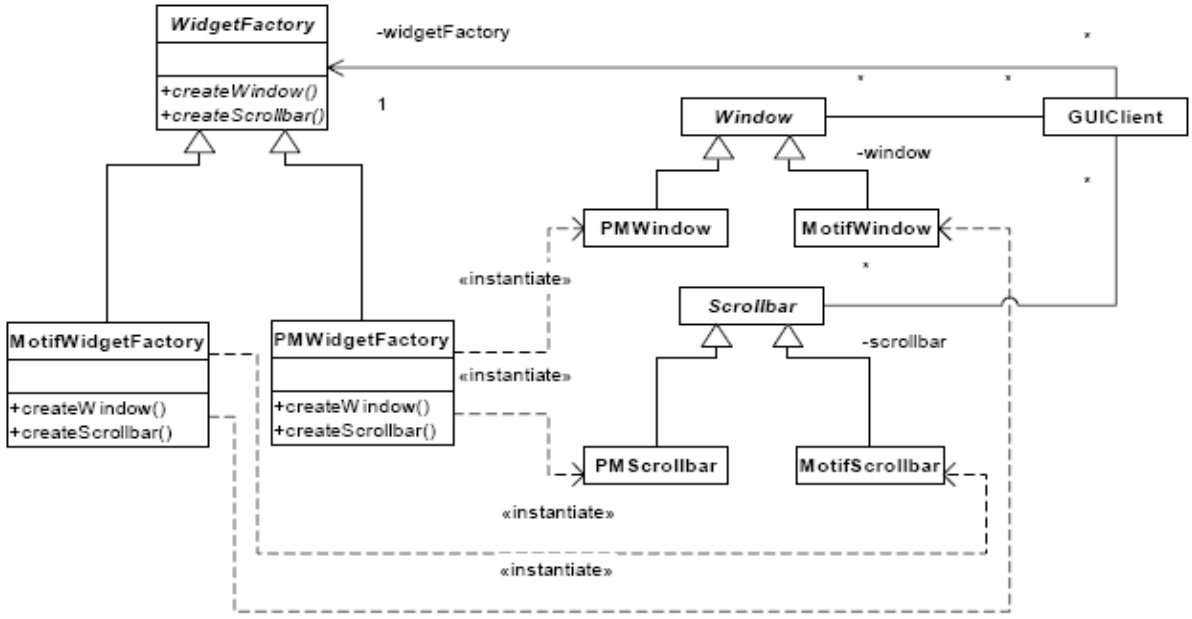
Projektni uzorci - kreacioni (Creational design patterns-DP)

Apstraktna fabrika (engl. Abstract factory) – kreacioni objektni DP

Drugo ime:
Kit

Namena:
Obezbeđuje interfejs za kreiranje familija logički ili funkcionalno povezanih objekata bez specificiranja konkretnih klasa familije objekata.

Kratak opis problema
1. razmatra se user interface toolkit, odnosno klasna biblioteka za realizaciju korisničkog interfejsa
-biblioteka podržava više standarda look-and-fill (izgleda-i-osećaja)
-primeri: Motif i Presentation Manager (PM)
2. različit izgled i osećaj definiše različite pojave i ponašanje komponenata (widget)
-primeri komponenata: prozor (window), klizači (scroll bars) i dugmad (buttons)
3. važno je da aplikacija bude portabilna između standarda look-and-fill
-da bi se ostvarila portabilnost ne smeju se fiksno (hard code) kodirati kontrole
-kreiranje objekata klasa koje definišu specifični look-and-fill svuda po aplikaciji čini aplikaciju loše portabilnom
4. problem se rešava definisanjem apstraktne klase fabrike komponenata WidgetFactory
-klasa definiše interfejs za kreiranje svih bazičnih vrsta komponenata
-za svaku bazičnu vrstu komponente postoji apstraktan metod za kreiranje (createWindow, createScrollBar)



5. potrebna je po jedna izvedena klasa stvarne fabrike za svaki standard look-and-fill (MotifWidgetFactory, PMWidgetFactory)
- te klase implementiraju operacije kreiranja koje je propisala apstraktna fabrika (createWindow(), createScrollBar())
- operacije kreiranja poštuju specifičan standard look-and-fill
6. potrebna je po jedna apstraktna klasa za svaku vrstu komponenata (npr. klase Window, ScrollBar)
- konkretne potklase implementiraju komponentu za specifičan standard izgleda (npr. MotifWindow, PMWindow)
7. klijenti pozivaju metode apstraktne fabrike (createWindow, createScrollBar) da proizvedu objekte komponenata
-klijenti nisu svesni konkretne klase koju koriste
8. ostaju nezavisni od look-and-fill
- klijenti moraju da poštuju interfejs koji definiše apstraktna fabrika
- osim apstraktne fabrike klijenti vide i apstraktne komponente (neopterećene specifičnostima izgleda i osećaja)

Primer: Kreirati aplikaciju koja će ispisivati poruku "MotifDugme" ili "PMDugme" u zavisnosti od upotrebljene Motif ili PM fabrike.

```
interface WidgetFactory {
    public Button createButton();
}


class MotifFactory implements WidgetFactory {
    public Button createButton() {
```

```
        return new MotifButton();
    }
}


class PMFactory implements WidgetFactory {
    public Button createButton() {
        return new PMButton();
    }
}

interface Button {
    public void paint();
}


class MotifButton implements Button {
    public void paint() {
        System.out.println("MotifDugme");
    }
}


class PMButton implements Button {
    public void paint() {
        System.out.println("PMDugme");
    }
}


class Application {
    public Application(WidgetFactory factory){
        Button button = factory.createButton();
        button.paint();
    }
}

public class ApplicationRunner {
    public static void main(String[] args) {
        new Application(createOsSpecificFactory());
    }

    public static WidgetFactory createOsSpecificFactory() {
        int sys = readFromConfigFile("User_Interface_Toolkit");
        if (sys == 0) {
            return new MotifFactory();
        } else {
            return new PMFactory();
        }
    }
}
```

Primenljivost

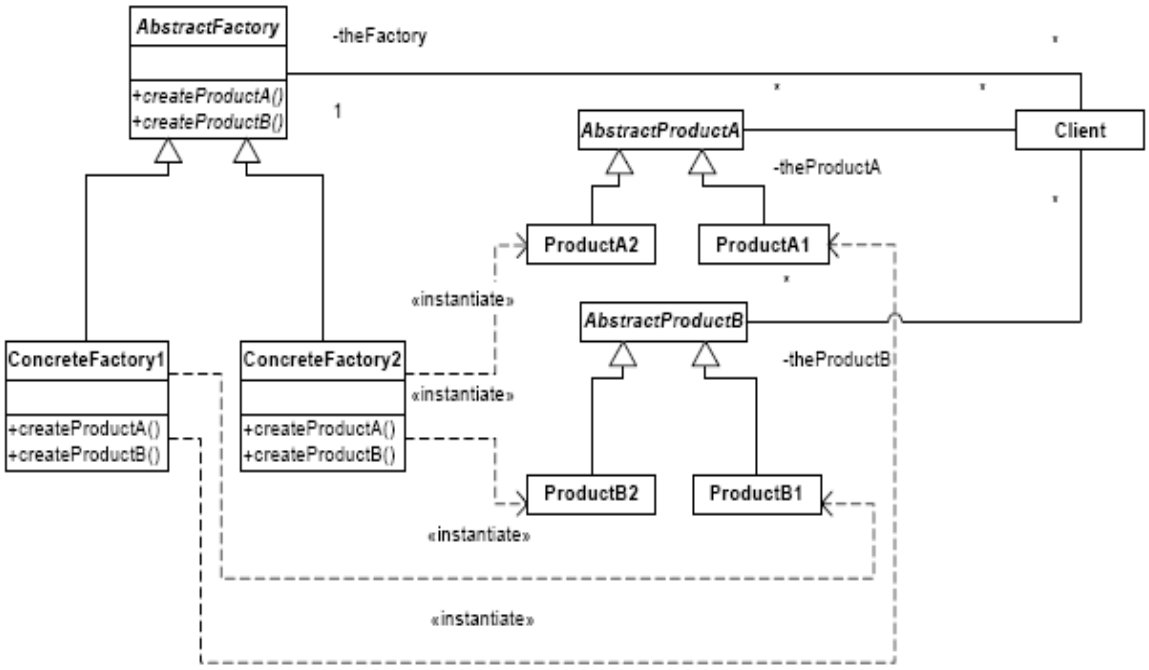
Ovaj DP treba koristiti kada:

• sistem treba da bude nezavisan od načina kreiranja i predstavljanja proizvoda

• sistem treba da bude konfigurisan jednom od više familija proizvoda

• potrebno je forsirati da proizvodi iz familije budu korišćeni zajedno
• potrebno je ponuditi klasnu biblioteku proizvoda otkrivajući samo interfejse, ne implementacije

DIJAGRAM PRIKAZA



Učesnici:

AbstractFactory klasa

deklariše interfejs za operacije koje kreiraju apstraktne objekte proizvoda

ConcreteFactory klasa

implementira operacije koje kreiraju konkretne objekte proizvoda

AbstractProduct klasa

deklariše interfejs za određeni tip objekta proizvoda

ConcreteProduct klasa

definiše objekat proizvoda koji će biti kreiran pomoću odgovarajuće konkretne fabrike i implementira interfejs apstraktnog proizvoda

Client klasa

koristi samo interfejse deklarisane pomoću apstraktne fabrike i apstraktnog proizvoda

Saradnja:

–apstraktna fabrika odlaže kreiranje objekata proizvoda do njenih potklasa, konkretnih fabrika

–uobičajeno je da se u vreme izvršavanja kreira samo po jedna instanca klase ConcreteFactory. Ta konkretna fabrika kreira objekte Product određene implementacije. Da bi se kreirali drukčiji objekti Product,

Client-i treba da koriste drugu KonkretnuFabriku.

Prednosti i mane:

Izolacija konkretnih klasa
- klijent manipuliše proizvodima kroz njihove apstraktne interfejse
- klijent ne koristi imena konkretnih klasa proizvoda (čak ni za njihovo stvaranje)

Olakšava izmenu familije proizvoda
- klasa konkretne fabrike se pojavljuje samo jednom u aplikaciji i to tamo gde se pravi njen objekat
- aplikacija može koristiti različite konfiguracije proizvoda menjanjem konkretne fabrike
- pošto apstraktna fabrika kreira kompletnu familiju, cela familija se menja odjednom

Unapređuje konzistenciju među proizvodima
- aplikacija koristi objekte proizvoda samo iz jedne familije u jednom trenutku

Mana: podrška novoj vrsti proizvoda nije jednostavna
- razlog je taj što apstraktna fabrika fiksira skup proizvoda kojise mogu kreirati
- podrška novog proizvoda zahteva proširenje apstraktne fabrike i svih potklasa

Bliski DP:

1. Abstract Factory se često implementira sa Factory Method ili koristeći Prototype
- konkretne fabrike mogu realizovati fabričke metode (uobičajeno), ali mogu se i paramerizovati prototipskim objektima proizvoda i praviti njihove kolonove

2. KonkretnaFabrika je često Singleton, jer aplikaciji je obično potrebna samo jedna instanca neke klase KonkretnaFabrika za jednu familiju proizvoda.

PRIMER: ..\2009primeri\ApstraktnaFabrika

1. Ilustracija: strukturalni primer koji demonstrira upotrebu Abstract Factory DP putem kreiranja paralelnih hijerarhija objekata. Kreiranje objekta je apstrahovana i nema potrebe da se hardkoduju nazivi klasa u kodu klijenta.

Izlaz:
ProizvodB1 interaguje sa ProizvodA1
ProizvodB2 interaguje sa ProizvodA2

2. Realan slučaj: koristeći različite fabrike demonstrira se kreacija različitih životinjskih svetova (npr. za potrebe računarske igrice). Iako fabrike kontineta Amerika i Afrika kreiraju različite životinje, interakcije među životinjama su iste (u npr. lancu ishrane).

Izlaz:
Lav jede Zebra
Puma jede Bizon

3. PrimerAbstracFactory.java: Napraviti klijentsku klasu koja ce kreirati razlicite dvocifrene brojeve pri cemu ce prva cifra biti neparna, a druga parna. Koristiti AbstractFactory u realizaciji.
Izlaz:
Kreiran je broj: 58
Kreiran je broj: 72

Fabrički metod (engl. Factory Method) – kreacioni klasni DP

Drugo ime:
Virtual Constructor

Namena:
Definiše interfejs za pravljenje objekata, ali dozvoljava klasi da prepusti pravljenje primerka podklasi.

Primenljivost
Ovaj DP treba koristiti kada:
• Klasa ne može da zna klasu objekta koji treba da pravi.
• Klasa namerno hoće da njene podklase odrede objekte koje prave.

Kratak opis problema

Često postoji potreba da se u objektno-orijentisanim sistemima kreiraju objekti čiji konkretan tip nije poznat na mestu kreiranja.
Neka, na primer, u sistemu postoji apstraktna klasa *Document*, čije su podklase *TextDocument* i *DrawingDocument* .
Dalje, neka postoji klasa *Application*, sa podklasama *TextApplication* i *DrawingApplication*.

Klasa *Application* sadrži zajednički kod manipulacije dokumentima. Taj kod je takav da ne zavisi od konkretne vrste dokumenata kojima se manipuliše. Na primer, *Application* može da obavlja operacije otvaranja, zatvaranja dokumenata, snimanja na disku, prikazivanje na ekranu ,...
Ali kako *Application* ima potrebu i da kreira nove dokumente, nastaje problem zbog toga što *Application* "ne poznaje" tip dokumenata koje će da kreira (*TextDocument* ili *DrawingDocument*), jer ovaj tip zavisi od konkretnog tipa aplikacije (*TextApplication* ili *DrawingApplication*) koja se izvršava. Zato je ovde koristan factory method DP.

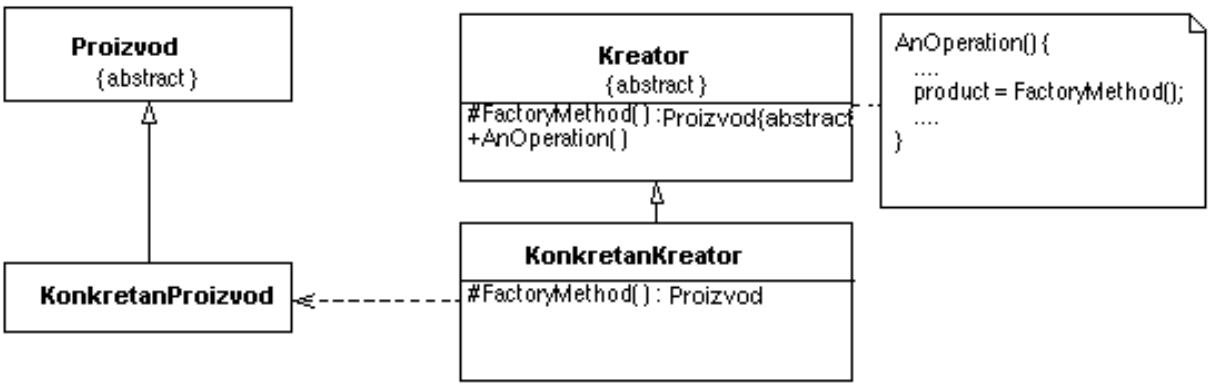
Ideja:
Kreiranje objekata preneti na konkretnu aplikaciju koja se izvršava i time "sakriti" od aplikacije koja koristi te objekte . I obezbedti factory metod *Create()* koji je dostupan aplikaciji, ali čija implementacija zavisi od tipa konkretne aplikacije koja se izvršava.

Primer: Kreirajte klasu PizzaAll koja ce kreirati raspoložive pice i njihove cene.

abstract class Pizza {

```
    public abstract double getPrice();
}

class SunkaPecurkaPizza extends Pizza {
    public double getPrice() {
        return 8.5;
    }
}

class DeluxePizza extends Pizza {
    public double getPrice() {
        return 10.5;
    }
}

class VegeterijanskaPizza extends Pizza {
    public double getPrice() {
        return 11.5;
    }
}

class PizzaFactory {
    public enum PizzaType {SunkaPecurka, Deluxe, Vegeterijanska}

    public static Pizza createPizza(PizzaType pizzaType) {
        switch (pizzaType) {
            case SunkaPecurka: return new SunkaPecurkaPizza();
            case Deluxe: return new DeluxePizza();
            case Vegeterijanska: return new VegeterijanskaPizza();
        }
        throw new IllegalArgumentException("Tip " + pizzaType + "nije
dozvoljen.");
    }
}

class PizzaAll {

    public static void main (String args[]) {
        for (PizzaFactory.PizzaType pizzaType : PizzaFactory.PizzaType.values())
{
            System.out.println("Cena za picu " + pizzaType + " iznosi " +
PizzaFactory.createPizza(pizzaType).getPrice());
        }
    }
}
```

DIJAGRAM PRIKAZA – klasni dijagram



Učesnici:

Proizvod (ili u našem primeru *Document*)
definiše interfejs objekta koji pravi FactoryMethod
KonkretanProizvod (u našem primeru konkretan dokument)
implementira interfejs proizvod
Kreator (ili u našem primeru *Application*)
deklariše FactoryMethod koji vraća objekat tipa Proizvod. Kreator može da pozove FactoryMethod radi pravljenja objekta tipa Proizvod.
KonkretanKreator (ili u našem primeru konkretna aplikacija)
može da nadjača factory method da bi vratio instancu KonkretnogProizvoda

Saradnja:
Kao što je već navedeno, od *Application* je "sakriveno" kreiranje različitih objekata tipa *Document*. Ovo kreiranje je ostavljeno FactoryMethod-u *Create()*, koji je apstraktan i koji je adekvatno definisan u potklasama (*TextApplication* i *DrawingApplication*). Time, dovoljno opšti kod (u smislu da je zadužen za manipulaciju dokumentima nezavisno od njhovog tipa i da je zajednički za sve aplikacije) unutar klase *Application* može da kreira odgovarajuće objekte. Ako je trenutna aplikacija *TextApplication*/ *DrawingApplication*, taj kod (putem metoda *Create()*) može da kreira *TextDocument* / *DrawingDocument* objekte.
Relativno je jednostavno zameniti delove ovakvog sistema (npr. zameniti objekte *TextDocument* objektima *WordProcessorDocument*), kao i proširiti sistem (npr. dodavanje *PaintDocument* i *PaintApplication* klasa).

Prednosti i mane:

-Factory Method eliminiše potrebu **da se u kôd ugrađuju klase specifične za pojedine aplikacije.** Naime, kôd radi sa interfejsom Proizvod, te on može da radi i sa ma kojom korisnički definisanom klasom KonkretanProizvod.

Pored ovoga postoje još dve dodatne posledice uzorka Factory Method:

• Obezbeđuje sredstvo za podklase. Pravljenje objekta unutar klase pomoću Factory metoda uvek je fleksibilnije od neposrednog pravljenja objekta. Factory metod daje podklasama sredstvo za obezbeđenje proširene verzije objekta.

•**omogućuje da i klijenti (a ne samo Proizvođači) mogu da iskoriste factory metode** što se naročito odnosi na pojavu paralelnih hijerarhija klasa

Nastanak paralelne hijerarhije klasa je u situaciji kada jedna klasa delegira neke svoje odgovorosti drugoj klasi. Na primer, pri implementaciji interaktivne manipulacije grafičkim objektima (istezanje,rotiranje pomoću miša,..) , često je potrebno sačuvati, ali i ažurirati informacije o stanju manipulisanja u zadatom trenuku. No, to stanje je potrebno samo za vreme manipulacije i nije nužno pamtiti to stanje u objektu oblik. Ali, mora se još voditi računa i da se različiti oblici različito ponašaju kada korisnik njima manipuliše (nije isto istezati liniju i istezati tekst).

MANA:

klijent pravi potklasu klase Proizvođač radi kreiranja objekta neke određene klase KonkretanProizvod.

Ovo nije problematično, ako klijent, inače, će praviti potklasu klase Proizvođač. U suprotnom, to je jedna obaveza više za klijenta.

Bliski DP:

–*Factory Method* se obično poziva iz *TemplateMethod*

–*Abstract Factory* se često implementira sa *Factory Method*

Primeri korišćenja factory method-a u realnim sistemima

* IDbCommand.CreateParameter (u ADO.NET) je primer upotrebe factory metoda za povezivanje paralelnih klasnih hijerarhija.

*QMainWindow::createPopupMenu (u Qt) je factory metod declarisan u framework-u koji se može nadjačati (overridde) u kodu aplikacije.

* u paketu javax.xml.parsers koriste se nekoliko fabrika, kao, na primer javax.xml.parsers.DocumentBuilderFactory ili javax.xml.parsers.SAXParserFactory.

- sistem Orbix ORB (Object Request Broker) upotrebljava Factory method radi generisanja adekvatne vrste proxy-ja u situacijama kada objekat zahteva referencu na udaljeni objekat.
- Java RMI (Remote Method Invocation)

Standardi za slanje poruka

Postoji nekoliko siroko prihvacenih standarda za slanje poruka udaljenim objektima. To su:

- CORBA (Common Object Request Broker Architecture)
- RMI (Remote Method Invocation)
- COM (DCOM)
- SOAP (Simple Object Acces Protocol)

CORBA

To je standard koji omogucava da proizvodi raznih proizvodaca medusobno sadraduju.
CORBA ima standarni API za vecinu funkcija ORB-a: inicijizacija ORB-a, pozivanje metoda udaljenih objekata, prevodjenje tipova podataka iz jednog u drugi programski jezik itd.
Objekti obavljaju komunikaciju preko Internet Inter-ORB protokola (IIOP) protokola.
Vecina velikih proizvodjaca podrzava CORBA-standard, osim Microsofta.
CORBA funkcionise tako sto kreira kopije udaljenih objekata (tj. distribuirani objekat u lokalnom adresnom prostoru) koji se naziva stab (eng. stub- klada, odsecak).
CORBA je projektovana da radi sa velikim skupom programskih jezika (C, C++, ADA, Smalltalk, Java, ...) i zbog toga nudi samo ogranicen skup osobina zajednicki svim jezicima.

RMI

RMI je ORB ugraden u JDK 1.1.
To je objektno orijentisana verzija RPC-a (Remote Procedure Call).
RMI stvara iluziju da se udaljeni objekti pozivaju lokalno (ne moramo da brinemo o osnovnim mehanizmima kao sto su sockets i dr.)
RMI je namenjen samo programskom jeziku Java.

PRIMERI:

http://www.matf.bg.ac.rs/~jelenagr/rs2/DPjhp/example/

1. Fabrika/Ilustracija: strukturalni primer koji demonstrira upotrebu Factory DP nudeći fleksibilno kreiranje različitih objekata. Apstraktna klasa može obezbediti podarazumevani objekat, ali svaka podklasa može instancirati proširenu verziju objekta.

Izlaz:

Kreiran je KonkretanProizvodA
Kreiran je KonkretanProizvodB

2. Fabrika/Realan slučaj: demonstrira upotrebu Factory DP nudeći fleksibilno kreiranje različitih dokumenata. Izvedene klase Izveštaj i Rezime „instanciraju“ proširenja klase Dokument. Factory Method se poziva u konstruktoru osnovne klase Dokument.

Izlaz:
Rezime -------
SekcijaVestina
SekcijaSkolovanje
SekcijaIskustvo

Izvestaj -------
UvodnaSekcija
SekcijaRezultati
SekcijaZakljucaka
SumarnaSekcija
SekcijaBibliografije

3. FactoryDP/FactoryPatter.java: Napraviti klijentsku klasu FactoryPattern koji ilustruje factory method DP. Za instanciranog zaposlenog/nezaposlenog u preduzecu na izlaz treba da se ispise poruka o mogucnosti da zaposleni/nezaposleni koristi neki resurs preduzeca