

PROGRAMSKO INŽENJERSTVO

UML dijagram razreda

doc. dr. sc. Aleksander Radovan

prof. struč. stud., dipl. ing.

Sadržaj

DIO 1

Arhitektura softvera

DIO 2

C4 Model

DIO 3

UML Class Diagram

DIO 4

Odnosi među razredima

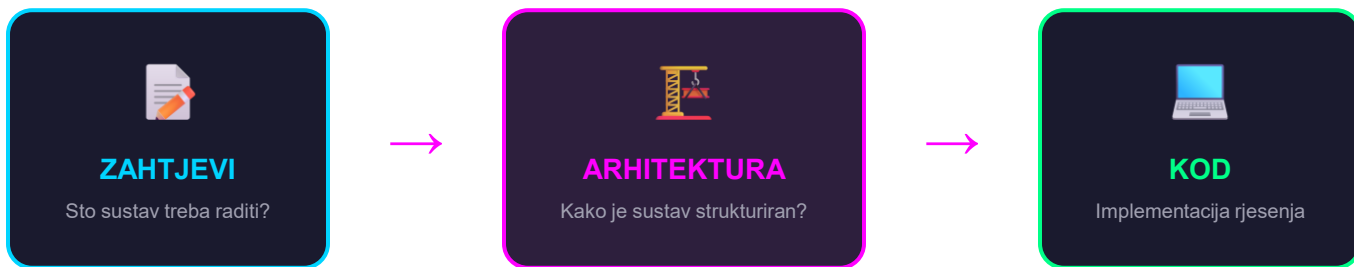
DIO 5

Sučelja i nasljeđivanje

DIO 6

SOLID principi

Uvod



Problem:

Veliki raskorak između problema i rješenja - arhitektura premošćuje taj jaz!

Oblikovanje arhitekture programske potpore

- Proces identificiranja i strukturiranja podsustava koji čine cjelinu te okruženja za upravljanje i komunikaciju između podsustava
 - rezultat procesa oblikovanja je opis/dokumentacija *arhitekture programske potpore*.

Prednosti definiranja arhitekture

Smanjuje troškove

Cijena razvoja i održavanja pada kad je struktura jasna od početka

Omogućuje re-use

Dobro strukturirane komponente možeš koristiti u drugim projektima

Razjašnjava zahtjeve

Prisiljava tim da jasno definira što sustav treba raditi

Poboljšava komunikaciju

Dijagrami su bolji od tisuće riječi za nove članove tima

Rano otkrivanje grešaka

Greška u arhitekturi je skupa – bitno rano otkrivanje

Olakšava skaliranje

TikTok ima milijardu korisnika - to je rezultat dobre arhitekture

C4 model arhitekture

C4 model je način vizualizacije softverske arhitekture na 4 razine apstrakcije - kao Google Maps za kod!

Level 1: Context - Tko koristi sustav?

Level 2: Container - Koji su glavni dijelovi?

Level 3: Component - Što je unutar dijelova?

Level 4: Code - Kako izgleda kod (UML)?

Google Maps vam pokazuje Hrvatsku, Zagreb, centar, pa ulicu - isto kao C4!

Level 1: Context



Level 2: Container



Level 3: Component



Level 4: Code (UML)

C4 model arhitekture

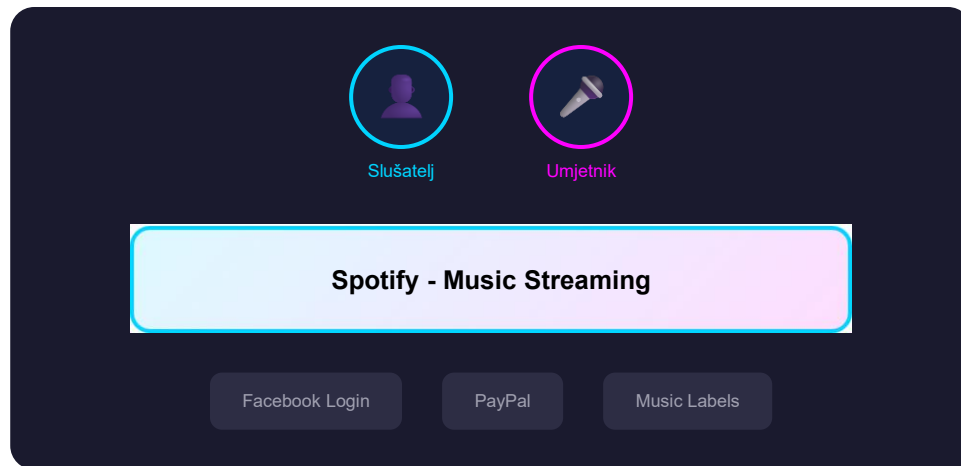
Pokazuje tko koristi sustav i s kojim vanjskim sustavima komunicira.

Primjer: Spotify

Korisnici: slusatelji, umjetnici

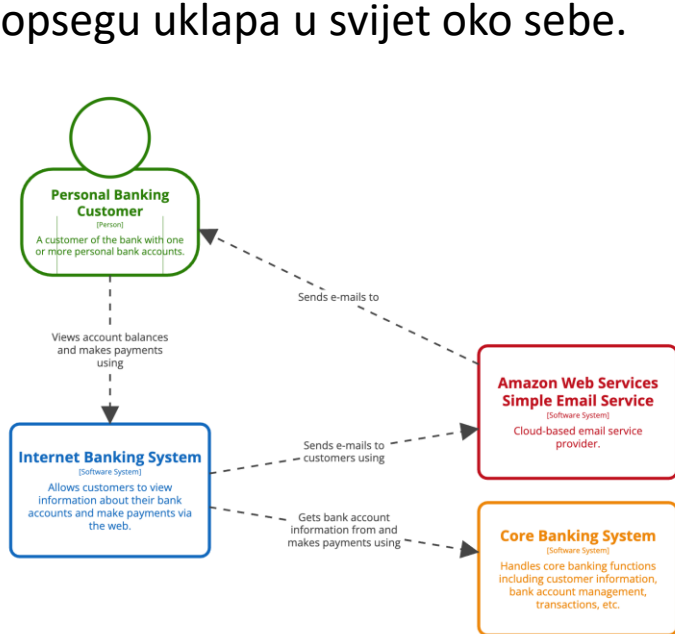
Vanjski sustavi: Facebook, PayPal, glazbene kuce

Pitanje: Tko koristi ovaj sustav i sa čime komunicira?



C4 model architecture – Level 1- Context

Dijagram konteksta sustava pruža početnu točku, pokazujući kako se softverski sustav u opsegu uklapa u svijet oko sebe.

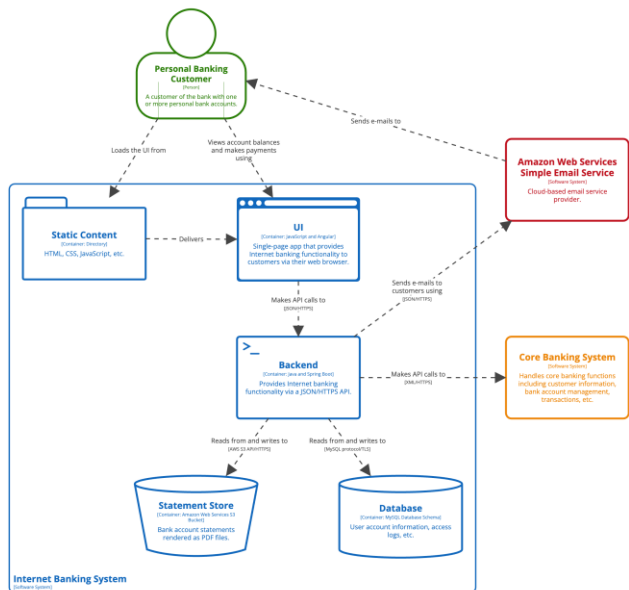


System Context View: Internet Banking System

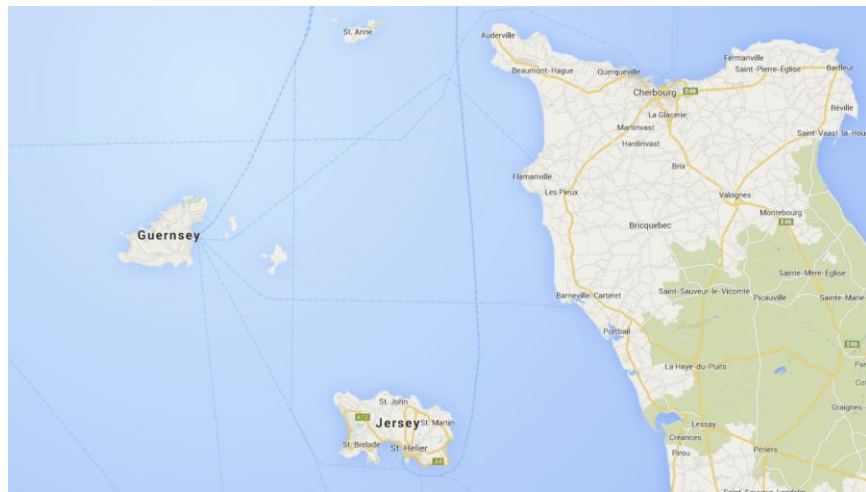
The system context diagram for a fictional Internet Banking System | Simon Brown | c4model.com | License: CC BY 4.0

C4 model architecture – Level 2 - Container

Dijagram kontejnera zumira softverski sustav u opsegu, prikazujući aplikacije i pohrane podataka unutar njega.

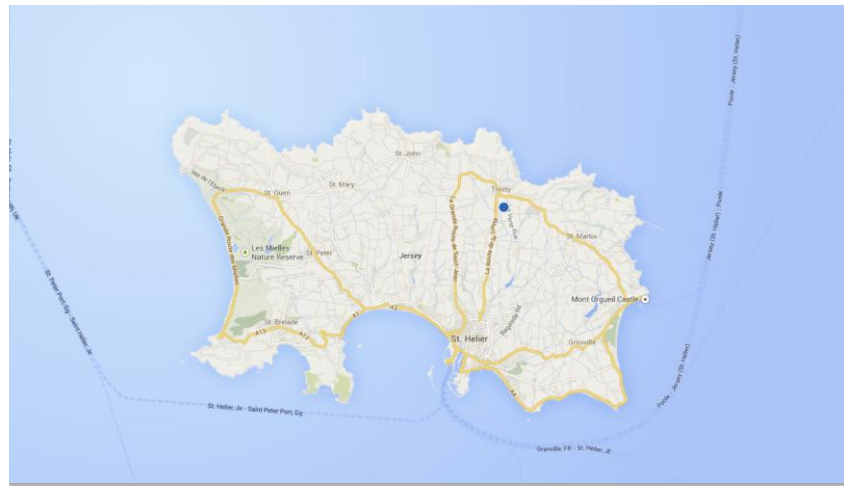
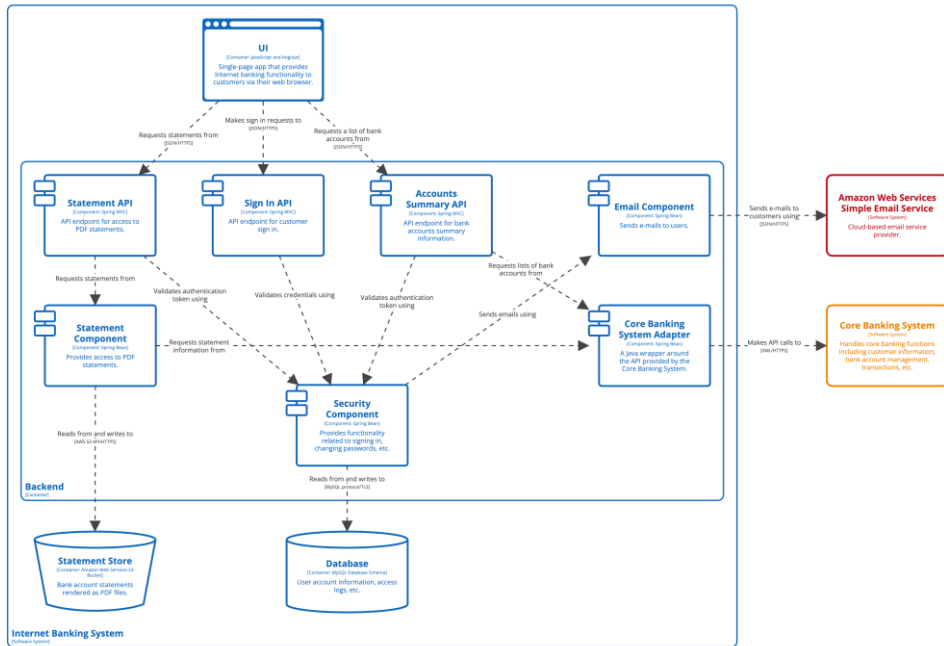


Container View: Internet Banking System
The container diagram for the Internet Banking System | Simon Brown | c4model.com | License: CC BY 4.0



C4 model architecture – Level 3 - Component

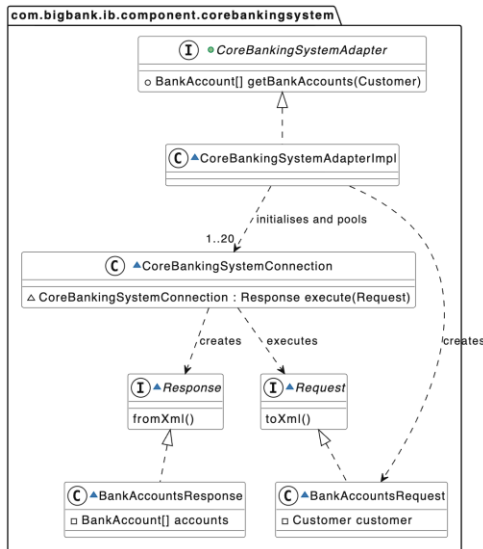
Dijagram komponenti zumira pojedinačni kontejner, prikazujući komponente unutar njega.



Component View: Internet Banking System - Backend

C4 model architecture – Level 4 - Code

Dijagram koda (npr. UML *class diagram*) može se koristiti za zumiranje pojedinačne komponente, prikazujući kako je ta komponenta implementirana na razini koda.



Code View: Internet Banking System - Backend - Core Banking System Adapter

A summary of the implementation details for the Core Banking System Adapter component | Simon Brown | c4model.com | License: CC BY 4.0

Uloga arhitekta

- Dobar arhitekt:
 - razumije potrebe poslovnog modela i zahtjeve projekta.
 - svjestan različitih tehničkih pristupa u rješavanju danog problema.
 - vrednuje dobre i loše strane tih pristupa.
 - preslikava potrebe i vrednovane zahtjeve u tehnički opis arhitekture programske potpore.
 - vodi razvojni tim u oblikovanju i implementaciji.
 - koristiti “meke” vještine kao i tehničke vještine.
- Pogled arhitekta na programsku potporu:
 - struktura kao skup implementacijskih zahtjeva
 - struktura i odnosi elemenata tijekom dinamičke interakcije
 - odnosi programskih struktura i okoline

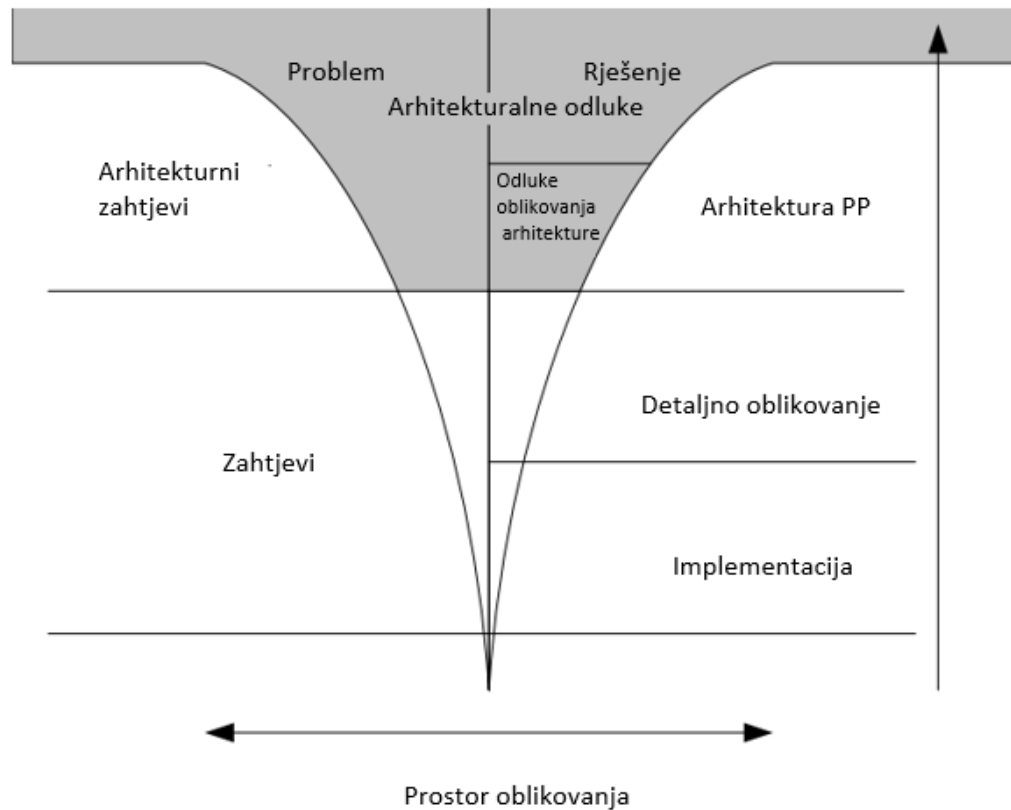
Proces izbora i vrednovanja

- Proces izbora i vrednovanja arhitekture \equiv proces donošenja odluka
- Alternativni stilovi arhitekture programske potpore
 - \Rightarrow **Oblikovanje kao niz odluka**
- Dizajner se sučeljava s rješavanjem niza problema (*engl. design issues*)
 - podproblemi ukupnog problema
- Više inačica rješenja
 - *engl. design options*
- Dizajner donosi odluke (*engl. design decision*) za rješavanje problema odabir najbolje opcije između više mogućih rješenja problema

Donošenje odluka

- Za donošenje odluka potrebna znanja:
 - zahtjeva
 - trenutno oblikovana arhitektura
 - raspoloživa tehnologija
 - principi oblikovanja i najbolja praksa (engl. *best practices*)
 - dobra rješenja iz prošlosti
- Zadaće donošenja odluka
 - postavljanje prioriteta sustava
 - dekompozicija sustava
 - definiranje svojstava sustava
 - postavljanje sustava u kontekst
 - cjelovitost sustava
- Tehnička i netehnička pitanja su isprepletena!

Prostor oblikovanja



Arhitekturno značajni zahtjevi

- engl. *Architecturally Significant Requirement ASR, Architecture Drivers*
- Neki zahtjevi imaju daleko dublji utjecaj na arhitekturu, a obično se definiraju kao arhitekturno značajni zahtjevi
 - izvedeni su iz funkcionalnih i nefunkcionalnih zahtjeva
 - ocijenjeni od dionika obzirom na prioritete i doseg
 - uobičajena tri stupnja ocjene
- **Stil arhitekture određuju nefunkcijski zahtjevi**, a funkcijski zahtjevi određuju instance elemenata definirane tim stilom

Svojstva oblikovanja

- Oblikovanje arhitekture
 - podjela u podsustave i komponente
 - Način povezivanja?
 - Način međudjelovanja?
 - Sučelja?
- Oblikovanje klasa
- Oblikovanje korisničkog sučelja
- Oblikovanje algoritma
 - za izračunavanje, upravljanje...
- Oblikovanje protokola
 - komunikacijski protokoli

Principi dobrog oblikovanja

- Cilj:
 - smanjenje cijene i povećanje profita
 - osiguranje sukladnosti sa zahtjevima
 - ubrzanje razvoja i implementacije
 - poboljšanje kvalitete
 - Uporabljivosti
 - Efikasnosti
 - Pouzdanosti
 - Lakoće održavanja
 - Ponovne uporabe

Razvoj modela arhitekture

- Započinje s grubom skicom arhitekture
 - zasnovanoj na osnovnim zahtjevima i obrascima uporabe.
- Određuje temeljne potrebne komponente sustava
- Izabire između raznih stilova arhitekture
 - savjet: nekoliko timova nezavisno radi grubu skicu arhitekture, a potom se spoje najbolje ideje
- Arhitektura se dopunjava detaljima tako da se:
 - identificiraju osnovni načini komunikacije i interakcije između komponenata
 - odredi kako će dijelovi podataka i funkcionalnosti raspodijeliti između komponenata
 - pokuša identificirati dijelove za ponovnu uporabu
 - vrati se na pojedini obrazac uporabe i podesi arhitekturu

Razvoj modela arhitekture

- Za izradu modela potreban je jezik i sintaksna pravila

→ UML:

dijagrami razreda

dijagrami komponenti

dijagram razmještaja

UML Class diagram

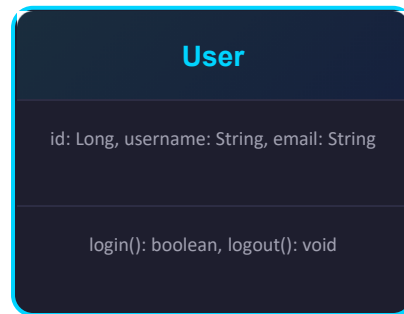
Što je UML Class Diagram?

Strukturni statički dijagram koji prikazuje klase (razrede), njihove atribute, operacije i odnose među njima.

Prikazuje:

Razrede i svojstva, atribute i operacije, odnose poput asocijacije i nasljeđivanja, sučelja i enumeracije

Alati: Astah, Visual Paradigm, draw.io, PlantUML, Mermaid



Primjer UML klase

Razred (klasa)

Što je razred?

Predložak za stvaranje objekata koji obuhvaća atribute (obilježja) i operacije (ponašanje).

Instagram primjer:

Razred Post ima atribute (slika, opis, broj lajkova) i metode (like, share, delete)

```
public class Post {  
    private String imageUrl;  
    private int likes;  
    public void like() { likes++; }  
}
```

Razred vs. Objekt



RAZRED

Nacrt / Predlozak



OBJEKT

Instanca razreda

Kao recept vs. gotovo jelo

Atributi

- Svojstva (engl. *property*):
 - Vidljivost (engl. *visibility*)
 - Naziv (engl. *name*)
 - Brojnost (engl. *multiplicity*)
 - Vrsta (engl. *type*)
 - Početna vrijednost (engl. *initial value*)
 - Ostala svojstva: promjenjivost, modifikatori

[visibility]name[[multiplicity]][:type][=initial value][{property}]

npr. – `MyAttribute : string = „Hello“;`

Atributi - vidljivost

- Stupanj vidljivosti atributa
 - **Public** (simbol: +)
 - Dostupan **svim razredima i paketima**
 - **Private** (simbol: -)
 - Dostupan samo unutar istog razreda
 - **Protected** (simbol: #)
 - Dostupan unutar istog i svih izvedenih razreda (koji nasljeđuju glavni razred)
 - **Package** (simbol: ~)
 - Dostupan svim razredima unutar paketa

Atributi - promjenjivost

- **changeable**

- Vrijednost atributa može se nesmetano mijenjati. Podrazumijevana (*default*) postavka – ne mora se posebno naznačiti.

- Rjeđe korišteni modifikatori*:

- **addOnly** - vrijednost atributa može se samo povećavati.
- **frozen** – vrijednost atributa može se promijeniti samo jednom tijekom života objekta (u praksi je to kod inicijalizacije)
- **read-only** – vrijednost atributa ne može se mijenjati izvan objekta kojemu pripada.

Atributi – modifikator static

- **static**

- vrijednost atributa ne ovisi o životu objekta - definirana je na razini razreda.
- atributu se pristupa preko **naziva razreda** (a ne instance!)
- vrijednost može biti promjenjiva ili konstantna, ovisno o tome koja mu je promjenjivost dodijeljena
 - *final* (Java) / *const* (C, C++, C#) – vrijednost atributa je konstantna*

Odgovornost, operacija, metoda

- **Odgovornost** (eng. *responsibility*) je nešto što sustav mora izvršiti.
 - Svaki funkcionalni zahtjev mora se pridijeliti nekom razredu (proizlaze iz obrazaca uporabe).
 - Realizira se jednom ili više operacija.
- **Operacije** ostvaruju odgovornosti pojedinog razreda i implementiraju se **metodama**:
 - vidljivost: *public, package, protected, private*
 - modifikatori: *static, abstract*
 - istodobnost: *sequential, guarded, concurrent*
 - parametri (argumenti)
 - povratna vrijednost

Terminologija

- Member
 - Member variable = atribut razreda
 - Isto što i member data, member field, attribute
 - **Instance** member variable
 - vrijednost svojstvena objektu razreda
 - **Static** member variable
 - vrijednost zadana na razini razreda
 - Member function = operacija razreda
 - Isto što i operation, method
 - **Instance** member function
 - **Static** member function (class member function)

Odnosi među razredima (klasama)

Asocijacija

"koristi" / "poznaje"

User koristi Post

Agregacija (dijamant prazan)

"ima" (slaba veza)

Playlist ima Song

Kompozicija (dijamant pun)

"sadrzi" (jaka veza)

Order sadrzi OrderItem

Nasljeđivanje (trokut)

"je vrsta" (IS-A)

Admin je vrsta User

Realizacija (isprekidano)

"implementira" sučelje

UserService impl IUserService

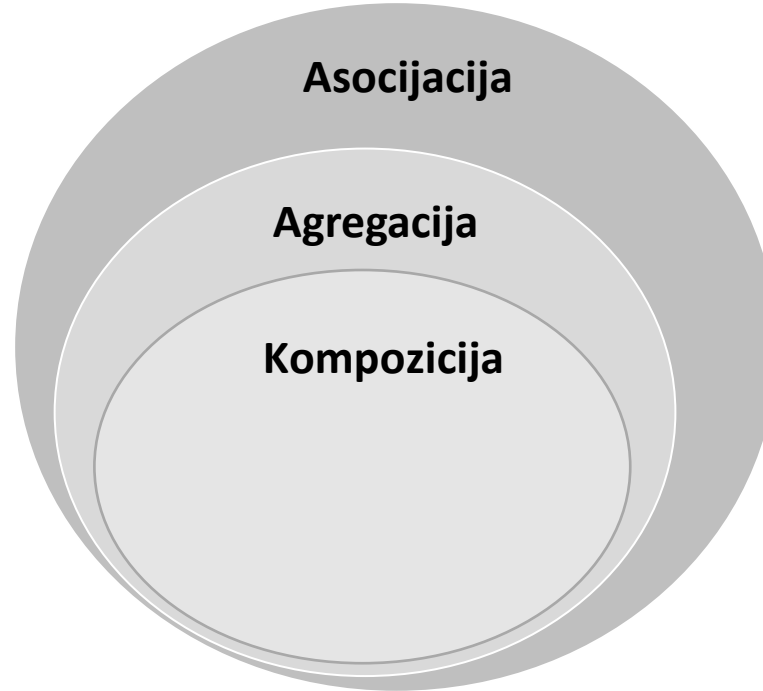
Ovisnost (isprekidano)

"ovisi o" (najslabija)

Controller ovisi o DTO

Odnosi među relacijama

- Asocijacija - Agregacija - Kompozicija



Asocijacija

Što je asocijacija?

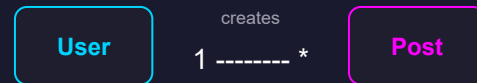
Veza između dva razreda gdje jedan "poznaje" ili "koristi" drugi. Objekti imaju vlastiti životni ciklus.

Viseštrukost (multiplicity):

1 = točno jedan, 0..1 = nula ili jedan, * = bilo koliko, 1..* = jedan ili više, n..m = između n i m

```
public class User {  
    private List<Post> posts; // 0..*  
}
```

Instagram primjer



Jedan user kreira 0 ili više postova

Jednosmjerna: samo User zna za Post. Dvosmjerna: oba razreda znaju jedan za drugog.

Nasljeđivanje

Sto je nasljeđivanje?

Koncept gdje podrazred preuzima atribute i metode nadrazreda. Odnos "JE-VRSTA" (IS-A).

Ključni koncepti:

extends - Java ključna riječ za nasljeđivanje. Polimorfizam - *override* metoda u podrazredu.
super - poziv metode nadrazreda. abstract - razred koji se ne može instancirati.

Java/C# ne podržavaju višestruko nasljeđivanje razreda (diamond problem)

TikTok primjer

Content (abstract)

extends

Video

Photo

Story

```
public class Video extends Content {  
    // nasljeđuje id, likes, createdAt...  
    private int duration;  
}
```


Sučelje

Što je sučelje?

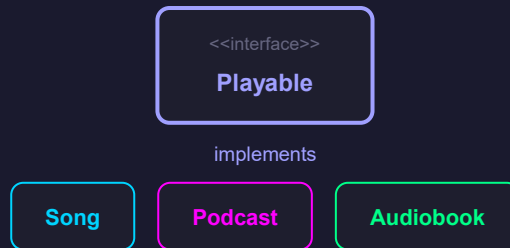
Ugovor koji definira metode koje razred mora implementirati. Nema atribute ni implementaciju!

Prednosti:

Višestruka implementacija - razred može implementirati više sučelja. Polimorfizam - isti kod radi s različitim impl. Loose coupling - manja ovisnost. Testabilnost - lakše mockanje.

```
public interface Playable {  
    void play();  
    void pause();  
    int getDuration();  
}
```

Spotify primjer - Polimorfizam



Player može reproducirati bilo što što implementira Playable - bez promjene koda!

SOLID principi

S - Single Responsibility

Razred ima samo jedan razlog za promjenu.

UserService - samo user logika

O - Open/Closed

Otvoren za proširenje, zatvoren za modifikaciju.

Dodaj novu impl. sučelja, ne mijenjaj staro

L - Liskov Substitution

Podrazred mora moci zamijeniti nadrazred.

Square ne smije extendati Rectangle

I - Interface Segregation

Vise manjih sučelja je bolje od jednog velikog.

Playable, Downloadable, Shareable

D - Dependency Inversion

Ovisi o apstrakcijama (sučeljima), ne o konkretnim razredima. High-level moduli ne ovise o low-level modulima.

Controller ovisi o IUserService, ne o UserServiceImpl

Loše prakse

God Class / Spaghetti

Jedan razred radi SVE - 5000+ linija koda, nemoguće za održavanje.

`AppManager.java` - 10000 linija

Tight Coupling

Razredi previše ovise jedni o drugima - promjena jednog lomi druge.

`new UserService(new MySQLDB())`

Copy-Paste Programming

Isti kod na 10 mjesta - bug fix = 10 promjena.

DRY princip: Don't Repeat Yourself

Wrong Inheritance

Nasljeđivanje samo radi code reuse, bez IS-A odnosa.

`Stack extends ArrayList` (krivo!)

Public Everything

Svi atributi public - nema enkapsulacije, bilo tko može promijeniti stanje.

`public String password;` (opasno!)

Rjesenje: Koristi SOLID principe, code review, refactoring

Usporedba dobre i loše prakse

LOSE - God Class

```
class AppManager {  
  // User stuff  
  createUser() {}  
  deleteUser() {}  
  // Video stuff  
  uploadVideo() {}  
  // Payment stuff  
  processPayment() {}  
  // ... 500+ metoda  
}
```

Problemi: Teško testirati, nemoguće za više developera, promjena u jednom dijelu može slomiti drugi

DOBRO - Single Responsibility

```
class UserService {  
  create() {}  
  delete() {}  
}  
class VideoService {  
  upload() {}  
}  
class PaymentService { ... }  
class EmailService { ... }
```

Prednosti: Svaki servis testiran zasebno, više developera paralelno, jasna odgovornost

Paketi

Što su paketi?

Nacin organiziranja razreda koji suraduju ili obavljaju sličnu funkcionalnost.

Ključne riječi:

`package` com.app.users; // Java

`namespace` App.Users; // C#

`import` com.app.users.User;

`using` App.Users;

Paketi omogućuju (~) package vidljivost - razredi unutar paketa vide jedni druge

TikTok struktura paketa

`com.tiktok`

`users/`

`User.java`

`UserService.java`

`UserRepository.java`

`videos/`

`Video.java`

`VideoService.java`

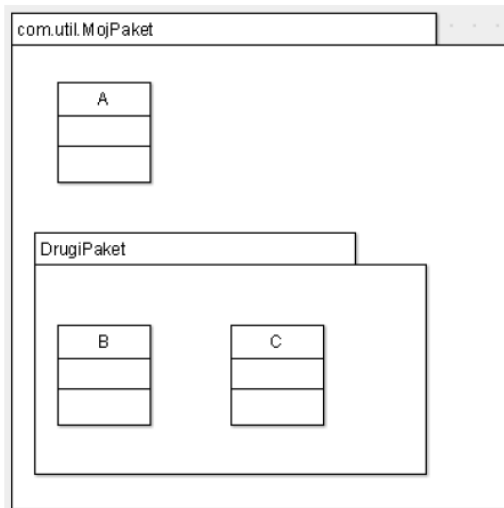
`comments/`

`notifications/`

Paketi

- Razredi koji međusobno surađuju/potrebni su za obavljanje određenog skupa funkcionalnosti organiziraju se u pakete.
 - Paketi mogu sadržavati druge razrede i pakete.
 - U programskom kodu za deklariranje naziva paketa koriste se ključne riječi namespace (C++ i C#), package (Java).
 - Za uvoz paketa koriste se ključne riječi using (C++ i C#) , import (Java).

Paketi - primjer



C++

```
namespace com {  
    namespace util {  
        namespace MojPaket {  
            namespace DrugiPaket {
```

```
                class B {};
```

```
            } /* End of namespace  
com.util.MojPaket::DrugiPaket */  
        } /* End of namespace com.util.MojPaket */  
    } /* End of namespace com.util */  
} /* End of namespace com */
```

Java

```
package com.util.MojPaket;
```

```
public class A {}
```

Java

```
package com.util.MojPaket.DrugiPaket;
```

```
public class B {}
```

Enumeracije

Sto je enumeracija?

Tip podatka koji sadrzi fiksni skup vrijednosti. Koristi se za diskretne, unaprijed poznate opcije.

Kada koristiti enum?

Status narudžbe: PENDING, PROCESSING, SHIPPED. Tip korisnika: FREE, PREMIUM, ADMIN. Dani u tjednu, mjeseci, smjerovi...

```
public enum VideoStatus {  
    DRAFT,  
    PUBLISHED,  
    REMOVED,  
    UNDER_REVIEW  
}
```

UML notacija

<<enumeration>>

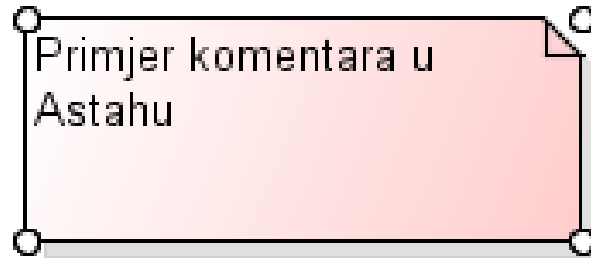
UserType

FREE
PREMIUM
CREATOR
ADMIN

Prednost pred String: kompajler javlja gresku ako upises krivu vrijednost!

Komentari

- Unatoč formalnoj ekspresivnosti UML dijagrama razreda ponekada su potrebni i komentari.
 - Koriste se za dodatni opis svrhe nekog razreda, atributa, veza, metoda i drugih elemenata dijagrama ukoliko je potrebno.
 - U komentarima je poželjno biti jasan i sažet, te obuhvatiti sve bitne aspekte UML elementa koji se opisuje, a koji nisu nedvosmisleno jasni iz samog dijagrama.





Alati za modeliranje

Visual Paradigm

Profesionalni CASE alat, podrška za sve UML dijagrame, code generation
visual-paradigm.com

draw.io / diagrams.net

Besplatan, web-based, integracija s Google Drive i GitHub
app.diagrams.net

PlantUML

Dijagrami iz teksta/koda! Verzioniranje u Gitu, CI/CD integracija
plantuml.com

Mermaid

Markdown dijagrami, GitHub/GitLab native podrška
mermaid.js.org

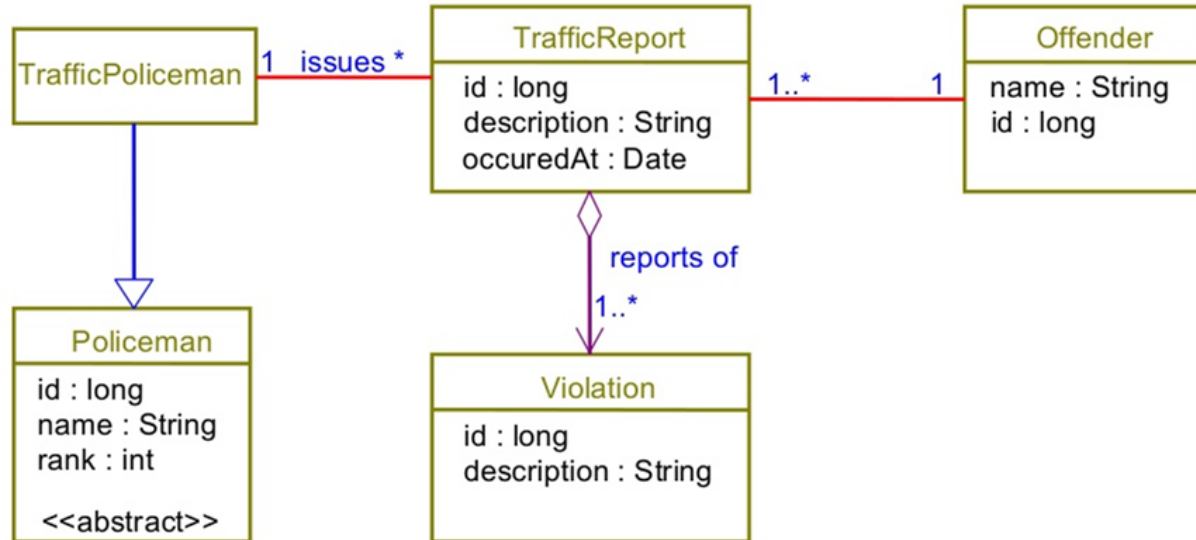
IntelliJ IDEA UML

Ugrađeno u IDE, generira dijagrame iz koda
Ultimate verzija

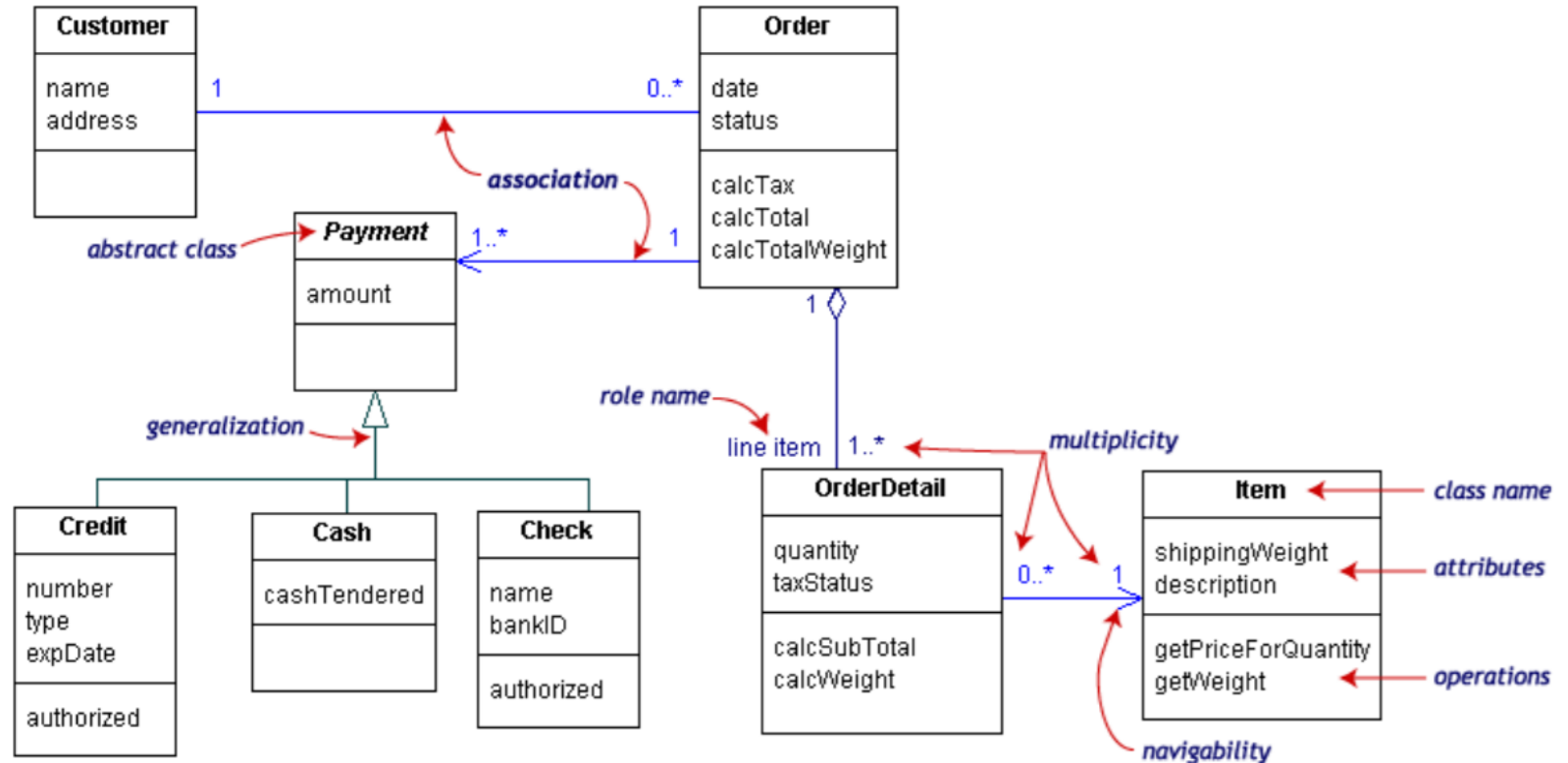
💡 Preporuka za studente: draw.io (besplatan) ili [Mermaid](https://mermaid.js.org) (u GitHubu)

Primjeri UML dijagrama

Traffic Violation Report System Example



Primjeri UML dijagrama





Sažetak

C4 Model

4 razine: Context → Container → Component → Code

UML Class Diagram

Razredi, atributi (+/-/#/~), operacije, odnosi

Odnosi

Asocijacija (—), Agregacija (◊), Kompozicija (◆), Nasljeđivanje (▷)

Sučelja

Ugovor za implementaciju, omogućuje polimorfizam

SOLID principi

S-R-P, O-C-P, L-S-P, I-S-P, D-I-P

Izbjegavati

God Class, Tight Coupling, Copy-Paste, Public Everything

 Cilj: Dobra arhitektura = manji troškovi, lakše održavanje, sretni developeri!



Pitanja?

Hvala na pažnji!

Dodatni resursi:

c4model.com, uml-diagrams.org, refactoring.guru

Martin Fowler: UML Distilled, Robert C. Martin: Clean Architecture