SOFTVERSKO INŽENJERSTVO

školska 2024/2025 godina

Vežba 14: JUnit testiranje koda

1. Šta je testiranje softvera?

Testiranje softvera je **proces verifikacije i validacije softverskog proizvoda**. Cilj je da se osigura da softver:

- Radi prema specifikacijama i zahtevima,
- Ne sadrži kritične greške,
- Ponaša se predvidljivo u svim uslovima.

Verifikacija znači proveru da li softver ispravno implementira funkcionalnosti.

Validacija znači proveru da li softver zadovoljava potrebe korisnika.

Testiranje se radi kako bi se smanjio broj grešaka koje bi korisnici mogli da otkriju u produkciji, što može dovesti do skupih popravki, gubitka poverenja i lošeg korisničkog iskustva. Kvalitetno testiranje pomaže timu da otkrije probleme u ranoj fazi razvoja, kada je njihovo otklanjanje znatno brže i jeftinije.

2. Šta je unit testiranje?

Unit testiranje (testiranje jedinica) fokusira se na najmanje delove softverskog koda – obično pojedinačne metode ili klase – i testira ih u izolaciji. Cilj je da se proveri da li **svaka jedinica koda radi kako je očekivano**.

Primer:
Zamislimo da imamo metodu add(int a, int b) koja sabira dva broja. Unit test bi proverio da li metoda zaista vraća zbir dva broja. Ako metoda nije ispravno implementirana, test će pasti i ukazati na problem. Na ovaj način se greške mogu uočiti odmah nakon promene u kodu, što olakšava održavanje i razvoj softvera.

Zašto je bitno testirati jedinice?

- Brzo se detektuju greške na najnižem nivou.
- Omogućava se pouzdano pravljenje kompleksnijih sistema jer su osnovni delovi provereni.
- Olakšava refaktorisanje i održavanje koda jer testovi brzo pokažu da li je nešto pokvareno.
- Unit testovi služe i kao dokumentacija, jer drugi programeri mogu brzo razumeti šta određena metoda treba da radi gledajući njene testove.
- Pravilno napisan skup testova pomaže da se smanji potreba za ručnim testiranjem, štedeći vreme tokom razvoja.

3. Šta je JUnit?

JUnit je **najpoznatija Java biblioteka za pisanje unit testova**. Ona programerima pruža alate i API-je za:

- Definisanje testnih metoda koje proveravaju ispravnost koda,
- Automatsko izvršavanje testova,
- Poređenje očekivanih i stvarnih rezultata,
- Prikazivanje informacija o uspehu ili neuspehu testova.

JUnit 5 je aktuelna verzija i sastoji se od tri ključna dela:

Deo	Opis
JUnit Platform	Infrastruktura za pokretanje testova
JUnit Jupiter	Novi API za pisanje testova i novih funkcionalnosti
JUnit Vintage	Kompatibilnost sa starim JUnit 3 i 4 testovima

JUnit koristi anotacije (kao što je @Test) da označi koje metode su testovi. Takođe, omogućava upotrebu **assert** metoda za proveru rezultata.

Pisanje testova uz JUnit je deo profesionalnih praksi koje se primenjuju u realnim projektima, jer obezbeđuje kontrolu kvaliteta i stabilnost koda.

Uz JUnit je moguće automatizovati hiljade testova i integrisati ih u CI/CD procese (npr. GitHub Actions, Jenkins).

4. Assert metode u JUnit Jupiter-u

Assert metode služe za proveru da li je stanje ili rezultat koda onakav kakav očekujemo.

Metoda	Opis	Primer korišćenja
assertEquals	Proverava da su očekivana i stvarna vrednost jednake	assertEquals(5, calc.add(2, 3))
assertTrue	Proverava da je logički izraz tačan	assertTrue(calc.isEven(4))
assertFalse	Proverava da je logički izraz netačan	assertFalse(calc.isEven(5))
assertNotNull	Proverava da objekat nije null	assertNotNull(someObject)
assertNull	Proverava da je objekat null	assertNull(maybeNullObject)
assertThrows	Proverava da li metoda baca određeni izuzetak	assertThrows(IllegalArgumentException.class, () -> ...)
assertArrayEquals	Poredi dva niza da li su identična	assertArrayEquals(new int[]{1,2}, arrayMethod())

Korišćenjem assert metoda, programer može eksplicitno da izrazi šta metoda treba da vrati i pod kojim uslovima. Na taj način, svaka greška se odmah prepoznaje tokom testiranja, bez potrebe za ručnim proverama. Na primer, ako očekujemo da metoda isEven(4) vrati true, a desi se suprotno, assertTrue će prijaviti neuspeh testa i prikazati korisne informacije o tome šta nije prošlo. Ova preciznost i automatizacija čine assert metode ključnim alatom za izgradnju pouzdanog, stabilnog i lako održivog softverskog sistema.

💻 Praktični deo

1. Podešavanje projekta sa JUnit 5

Ukoliko koristite Maven za upravljanje projektom, u fajl pom.xml dodajte:

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.10.0</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Ova zavisnost omogućava korišćenje JUnit 5 biblioteke samo u testnom okruženju, bez uticaja na produkcijski kod.

U IDE-u kao što su IntelliJ IDEA ili Eclipse, testove možete pokretati direktno klikom na metodu ili klasu.

---

2. Klasa koju testiramo – Calculator

Napravimo klasu Calculator koja sadrži jednostavne matematičke metode:

```
public class Calculator {
    // Metoda za sabiranje dva broja
    public int add(int a, int b) {
        return a + b;
    }
    // Metoda za deljenje dva broja. Baca grešku ako se deli sa nulom
    public int divide(int a, int b) {
        if (b == 0) throw new
                        IllegalArgumentException("Cannot divide by zero!");
        return a / b;
    }
```

```
    // Metoda proverava da li je broj paran
    public boolean isEven(int number) {
        return number % 2 == 0;
    }
}
```

---

3. JUnit test klasa – CalculatorTest

Sada ćemo napisati testove koji proveravaju da li metode u klasi Calculator rade ispravno.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class CalculatorTest {

    // Kreiramo instancu klase Calculator koju ćemo testirati
    Calculator calc = new Calculator();

    // Testira metodu add
    @Test
    public void testAddition() {
        // Očekujemo da zbir 2 i 3 bude 5
        assertEquals(5, calc.add(2, 3), "2 + 3 should be 5");
    }

    // Testira metodu divide sa ispravnim argumentima
    @Test
    public void testDivision() {
        // Očekujemo da 10 podeljeno sa 5 bude 2
        assertEquals(2, calc.divide(10, 5));
    }
```

```
    // Testira da li metoda divide baca grešku kada delimo sa nulom
    @Test
    public void testDivisionByZero() {
        // Očekujemo IllegalArgumentException ako je drugi parametar 0
        assertThrows(IllegalArgumentException.class, () ->
                                                    calc.divide(10, 0));
    }

    // Testira da li isEven vraća true za paran broj
    @Test
    public void testIsEvenTrue() {
        assertTrue(calc.isEven(8)); // 8 je paran broj, test treba da prođe
    }

    // Testira da li isEven vraća false za neparan broj
    @Test
    public void testIsEvenFalse() {
        assertFalse(calc.isEven(7)); // 7 je neparan br, test treba da prođe
    }
}
```

Test testDivisionByZero

- Ova metoda koristi assertThrows, što znači da očekujemo da će se u telu lambda izraza baciti određeni izuzetak.
- Ako metoda divide ne baci IllegalArgumentException kada delimo sa nulom, test će pasti.
- Ovo je važno za proveru da li je kod zaista bezbedan i da ne dozvoljava nevalidne operacije.

Testovi testIsEvenTrue i testIsEvenFalse

- Pokazuju da možemo testirati i logičke vrednosti, odnosno true/false izraze.
- Ovim testovima osiguravamo da metoda isEven radi kako treba za različite ulazne vrednosti.