SOFTVERSKO INŽENJERSTVO

školska 2024/2025 godina

Vežba 10: Builder Pattern

Builder patern pripada **kreacionim (stvaralačkim)** dizajn paterima. Njegova osnovna svrha je da **razdvoji proces pravljenja objekta od njegove reprezentacije**, tako da isti proces može kreirati različite varijante objekta.

Ovaj patern je posebno koristan kada radimo sa **objektima koji imaju mnogo opcionalnih parametara** ili kada imamo više načina na koje se objekat može napraviti.

🔍 Problem koji Builder rešava

U Javi i drugim objektno orijentisanim jezicima često se dešava sledeći problem:

Zamislimo klasu User koja ima puno atributa:

```
public class User {
    private String name;
    private String email;
    private int age;
    private boolean subscribed;
    private String phone;
    private String city;
}
```

Ako pokušamo da koristimo klasične konstruktore, možemo završiti sa:

- **Više konstruktora** sa različitim kombinacijama parametara (konstruktor overload)
- **Zbunjujući redosled parametara**
- **Teško čitljivim kodom**

Primer:

```
User u = new User("Ana", "ana@gmail.com", 25, true, "064123456", "Beograd");
```

Ali šta ako samo želimo da postavimo ime i email? Moramo da dodajemo null, 0, false... što postaje nečitljivo i krhko.

🧠 Rešenje: Builder

Builder patern omogućava kreiranje objekta tako da:

- Gradimo objekat **korak po korak**
- **Samo ono što želimo postavimo**
- Na kraju pozivamo .build() da završimo objekat

Sintaksa izgleda ovako:

```
User u = new User.Builder("Ana", "ana@gmail.com")
         .age(25)
         .phone("064123456")
         .build();
```

🏗️ Struktura Builder paterna

Element	Opis
Product	Objekat koji se gradi (npr. User)
Builder	Interfejs ili apstraktna klasa koja definiše korake za gradnju
ConcreteBuilder	Implementira Builder i sadrži stanje koje se gradi
Director	(opciono) Odgovoran za definisanu sekvencu pozivanja Builder-a
Client	Koristi Builder za pravljenje objekta

U praksi u Javi često koristimo **unutrašnju statičku klasu** Builder unutar klase koju gradimo (kao u prethodnom primeru sa User).

✅ Prednosti Builder paterna

- Jednostavan za korišćenje – nema dugačkih konstruktora
- Izbegava **telescoping constructors** (preklapanje konstruktora)
- Povećava čitljivost koda
- Lako se dodaju nova polja u budućnosti
- Omogućava **immutability** – krajnji objekat se ne može menjati posle kreacije

---

🧰 Gde se koristi Builder?

- U Java bibliotekama: StringBuilder, StringBuffer, Stream.Builder, HttpClient.Builder
- U složenim konfiguracijama: npr. kreiranje kompleksnih formi, konfiguracija baze, izveštaja, ali i u objektima sa **puno opcionih parametara**

💡 Napredne osobine i obrasci korišćenja Builder paterna

Builder patern u Javi se najčešće implementira pomoću **statičke unutrašnje klase** unutar same klase koju gradimo. Ovaj pristup ima nekoliko važnih prednosti:

🧱 1. Fluent interfejs

Metode u Builder klasi često vraćaju instancu samog Builder-a (return this), čime omogućavaju **ulančavanje metoda** (method chaining):

```
User u = new User.Builder("Ana", "ana@gmail.com")
                .age(25)
                .city("Beograd")
                .phone("064123456")
                .build();
```

Ovaj stil čini kod:

- **Deklarativnim** (lepo prikazuje šta se tačno postavlja)
- **Jasnim** (nema potrebe za dodatnim komentarima)
- **Manje podložnim greškama** (nema potrebe za pamćenjem redosleda argumenata)

🔐 2. Podrška za imutabilnost

Builder omogućava da kreirani objekat bude **immutable** (nepromenljiv), jer se sva postavka atributa vrši **unutar Builder-a**, a finalna klasa (User) može imati samo private final polja i bez set metoda:

```
public final class User {
    private final String name;
    private final String email;
    private final int age;
    private final String phone;

    private User(Builder builder) {
        this.name = builder.name;
        this.email = builder.email;
        this.age = builder.age;
        this.phone = builder.phone;
    }
    // ... getter metode, bez setera
}
```

Ova osobina je ključna za:

- Sigurnost u višenitnim (thread-safe) okruženjima
- Predvidivo ponašanje aplikacija
- Lakše testiranje i debagovanje

🛠️ 3. Razdvajanje kompleksne logike

Ukoliko je logika za konstrukciju složenija (npr. validacija kombinacija parametara, podešavanje po tipu korisnika itd.), Builder je idealno mesto za to — jer omogućava da gradnja bude **kapsulisana** u posebnu klasu.

Primer:

```
public Builder age(int age) {
    if (age < 0) throw new IllegalArgumentException("Godine ne mogu biti
                                                    negativne.");
    this.age = age;
    return this;
}
```

🧭 4. Opcioni Director – kada imamo više unapred definisanih konfiguracija

Iako se u praksi ređe koristi, **Director** klasa može sadržati unapred definisane šablone kako se objekat gradi (npr. korisnik početnik, premium korisnik...).

```
public class UserDirector {
    public User createDefaultUser(String name, String email) {
        return new User.Builder(name, email)
                       .age(18)
                       .subscribed(false)
                       .build();
    }
}
```

Pošto Builder koristi male metode za svako polje, olakšava se:

- Testiranje pojedinačnih delova
- Praćenje promena (nema velikih konstruktora)
- Dodavanje novih parametara bez menjanja postojećeg koda (Open/Closed princip

Builder je prisutan u mnogim popularnim Java bibliotekama i okruženjima:

- StringBuilder i StringBuffer (JDK)
- Stream.Builder (JDK)
- HttpRequest.Builder i HttpClient.Builder (Java 11+)
- Lombok @Builder anotacija – automatski generiše Builder klasu

Vežba: Kreiranje Pizza objekta pomoću Builder paterna

Kreirati objekat Pizza sa mnogo opcionalnih dodataka (sir, pečurke, masline itd.), ali tako da gradnja ostane **čitljiva**, **sigurna** i **prilagodljiva**.

Definišemo glavnu klasu `Pizza` i njenu `Builder` klasu:

```
public class Pizza {
    // Obavezna polja
    private final String size;

    // Opcionalna polja
    private final boolean cheese;
    private final boolean pepperoni;
    private final boolean mushrooms;
    private final boolean olives;

    // Privatni konstruktor koji se poziva iz Builder-a
    private Pizza(Builder builder) {
        this.size = builder.size;
        this.cheese = builder.cheese;
        this.pepperoni = builder.pepperoni;
        this.mushrooms = builder.mushrooms;
        this.olives = builder.olives;
    }

    // Getteri za čitanje vrednosti
    public String getSize() { return size; }
    public boolean hasCheese() { return cheese; }
    public boolean hasPepperoni() { return pepperoni; }
    public boolean hasMushrooms() { return mushrooms; }
    public boolean hasOlives() { return olives; }

    // Statička unutrašnja Builder klasa
    public static class Builder {
        // Obavezna polja
        private final String size;

        // Opcionalna polja (podrazumevano false)
        private boolean cheese = false;
        private boolean pepperoni = false;
        private boolean mushrooms = false;
        private boolean olives = false;

        // Konstruktor Builder-a sa obaveznim parametrom
        public Builder(String size) {
            this.size = size;
        }
```

```
        // Metode za dodavanje dodataka
        public Builder cheese(boolean value) {
            this.cheese = value;
            return this;
        }

        public Builder pepperoni(boolean value) {
            this.pepperoni = value;
            return this;
        }

        public Builder mushrooms(boolean value) {
            this.mushrooms = value;
            return this;
        }

        public Builder olives(boolean value) {
            this.olives = value;
            return this;
        }

        // Metoda koja vraća gotov Pizza objekat
        public Pizza build() {
            return new Pizza(this);
        }
    }
}
```

Glavni program

```
public class Main {
    public static void main(String[] args) {
        // Kreiramo Pizzu koristeći Builder patern
        Pizza pizza = new Pizza.Builder("Velika")
                            .cheese(true)
                            .pepperoni(true)
                            .mushrooms(false)
                            .olives(true)
                            .build();

        System.out.println("Veličina: " + pizza.getSize());
        System.out.println("Sir: " + pizza.hasCheese());
        System.out.println("Peperoni: " + pizza.hasPepperoni());
        System.out.println("Pečurke: " + pizza.hasMushrooms());
        System.out.println("Masline: " + pizza.hasOlives());
    }
}
```