Projektni uzorci (Design Patterns-DP)

Projektni uzorci, obrazci ili šabloni predstavljaju opšta, isprobana i ponovo iskoristiva dizajnerska rešenja koja mogu da se primene na uobičajene probleme u (objektno orijentisanom) razvoju softvera.

Osnovni elementi projektnog uzorka su:
- naziv uzorka
- postavka problema
- opis rešenja
- diskusija konsekvenci

Inače, sama specifikacija je predložena od strane Gamma, Eric; Helm, Richard; Johnson, Ralph ; Vlissides, John u knjizi **Design Patterns. Elements of Reusable Software**, Addison-Wesley, 1995, ali nije imperativ pri opisu design pattern-a.
Tim pre što su se autori u izboru programskih jezika opredelili za Smalltalk i C++, te je takav izbor uticao i na procenu i komentare šta se može, a šta ne relativno jednostavno implementirati.

Uzorci

Postoje tri nivoa apstrakcije u OO programiranju kojima može biti pridružen neki uzorak.

Idiomi su uzorci najnižeg nivoa koji zavise od specifične implementacione tehnologije, kao što je npr. programski jezik. Na primer, foreach idiom u C#: Pri obilasku niza foreach petlje su brže od for petlji, jer ne proveravaju granice niza više puta.

```
static void Main(string[] args)
        {
            int [] list = new int[] { 1,2,3,4,5,6,7,8,9,10};
            // slow:
            for (int index = 0; index < list.Length; index++)
                Console.WriteLine(list[index]);
            // fast:
            foreach (int el in list)
                Console.WriteLine(el);
        }
```

Projektni uzorci su uzorci koji su nezavisni od konkrente implementacione tehnologije. Oni su mikroarhitektura: oni sadrže strukturu koja može biti složena ali ne i dovoljno velika da bi se za nju reklo da je podsistem. Kada kažemo podsistem mislimo na jednu od osnovnih funkcija koje sačinjavaju jedan sistem (program).

Uzorci

Okviri su uzori sistemskog nivoa. Panu Viljam za okvire kaže: “ Okviri ukazuju na kolekciju konkretnih klasa, koje zajedno rade kako bi izvršili postavljeni parametrizirani zadatak”.
Oni su projektovani tako da sadrže kompletan kod jedne od osnovnih funkcija sistema ili celog sistema, koji može biti proširen za konkretnu aplikaciju. Navedeni kod pokriva okvir problema. Dakle, okvir obezbeđuje opšte rešenje a različite aplikacije ga koriste i proširuju svojim specifičnostima. Jedan okvir može da sadrži druge uzore različitih nivoa apstrakcije (idiome, uzore projektovanja i okvire).

SPECIJALNI UZORI

Anti-uzori – Oni uzori koji su se u praksi pokazali kao loši (ili ne rade ili delimično rade) nazivaju se anti-uzori. Na anti-uzore ukazali su u nezavisnim istraživanjima Sam Adams i Andrew Koenig. U tim istraživanjima navedeni autori su obrazlagali i objašnjavali posledice loših rešenja kod anti-uzora.

DP-sažet pogled unazad

- *Svaki pattern opisuje problem koji se stalno ponavlja u našem okruženju i potom opisuje suštinu rešenja problema tako da se to rešenje može iskoristiti milion puta, a da se dva puta ne ponovi na isti način.* Cristopher Alexander
- Cristopher Alexander je arhitekta koji je prvi proučavao pattern-e u građevinarstvu i sa grupom kolega razvijao pattern language za izgradnju pattern-a. "*A Pattern Language*" 1977, Oxford University Press, New York Cristopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King , Shlomo Angel
- Donnald E. Knuth je radio na katalogizaciji softverskog znanja, ali se usredsredio samo na opisivanje algoritama.
- Kent Back and Ward Cunningham, Textronix, OOPSLA'87 (propagiranje rada C.Alexandera u oblasti softvera i objavljivanje članaka o Smalltalk pattern-ima).
- Erich Gamma -doktorska disertacija *"Object-Oriented Software Development based on ET++ :Design Patterns, Class Library, Tools"* - Univerzitet u Cirihu, 1991.
- James O. Coplien (*"Advanced C++:Programming Styles and Idioms"* , 1992.) se bavi pattern-ima specifičnim za C++ .
- **GoF:** Gamma, Helm, Johnson, Vlissides *"Object Oriented Patterns Book"* , 1994 (JOOP (Journal of Object-Oriented Programming) je proglasio najboljom OO knjigom protekle decenije).
- počev od 1994: PLoP konferencije

Klasifikacija projektnih uzoraka

Klasifikacija uzoraka doprinosi boljem i bržem snalaženju prilikom traženja odgovarajućeg uzorka, a istovremeno i usmerava napore ka otkrivanju novih uzoraka.

Za klasifikaciju uzoraka koriste se dva kriterijuma:

- kriterijum namene, koji razvrstava uzorke u zavisnosti od toga čime se bave
  - kreiranjem objekata (*creational patterns*)
  - strukturom, tj. kompozicijom klasa i objekata (*structural patterns*)
  - ponašanjem, tj načinom interakcije objekata i klasa (*behavioral patterns*)

- kriterijum domena, koji razvrstava uzorke zavisno od toga da li se primarno primenjuju na klase ili objekte
  - klasni uzorci koji se fokusiraju na relacije između klasa i podklasa (*class patterns*)
  - objektni uzorci koji se fokusiraju na relacije između objekata (*object patterns*)

GoF katalog projektnih uzoraka

Prostor uzoraka		**Namena**		
		uzorci kreiranja	**uzorci strukture**	**uzorci ponašanja**
Domen	**klasni**	Factory Method	Adapter (class)	Interpreter Template Method
	objektni	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Odnosi među DP-ima

- neki uzorci se često koriste zajedno

npr. Kompozicija (*Composite*) i Iterator(*Iterator*) ili Posetilac (*Visitor*)

- neki uzorci su alternative jedni drugima

npr. Prototip (*Prototype*) i Apstraktna fabrika (*Abstract Factory*)

- neki uzorci različitih namena rezultuju u sličnoj implementaciji

npr. Kompozicija (*Composite*) i Dekorater (*Decorator*)

Uloga DP-a pri projektovanju više puta upotrebljivog objektno orijentisanog softvera

- **Iskusni softverski projektanti znaju da nije uvek neophodno svaki problem rešavati od početka. Zbog toga se u mnogim objektno orijentisanim sistemima često naliazi na iste uzorke klasa i objekata koji komuniciraju.** Projektant koji poznaje design pattern-e može ih iskoristiti pri rešavanju problema, bez potrebe da ih ponovo otkriva, jer primenjuje rešenja, koja su već ranije bila uspešno korišćena.
- Dakle, DP su pogodni za više puta upotrebljivi objektno orijentisani projekat, jer imenuju, apstrahuju i identifikuju ključne aspekte široke klase problema koji imaju slični pojavni oblik.
- Dalje, **DP identifikuju klase i primerke, te njihove uloge, interakcije, ali i distribuciju njihovih odgovornosti, te mogu da unaprede i dokumentaciju i održavanje postojećih sistema.**

Uloga DP-a pri projektovanju koje uzima u obzir promene

Projektovanje koje ne uzima u obzir promene rizikuje da se u budućnosti redizajnira. Same promene mogu da se odnose na novo definisanje ili izmenu implementacije klasa, promenu klijenata i ponovno testiranje. **Tome valja dodati da redizajniranje utiče na mnoge delove softverskog projekta, a nepredviđene promene su, uglavnom, skupe.**

- Design pattern-i su od velike pomoći i kod ovog aspekta projektovanja, jer obezbeđuju da sistem može da se na određeni način menja. Upravo ključ za što veću mogućnost ponovnog korišćenja i jeste predviđanje novih zahteva i promene postojećih zahteva. **Svaki design pattern dozvoljava variranje nekog aspekta strukture sistema nezavisno od drugih aspekata.**

Lista nekih od konkretnih i relativno čestih razloga za redizajniranje (praćen DP-om adekvatnim u opisanoj situaciji):

- **zavisnost od algoritma**
  Ako objekti zavise od nekog algoritma, onda pri promeni algoritma, moraju se menjati i objekti koji zavise od algoritma. Inače, tokom razvoja i kod ponovnog korišćenja, algoritmi se često proširuju, optimizuju i zamenjuju. Dakle, valja izolovati algoritme koji bi mogli da se menjaju.
  Design pattern-i od koristi: Iterator, Template Method, Visitor
- **pogodna promena klasa**
  Nekada je nužno zameniti klasu koja se relativno teško menja. Na primer, potreban Vam je izvorni kod, a nemate ga na raspolaganju. Ili promena te klase zahteva da se izmeni čitav niz postojećih klasa. Otuda je korisno upotrebiti one design pattern-e koji nude rešenja za takve situacije.
  Design pattern-i od koristi: Adapter, Decorater, Visitor

Lista nekih od konkretnih i relativno čestih razloga za redizajniranje (praćen DP-om adekvatnim u opisanoj situaciji):

- **posredno pravljenje objekata**
  Ponekad pravljenje objekata eksplicitnim navođenjem imena klase, zapravo, je vezivanje za određenu implementaciju, umesto za određeni interfejs, što može da iskomplikuje buduće izmene. U toj situaciju nudi se pravljenje objekata posredno.
  Design pattern-i od koristi: Factory Method, Abstract Factory, Prototype
- **delegiranje**
  Pod delegiranjem se podrazumeva mehanizam implementacije u kom jedan objekat prosleđuje/delegira zahtev drugom objektu. Sam delegat će izvršiti zahtev za potrebe originalnog objekta.
  Delegiranje je sredstvo da sastavljanje objekata postane podjednako kvalitetno za višestruko korišćenje kao i nasleđivanje. Mnogi DP-i proizvode dizajn u kome se proširivanje funkcionalnosti vrši pomoću potklasa. Inače, često postoje teškoće u prilagođavanju objekata pomoću potklase. Svaka nova klasa ima dodatne poslove implementiranja (inicijalizacija, finalizacija,...). Potom definisanje potklase zahteva i razumevanje roditeljske klase, a kreiranje potklasa može da dovede do ekspanzije klasa (npr. čak i za malo proširenje možda će se morati uvesti mnogo novih potklasa). Otuda su od značaja design pattern-i kod kojih se proširivanje funkcionalnosti vrši jednostavnim definisanjem jedne potklase i sastavljanjem njenih primeraka sa postojećim.
  Design pattern-i od koristi: Bridge, Composite, Observer

Uloga DP u JAVA AWT/SWING aplikacijama

- Observer DP, Composite DP, Memento DP, Strategy DP, Decorater DP su se pokazali kao vrlo korisni mehanizmi za upotrebu pri izgradnji user interface-a (radi obezbedjenja undo-redo mehanizma, ugnježdenih kontejnera, uskladjivanja funkcionalnosti medjusobno zavisnih delova, ...) o čemu već sada postoje izvesne dokumentacije i saopštenja sa workshop-ova sa ciljem da se odmakne i dalje od ideja i skica rešenja.

Naziv uzorka

Naziv uzorka se koristi da opiše projektni problem, njegova rešenja i konsekvence u par reči.

Uvođenje naziva uzorka:

- omogućava projektovanje na višem nivou apstrakcije
- pojednostavljuje komunikaciju u timu
- olakšava dokumentovanje projekta

Ponekad se u literaturi sreće više naziva za isti uzorak.

Postavka problema

Postavka problema:

- objašnjava problem i njegov kontekst
- opisuje u kojim situacijama se razmatrani uzorak primenjuje
- daje listu uslova koji se moraju ispuniti da bi se mogao primeniti dati uzorak
- opisuje strukture klasa i objekata

Opis rešenja

Opis rešenja:

- opisuje elemente koji predstavljaju dizajn, njihove relacije, odgovornosti i saradnje

- ne opisuje konkretan dizajn ili implementaciju, jer uzorak treba da posluži kao šablon koji se može primeniti u konkretnim slučajevima

Posledice

Posledice su rezultati primene projektnog uzorka. One su kritične za ispitivanje projektnih alternativa i razumevanje cene i dobiti zbog primene uzorka.

Posledice se obično odnose na:

- prostor
- vreme
- jezik
- implementacione detalje

Posledice utiču na:

- fleksibilnost sistema
- proširivost sistema
- portabilnost sistema

Katalog projektnih uzoraka

Za svaki uzorak, **katalog sadrži sledeće stavke**:

Ime uzorka i klasifikacija

Namena – odgovori na pitanja "šta radi?“, “čemu služi? “ i koji problem rešava? “

Drugi nazivi – isti uzorak može imati više naziva

Motivacija – scenario koji ilustruje projektni problem

Primenljivost – situacije u kojima se uzorak može primeniti

Struktura –grafička reprezentacija klasnih i objektnih dijagrama koji opisuju uzorak

Učesnici – klase i objekti koji učestvuju u uzorku i njihove odgovornosti

Kolaboracije – kako učesnici sarađuju da bi ispunili svoje odgovornosti

Posledice – diskusija dobrih i loših strana primene uzorka

Implementacija – preporuke i tehnike kojih treba biti svestan pri implementaciji

Primer koda – deo koda koji pokazuje kako se uzorak može implementirati

Poznate primene – primeri primene uzorka u realnim sistemima

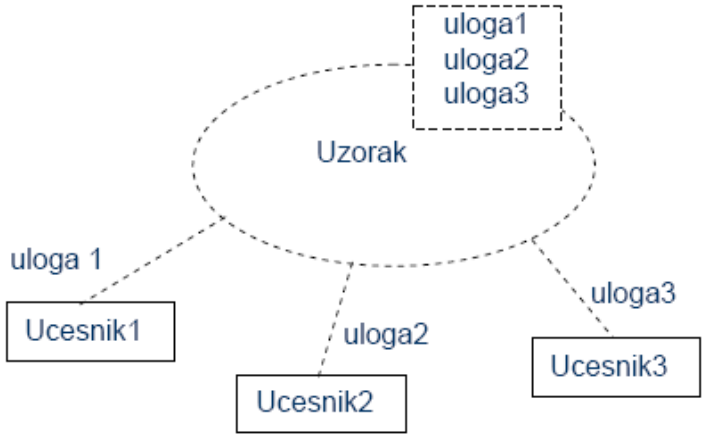
Korelisani uzorci – uzorci bliski sa datim, razlike među njima

UML notacija – grafički simbol za saradnju koja realizuje uzorak

UML 1 notacija za opis DP

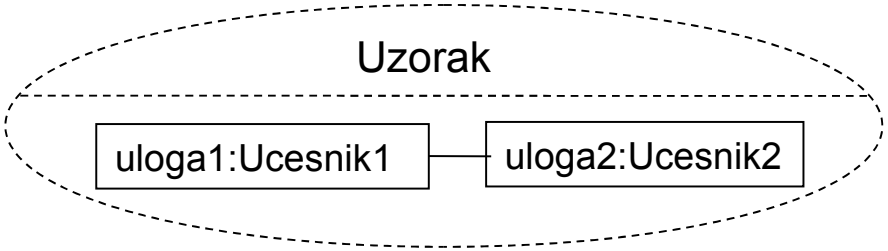
- Grafički simbol predstavlja parametrizovanu saradnju, gde

formalni parametri saradnje su uloge učesnika,
stvarni parametri su učesnici – konkretne klase

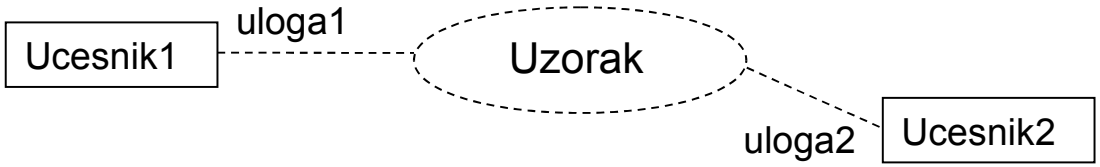


UML 2 notacija za opis DP

- UML 2: ne koristi se parametrizovana saradnja
- Prvi način: klase učesnici se crtaju u okviru saradnje



- Drugi način: klase učesnici se crtaju izvan saradnje



Singleton DP

Ime uzorka i klasifikacija

- *Singleton* – kreacioni objektni DP

Namena

- obezbeđuje da klasa ima samo jednu instancu i daje globalni pristup toj instanci

Motivacija

za neke klase je važno obezbediti da imaju samo po jednu instancu (u sistemu može da postoji više štampača, ali treba da postoji samo jedan dispečer štampe).

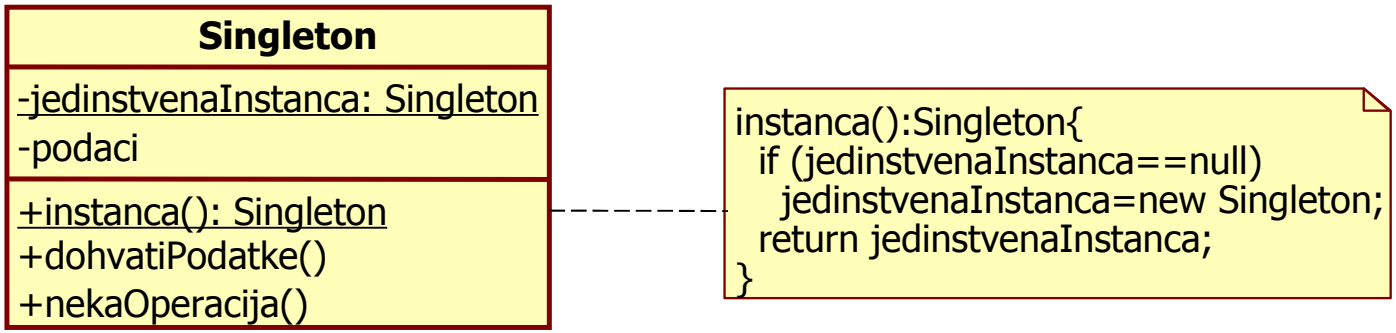
Da li je rešenje globalna promenljiva? Ne! Iako globalna promenljiva obezbeđuje pristup objektu, ali ne sprečava kreiranje više instanci objekta

Bolje rešenje je da klasa bude sama odgovorna za jedinstvenost svoje instance =>Singleton DP

Primenljivost

- *Singleton* uzorak se koristi kada:
  - ✓ Mora postojati tačno jedna instanca klase i ona mora biti pristupačna klijentima preko svima poznate tačke pristupa
  - ✓ Ta jedina instanca može da se proširuje potklasama i treba da važi da klijenti mogu da koriste instancu izvedene klase bez potrebe da modifikuju svoj kod

Struktura



Učesnici

- Singleton klasa
  - ✓ Definiše operaciju `instanca()` koja omogućava klijentima pristup do jedinstvene instance.
  - ✓ Može biti odgovorna za kreiranje vlastite instance

Saradnja

- Klijenti pristupaju objektu *Singleton* klase isključivo kroz klasnu operaciju `instanca()`

Posledice – dobre strane

- Kontrolisani pristup do jedine instance
  - ✓ pošto klasa Singleton (sa slike) enkapsulira jedinu instancu, ona može imati striktnu kontrolu nad tim kako i kada klijenti pristupaju instanci
- Redukovan prostor imena
  - ✓ *Singleton* uzorak je bolji koncept od globalne promenljive; on izbegava opterećivanje prostora imena globalnom promenljivom koja čuva jedinu instancu
- Dozvoljeno doterivanje operacija i reprezentacije
  - ✓ iz Singleton klase se može izvoditi
  - ✓ aplikaciju je lako konfigurisati instancom izvedene klase, čak i u vreme izvršenja
- Dozvoljava kontrolisano povećanje broja instanci
  - ✓ uzorak omogućava projektantu da se po maloj ceni predomisli i poveća broj dozvoljenih instanci
- Fleksibinost u odnosu na operacije klase
  - ✓ drugi način za realizaciju funkcionalnosti *Singleton*-a je uslužna (*utility*) klasa
  - ✓ sa uslužnom klasom je komplikovano promeniti broj instanci

Implementacija Java

```
class Singleton {
    public static Singleton instance(){
     if (_instance == null) _instance = new Singleton();
     return _instance;
    }
    public void singletonOperation(){/*...*/};
         protected Singleton(){}
    private static Singleton _instance=null;
    };
    // koriscenje:
    Singleton.instance().singletonOperation();
```

Poznate primene

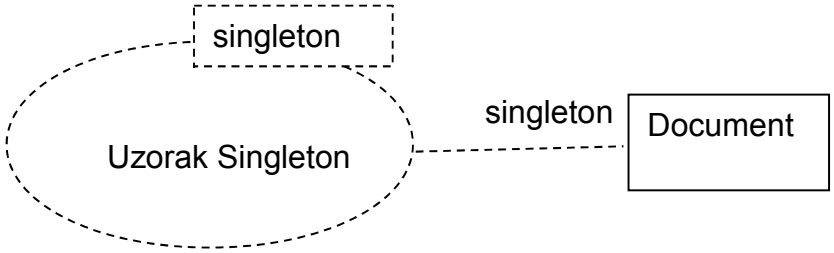
- u single-document aplikacijama klasa Document je *Singleton*

Korelisani uzorci

mnogi uzorci koriste *Singleton* (*Abstract Factory*, *Builder*, *Prototype*, *Facade*). Postoje primeri kada Factory mora biti jedinstven, jer u suprotnom pojavili bi se bočni efekti pri kreiranju objekata. Na taj način se i sprečava instanciranje klase pre njene upotrebe.

Notacija

- UML 1



- UML 2