UML

- Objedinjeni jezik za modelovanje - UML (Unified Modeling Language)
- UML predstavlja standardizovani jezik i grafičku notaciju za
  - vizuelizaciju,
  - specifikaciju,
  - modelovanje i
  - dokumentovanje

  delova softverskog sistema koji se projektuje
- UML predstavlja zajednički “rečnik” za sporazumevanje između osoba uključenih u projekovanje i razvoj nekog softverskog sistema

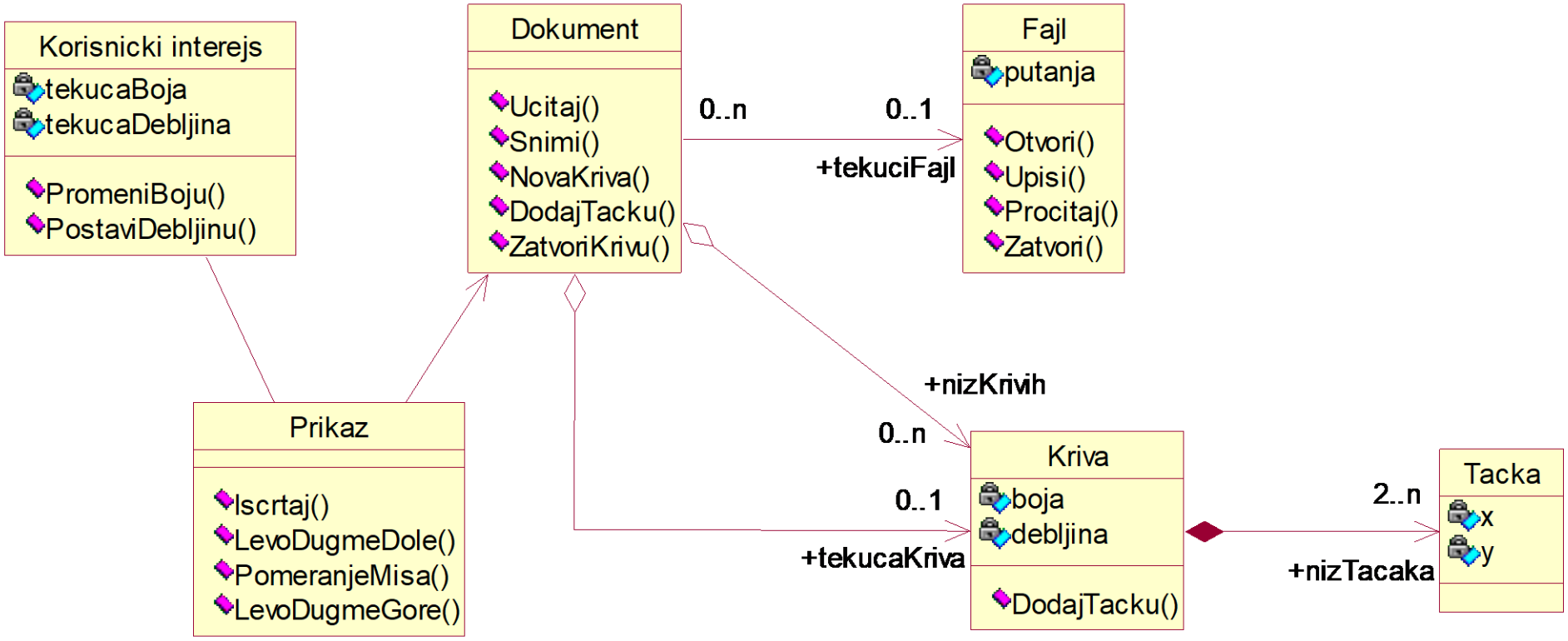
Tipovi UML dijagrama

- Dijagrami klasa (engl. *Class Diagram*)
- Dijagrami slučajeva korišćenja (engl. *Use-Case Diagram*)
- Dijagrami sekvence (engl. *Sequence Diagram*)
- Dijagrami saradnje (engl. *Collaboration Diagram*)
- Dijagrami stanja (engl. *Statechart Diagram*)
- Dijagrami aktivnosti (engl. *Activity Diagram*)
- Dijagrami komponenti (engl. *Component Diagram*)
- Dijagrami razmeštaja (engl. *Deployment Diagram*)

Dijagrami klasa

- Koriste se za predstavljanje klasa i njihove organizacije u pakete
- Dijagrami klasa se koriste za modelovanje
  - domena sistema
  - aplikacije
- Elementi dijagrama su:
  - klase
  - veze između klasa
    - nasleđivanje
    - asocijacija
    - agregacija
    - kompozicija
  - paketi
  - veze zavisnosti između paketa

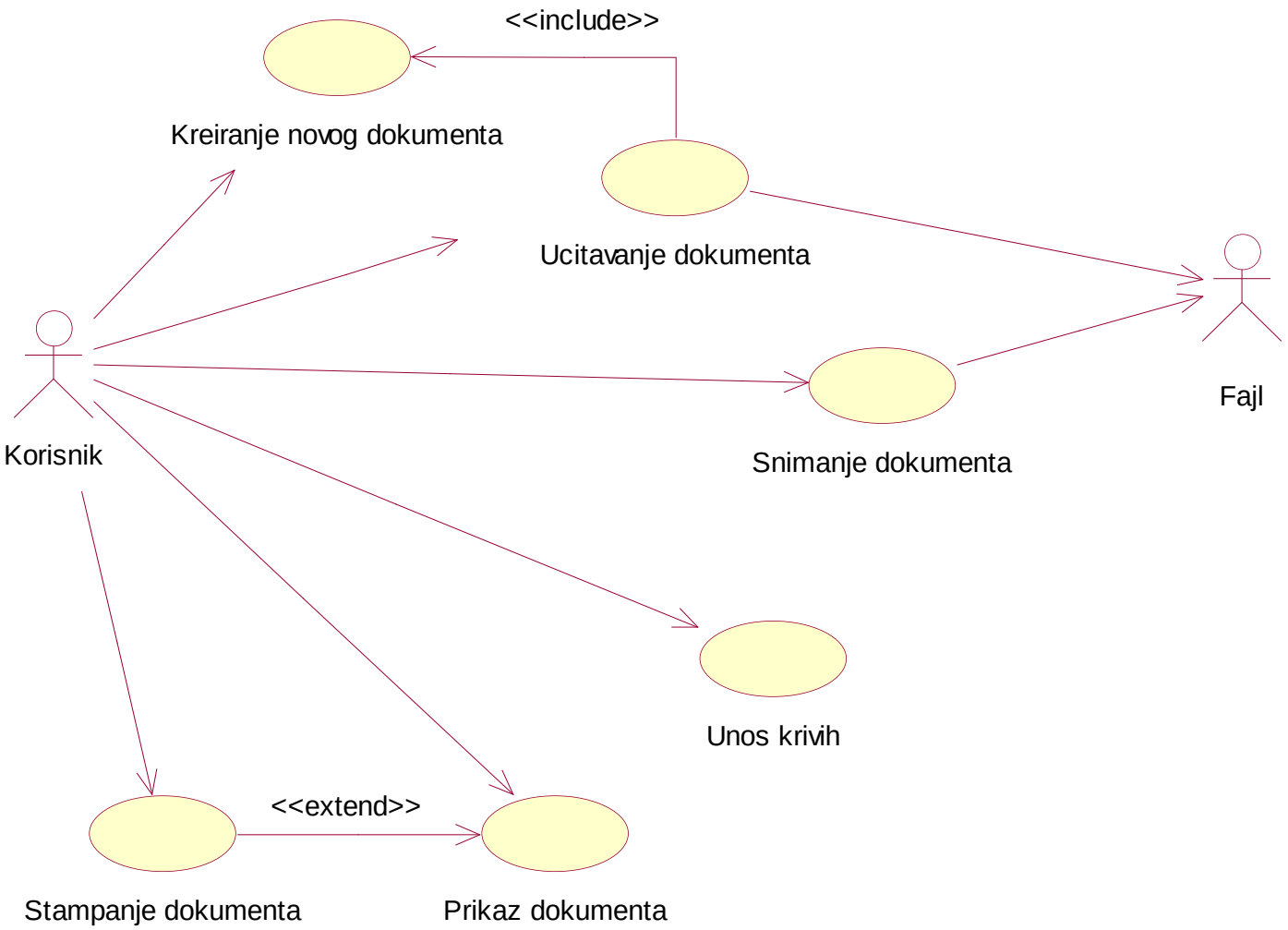
Dijagram klasa - primer



Dijagrami slučajeva korišćenja

- Koriste se u procesu prikupljanja i dokumentovanja korisničkih zahteva
- Elementi dijagrama su:
  - akteri
    - korisnici sistema
    - drugi sistemi iz okruženja
  - slučajevi korišćenja sistema
  - veze između aktera i slučajeva korišćenja
    - asocijacija
    - generalizacija
  - paketi
  - veze zavisnosti između paketa

Dijagram slučajeva korišćenja - primer

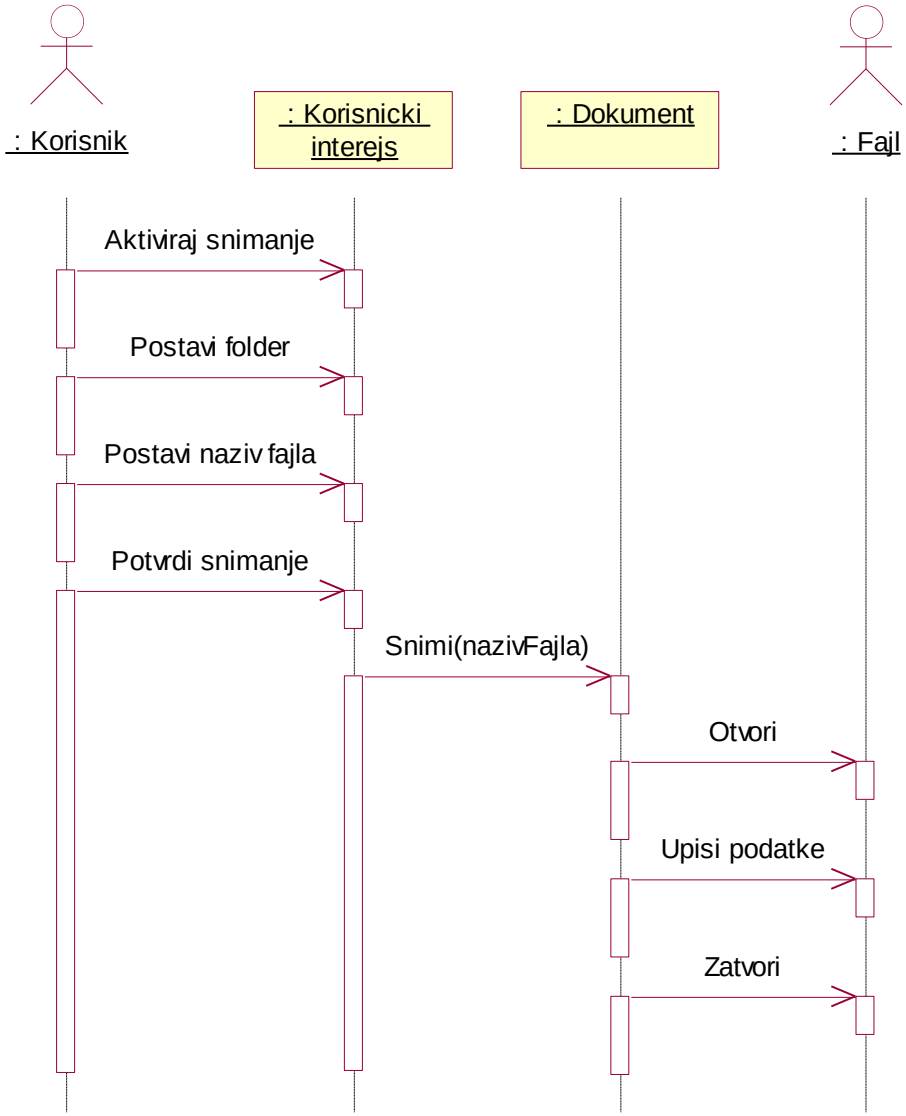


Dijagrami sekvence

- Koriste se za predstavljanje scenarija interakcije između objekata u sistemu
  - Najčešće se ovi scenariji odnose na slučajeve korišćenja sistema

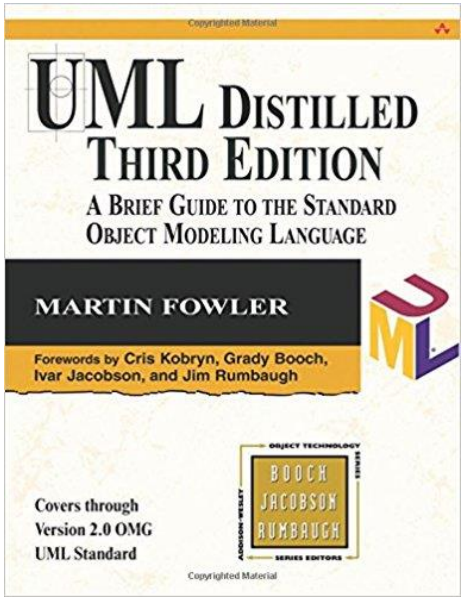
- Elementi dijagrama su:
  - objekti
  - vremenska linija
  - poruke između objekata

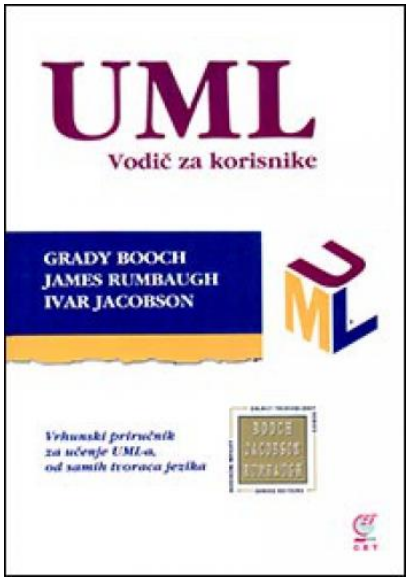
Dijagram sekvence - primer



Literatura za UML

- Martin Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd edition, Addison Wesley, 2003.
- Grady Booch, James Rumbaugh, Ivar Jacobson, UML - Vodič za korisnike, CET, 2001.





Zadaci za vežbanje

- Nacrtati UML dijagrame klasa i dijagrame slučajeva korišćenja za klase ranije implementirane u okviru paketa `zaposleni`, `matematika` i `institucija`. Identifikovati tipične slučajeve korišćenja za aplikacije koje koriste prethodne pakete.

OBJEKTNO-ORIJENTISANO PROJEKTOVANJE SOFTVERA

Projektovanje OO programa

"*Postoje dva načina za projektovanje i razvoj programa. Jedan je da ih učinite toliko jednostavnim da je očigledno da nemaju nedostatke. Drugi je da ih učinite toliko komplikovanim da nemaju* ***očiglednih*** *nedostataka.*"

C.A.R. Hoare

- Ključ za **uspešno OO projektovanje programa** je **dobro osmišljena apstrakcija problema** kroz objektne koncepte:

  "*Suština apstrakcije je čuvanje informacija koje su relevantne u datom kontekstu i ignorisanje informacija koje su irelevatne za dati konkteks*"

  John Guttag

Ulazni podaci za OO projektovanje

- **Konceptualni model** je rezultat objektno-orijentisane analize i opisuje koncepte u problemskom domenu. Eksplicitno se kreira tako da bude nezavisan od implementacionih detalja, kao što su konkurentnost ili čuvanje podataka
- **Slučajevi korišćenja** (use cases) su opisi sekvence događaja koji zajedno dovode do toga da sistem realizuje neku korisnu aktivnost
- **Dijagrami sekvence** grafički prikazuju, za određeni scenario u okviru nekog slučaja korišćenja, događaje koje generišu eksterni akteri, njihov redosled i moguće događaje unutar sistema

Ulazni podaci za OO projektovanje

- **Dokumentacija za korisnički interfejs** (engl. user interface – UI) prikazuje i opisuje izgled i tok rada sa programom putem korisničkog interfejsa finalnog softverskog proizvoda
- **Relacioni model podataka** – model podataka je apstraktni model koji opisuje kako se podaci predstavljaju i koriste. Ako se ne koristi objektna baza podataka, tada je relacioni model podataka neophodno unapred kreirati, pošto je izabrana strategija za objektno-relaciono mapiranje jedan od izlaza procesa objektno-orijentisanog projektovanja

Projektovanje OO programa

- **Definisati objekte** (kreirati dijagrame klase na osnovu konceptualnih dijagrama). Tom prilikom se entiteti tipično mapiraju na klase
- **Definisati elemente klasa** (atributi i metode)
- **Definisati broj objekata**, **trenutak** njihovog **nastajanja** i **nestajanja**, kao i **način medjusobne interakcije** tokom vremena
- **Definisati odgovornosti** svakog dela sistema

Projektovanje OO programa

- **Upotrebiti projektne obrazce** (ako su primenljivi) - glavna prednost primene projektnih obrazaca je mogućnost njihovog ponovnog korišćenja u više aplikacija. Objektno-orijentisani projektni obrazci obično prikazuju odnose i interakcije između klasa i objekata, bez specifikacije konačnih aplikacionih klasa i objekata
- **Izabrati radno okruženje** (ako je primenljivo) - radna okruženja uključuju veliki broj bibilioteka i klasa koje se mogu iskoristi za implementaciju standarnih struktura u aplikaciji. Na ovaj način se može dosta uštedeti na vremenu razvoja softvera pošto se izbegava ponovno pisanje velikog dela koda prilikom razvoja novih aplikacija
- **Identifikovati perzistentne objekte/podatke** (ako je primenljivo) – potrebno je identifikovati objekte koji treba da postoje duže od trajanja jednog izvršenja aplikacija. Ako aplikacija koristi relacionu bazu podataka, projektovati objektno-relaciono mapiranje

Projektni obrasci

- **Projektni obrasci** su **opšta, ponovo upotrebljiva rešenja za probleme koji se često javljaju** u određenom kontekstu projektovanja softvera
- Oni **nisu kompletan projekat koji se može direktno transformisati u izvorni ili mašinski kod**, već su opis ili šablon za rešavanje problema koji može da se koristi u mnogo različitih situacija
- Projektni obrasci su **formalizovani postupci najbolje prakse** (engl. best practices) koje programeri mogu koristiti kako bi efikasno rešili tipične probleme koji se javljaju prilikom projektovanja aplikacije ili sistema

Projektni obrasci

- Projektni obrasci su originalno (prema GoF) podeljeni na: **stvaralačke** (engl. creational), **strukturalne** (engl. structural) i **bihevijoralne** (engl. behavioral), a danas se koriste i **konkurentni** (npr. blockchain) i **arhitekturalni** (npr. Model-View-Controller - MVC)
- **Stvaralački**: Singleton (osigurava da klasa ima samo jednu instancu za koju postoji globalni pristup), Builder, Factory...
- **Strukturalni**: Adapter, Facade, Decorator...
- **Bihevijoralni**: Iterator, Interpreter, Visitor...

Primer 5.1 – obrazac Singleton u Javi

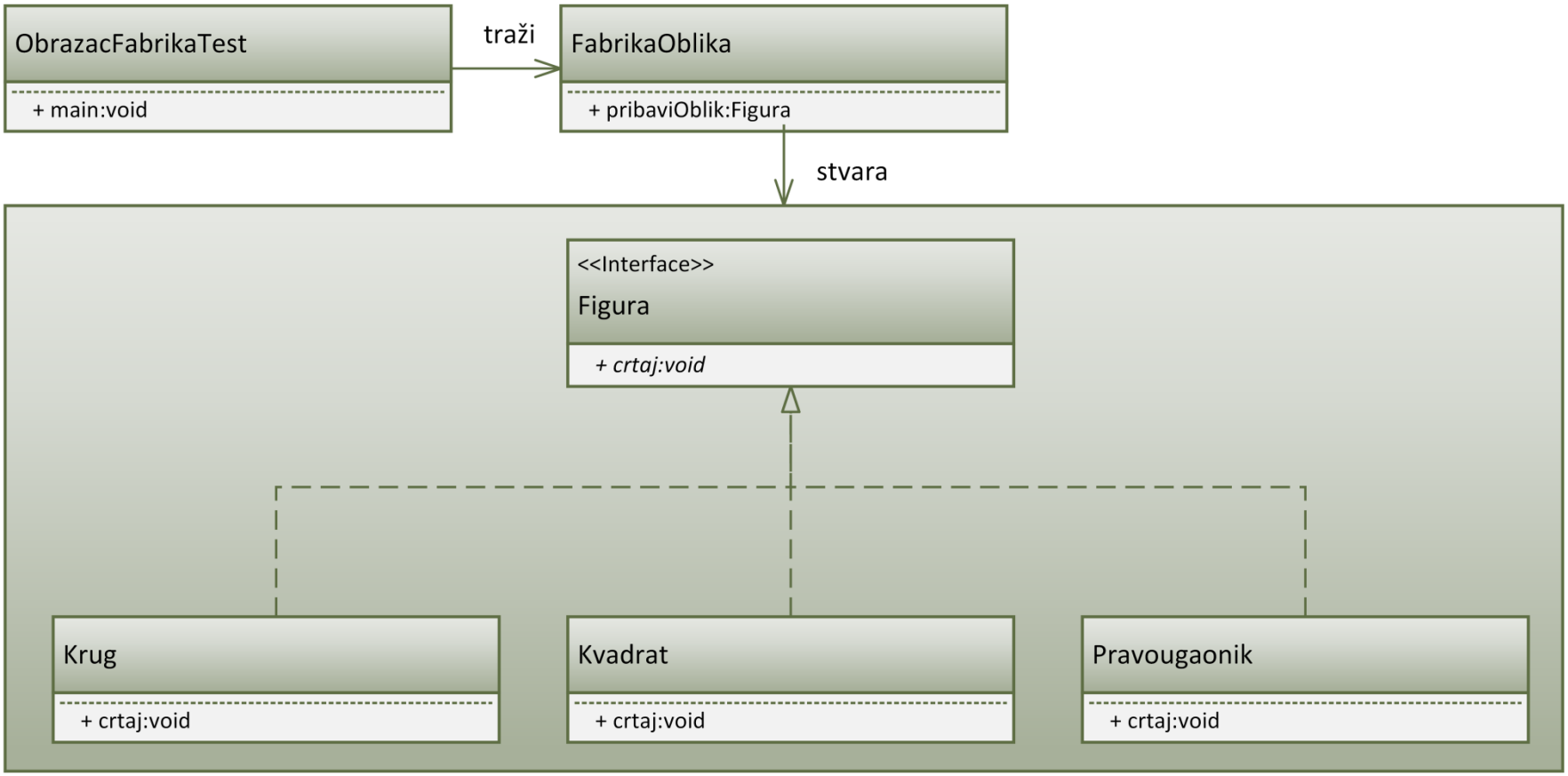
```
public class Singleton {

	private static Singleton instanca = null;

	protected Singleton() {
		// Postoji samo kako bi sprecili instanciranje
	}

	public static Singleton pribaviInstancu() {
		if(instanca == null) {
			instanca = new Singleton();
		}
		return instanca;
	}
}
```

Tipične primene: čuvanje podešavanja aplikacije (konfiguracioni fajlovi), logovanje podataka, itd.

Primer 5.2 – obrazac Factory u Javi

- Sa Factory obrascem, kreiramo objekat bez potrebe da izložimo logiku kreiranja objekta (engl. *creation logic*) klijentu i potom se obraćamo novostvorenom objektu korišćenjem zajedničkog interfejsa
- Kreiraćemo interfejs `Figura` i konkretne klase koje implementiraju ovaj interfejs. Potom ćemo u narednom koraku definisati klasu „fabriku“ `FabrikaOblika`
- Napravićemo i test klasu `ObrazacFabrikaTest` koja će koristiti klasu `FabrikaOblika` kako bi pribavila odgovarajući objekat nekog oblika. Test klasa će samo prosleđivati informaciju fabrici oblika da li je u pitanju krug, kvadrat ili pravouganik, a klasa `FabrikaOblika` će potom „isporučivati“ traženi oblik test klasi

Primer 5.2 – obrazac Factory u Javi



Primer 5.2 – interfejs Oblik, klase Krug, Pravougaonik i Kvadrat

```
public interface Figura {                                   //Figura.java
        void crtaj();
}

public class Pravougaonik implements Figura {               //Pravougaonik.java
        @Override public void crtaj() {
            System.out.println("Unutar Pravougaonik::crtaj() metode!");
        }
}

public class Kvadrat implements Figura {                    //Kvadrat.java
        @Override public void crtaj() {
            System.out.println("Unutar Kvadrat::crtaj() metode!");
        }
}

public class Krug implements Figura {                       //Krug.java
        @Override public void crtaj() {
            System.out.println("Unutar Krug::crtaj() metode!");
        }
}
```

Primer 5.2 – klasa FabrikaOblika

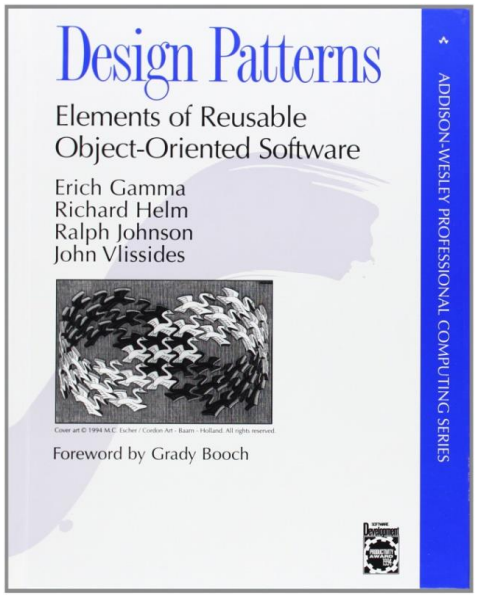
```
public class FabrikaOblika {
        //metod pribaviOblik dobavlja oblik zeljenog tipa
        public Figura pribaviOblik(String tipOblika){
                if (tipOblika == null){
                        return null;
                }
                if (tipOblika.equalsIgnoreCase("KRUG")){
                        return new Krug();
                } else if (tipOblika.equalsIgnoreCase("PRAVOUGAONIK")){
                        return new Pravouganik();
                } else if (tipOblika.equalsIgnoreCase("KVADRAT")){
                        return new Kvadrat();
                }
                return null;
    }
}
```

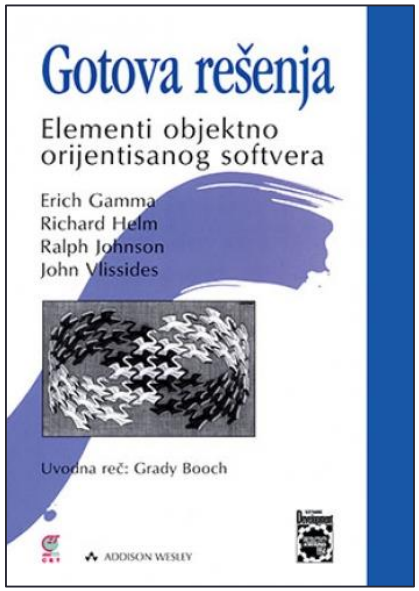
Primer 5.2 – klasa ObrazacFabrikaTest

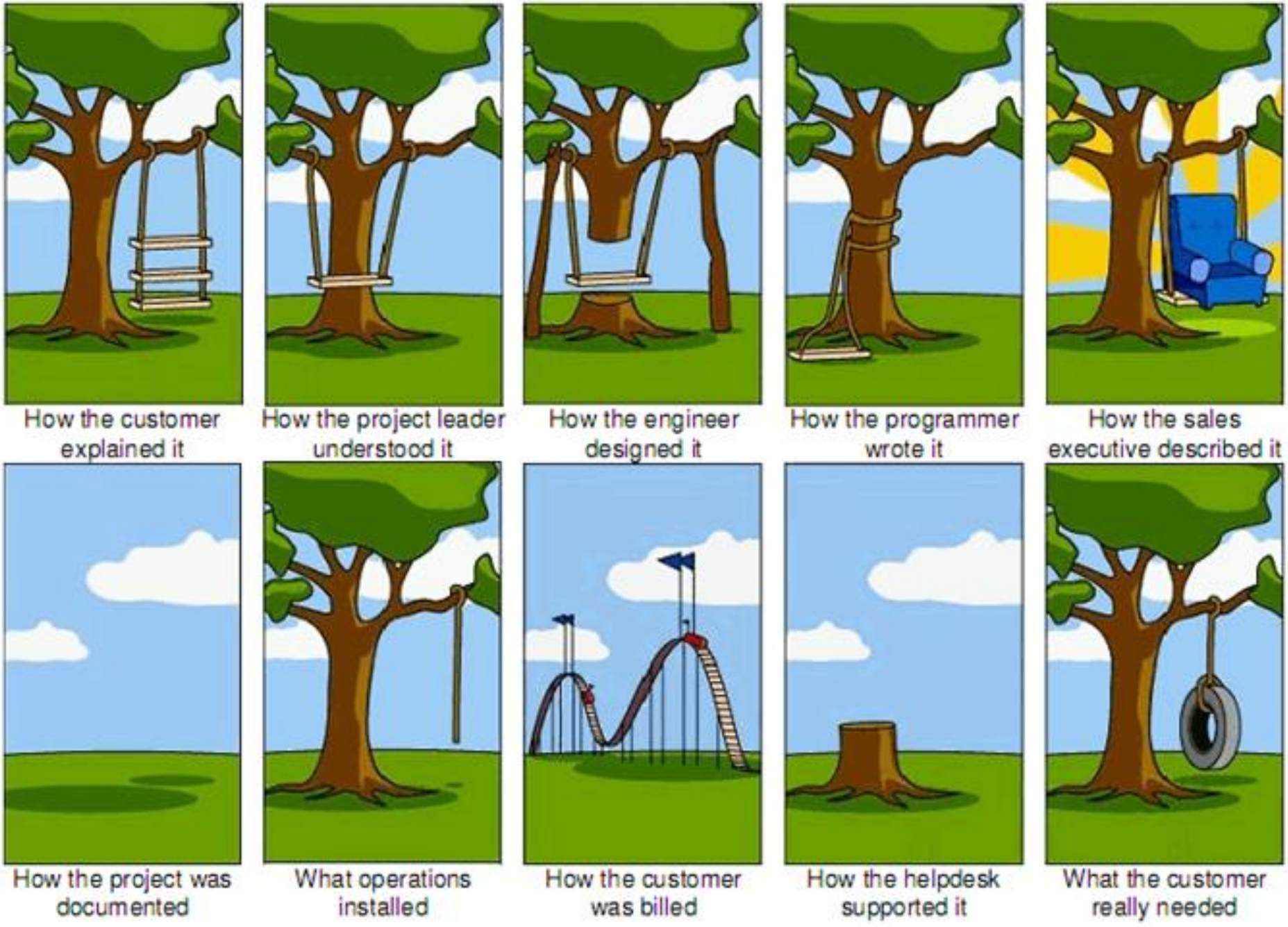
```
public class ObrazacFabrikaTest {

        public static void main(String[] args) {
                FabrikaOblika fabrikaOblika = new FabrikaOblika();

                // pribavi oblik Krug i pozovi njegov metod crtaj
                Figura oblik1 = fabrikaOblika.pribaviOblik ("KRUG");
                // pozovi metod crtaj za krug
                oblik1.crtaj();

                // pribavi oblik Pravougaonik i pozovi njegov metod crtaj
                Figura oblik2 = fabrikaOblika.pribaviOblik ("PRAVOUGAONIK");
                // pozovi metod crtaj za pravougaonik
                oblik2.crtaj();

                // pribavi oblik Kvadrat i pozovi njegov metod crtaj
                Figura oblik3 = fabrikaOblika.pribaviOblik ("KVADRAT");
                // pozovi metod crtaj za kvadrat
                oblik3.crtaj();
        }
}
```

Projektni obrasci

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley 1994. – poznata kao “Gang of Four” – “GoF”
- Srpsko izdanje: Gotova rešenja, CET Beograd, 2002.






How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

Zadatak za rad na času

- Korišćenjem projektnog obrasca Factory napraviti “fabriku vozila”.
- Treba kreirati interfejs `Proizvod` i konkretne klase koje implementiraju ovaj interfejs. Potrebno je i definisati klasu „fabriku“ - `FabrikaVozila`
- Treba kreirati i test klasu `ObrazacFabrikaTest` koja će koristiti klasu `FabrikaVozila` kako bi pribavila odgovarajući objekat nekog tipa vozila. Test klasa će samo prosleđivati informaciju fabrici vozila da li je u pitanju automobil, kamion ili motocikl, a klasa `FabrikaVozila` će potom „isporučivati“ traženo vozilo test klasi

Zadaci za vežbanje

- Proučiti ostale najvažnije projektne obrasce (pored `Singleton` i `Factory`, to su npr. `Builder`, `Adapter`, `Facade`, `Iterator`, `Visitor`, MVC) i modifikovati klase u paketu `zaposleni` tako da koriste projektne obrasce u situacijama gde je to adekvatno.
- Modifikovati preostale klase razvijene tokom kursa iz OOP tako da se primene projektni obrasci u situacijama gde je to adekvatno.