SOFTVERSKO INŽENJERSTVO

školska 2024/2025 godina

Vežba 9: Factory i Abstract Factory patterni

1. Factory Method Pattern – *"Fabrika za pravljenje objekata"*

Factory Method je **kreacioni dizajn šablon** koji se koristi kada želimo da **odvojimo kreiranje objekata od njihove konkretne upotrebe**. U ovom šablonu, umesto da direktno koristimo new za instanciranje objekata, mi pozivamo metodu fabrike koja vraća objekat željenog tipa.

To omogućuje da:

- lako dodajemo nove tipove bez menjanja klijentskog koda,
- smanjimo zavisnost od konkretnih klasa,
- omogućimo **runtime** izbor tipa objekta.

Ovaj šablon promoviše **otvorenost za proširenje, ali zatvorenost za izmene** (princip SOLID dizajna). Drugim rečima, možemo dodavati nove klase proizvoda bez da diramo postojeći kod — što povećava održivost i fleksibilnost sistema.

Factory Method je posebno koristan u situacijama kada:

- imamo **hijerarhiju klasa** sa zajedničkim interfejsom ili apstraktnom klasom,
- želimo da izbegnemo **dug if-else ili switch blok** koji bira tip objekta na osnovu neke vrednosti,
- očekujemo da će se u budućnosti pojaviti **novi tipovi proizvoda** koji treba lako dodati.

Iako Factory Method povećava fleksibilnost, može dovesti do većeg broja klasa u sistemu. Zato je važno balansirati primenu, da se pattern koristi kada zaista postoji potreba za varijacijom u tipovima objekata.

🧠 Analogija

Zamisli da naručuješ picu telefonom. Ne moraš znati koji tačno radnik pravi picu, samo kažeš "hoću kapričozu". Fabrika (picerija) zna koji konkretan objekat (radnik) pravi tu picu. Klijent (ti) koristi proizvod bez da zna unutrašnju logiku kreacije. Važno je šta dobiješ, a ne ko ga pravi i kako je napravljen.

Ključne komponente

Uloga	Opis
Product	Interfejs ili apstraktna klasa za sve proizvode
ConcreteProduct	Konkretna implementacija proizvoda
Creator	Apstraktna klasa ili interfejs koja sadrži factory metodu
ConcreteCreator	Implementira factory metodu i kreira konkretan proizvod

✅ Kada koristiti

- Kada kreacija objekata zavisi od nekih **uslova ili izbora korisnika**.
- Kada ne želiš da hardkodiraš konkretne klase unutar poslovne logike.
- Kada se novi tipovi objekata često dodaju u sistem.

💻 Primer: Kreiranje oblika

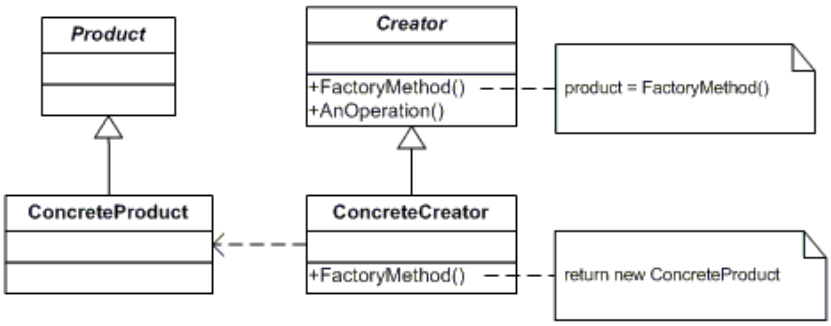
```
// 1. Apstraktni proizvod
interface Shape {
    void draw();
}

// 2. Konkretne implementacije
class Circle implements Shape {
    public void draw() {
        System.out.println("Crtam krug.");
    }
}

class Rectangle implements Shape {
    public void draw() {
        System.out.println("Crtam pravougaonik.");
    }
}
```

```
// 3. Fabrika (kreator)
class ShapeFactory {
    public Shape createShape(String shapeType) {
        if (shapeType == null) return null;
        if (shapeType.equalsIgnoreCase("circle")) return new Circle();
        else if (shapeType.equalsIgnoreCase("rectangle")) return new
                                                            Rectangle();
        return null;
    }
}

// 4. Klijent
public class Main {
    public static void main(String[] args) {
        ShapeFactory factory = new ShapeFactory();

        Shape shape1 = factory.createShape("circle");
        shape1.draw(); // Crtam krug.

        Shape shape2 = factory.createShape("rectangle");
        shape2.draw(); // Crtam pravougaonik.
    }
}
```

Ovaj primer prikazuje Factory Method dizajn šablon na jednostavnom primeru kreiranja oblika (Shape). Interfejs Shape definiše apstraktnu operaciju draw(), a klase Circle i Rectangle je konkretno implementiraju.

Klasa **ShapeFactory** predstavlja fabriku koja na osnovu unetog stringa ("circle" ili "rectangle") kreira i vraća odgovarajući objekat. Klijent (Main klasa) ne zna koji konkretan objekat se stvara — on samo poziva createShape() i koristi metod draw() bez brige o detaljima instanciranja.

Na taj način postižemo odvajanje logike kreiranja objekata od njihove upotrebe, što povećava fleksibilnost i proširivost aplikacije.



2. Abstract Factory Pattern – *"Fabrika fabrika"*

Abstract Factory šablon je unapređena verzija Factory Method šablona. Umesto da kreira **jedan** proizvod, ovaj šablon omogućuje kreaciju **porodice povezanih objekata**.

Ovaj patern omogućuje **sistemu da bude nezavisan od konkretnih klasa koje koristi**, jer klijentski kod koristi samo interfejse i metode apstraktne fabrike. Na taj način se lako može promeniti čitav skup povezanih objekata bez izmene postojećeg koda — dovoljno je zameniti konkretnu implementaciju fabrike.

Na primer, ako praviš korisnički interfejs za različite operativne sisteme (Windows, macOS), svaki OS ima svoj izgled za dugmad, prozore i menije. Abstract Factory ti omogućuje da sve te komponente praviš u paketu — sve Windows komponente zajedno, sve Mac komponente zajedno, itd.

Zbog toga je Abstract Factory posebno koristan u **aplikacijama koje moraju podržavati više tema, platformi, baza podataka ili konfiguracija**, gde svaka od njih ima svoj set komponenti koje moraju raditi skladno zajedno.

🧠 Analogija

Zamisli da kupuješ nameštaj za dnevnu sobu. Ako hoćeš klasični stil, uzmeš fotelju, sto i luster iz iste kolekcije. Ne želiš da kombinuješ modernu fotelju i retro sto. Abstract Factory je kao salon koji prodaje kompletne kolekcije nameštaja. Na taj način, svi elementi se slažu stilski i funkcionalno jer dolaze iz iste porodice proizvoda, što osigurava konzistentnost u izgledu i ponašanju.

Ključne komponente

Uloga	Opis
AbstractFactory	Interfejs sa metodama za kreiranje različitih vrsta objekata
ConcreteFactory	Implementacije koje proizvode specifične varijante porodice objekata
AbstractProduct	Interfejsi za svaku klasu proizvoda
ConcreteProduct	Konkretna implementacija proizvoda
Client	Koristi fabriku za kreaciju objekata, bez znanja o konkretnim klasama

✅ Kada koristiti

- Kada želiš da obezbediš **doslednost proizvoda** koji moraju raditi zajedno.
- Kada želiš da sakriješ konkretnu implementaciju proizvoda.
- Kada postoji više “porodica” proizvoda koje treba lako zameniti.

💻 Primer: GUI elementi

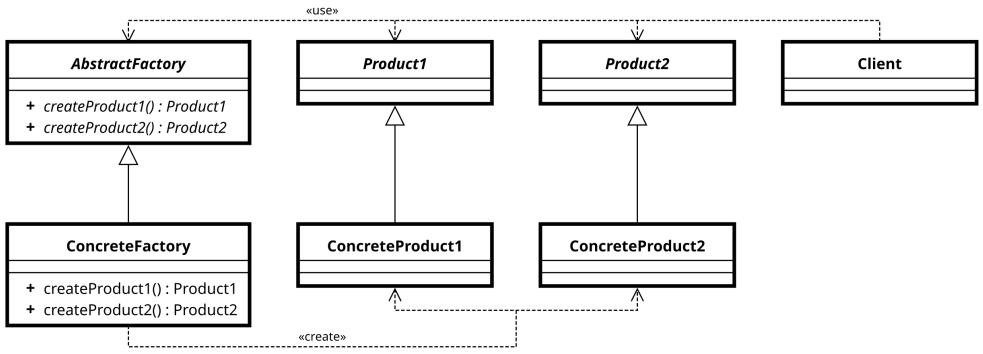
```
// Apstraktni proizvodi
interface Button {
    void render();
}

interface Checkbox {
    void render();
}

// Konkretni proizvodi – Windows stil
class WindowsButton implements Button {
    public void render() {
        System.out.println("Windows dugme.");
    }
}

class WindowsCheckbox implements Checkbox {
    public void render() {
        System.out.println("Windows čekboks.");
    }
}

// Konkretni proizvodi – Mac stil
class MacButton implements Button {
    public void render() {
        System.out.println("Mac dugme.");
    }
}

class MacCheckbox implements Checkbox {
    public void render() {
        System.out.println("Mac čekboks.");
    }
}

// Apstraktna fabrika
interface GUIFactory {
    Button createButton();
    Checkbox createCheckbox();
}
```

```
// Konkretne fabrike
class WindowsFactory implements GUIFactory {
    public Button createButton() {
        return new WindowsButton();
    }

    public Checkbox createCheckbox() {
        return new WindowsCheckbox();
    }
}

class MacFactory implements GUIFactory {
    public Button createButton() {
        return new MacButton();
    }

    public Checkbox createCheckbox() {
        return new MacCheckbox();
    }
}

// Klijent
class Application {
    private Button button;
    private Checkbox checkbox;

    public Application(GUIFactory factory) {
        button = factory.createButton();
        checkbox = factory.createCheckbox();
    }

    public void renderUI() {
        button.render();
        checkbox.render();
    }
}

// Glavni program
public class Main {
    public static void main(String[] args) {
        GUIFactory factory;
        String os = "Mac";

        if (os.equalsIgnoreCase("Windows")) {
            factory = new WindowsFactory();
        } else {
            factory = new MacFactory();
        }

        Application app = new Application(factory);
        app.renderUI();
    }
}
```

Ovaj primer demonstrira Abstract Factory dizajn šablon kroz kreiranje korisničkog interfejsa (UI) za različite operativne sisteme.

- Imamo apstraktne proizvode: Button i Checkbox, koji definišu zajednički interfejs za sve stilove dugmadi i čekbokseva.
- Kreiramo konkretne proizvode za dva različita stila: WindowsButton i WindowsCheckbox za Windows, te MacButton i MacCheckbox za Mac.
- Zatim definišemo apstraktnu fabriku **GUIFactory** sa metodama za pravljenje dugmadi i čekbokseva.
- Svaki operativni sistem ima svoju konkretnu fabriku: WindowsFactory i MacFactory, koje kreiraju odgovarajuće stilizovane komponente.

U klasi **Application**, korisnik koristi interfejs GUIFactory da kreira i renderuje dugmad i čekboks, bez potrebe da zna koji konkretan stil se koristi. U metodi **Main**, na osnovu vrednosti promenljive os odlučuje se koja fabrika će se koristiti. Time omogućavamo da se čitava porodica povezanih objekata (dugme + čekboks) kreira u skladu sa odabranim operativnim sistemom, bez menjanja ostatka koda.



Zaključak o Factory pristupu

Osobina	**Factory Method**	**Abstract Factory**
Kreira	Jedan proizvod	Više povezanih proizvoda (porodicu)
Fleksibilnost	Dodavanje novih proizvoda je lako	Teže dodavanje novih proizvoda, ali lako menjanje porodica
Kompleksnost	Jednostavniji dizajn šablon	Složeniji i organizovaniji za veće sisteme
Primer u praksi	Oblik (krug, kvadrat, trougao)	GUI komponente za različite platforme