# Projektni uzorci - kreacioni (Creational design patterns-DP)

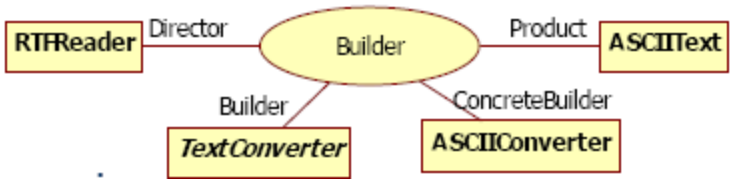
## Builder – kreacioni objektni DP

Drugo ime:

Namena:
–razdvaja konstrukciju kompleksnog objekta od njegove reprezentacije
–posledica je da isti proces konstrukcije može da kreira različite reprezentacije

Primenljivost
uzorak treba koristiti kada:
1. algoritam za kreiranje složenog objekta treba da bude nezavisan
   a) od delova koji čine objekat
   b) od načina na koji se delovi sklapaju u celinu
2. proces konstrukcije mora da dopusti različite reprezentacije za objekat koji se konstruiše

**Kratak opis problema**




1. posmatra se aplikacija za čitanje RTF dokumenata (čitač)
2. potrebno je da učitane dokumente može da konvertuje u:
   - čisti ASCII tekst
   - TeX dokument
   - tekstualnu komponentu koja može da se interaktivno edituje
   - ...
3. potrebno je da se lako dodaje nova konverzija, bez izmene čitača
4. rešenje je primena Builder DP-a:
- konfigurisanje RTFReader klase objektom TextConverter
- RTFReader čita RTF tekst i parsira ga
- TextConverter konvertuje RTF u drugu tekstualnu reprezentaciju

- kad RTFReader prepozna RTF token, vrši se izdavanje zahteva TextConverter objektu
-TextConverter konvertuje podatak i reprezentuje token u datom formatu
- potklase TextConverterspecijalizuju korake u konverziji
5. svaka vrsta konvertora, iza apstraktnog interfejsa,
ima mehanizam za kreiranje i sastavljanje složenog objekta
6. konvertor je razdvojen od čitača
- čitač je odgovoran samo za parsiranje ulaznog RTF dokumenta
7. svaka klasa konvertora u uzorku se naziva graditeljem (builder)
8. klasa čitača se naziva upravljačem (director)
9. u ovom primeru uzorak graditelja je razdvojio
-algoritam interpretacije tekstualnog formata (parser za RTF dokumente)
-način kako se konvertovani format stvara i reprezentuje
10. posledica je ponovna upotreba algoritma parsiranja pri kreiranju različitih tekstualnih reprezentacija od RTF dokumenata
- RTFReader se samo konfiguriše objektom neke od potklasa RTFConverter

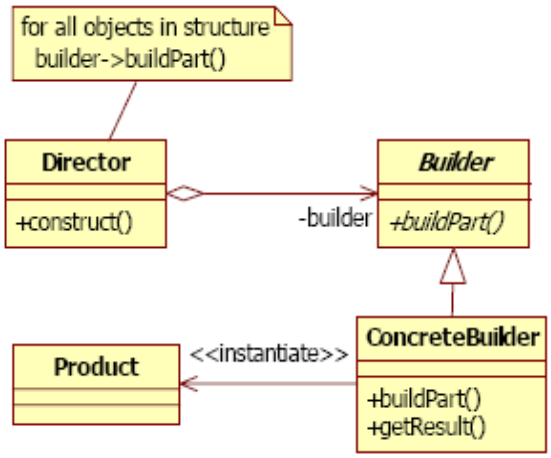
Primer: Kreirajte klasu PizzaKreacija koja ce konstruisati delove i sastaviti različite vrsta pica.

```
/** "Proizvod" */
class Pizza {
        private String testo = "";
        private String sos = "";
        private String prilog = "";

        public void setTesto(String testo) {
                this.testo = testo;
        }

        public void setSos(String sos) {
                this.sos = sos;
        }

        public void setPrilog(String prilog) {
                this.prilog = prilog;
        }
}

/** "Abstract Builder" */
abstract class PizzaBuilder {
        protected Pizza pizza;

        public Pizza getPizza() {
                return pizza;
        }

        public void createNewPizzaProduct() {
                pizza = new Pizza();
```

```
        }

        public abstract void buildTesto();

        public abstract void buildSos();

        public abstract void buildPrilog();
}

/** "ConcreteBuilder" */
class VegeterijanskaPizzaBuilder extends PizzaBuilder {
        public void buildTesto() {
                pizza.setTesto("posno");
        }

        public void buildSos() {
                pizza.setSos("pelat");
        }

        public void buildPrilog() {
                pizza.setPrilog("pecurke+maslinke");
        }
}

/** "ConcreteBuilder" */
class MexicanaPizzaBuilder extends PizzaBuilder {
        public void buildTesto() {
                pizza.settesto("palacinka");
        }

        public void buildSos() {
                pizza.setSos("ljut");
        }

        public void buildPrilog() {
                pizza.setPrilog("sunka+feferoni");
        }
}

/** "Director" */
class Cook {
        private PizzaBuilder pizzaBuilder;

        public void setPizzaBuilder(PizzaBuilder pb) {
                pizzaBuilder = pb;
        }

        public Pizza getPizza() {
                return pizzaBuilder.getPizza();
        }

        public void constructPizza() {
                pizzaBuilder.createNewPizzaProduct();
                pizzaBuilder.buildTesto();
                pizzaBuilder.buildSos();
                pizzaBuilder.buildPrilog();
        }
}

/** Konstrukcija pice zadatog tipa */
public class PizzaKreacija {
        public static void main(String[] args) {
```

```
                Cook cook = new Cook();
                PizzaBuilder vegeterijanskaPizzaBuilder = new
VegeterijanskaPizzaBuilder();
                PizzaBuilder mexicanaPizzaBuilder = new MexicanaPizzaBuilder();

                cook.setPizzaBuilder(vegeterijanskaPizzaBuilder);
                cook.constructPizza();

                Pizza pizza = cook.getPizza();
        }
}
```

## DIJAGRAM PRIKAZA




## Učesnici:

**Builder**

specificira apstraktan interfejs za kreiranje delova objekta Product

**ConcreteBuilder**

- konstruiše i sastavlja delove proizvoda implementiranjem interfejsa Builder
- definiše i čuva proizvod koji kreira
- obezbeđuje interfejs za dohvatanje proizvoda

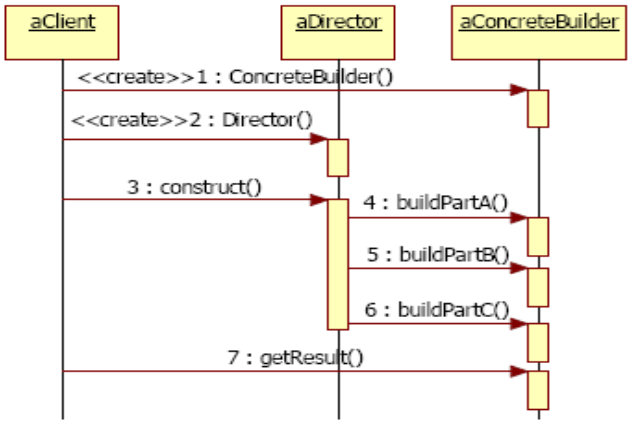
**Director**

konstruiše objekat koristeći interfejs Builder

**Product**

-predstavlja složeni objekat koji se konstruiše
-uključuje klase koje definišu sastavne delove (uključujući interfejse za sastavljanje delova u finalan rezultat)

## Saradnja:




Prednosti i mane:

**dopušta izmene interne reprezentacije i načina sklapanja složenog proizvoda**

-graditelj pruža upravljaču (Director) apstraktan interfejs za konstrukciju proizvoda

-reprezentacija i interna struktura, kao i način sklapanja su sakriveni

-za promenu interne reprezentacije potrebna je samo nova vrsta graditelja

**izolovanje koda za konstrukciju i reprezentaciju**

-bolja modularnost kroz kapsuliranje načina konstrukcije složenog objekta

- klijenti ne treba da znaju ništa o klasama koje definišu unutrašnju strukturu proizvoda (te klase se ne pojavljuju u interfejsu graditelja)

**daje finiju kontrolu nad procesom konstrukcije od drugih fabrika**

- drugi DP konstruišu proizvod u jednom potezu

- graditelj konstruiše proizvod korak-po-korak, pod kontrolom upravljača

Bliski DP:

1. Apstraktna fabrika je slična Builder-u po tome što može da konstruiše kompleksne objekte. Osnovna razlika je što se Builder fokusira na konstrukciju kompleksnog objekta korak-po-korak, a kod Apstraktne fabrike je naglasak na familijama objekata proizvoda (jednostavnih ili kompleksnih). Inače, Builder vraća proizvod kao finalni korak, dok ga Apstraktna fabrika vraća odmah

2. Builder često gradi objekat Composite

3. Kod uzorka Builder važi da se Director parametrizuje objektom Builder-a, kao što se klasa Context parametrizuje objektom Strategy u odgovarajućem DP Strategy. Osim u nameni, razlika je što se Strategy

najčešće implementira u jednoj metodi, dok su kod Builder-a implementirani pojedini koraci gradnje kao posebne metode

4. Projektni uzorak Template Method poziva apstraktne metode sopstvene klase, dok kod uzorka Builder, klasa Director sadrži konkretnu metodu koja poziva apstraktne metode Builder-a

**PRIMERI:** \2009primeri\Graditelj

1. Ilustracija: strukturalni primer koji demonstrira Builder DP u kom su kompleksni objekti kreirani korak-po-korak. Kreiraju se različite reprezentacije objekta i obezbeđen je visok nivo kontrole izgradnje reprezenata deo-po-deo.

Izlaz:
Delovi Proizvoda -------
DeoA
DeoB

Delovi Proizvoda -------
DeoX
DeoY

2. Realan slučaj: demonstrira Builder putem konstrukcije korak-po-korak različitih vozila. (motorcikl, automobil, motor). Radnja (Direktor) koristi GraditeljVozila da konstruiše različita vozila u nizu sekvencijalnih koraka.

Izlaz:
---
Tip vozila prosiruje: Biciklo
Okvir prosiruje : Okvir bicikla
Motor prosiruje : nema
Broj tockova: 2
Broj vrata : 0

---
Tip vozila prosiruje: Automobil
Okvir prosiruje : Automobilska skoljka
Motor prosiruje : 2500 cc
Broj tockova: 4
Broj vrata : 4

---
Tip vozila prosiruje: Motorcikl
Okvir prosiruje : Okvir motorcikla
Motor prosiruje : 500 cc
Broj tockova: 2
Broj vrata : 0

3. Napisati program koja ce kreirati razlicite dvocifrene brojeve pri cemu ceprva cifra biti neparna a druga parna. Koristiti AbstractFactory, Factory, Builder DP u realizaciji.

# Prototip – kreacioni objektni DP

## Drugo ime:

Polimorfna kopija ( engl. Prototype)

## Namena:

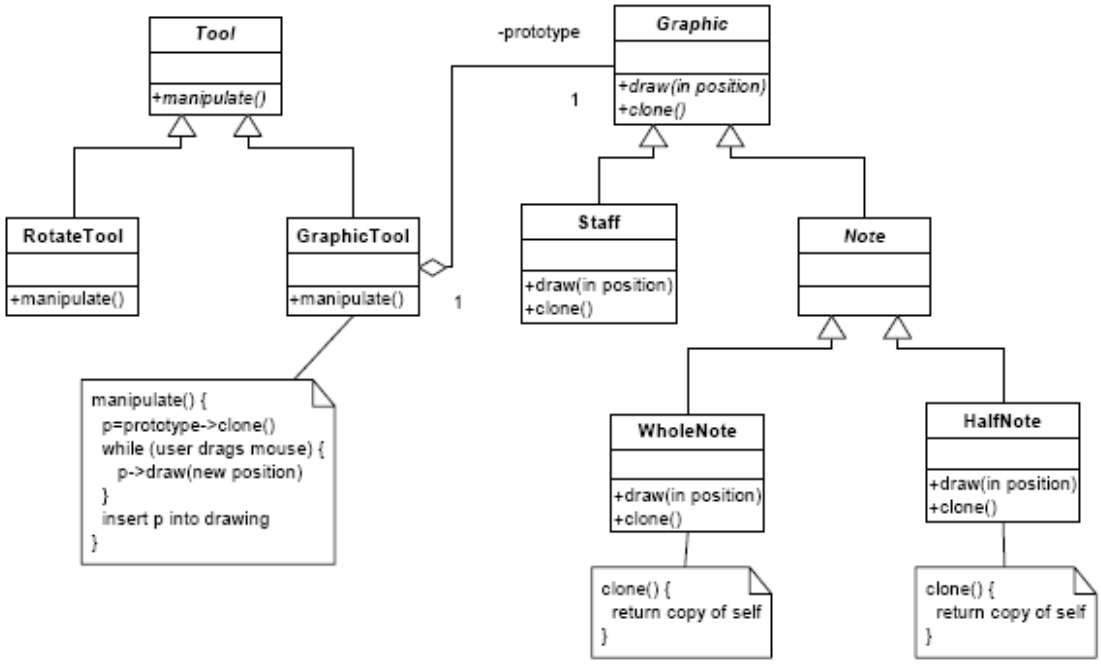
– kreira nove objekte kopiranjem prototipa

## Primenljivost: uzorak treba koristiti

–kada sistem treba da bude nezavisan od toga kako se njegovi proizvodi kreiraju i predstavljaju
–kada se klase specificiraju u vreme izvršenja, npr. dinamičkim učitavanjem
–kada treba izbeći izgradnju hijerarhije fabrika paralelno sa hijerarhijom proizvoda
- kada instance jedne klase mogu da imaju jednu od malog broja različitih kombinacija stanja. Može da bude pogodnije instalirati odgovarajući broj prototipova i klonirati ih, nego ručno instancirati klasu, svaki put są odgovarajućim stanjem.

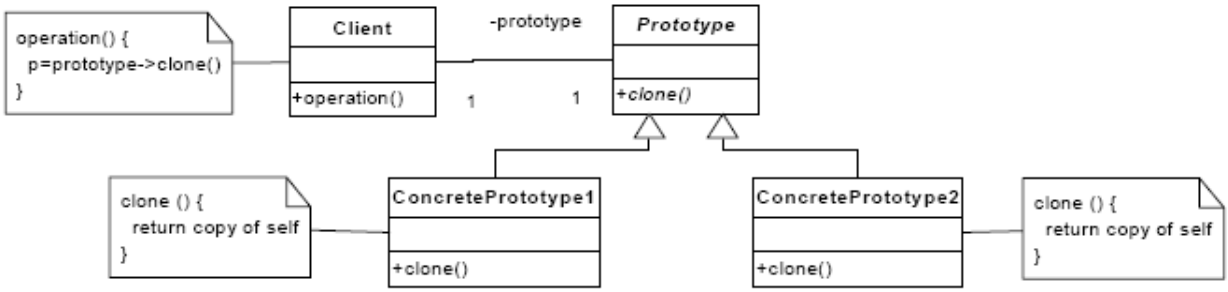
**Kratak opis problema**

1. editor za muzičke partiture bi se mogao realizovati tako što se:
- prilagodi opšti radni okvir za grafičke editore
- dodaju novi objekti koji predstavljaju note, pauze i druge muzičke simbole
2. radni okvir editora može imati paletu alata za dodavanje muzičkih objekata partituri
- paleta bi takođe uključila i alate za selektovanje, pomeranje i druge radnje nad objektima
- korisnik bi kliknuo na alat "četvrtinka" i dodao ovu u partituru
- korisnik bi mogao da koristi alat za pomeranje da premesti notu na drugu notnu liniju
3. radni okvir nudi
- apstraktnu klasu Graphics za grafičke komponente (npr. note, linije)
- apstraktnu klasu Tool za definisanje alata kakvi su u paleti (dodavanje, pomeranje, rotacija,...)
- klasu GraphicTool, potklasu klase Tool, za kreiranje i dodavanje objekata u dokument
4. problem za projektanta radnog okvira je klasa GraphicTool
-klase za note i notne linije su specifične za aplikaciju, nepoznate radnom okviru
-mogla bi se praviti potklasa GraphicTool za svaki muzički objekat, ali ovih ima mnogo




5. rešenje:
-GraphicTool kreira novi objekat klase Graphic kopiranjem (kloniranjem) objekta potklase Graphic
-dotični objekat (instanca potklase Graphic) se naziva prototipom
-objekat GraphicTool je parametrizovan prototipom koji klonira i dodaje u dokument
6. potrebno da sve potklase klase Graphic imaju operaciju clone()
7. u primeru s početka, muzičkom editoru, alati za kreiranje muzičkih objekata su objekti GraphicTool
-dotična instanca GraphicTool se inicijalizuje različitim prototipom
- svaki objekat GraphicTool proizvodi muzički objekat kloniranjem njegovog prototipa

DIJAGRAM PRIKAZA




Učesnici:
**Prototype klasa(Graphic)** deklariše interfejs za sopstveno kloniranje
**ConcretePrototype klasa (Staff, WholeNote, HalfNote)**
implementira operaciju za sopstveno kloniranje
**Client klasa (GraphicTool)**
kreira novi objekat zahtevom prototipu da se klonira

Saradnja:
klijent zahteva od prototipa da se klonira

Prednosti i mane:
dobre strane:
- redukuje broj imena koje klijent treba da poznaje
- dodavanje i uklanjanje proizvoda u vreme izvršenja (registracija prototipske instance)

MANE:
- svaka potklasa mora implementirati operaciju clone()
-to je komplikovano ako klasa već postoji ili ako sadrži objekte koji ne podržavaju kopiranje

Bliski DP:

1. Abstract Factory i Prototype su (na neki način) konkurentni DP, ali se AbstractFactory može i dopuniti sa Prototype:
-fabrika može sadržati skup prototipova na osnovu kojih klonira i vraća proizvode
2. projekti koji dosta koriste Composite i Decorator mogu imati koristi i od Prototype

**PRIMERI:** \2009primeri\Prototip
1. Ilustracija: strukturalni primer demonstrira Prototip DP u kom su novi objektim postojećih objekata (prototipova) iste klase.

Izlaz:
Cloned: I
Cloned: II

2. Realan slučaj: demonstrira Prototip u kom su kreirani novi Color objekti kopiranjem prototipa korisnički definisanih boja.

Izlaz:
RGB values are: 255, 0, 0
RGB values are: 128,211,128
RGB values are: 211, 34, 20