***MODELOVANJE***

Model je ***reprezentacija*** sistema koja omogućava ***istraživanje*** njegovih ***osobina*** i u nekim slučajevima ***predviđanje*** budućih ***ishoda***.

***Osnovni cilj***

Cilj modelovanja jeste da imamo nešto što možemo da ***koristimo umesto realnog sistema***, nešto što će ***zameniti*** realni sistem. Nešto što će nam omogućiti da ***utvrdimo*** neke ***činjenice*** vezane za model i ako je model dobar da na osnovu toga zaključimo nešto i o realnom sistemu. Ukoliko je model dobar moćićemo takve zaključke da donesemo pre nego što to učinimo na realnom sistemu. U mnogo slučajeva nam je to mnogo bolje jer je ***brže*** možda ***jeftinije***, ***bezbednije*** da proveravamo na modelu nego na realnom sistemu.

***Formalna definicija***

„Model predstavlja ***pojednostavljenje*** sistema načinjeno za ***određenu svrhu***. Model treba da pruži ***odgovore umesto realnog sistema*** i ti odgovori treba da budu ***isti kao*** da su prikupljeni ***na*** ***realnom sistemu*** uz uslov da su pitanja u okviru domena svrhe za koju je taj model kreiran.”

***Da bi*** model bio ***koristan***, model ***mora biti jednostvniji*** za korišćenje od stvarnog sistema. Da bi se to postiglo, mnogi detalji se apstrahuju. Ovo pojednostavljenje je suština modela.

Modeli se prave tako da ***uključe samo određene detalje*** samo određene aspekte sistema koji modelujemo.

Modeli se najčešće koriste :

* Da bi ljudi mogli ***lakše*** ***da sagledaju*** šta se dešava u samom sistemu.

Detalji koje uključujemo u model treba da budu ***tako odabrani*** da za neki aspekt sagledavanja realnog sitema koji modelujemo ti detalji budu ***dovoljni*** da kroz njih možemo ***da sagledamo*** kako se ***realni sistem*** ponaša ili kako on funkcioniše.

* Ukoliko želimo da model koristimo ***za neku simulaciju*** za neko izračunavanje ovo uprošćenje može da bude zato da bi model mogao da se izvršava na nekom računarskom sistemu, kako bi njegovo ***izvršavanje*** bilo ***brže***.

***„Svi modeli su pogrešni, ali neki modelu su korisni.“***

Svaki model je neka ***aproksimacija*** sistema koji modelujemo, oni prikazuju samo jedan deo sistema, samo neke detalje vezane za sistem dok su oni drugi detalji sakriveni od pogleda nisu pristupačni iz tog modela.

To da li će model biti ***koristan*** zavisi od onog ko modeluje, od njegove veštine, znanja. Takođe od toga i zavisi da li modelovanje u softverskog inžinjerstvu ima smisla ili je suvišno.

Inžinjeri koriste modele:

* Kako bi ***razumeli*** realne sisteme
  + U nekim ***drugim*** ***disciplinama***, model omogućava da u nekom trenutku posmatramo sistem samo iz jednog ugla i da na osnovu tog modela razumemo kako neki kompleksan sistem funckioniše.
  + U ***softverskom inžinjerstvu*** se modeluje neki realni sistem koji naš model treba da podrži da bi razumeli kako taj sistem funkcioniše i da bi odredili koje detalje iz tog realnog sistema treba da reflektuje naš softver.
* Kako bi ***predvideli neko ponasanje*** sistema
  + U ***gradjevini***, dizajn mostova, visokih objekata obično podrazumeva i neko modelovanje da bi videli kako vetar utiče na konstrukciju.
  + Slučno i u ***softverskom inžinjerstvu***, možemo modelovati da bi predvideli kako će recimo komunikacija određenih komponenti da izgleda u nekim specifičnim uslovima, da na modelu ispitamo razne tokove ponašanja, tokove koji će se odigravati u sistemu koji podržava neki stvarni proces.
* Kako bi ***komunicirali***
  + Koristimo model da nekom drugom objasnimo svoje shvatanje sistema. Razlog tome može biti da želimo da neko drugi uradi dizajn našeg softvera, možda nam je potrebno nečije mišljenje kako nešto implementirati ili želimo da neko drugi to implementira, a ne mi. Za sve to koristimo model našeg softvera kako bi se bolje razumeli, ali takođe model možemo proslediti krajnjem korisniku da proverimo da li je naše razumevanje realnog sistema isto. Tada takođe moramo voditi računa da i krajnji korisnik razume naš model, kako bi takva komunikacija bila korisna.
  + Modeli ,ukoliko su dovoljno bogati, mogu da budu korišćeni i kao izvor specifikacije. Na osnovu modela dizajna softver neko drugi može tako nešto i da implementira, za šta je takođe neophodno jednako razumevanje modela.

***Apstrakcija***

Jedna od slobodnih definicija apstrakcije jeste ***izdvajanje bitnog od nebitnog***. Svodi se na namerno sakrivanje nekih detalja proces ili nekog artefakta kako bi jasnije sagledali neki drugi aspekt, detalje ili strukturu. (često od naše veštine apstrakcije zavisi i uspešnost celokupnog dizajna)

***Primer***: Prilikom projektovanja sistema, prvi korak je da identifikujemo pojmove koji su ***od interesa*** za nas sistem i da ih predstavimo u modelu. Takođe moramo izdvojiti samo one osobine pojmova koje su nama ***od interesa***, a kasnije i da uočimo koje interakcije tih entiteta su nama ***relevantne za naš sistem***, kakvi su im međusobni odnosi.(studentska služba, student, prijava ispita,..)

Termin apstarakcija se koristi i ***za rezultat procesa apstahovanja***. (klasa je apstrakcija neke realne pojave iz realnog sistema)

***Rafinacija***

***Aprstakcija*** predstavlja prikaz samo nekih detalja o stvarnom sistemu što nam je u početku sasvim dovoljno da razumemo čime se naš sistem bavi, ali ***ne i da ga implemtantiramo***. Apstrakcija nam je bila korisna u jednu svrhu, međutim sada moramo da sagledamo koje osobine svakog entiteta su nam potrebne.

***Rafinacija*** je proces obrnut od apstrakcije. Osnovni cilj rafinacije jeste da ***obogatimo*** ***postojeći model*** da bi nam isti bio dovoljan ***za neku drugu svrhu***. Na ovaj način možemo transformisati modele iz jednog oblika u drugi i možemo takođe ***očuvati vezu*** između ***modela*** jednog ***istog dela realnog sistema*** na različitim nivoima apstrakcije.

Modeli na ***višem nivou*** apstrakcije su ***bliži*** domenu ***problema***, oni se služe pojmovima koji su bliži samom sistemu koji mi želimo da podržimo. Ti modeli nam služe ***da razumemo*** taj sam sistem, da ga ***dokumentujemo*** i proverimo da li smo ga dobro dokumentavali, ali oni ***ne mogu*** sami po sebi da budu ***izvor specifikacije*** za implemetnaciju (ne znamo šta će biti klasa šta osobina klase, kakvi će biti odnosi tih klasa, nekada moramo da dodamo i uvedemo neke pomoćne entitete koji u realnom sistemu ne postoje, a takve stvari se na nalaze u modelu na visokom nivou apstrakcije).

Modeli na ***nižem nivou*** apstrakcije su ***bliži*** samoj ***implementaciji*** i na njima možemo prikazati koje će klase postojati (ukoliko implementiramo u objektno orjentisanom jeziku).

***Načini modelovanja dizajna softverskog sistema***

1. ***FORMALNI MODELI***

Podrazumeva postojanje ***tačno definisanih pravila*** kako modelujemo i sta određeni elementi tog modela znače.Tačno je definisano značenje elemenata modela, veza izmedju njih itd.

Međutim, iako neki jezik za modelovanje ima formalnu definiciju i neki konzistentan skup pravila, ne mora da znači da ćemo imati manje ***problema*** u ***komunikaciji***. Formalne definicije su obično po prirodi kompleksne i u praksi, ***razumevanje*** istog ***pojma*** modela dve različite strane ***zavisi*** od ***njihovo razumevanja formalne specifikacije*** ***jezika*** za modelovanje.

***Prednost*** formalnih jezika je što sam model možemo na neki ***automtaizovan*** način koristiti za neku dalju ***validaciju***. ( automatizovane provere konzistentnosti modela, provere modela da li prati formalno def inisana pravila) Takođe veoma značajno je što je korišćenjem ovakvih jezika omogućeno da ***automatizujemo*** i ***transformaciju*** jedne vrste modela u neku drugu vrstu modela. Čak ,ukoliko imamo model sa dovoljnim nivoom detalja, i imamo potrebne mehanizme, postoje načini i da modele ***transformišemo*** direktno ***u kod koji će biti izvršiv*** i da tako dodatno ubrzamo proces od dizajna do implementacije.

1. ***NEFORMALNI MODELI***

Modeli koji mogu biti mnogo ***slobodniji***, mogu biti ***brži za realizaciju*** jer ne podrazumevaju neke posebne alate, već slobodno crtanje na papiru.

Osnovi ***problem*** neformalnih modela je sto ne postoji tačno utvrđeno značenje simbola koje koristimo, nama može delovati da su oni jasni i da se podrazumeva šta oni znače, ali ne mora svako jednako da ih razume.

***Jezici za modelovanje***

Formalni jezici za modelovanje kao i svi drugi jezici imaju:

* Sintaksu – koja definiše kako izgledaju pojmovi i elementi samog jezika. Često isti jezik ima više sintaksi pa je moguće isti pojam predstaviti na više različitih načina, a nekada čak i na automatizovan način transformisati sintaksu iz jedne u drugu. Sinatksa može bit:
  + Grafička
  + Tekstualna
  + Kombinovana
* Semantiku – koja definiše šta pojmovi jezika znače

Jezici mogu biti:

* ***JEZICI OPŠTE NAMENE***

Ovi jezici ***nemaju usko specificiranu namenu***, i pravljeni su tako da se mogu ***koristiti u bilo koju svrhu***, za razvoj bilo kakvog softverskog sitema nezavisno od polja primene (vrste softvera koji razvijamo), ***nezavisno od realnog sistema***.

Obično su ***mnogo kompleksniji*** jer su pravljeni da omoguće modelovanje bilo čega u bilo kakvoj situaciji na bilo kakav način, te je njihovo razumevanje ponekada teže, i zahteva veći napor.

* ***JEZICI SPECIFICIRANI ZA DOMEN***

Jezici koji su ***usko vezani za odeđenu tematiku***, oblast primene, u tehničkom smislu razvoj softvera ili oblast primene u domenu problema. (jezik specificiran za domen modelovanja u oblasti bankarsta, bezbednosti, konačnih automata,..) Njihova svrha nije da omoguće modelovanje bilo čega. Ovi jezici omogućavaju da formalizme tog jezika zadržimo unutar domena, što nam daje ***veću šansu*** da će ga i ***ekspert iz oblasti*** za koju modelujemo ***razumeti***.

***Unified Modeling Language***

***Najrasprostranjeniji*** jezik za modelovanje u oblasti sofverskog inžinjestva je UML. ***Opšti jezik*** čija je ideja bila da se uspostavi i osimisli ***standardizovani način za prikaz dizajna sistema***. UML danas podržava i neke vidove dizajna koji su nezavisni od toga kojom paradigmom implementiramo sam sistem.

UML ***definiše više vrsta dijagrama***, gde je svaka vrsta namenjena da ***prikaže neki aspekt*** ***strukture*** ili ***ponašanja*** softvera.

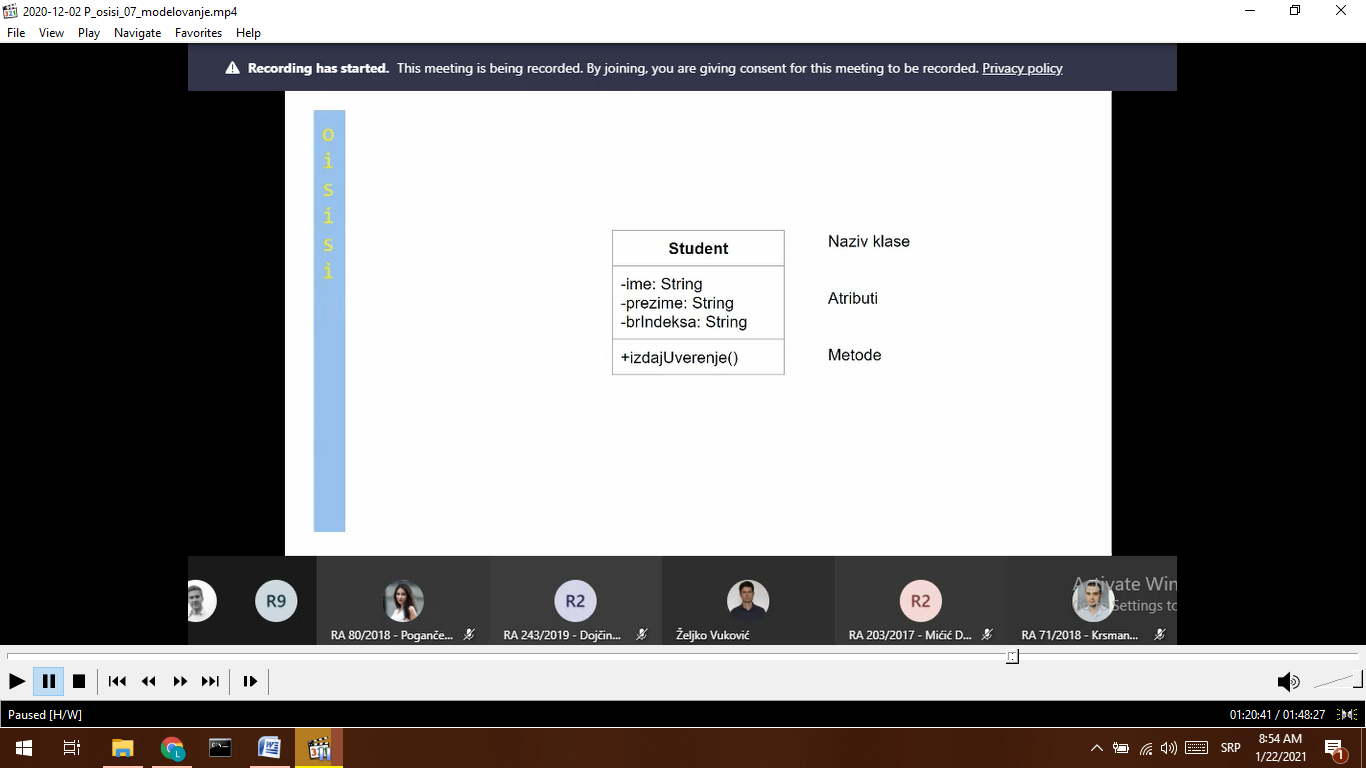
Osnovni tipovi dijagrama strukture:

* Dijagram klasa
* Dijagram komponeni - koristi se prilikom razmatranja korišćenja određenih komponenti, kada je razvoj baziran na komponetama, omogućava da se na višem nivou predstave sve potrebne komponente i njihove međusobne veze
* Dijagram objekata - ponekada je prilično teško na nivou klase opisati neku strukturu pa je neophodno da se to uradi na nivou instanci klasa
* Deployment dijagram - za modelovanje toga kako će konačni proizvod biti raspoređen na nekim serverima i slično
* ...

Dijagrami koji služe za prikaz ponašanja, interakcije raznih elemenata sistema na određeni način:

* Use case dijagrami - polazna osnova u inžinjerskih zahteva, tj. služe da modelujemo grupe korisnika koje imamo u sistemu i šta svaka od tih grupa korisnika treba da može da radi, sta su to funkcionalnosti koje sistem treba da podrži.
* Dijagram aktivnosti
* Dijagram sekvence - prikazuje međusobnu saradnju više klasa, koja metoda jedne klase poziva drugu,..
* ...

***Dijagram klasa***

 Ukoliko je klasa ***apstraktna*** onda se ime klase pise italic ili se pored imena klase u gornjem desnom ugu u vitičastim zagradama napise apstract.

Svaki atribut ima određenu vidljivost što je na primeru naznačeno sa + ili -. Nakon simbola koji označava vidljivost navodi se naziv atributa nakon čega sledi dvotačka pa tip atributa. Opciono , nakon tipa se može navesti i podrazumevana vrednost.

Metode takođe imaju vidljivost nakon čega sledi naziv. U zagradama se mogu navesti parametri metode ukoliko ih ona ima (par1:tip1, par2:tip2). Ukoliko metoda ima povratnu vrednost navodi se tip povratne vrednosti metode nakon zagrada.

***Vidljivost***:

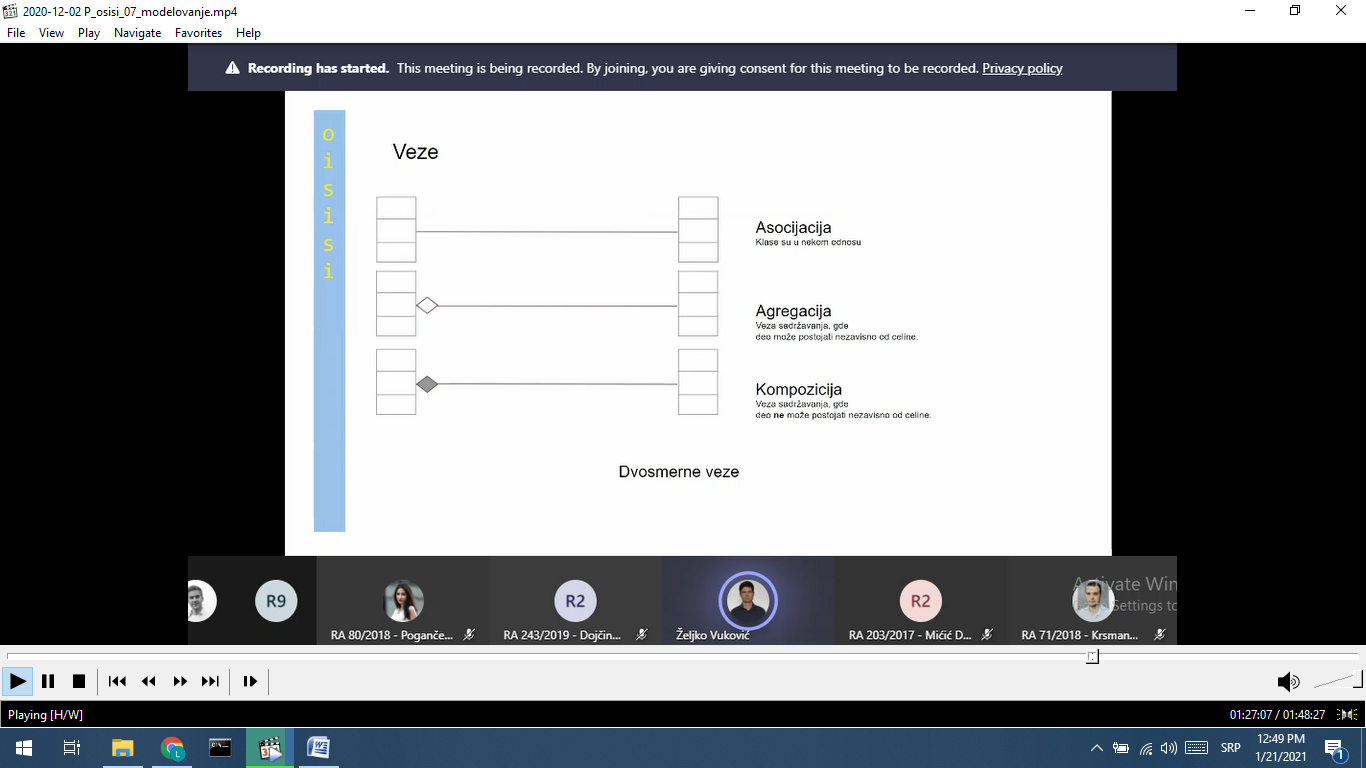
+ public

# protected

~ package

- private

***Veze između klasa***



***Asocijacija***

Klase su na neki način logički povezane, koriste na neki način jedna drugu pri čemu ***nije obavezno*** da jedna klasa ima u implementaciji ***atribut*** koji ce biti tipa druge klase.

***Agregacija – Kompozicija***

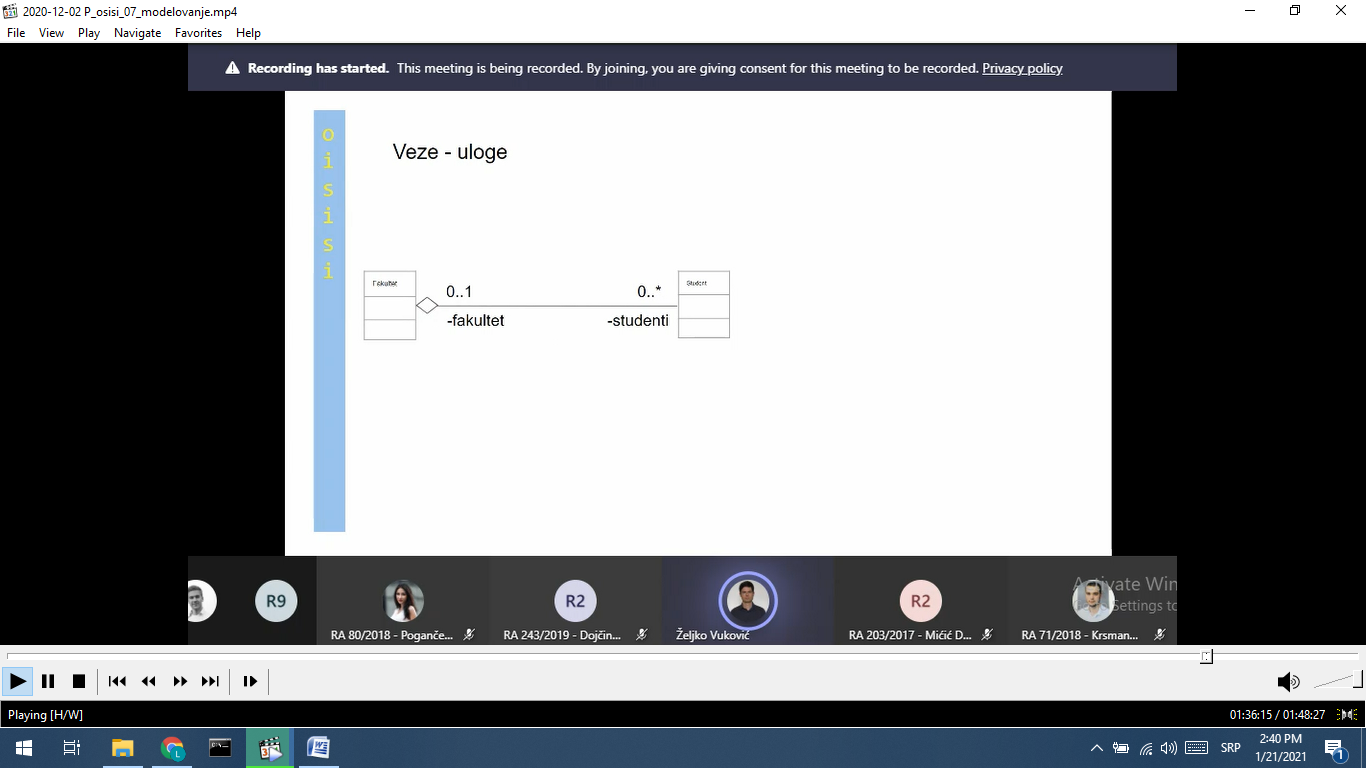
Podvrste asocijacije. U ovim vezama klase su na neki način povezane, ali su u pitanju ***veze*** ***sadržavanja*** što znači da će kao rezultat ove veze na modelu u implementaciji klase i na jednoj i na drugoj strani ***imati atribut*** tipa one klase sa kojim je ta klasa u vezi .

***Agregacija*** predstavlja vezu gde deo može postojati nezavisno od celine, dok kod ***kompozicije*** delovi nemaju smisla bez celine, ne mogu postojati zasebno.

Svaka kod ovih veza moze biti:

* ***Dvosmerna*** - obe strane su svesne sa kim su u vezi
* ***Jednosmerne*** - jedna od srana ne vodi računa o tome sa kim je u vezi (na grafiku ima strelica na strani kod klase koja ne zna za objekat klase sa druge strane)

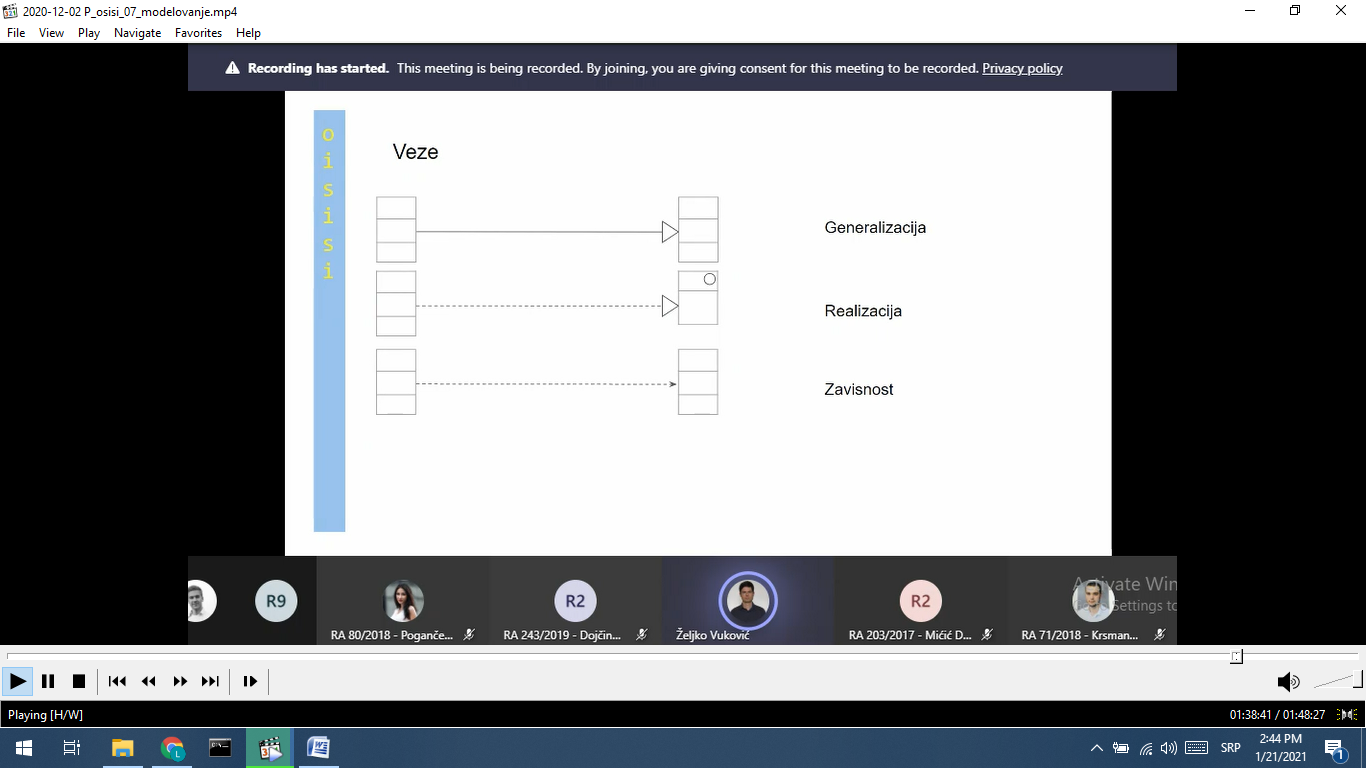
***Kardinalitet***

Kardinalitet govori o tome sa koliko se jedna instanca te klase može naći u vezi sa instancama druge klase. Pri tome,kardinaliteti za tu klasu su napisani na drugom kraju veze.

Na samoj vezi možemo da definišemo kako se zove ***uloga*** neke strane veze.Ako je fakultet u vezi sa entitetom student, ako posmatramo fakultet i on ima neku kolekciju studenata onda će uloga tog suprotnog kraja veze zvati studneti. To znači da će u implementaciji kada ova veze postane atribut klase fakulet, neka kolekcija ili lista, da će se taj property u implementaciji zvati studenti. I obrnuto da će student imati property fakultet.

To ce biti rezultat u implementaciji. U samom UML i ako smo prikazali tu vezu i tako to definisali u modelu necemo dodavati taj atribut i u polje za atribute. Tamo idu isključivo lične osobine te klase.

***Genrelizacija*** (nasleđivanje)

Leva klasa nasleđuje desnu. U UML možemo prikazati i višestruko nasleđivanja sa više strelica.

***Realizacija***

Neka klasa može da realizuje interface(u javi implementacija). U UML interfejs se prikazuje slično kao klasa ali ima kružć gore desno. I interfejsi među sobom omogu da koriste vezu generalizacije, tj nasleđivanje intefejsa.

***Veza zavisnosti***

Jedna klasa na neki način koristi drugu, iz nekog razloga je ta druga klasa neophodna da postoji. To u implementaciji može a ne mora da bude realizovano ni atributom druge klase ni pozivom metode druge klase, ali to želimo da prikažemo na modelu.