

DODATNI PRIMERI JQWIK GENERATORA

Primer 1. Najviše jedna tačka bilo gde u lokalnom delu

```
private static final Pattern EMAIL_REGEX =  
    Pattern.compile("^[a-zA-Z0-9]*\\?.*[a-zA-Z0-9]*@[a-zA-Z0-9]+\\. [a-zA-Z]{2,6}$");
```

Jqwik Generator i Kombinator

```
// Generator za čiste tekstualne delove (samo slova i brojevi, bez tačke)
```

```
@Provide
```

```
Arbitrary<String> cistTekst() {  
    return Arbitraries.strings().alpha().numeric().ofMinLength(0).ofMaxLength(7);  
}
```

```
// Generator koji nasumično bira da li će vratiti prazan string "" ili tačku "."
```

```
@Provide
```

```
Arbitrary<String> opcionaTacka() {  
    return Arbitraries.of("", ".");  
}
```

```
@Provide
```

```
Arbitrary<String> legalneEmailAdreseSaNajviseJednomTackom() {  
    return Combinators.combine(  
        cistTekst(), // Prvi deo lokalnog dela (npr. "ime")  
        opcionaTacka(), // Tačka koja može biti "" ili "."  
        cistTekst(), // Drugi deo lokalnog dela (npr. "prezime")  
        validanDomen(),  
        validnaEkstenzija()  
    ).as((d1, tacka, d2, domen, ekstenzija) -> {  
        String lokalniDeo = d1 + tacka + d2;  
  
        // Ivica: jqwik može da generiše d1="", tacka="" i d2="", što daje prazan lokalni deo.  
        // Pošto imejl mora imati bar jedan karakter pre @, ako je prazan, ubacićemo jedno slovo.  
        if (lokalniDeo.isEmpty()) {  
            lokalniDeo = "a";  
        }  
  
        return lokalniDeo + "@" + domen + "." + ekstenzija;  
    });  
}
```

Primer 2. Opciona tačka ne sme biti na prvom i poslednjem mestu lokalnog dela

```
private static final Pattern EMAIL_REGEX =  
    Pattern.compile("^( [a-zA-Z0-9]+| [a-zA-Z0-9]+\\. [a-zA-Z0-9]+ )@[a-zA-Z0-9]+\\. [a-zA-Z]{2,6}$");
```

- [a-zA-Z0-9]+ Prva alternativa (čist tekst bez tačke, dužine bar 1).
- | Znak za ILI.
- [a-zA-Z0-9]+\\. [a-zA-Z0-9]+ Druga alternativa (tekst od bar jednog karaktera, tačka, pa opet tekst od bar jednog karaktera).

Jqwik Generator sa Alternativama (Arbitraries.oneOf)

```
// Pomoćni generator za čist tekst (dužine bar 1)
```

```
private Arbitrary<String> cistTekst() {  
    return Arbitraries.strings().alpha().numeric().ofMinLength(1).ofMaxLength(10);  
}
```

@Provide

```
Arbitrary<String> lokalniDeoSaliBezJedneTacke() {  
    // Alternativa 1: Samo čist tekst (npr. "korisnik")  
    Arbitrary<String> bezTacke = cistTekst();  
  
    // Alternativa 2: Tekst + tačka + tekst (npr. "ime" + "." + "prezime")  
    Arbitrary<String> saJednomTackomU Sredini = Combinators.combine(cistTekst(), cistTekst())  
        .as((deo1, deo2) -> deo1 + "." + deo2);  
  
    // jqwik će u 50% slučajeva birati prvu, a u 50% slučajeva drugu alternativu!  
    return Arbitraries.oneOf(bezTacke, saJednomTackomUSredini);  
}
```

Primer 3: Generatori kolekcija

U ovom primeru testiramo sistem za grupno slanje imejllova (npr. *Newsletter*). Pravilo je da lista primalaca mora imati **između 1 i 10 imejllova**, i svi imejlovi u listi moraju biti **jedinstveni** (kako sistem ne bi istom čoveku poslao dve iste poruke).

```
import net.jqwik.api.*;  
import net.jqwik.api.constraints.*;  
import java.util.List;
```

```
public class NewsletterTest {
```

```
    // Generator koji pravi LISTU jedinstvenih legalnih imejllova
```

```
    @Provide
```

```
    Arbitrary<List<String>> listaPrimalaca() {
```

```
        return Arbitraries.strings().alpha().map(s -> s + "@domen.com") // pravi imejl
```

```
            .list() // pretvara u generator liste
```

```
            .ofMinSize(1).ofMaxSize(10) // @Size ekvivalent kroz kod
```

```
            .uniqueElements(); // @UniqueElements ekvivalent kroz kod
```

```
    }
```

```
    // --- TEST POMOĆU ANOTACIJA ---
```

```
    @Property
```

```
    void slanjeNewsletteraMoraUspeti()
```

```
        @ForAll
```

```
        @Size(min = 1, max = 10)
```

```
        @UniqueElements
```

```
        List<@ForAll("legalneEmailAdrese") String> primaoci
```

```
    ) {
```

```
        // Test uspešno prolazi jer jqwik garantuje da lista ima 1-10 elemenata i nema duplikata
```

```
        assertTrue(sistemZaSlanje.posaljiGrupno(primaoci));
```

```
    }
```

```
}
```

Kako radi @Size(min = 1, max = 10):

- Jqwik neće generisati praznu listu (jer je min = 1).

- Jqwik nasumično varira dužinu liste (npr. u jednom pokušaju lista ima 2 elementa, u sledećem 9).

Kako radi @UniqueElements:

- Ako ugrađeni generator napravi npr. imejl `ana@domen.com`, jqwik-ov mehanizam će osigurati da se taj isti string `ana@domen.com` **ne pojavi ponovo unutar te iste liste**.
- Pogodno za ključeve čije vrednosti moraju biti unikatne.

Dva načina pisanja:

- Filteri za kolekcije se mogu pisati i kroz fluentni API
`(.list()).ofMinSize(1).uniqueElements())` unutar `@Provide` metode, ili direktno kao **anotacije iznad parametra** u samom testu, što kod čini izuzetno čitljivim.

STRATEGIJA GENERISANJA VREDNOSTI KOD JQWIK

PBT (Property-Based Testing) alati poput **jqwik**-a ne biraju brojeve nasumično kao klasičan *Random* generator. Oni koriste takozvani **pametni (inteligentni) odabir sa težinama** i strategiju koja se fokusira na granične vrednosti i ekstremne situacije.

Evo kako to tačno funkcioniše kroz dva primera.

Primer 1: Prosti primer – Opseg celih brojeva [m, n]

Zamislamo da je zadat opseg celih brojeva od **-50 do 150**. Klasičan nasumični generator bi podjednako birao bilo koji broj (npr. 42, 87, -12...).

jqwik, sa druge strane, generiše npr. 1000 test slučajeva, ali unutar tog skupa primenjuje sledeću pametnu raspodelu:

1. **Apsolutne granice (Prvo i najvažnije):** Prvo će generisati tačno **-50 i 150** da proveriti same ivice.
2. **Nula i okolina nule:** Ako opseg obuhvata nulu (kao ovde), jqwik će generisati **0, 1, i -1**. Nula je istorijski najveći izvor bagova u programiranju.
3. **Prelazi znakova:** Generisaće nekoliko brojeva tik ispod nule (npr. -2, -3) i tik iznad nje.
4. **"Skrivene" granice tipova podataka:** Ako opseg dodiruje ekstreme (poput `Integer.MAX_VALUE` ili `Integer.MIN_VALUE`), on će obavezno testirati i njih.
5. **Nasumična statistička pokrivenost:** Tek preostali procenat testova se ravnomerno raspoređuje na "dosadne" brojeve u sredini (npr. 34, 78, 112).

Primer 2: Složeniji primer – Validacija lozinke (Strings)

Uzmimo primer testiranja metoda koji prihvata tekst (`String`) dužine između **8 i 20 karaktera**.

```
@Property
void proveriloZinku(@ForAll @StringLength(min = 8, max = 20) String lozinka) {
    // test kod
}
```

jqwik će u pozadini kreirati masu test primera, ali **neće** generisati samo obične reči od 10 slova. On će namerno kreirati sledeće "pametne" kombinacije:

- **Dužinske ivice:** Tekst koji ima tačno **8 karaktera** i tekst koji ima tačno **20 karaktera**.
- **Specijalni karakteri:** Tekstualne nizove koji sadrže prazna mesta (space), tabulatore (`\t`), prelome reda (`\n`).
- **Unikod i emodžiji:** Generisaće stringove sa ćirilicom, kineskim navodnicima ili emodžijima (npr. " "). Mnogi sistemi padaju ovde jer emodžiji zauzimaju više bajtova nego obična slova, što može da sruši validaciju dužine!
- **Ponavljanja:** Stringove sastavljene od samo jednog ponovljenog karaktera (npr. "aaaaaaa").

"Shrinking" (Skupljanje)

Ova pametna raspodela ide ruku pod ruku sa još jednom moćnom osobinom. Ako jqwik tokom tog "pametnog" generisanja nađe kompleksan primer koji ruši program (npr. string `"A B!d9#qx"`), on ga neće samo ispisati.

On će u pozadini automatski početi da **smanjuje i pojednostavljuje** taj ulaz (odbacivaće karakter po karakter, smanjivaće brojeve ka nuli) sve dok ne pronade **najmanji mogući primer** koji i dalje ruši program (npr. samo emodži `" "` ili samo razmak `" "`). Programeru tek onda isporuči tačan uzrok kvara.