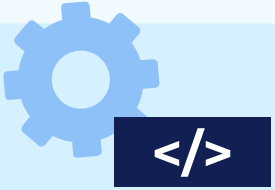
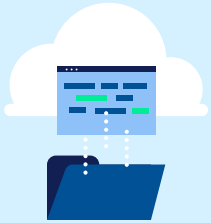
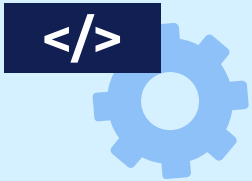
UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI
FAKULTET



TESTIRANJE BAZIRANO NA SVOJSTVIMA

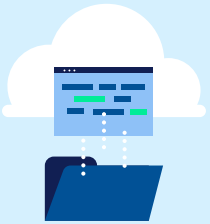


TESTIRANJE SOFTVERA



01

TESTIRANJE SVOJSTAVA



PODSETNIK

- Metoda klasa ekvivalencije podrazumeva da se ulazni podaci dele na klase za koje se očekuje slično ili isto ponašanje programa.
  - Test primere smo konstruisali na osnovu reprezentativnih vrednosti iz legalnih i nelegalnih klasa.
- Metoda graničnih vrednosti proširuje metodu klasa ekvivalencije, tako što stavlja akcenat na granične vrednosti među klasama.
  - Test primere smo konstruisali tako da se obuhvati svaka granica i vrednosti u njenoj okolini.
- Da li smo testirali svaki mogući ulaz biranjem najmanje jednog reprezentativnog predstavnika svake klase?



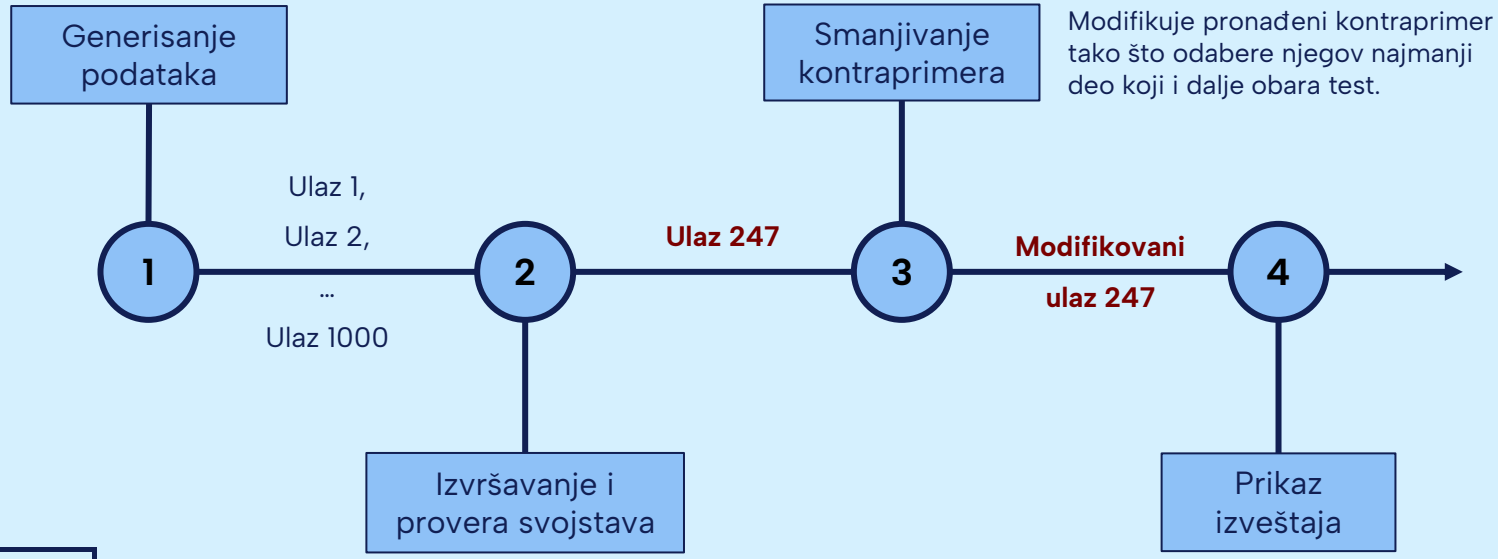
PRISTUPI TESTIRANJU

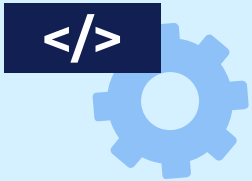
- **Klasičan pristup testiranju** (engl. *example-based testing*) podrazumeva pisanje konkretnih primera kako program treba da se ponaša za odgovarajući ulaz:
  - Definiše se ulaz u program i očekivani izlaz sa kojim se poredi pravi izlaz programa.
  - Proverava se samo jedan konkretan slučaj.
  - Šta ako želimo da napišemo mnogo više test primera za svaku klasu ekvivalencije?
- **Testiranje bazirano na svojstvima** (engl. *property-based testing*) omogućava automatizovanu proveru definisanih pravila za podatke:
  - Definiše se svojstvo (pravilo) koje podatak ili program mora da zadovolji.
  - Korišćenjem generatora podataka se generiše veliki broj ulaznih podataka za koje se ispituje zadovoljenost napisanih pravila.

TESTIRANJE BAZIRANO NA SVOJSTVIMA

- Kako se testiranje bazirano na svojstvima nadovezuje na metode klasa ekvivalencije i graničnih vrednosti?
  - Ovim metodama zapravo definišemo pravila koje podaci moraju da zadovolje:
    - Klase ekvivalencije: „Broj godina je ceo broj.“, „Broj godina je između 1 i 100.“
    - Svojstvo: „Ovaj podatak mora da bude ceo broj između 1 i 100.“
    - Generator podataka generiše cele brojeve u opsegu od 1 do 100.
- Prednost ovog pristupa je mnogo veći broj testiranih ulaznih podataka u odnosu na broj napisanih testova.
- Ograničenje ovog pristupa je da svojstva (pravila) moraju biti precizno definisana.
- Testiranje bazirano na svojstvima nije zamena za pažljivo projektovanje testova!

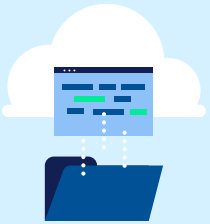
IZVRŠAVANJE TESTOVA





02

ALATI JUNIT I JQWIK



ALAT JUNIT

- **JUnit** je radni okvir i biblioteka za pisanje i pokretanje testova u programskom jeziku Java.
- Uglavnom se koristi za pisanje jediničnih testova.

- Jedan test primer napisan u JUnit alatu čine sledeće tri celine:
  1. **Priprema** ulaznih podataka i objekata potrebnih za izvršavanje testa.
  2. **Izvršavanje** obrade kojom se neka jedinica testira (poziv metode, itd.).
  3. **Provera** da li se dobijeni rezultat i očekivani rezultat poklapaju.



PISANJE TESTOVA U JUNIT-U

- Metoda označena anotacijom **@Test** predstavlja test primer.
- Jedna klasa za testiranje može sadržati više test metoda.
- U telu test metode se pozivaju funkcije koje se testiraju.
- Kada se obrada završi, proverava se poklapanje očekivanog i dobijenog rezultata **assert** metodama.
- Ukoliko **assert** tvrdnja nije zadovoljena, test je pao.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import static rs.ac.bg.etf.ts.Kalkulator;

class TestoviSabiranje {
    @Test
    public void sabiranje() {
        int rezultat = Kalkulator.saberi(2, 3);
        assertEquals(5, rezultat);
    }
}
```

ASSERT METODE



`assertEquals(expected, actual)`

Proverava da li se očekivani rezultat **`expected`** i dobijeni rezultat **`actual`** poklapaju.



`assertTrue(condition)`

Proverava da li je prosleđeni uslov **`condition`** jednak *true*.



`assertFalse(condition)`

Proverava da li je prosleđeni uslov **`condition`** jednak *false*.



`assertNotNull(value)`

Proverava da li je prosleđena vrednost **`value`** različita od *null*.

- Opciono se može proslediti argument `message`, koji obezbeđuje da se ispiše prosleđena poruka u slučaju pada test primera.

ALAT JQWIK

- Alat **jqwik** nadograđuje JUnit i na taj način omogućava testiranje bazirano na svojstvima.

- Osnovno proširenje ovog alata čine tri nove anotacije:
  1. **`@Property`** – Naznaka da metoda opisuje svojstvo. Koristi se umesto **`@Test`** anotacije.
  2. **`@ForAll`** – Naznaka da jqwik treba da generiše ulazne podatke za datu metodu.
  3. **`@Provide`** – Naznaka da metoda predstavlja generator podataka.



PISANJE TESTOVA U JQWIK-U

- Metoda označena anotacijom **@Property** predstavlja svojstvo za testiranje.
- Anotacije **@ForAll** i **@IntRange** nekog parametra označavaju da se taj parametar generiše.
- U ovom slučaju generisaće se celi brojevi između 0 i 100 i telo metode će biti izvršeno za svaki od generisanih podataka.

```
import net.jqwik.api.*;
import net.jqwik.api.constraints.*;
import static rs.ac.bg.etf.ts.Kalkulator;

class TestoviPoeni {
    @Property
    void validniPoeniSuUOpsegu(
        @ForAll
        @IntRange(min = 0, max = 100)
        int poeni
    ) {
        assertTrue(Kalkulator.proveri(poeni));
    }
}
```

GENERATORI PODATAKA U JQWIK-U

OSNOVNE ANOTACIJE

- Metoda označena anotacijom **@Provide** predstavlja odvojeni generator podataka.
- Klasa **Arbitrary<T>** predstavlja sam generator.
- Anotaciji **@ForAll** može da se doda generator korišćenjem naziva metode koja je anotirana sa **@Provide**.
- Za jednostavne opsege dovoljan je **@IntRange**, dok je za složenije podatke pogodnije koristiti generatore sa **@Provide**.

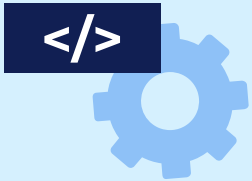
```
@Provide
Arbitrary<Integer> validniPoeni() {
    return Arbitraries.integers().between(0, 100);
}

@Property
void validniPoeniSuUOpsegu(
    @ForAll("validniPoeni") int poeni
) {
    assertTrue(Kalkulator.proveri(poeni));
}
```

GENERATORI PODATAKA U JQWIK-U

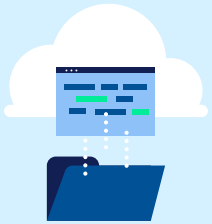
NAJČEŠĆA OGRANIČENJA

- Najčešća ograničenja za **brojeve**:
  - **`@IntRange/@DoubleRange(min=3, max=50)`** – Opseg brojeva od `min` do `max`.
  - **`@Positive/@Negative`** – Isključivo pozitivni/negativni brojevi.
- Najčešća ograničenja za **stringove**:
  - **`@StringLength(min=3, max=50)`** – String dužine od `min` do `max`.
  - **`@NotBlank`** – String neće biti prazan.
  - **`@AlphaChars/@NumericChars`** – String sadrži samo slova/brojeve.
- Najčešća ograničenja za **kolekcije**:
  - **`@Size(min=1, max=10)`** – Broj elemenata kolekcije je od `min` do `max`.
  - **`@UniqueElements`** – Neće biti duplikata u kolekciji.



03

ZADACI



ZADATAK 1 – POPUST

POSTAVKA

- Data je metoda **`double applyDiscount(double price, int discount)`** koja određuje konačnu cenu proizvoda koja se dobija kada se prosleđena cena **`price`** umanji za **`discount`** procenata.
- Potrebno je testirati datu metodu korišćenjem alata JUnit u kombinaciji sa alatom jqwik. Napisati segment programskog koda u Javi sa jednim jqwik svojstvom za validne klase ekvivalencije ulaznih podataka.



ZADATAK 1 – POPUST

REŠENJE

```
@Property
void checkFinalPrice(
    @ForAll
    @DoubleRange(min = 1.0, max = 1000000.0)
    double price,
    @ForAll
    @IntRange(min = 0, max = 100)
    int discount
) {
    double result = applyDiscount(price, discount);
    double expected = price * (1 - discount / 100.0);
    assertEquals(expected, result);
}
```

ZADATAK 2 – KORISNIČKO IME

POSTAVKA

- Data je metoda **`boolean isValidUsername(String username)`** koja vrši proveru ispravnosti unetog korisničkog imena pri registraciji korisnika. Korisničko ime je validno ukoliko se sastoji od najmanje 5, a najviše 12 karaktera i ukoliko sadrži isključivo slova i cifre. U nastavku su primeri validnih i nevalidnih korisničkih imena:
  - „nikola7“ – validno korisničko ime
  - „ab12“ – nevalidno korisničko ime
  - „ana_marija“ – nevalidno korisničko ime
- Potrebno je testirati datu metodu korišćenjem alata JUnit u kombinaciji sa alatom jqwik. Napisati segment programskog koda u Javi sa jqwik svojstvima za dve nevalidne klase ekvivalencije: korisničko ime je kraće od 5 karaktera i korisničko ime sadrži karakter „_“. Za svaku klasu koristiti poseban generator podataka i svojstvo koje proverava ispravnost korisničkog imena.

ZADATAK 2 – KORISNIČKO IME

REŠENJE (1)

```
@Property
void shortUsernamesInvalid(
    @ForAll
    @StringLength(min=1, max=4)
    @AlphaChars
    @NumericChars
    String username
) {
    assertFalse(isValidUsername(username));
}
```

```
@Property
void underscoreUsernamesInvalid(
    @ForAll
    @StringLength(min=5, max=12)
    @AlphaChars
    @NumericChars
    @Chars({'_'})
    String username
) {
    assertFalse(isValidUsername(username));
}
```

- Da li je drugo svojstvo ispravno?

ZADATAK 2 – KORISNIČKO IME

REŠENJE (2)

- Kada se generatoru prosledi više skupova simbola koje on treba da uključi u string, on pravi uniju tih skupova i iz unije tih skupova izvlači simbole za generisanje.
- Gde nastaje problem? Može da se desi da se „_“ uopšte ne nađe u rezultujućem stringu. U tom slučaju bi korisničko ime bilo ispravno!
- Generator ne garantuje da će se svaki karakter iz svakog skupa pojaviti u rezultujućem stringu.

```
@Property
void underscoreUsernamesInvalid(
    @ForAll
    @StringLength(min=5, max=12)
    @AlphaChars
    @NumericChars
    @Chars({'_’})
    String username
) {
    if (username.contains("_"))
        assertFalse(isValidUsername(username));
}
```

ZADATAK 3 – LOZINKA

POSTAVKA

- Data je metoda **`boolean isValidPassword(String password)`** koja vrši proveru ispravnosti unete lozinke pri registraciji korisnika. Lozinka je validna ukoliko se sastoji od najmanje 8, a najviše 16 karaktera i ukoliko sadrži barem jedno veliko slovo i barem jednu cifru. U nastavku su primeri validnih i nevalidnih lozinki:
  - „Pass1234“ – validna lozinka
  - „passw0rd“ – nevalidna lozinka
  - „PASSword“ – nevalidna lozinka
- Potrebno je testirati datu metodu korišćenjem alata JUnit u kombinaciji sa alatom jqwik. Napisati segment programskog koda u Javi sa jqwik svojstvima za validne klase ekvivalencije, odnosno za validne lozinke. Koristiti jedan generator podataka i jedno svojstvo koje proverava ispravnost unete lozinke.



ZADATAK 3 – LOZINKA

REŠENJE (1)

```
@Provide
Arbitrary<String> validPasswords() {
    Arbitrary<String> chars = Arbitraries.strings().alpha().ofMinLength(7).ofMaxLength(13);
    Arbitrary<String> digits = Arbitraries.strings().numeric().ofMinLength(1).ofMaxLength(3);

    return Combinators.combine(chars, digits)
        .as((c, d) -> c.substring(0, 1).toUpperCase() + c.substring(1) + d);
}

@Property
void validPasswordsCheck(@ForAll("validPasswords") String password) {
    assertTrue(isValidPassword(password));
}
```

- Ovakav generator generiše stringove u kojima se uvek prvo nalazi jedno veliko slovo, pa mala ili velika slova, i tek onda cifre na kraju. Može li ovo još da se generalizuje?

ZADATAK 3 – LOZINKA

REŠENJE (2)

```
@Provide
Arbitrary<String> validPasswordsShuffled() {
    Arbitrary<String> chars =
        Arbitraries.strings().alpha().ofMinLength(7).ofMaxLength(13);

    Arbitrary<String> digits =
        Arbitraries.strings().numeric().ofMinLength(1).ofMaxLength(3);

    return Combinators.combine(chars, digits)
        .as((c, d) -> {
            String rawPassword = c.substring(0, 1).toUpperCase() + c.substring(1) + d;
            List<String> characterList = Arrays.asList(rawPassword.split(""));
            Collections.shuffle(characterList);
            return String.join("", characterList);
        });
}
```

ZADATAK 4 – SORTIRANJE LISTE

POSTAVKA

- Data je metoda **`sortNumbers(List<Integer> numbers)`** koja sortira prosleđenu listu celih brojeva u neopadajućem poretku. Metoda kao povratnu vrednost vraća novu listu istog tipa.
- Potrebno je testirati datu metodu korišćenjem alata JUnit u kombinaciji sa alatom jqwik. Napisati segment programskog koda u Javi sa jqwik svojstvima za proveru ispravnosti date metode. Koristiti jedan generator podataka i jedno svojstvo koje proverava ispravnost metode za sortiranje. Generator podataka treba da generiše liste od maksimalno 30 elemenata i sa pozitivnim i sa negativnim brojevima.
- Primeniti *model-based oracle* pristup, tako da referentni model za sortiranje bude metoda **`sort`** iz Javine biblioteke **`Collections`**.



ZADATAK 4 – SORTIRANJE LISTE

REŠENJE

```
@Provide
Arbitrary<List<Integer>> numberList() {
    return Arbitraries.integers()
        .between(-1000, 1000)
        .list().ofMinSize(0).ofMaxSize(30);
}

@Property
void sortListCheck(@ForAll("numberList") List<Integer> list) {
    List<Integer> expected = new ArrayList<>(list);
    Collections.sort(expected);

    List<Integer> actual = sortNumbers(list);
    assertEquals(expected, actual);
}
```

UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI
FAKULTET

PITANJA?

https://si3ts.etf.bg.ac.rs/

nikolas@etf.bg.ac.rs

