

  - **Killed** – mutant detektovan
  - **Survived** – mutant nije detektovan
  - **No coverage** – nijedan test nije izvršio datu metodu čiji je mutant kreiran
  - **Non viable** – nevalidan bajtkod za JVM datog mutanta (ne može da se izvrši)
  - **Timed out** – predugo izvršavanje, npr. mutacijom se generisala beskonačna petlja



  - **Memory error** – previše memorije iskorišćeno, npr. call stack se prepunio u mutantu
  - **Run error** – greška pri izvršavanju mutanta

- Treba imati što manje mutanata čiji je status Survived ili No coverage! Na osnovu generisanog izveštaja treba dodati nove ili izmeniti postojeće testove tako da se mutacioni skor poveća.

MUTACIONO TESTIRANJE SA PIT-OM (5)

- Na kraju izveštaja su navedene sve mutacije koje su dolazile u obzir prilikom generisanja mutanata. Podrazumevano ima 11 tipova mutacija i to su:

1. **Conditionals Boundary** – mutacija relacionih operatora (<, <=, >, >=)
2. **Empty Returns** – vraća „praznu“ vrednost u zavisnosti od tipa povratne vrednosti (0, prazan string, prazna lista, …)
3. **False Returns** – vraća False kao povratnu vrednost za metode ako im je tip povratne vrednosti boolean
4. **True Returns** – vraća True kao povratnu vrednost za metode ako im je tip povratne vrednosti boolean
5. **Null Returns** – vraća Null kao povratnu vrednost
6. **Primitive Returns** – vraća 0 ukoliko je tip povratne vrednosti int, short, long, char, float ili double

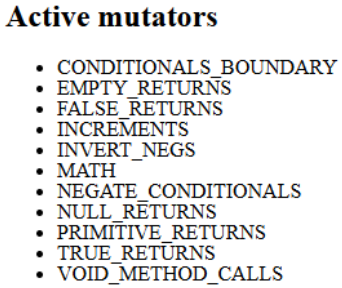
Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

MUTACIONO TESTIRANJE SA PIT-OM (6)

- Na kraju izveštaja su navedene sve mutacije koje su dolazile u obzir prilikom generisanja mutanata. Podrazumevano ima 11 tipova mutacija i to su:

7. **Increments** – menja inkrement sa dekrementom i obrnuto (i++, ++i, i--, --i)
8. **Invert Negatives** – uklanja minus kod negiranih promenljivih (npr. –i se mutira u i)
9. **Math** – mutacija aritmetičkih operatora (+ u -; - u +; * u /; / u *; % u *; & u |; | u &; ^ u &; << u >>; >> u <<; >>> u <<)
10. **Negate Conditionals** – slično mutaciji relacionih operatora, ali se mutira u kontra uslov (== u !=; != u ==; > u <=; >= u <; < u >=; <= u >)
11. **Void Method Calls** – uklanjanje poziva metode koja nema povratnu vrednost (void)



MUTACIONO TESTIRANJE SA PIT-OM (7)

- Mutacije koje se uzimaju u obzir se mogu konfigurisati u *pom.xml* fajlu:

```
<configuration>
        <mutators>
                <mutator>PAKET_MUTACIJA</mutator>
        </mutators>
</configuration>
```

- Pri čemu paketi mutacija mogu biti:

  - *DEFAULTS (podrazumevano)*
  - *OLD_DEFAULTS*
  - *STRONGER*
  - *ALL*

KORISNI LINKOVI

- Mockito dokumentacija:

  https://javadoc.io/doc/org.mockito/mockitocore/latest/org.mockito/org/mockito/Mockito.html

- PIT dokumentacija:

  https://pitest.org/quickstart/

- Više informacija o mutacionim operatorima PIT alata:

  https://pitest.org/quickstart/mutators/

HVALA NA PAŽNJI! ☺

PITANJA ?

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