

TESTIRANJE BAZIRANO NA SVOJSTVIMA

Šta je testiranje bazirano na svojstvima (Property based testing)

Definicija:

- Property-Based Testing je tehnika automatizovanog testiranja softvera gde umesto pisanja pojedinačnih test primera (eng. Example-Based Testing), programer definiše opšta pravila (svojstva) koja kod mora da zadovolji za bilo koji ispravan ulaz.
- **Glavna razlika** u pristupu:
 - Tradicionalni test primer:** „Ako funkciji prosledim e-mail petar@gmail.com, očekujem rezultat true.“
 - PBT (Svojstvo):** „Ako funkciji prosledim bilo koji string koji pripada klasi Validna e-mail adresa, rezultat uvek mora biti true.“
- **Zašto ga koristimo?**
- Čovek pri pisanju testova često bira „srećne putanje“ i predvidive granične vrednosti. PBT alat generiše stotine ili hiljade nasumičnih i ekstremnih kombinacija unutar zadatih klasa ekvivalencije, pokrivajući granične slučajeve (edge cases) koje programer možda ne bi ručno napisao

Proces rada jqwik alata (dodatak JUnit-u)



Proces rada jqwik alata (dodatak JUnit-u)

1. Generisanje

- Na osnovu pravila koja smo zadali (klase ekvivalencije), alat pokreće generator podataka. On u sekundi kreira npr. 1000 različitih varijacija e-mail adresa: od najkraćih (a@b.rs), preko standardnih, pa sve do ogromnih stringova na samoj granici memorije.

2. Izvršavanje i praćenje

- Alat „zasipa“ našu metodu generisanim podacima i prati šta se dešava. Sve dok se tvrdnja (propertija) ispunjava, test ide dalje.

3. Smanjivanje kontraprimera (Shrinking)

Ovo je najmoćniji deo alata. Pretpostavimo da je alat poslao e-mail sa 500 karaktera i čudnim rasporedom tačaka i da je naš kod pao (pronađen je bag). Za programera je užasno teško da debuguje string od 500 karaktera.

- Alat tada preuzima inicijativu: Počinje automatski da skraćuje taj string, izbacuje viškove i traži **najmanji mogući unos** koji i dalje obara naš kod.

4. Izveštavanje

- Umesto onog džinovskog stringa sa početka, jqwik nam u konzoli ispisuje minimalni primer (npr. adresa a@.co – gde nedostaje domen). Programer odmah vidi gde je logika zakazala i popravlja kod.

@Property	// 1. ANOTACIJA SVOJSTVA (Property Annotation)
void popustUvekValidan(	// 2. METODA SVOJSTVA (Property Method)
@ForAll	// 3. PROVAJDER PARAMETRA (Parameter Provider)
@IntRange(min = -50, max = 200)	// 4. OGRANIČENJE GENERATORA (Generator Constraint)
int procenat	// 5. TEST PODATAK / ULAZNI PARAMETAR (Input Parameter)
) {	
// 6. PRETPOSTAVKA / FILTER (Assumption)	
// Ako je procenat npr. deljiv sa 13, preskoči ovaj trial i generiši drugi broj	
Assume.that(procenat % 13 != 0);	
 Cenovnik c = new Cenovnik();	// ┌
c.postaviPopust(procenat);	// └ 7. STIMULUS / IZVRŠNA FAZA (Stimulus / Execution)
 Assertions.assertTrue(	// ┌
c.getPopust() >= 0 &&	// └ 8. INVARIJANTA / TVRDNJA (Invariant / Assertion)
c.getPopust() <= 100	// ┌
);	// └
}	

Anatomija PBT jqwik pravila

Element	Naziv u terminologiji	Kratka uloga u testu
@Property	Anotacija svojstva (<i>Property Annotation</i>)	Menja @Test. Pokreće metodu stotinama puta sa različitim podacima. Jedno izvršavanje se zove Trial.
void imeMetode	Metoda svojstva (<i>Property Method</i>)	Naziv poslovnog pravila koje proveravamo.
@ForAll	Provajder parametra (<i>Parameter Provider</i>)	Signal jqwik-u da automatski generiše podatke za ovaj parametar.
@IntRange / @NotBlank	Ograničenje generatora (<i>Generator Constraint</i>)	Grubo podešavanje i sužavanje opsega test podataka na nivou anotacije.
int procenat	Ulazni parametar (<i>Input Parameter</i>)	Promenljiva u koju se ubrizgava generisana vrednost u svakom pokušaju (<i>trial</i>).
Assume.that(...)	Pretpostavka / Filter (<i>Assumption</i>)	Fino filtriranje podataka tokom izvršavanja. Ako uslov nije ispunjen, jqwik odbacuje taj podatak i odmah traži sledeći, bez rušenja testa.
Kod unutar metode	Stimulus / Izvršna faza (<i>Stimulus / Execution</i>)	Priprema objekata i pozivanje metode koja vrši radnju.
Assertions.assert t...	Invarijanta / Tvrdnja (<i>Invariant / Assertion</i>)	Logički iskaz (istina) koji nikada ne sme biti prekršen , bez obzira na ulaz.

Transformacija zahteva programa u PBT šablone

Ako u specifikaciji piše...	...to je ovaj PBT pristup	...i piše se u ovom logičkom formatu / kodu
„Sistem mora uvek biti u bezbednom stanju...”	Validity (Validnost)	<code>assert objekat.isValid();</code>
„Nakon što se desi X, rezultat mora biti Y...”	Postcondition (Postuslov)	<code>objekat.uradiX(); assert objekat.stanje == Y;</code>
„Nebitno je kojim redom se unose podaci...”	Metamorphic (Komutativnost)	<code>assert f(A, B) == f(B, A);</code>
„Ponovljeni klik/unos ne menja stanje...”	Metamorphic (Idempotentnost)	<code>assert f(f(X)) == f(X);</code>
„Modul radi isto kao staro rešenje, ali brže...”	Model-based / Oracle	<code>assert nasKompleksniKod() == jednostavanModel();</code>
„Dodavanje novog elementa/korak ne narušava strukturu...”	Invariant / Indukcija	<code>assert obj.imaOsobinu(); obj.korakIndukcije(); assert obj.imaOsobinu();</code>

Tri ključne jqwik anotacije

- Da bismo PBT pipeline pustili u rad, koristimo tri glavne anotacije:
 - **@Property** (Šta testiramo)
 - **@ForAll** (Šta ulazi u test)
 - **@Provide** (Kako pravimo specifične podatke)
- One govore alatu šta je test, koji podaci ulaze, i kako se ti podaci prave.

@Property (Šta testiramo)

- **Čemu služi:** Menja standardnu JUnit `@Test` anotaciju. Ona govori jqwik-u: *„Ovo nije običan test primer, ovo je opšte pravilo (svojstvo) koje želim da izvršiš 100 puta sa različitim podacima.“*
- **Primer u kodu:**

`@Property`

```
void popustNikadaNeSmeBitiNegativan(...) { ... }
```

@ForAll (Šta ulazi u test)

- **Čemu služi:** Stavlja se ispred parametara u metodi testa. Ona naređuje alatu: „Za ovaj parametar mi nasumično izgeneriši gomilu podataka.“ Ako u zagradi navedemo ime (npr. @ForAll("validniMejlovi")), povezujemo parametar sa našom definisanom generatorskom funkcijom.

- **Primer u kodu:**

@Property

```
void proveriVaridnost(@ForAll("validniMejlovi") String email) {  
    Assertions.assertTrue(validator.isValid(email));  
}
```

@Provide (Kako pravimo specifične podatke)

- **Čemu služi:** Stavlja se iznad pomoćne metode koja vraća `Arbitrary<T>` (objekat koji u jqwik-u služi kao fabrika/generator podataka). Ovde mapiramo naše klase ekvivalencije.
- **Primer u kodu:**

@Provide

```
Arbitrary<String> validniMejlovi() {  
    // Koristimo jqwik-ov fluent API za kreiranje stringova po našim pravilima  
    return Arbitraries.strings()  
        .alpha() // samo slova  
        .ofMinLength(2)  
        .ofMaxLength(6)  
        .map(ext -> "korisnik@domen." + ext); // sklapanje u klasu ekv. ekv  
}
```

Najčešća ograničenja generatora

Za filtriranje podataka, koriste se ugrađene jql anotacije pored @ForAll:

Za brojeve (int, long, double...)

@IntRange(min = 1, max = 100) – Ograničava opseg (uključujući granične vrednosti).

@Positive / @Negative – Samo brojevi veći od nule, odnosno manji od nule.

Za tekst (String)

@StringLength(min = 3, max = 50) – Definise dužinu stringa.

@NotBlank – Osigurava da string ne bude prazan niti da sadrži samo razmake.

@AlphaChars / @NumericChars – Generiše samo slova, odnosno samo brojeve.

Za kolekcije (List, Set...)

@Size(min = 1, max = 10) – Ograničava broj elemenata u kolekciji.

@UniqueElements – Osigurava da u listi nema dupliranih vrednosti.

Primer: Automatizacija testiranja klasa ekvivalencije

- Cilj tehnike klasa ekvivalencije je da beskonačan broj mogućih e-mail adresa grupišemo u nekoliko kategorija (klasa). Ako test prođe za jednog predstavnika klase, pretpostavljamo da radi za sve u toj klasi.
- Analiziraćemo problem provere ispravnosti imejl adrese `lokalni_deo@domen.ekstenzija`
- Prvo ćemo definisati legalne i nelegalne klase ekvivalencije
- Zatim ćemo napisati PBT pravila koja za testiranje na osnovu ovih klasa ekvivalencije

Definisanje klasa ekvivalencije za imejl adresu

- Pravila za jednostavnu e-mail adresu (format: lokalni_deo@domen.ekstenzija):
- **Lokalni deo** (pre @): sme da sadrži slova, brojeve i tačke. Ne sme biti prazan.
- **Domen** (između @ i .): sme da sadrži slova i brojeve. Ne sme biti prazan.
- **Ekstenzija** (posle zadnje .): samo slova, dužine između 2 i 6 karaktera (npr. .com, .rs, .edu).

Legalne klase ekvivalencije

- 1) **Standardna adresa:** Samo obična slova i jedan domen (npr. test@gmail.com).
- 2) **Adresa sa brojevima i tačkama:** Sadrži brojeve i tačke u lokalnom delu (npr. petar.petrovic12@fedoraproject.org).
- 3) **Kratka ekstenzija:** Ekstenzija od tačno 2 slova (npr. user@kompanija.rs).
- 4) **Dugačka ekstenzija:** Ekstenzija od tačno 6 slova (npr. kontakt@muzej.museum).

Nelegalne klase ekvivalencije

- 5) **Nedostaje @ znak:** (npr. korisnikgmail.com).
- 6) **Nedostaje tačka u domenu:** (npr. korisnik@gmailcom).
- 7) **Prazan lokalni deo:** Počinje direktno sa @ (npr. @gmail.com).
- 8) **Prazan domen:** Nema ničega između @ i . (npr. test@.com).
- 9) **Prekratka ekstenzija:** Ekstenzija ima manje od 2 slova (npr. test@gmail.c).
- 10) **Predugačka ekstenzija:** Ekstenzija ima više od 6 slova (npr. test@gmail.commerce).
- 11) **Zabranjeni karakteri:** Sadrži znakove poput !, #, \$ u domenu (npr. test@gma!l.com).

EdukativniTest.java

```
import org.junit.jupiter.api.Test;
import net.jqwik.api.*;
import net.jqwik.api.constraints.*;
import java.util.regex.Pattern;

import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.junit.jupiter.api.Assertions.assertFalse;

class EdukativniTest {

    // Jednostavan Regex koji striktno prati naša pravila klasa ekvivalencije
    private static final Pattern EMAIL_REGEX =
        Pattern.compile("^([a-zA-Z0-9.] +@[a-zA-Z0-9]+\\. [a-zA-Z]{2,6})$");

    // Funkcija koju testiramo
    boolean isValidEmail(String email) {
        if (email == null) return false;
        return EMAIL_REGEX.matcher(email).matches();
    }
}
```

EdukativniTest.java (nastavak)

```
// --- GENERATORI ---
```

```
@Provide
```

```
Arbitrary<String> validniLokalniDeo() {  
    return Arbitraries.strings().alpha().numeric().withChars('.')  
        .ofMinLength(1).ofMaxLength(15);  
}
```

```
@Provide
```

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

```
@Provide
```

```
Arbitrary<String> validnaEkstenzija() {  
    return Arbitraries.strings().alpha()  
        .ofMinLength(2).ofMaxLength(6);  
}
```

```
@Provide
```

```
Arbitrary<String> legalneEmailAdrese() {  
    return Combinators.combine(validniLokalniDeo(), validanDomen(), validnaEkstenzija())  
        .as((lokalni, domen, ekstenzija) -> lokalni + "@" + domen + "." + ekstenzija);  
}
```

EdukativniTest.java (nastavak)

```
// --- NOVI GENERATOR ZA NELEGALNE KARAKTERE (Klasa 11) ---  
// Biramo specijalne znakove koji nikako ne smeju biti u domenu  
@Provide  
Arbitrary<String> nelegalniKarakter() {  
    return Arbitraries.of("!", "#", "$", "%", "*", "?", "/" );  
}  
  
// --- TESTOVI ---  
  
// 1. Gledamo šta jqlwik generiše u pozadini pomoću @Report  
@Property  
@Report(Reporting.GENERATED) // MAGIJA: Ispisuje generisane primere u terminal!  
void sveLegalneAdreseMorajuBitiValidne(@ForAll("legalneEmailAdrese") String email) {  
    assertTrue(isValidEmail(email));  
}
```

EdukativniTest.java (nastavak)

```
// 2. Raniji test za predugu ekstenziju (Klasa 10)
@property
void adreseSaPredugomEkstenzijomNisuValidne(
    @forall @AlphaChars @StringLength(min=1, max=10) String lokalni,
    @forall @AlphaChars @StringLength(min=1, max=10) String domen,
    @forall @AlphaChars @StringLength(min=7, max=15) String predugaEkstenzija
) {
    String nelegalniEmail = lokalni + "@" + domen + "." + predugaEkstenzija;
    assertFalse(isValidEmail(nelegalniEmail));
}

// 3. NOVI TEST: Specijalni karakteri u domenu (Klasa 11)
// Spajamo legalan lokalni deo, legalan domen, ali ubacujemo zabranjeni znak unutra!
@property
void adreseSaSpecijalnimKarakterimaUDomenuNisuValidne(
    @forall("validniLokalniDeo") String lokalni,
    @forall("validanDomen") String domen,
    @forall("nelegalniKarakter") String losZnak,
    @forall("validnaEkstenzija") String ekstenzija
) {
    // Pravimo domen koji u sebi ima npr. "gma!l" ili "gmail?"
    String pokvareniDomen = domen + losZnak;
    String nelegalniEmail = lokalni + "@" + pokvareniDomen + "." + ekstenzija;

    assertFalse(isValidEmail(nelegalniEmail), "Regex je greškom pustio loš karakter: " + nelegalniEmail);
}
}
```

Pokretanje testova

- Komanda `mvn clean test` daje nam standardni ispis iz jqwik dok pravi adrese (podrazumevano 100, broj se može zadati sa `@Property(tries = 500)` za na primer 500 adresa)
- Videće se da su svi testovi prošli i dodatan ispis adresa:

```
EdukativniTest > sveLegalneAdreseMorajuBitiValidne() GENERATED DATA
```

```
-----
```

```
...
```

```
count: 1, value: 3.c@f.as
```

```
count: 2, value: .@mE86.lh
```

```
count: 3, value: a89.X7@A9c.comm
```

```
...
```

```
count: 100, value: petar99.@domain.online
```