АКАДЕМИЈА ТЕХНИЧКО-ВАСПИТАЧКИХ СТРУКОВНИХ СТУДИЈА

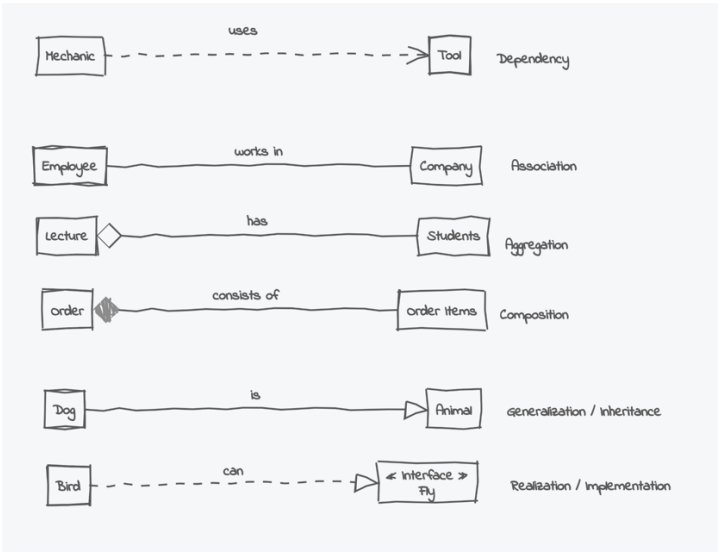
ОДСЕК НИШ

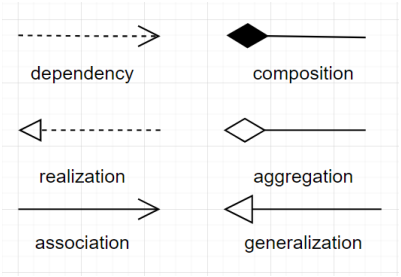
PROGRAMSKI ALATI ZA RAZVOJ SOFTVERA

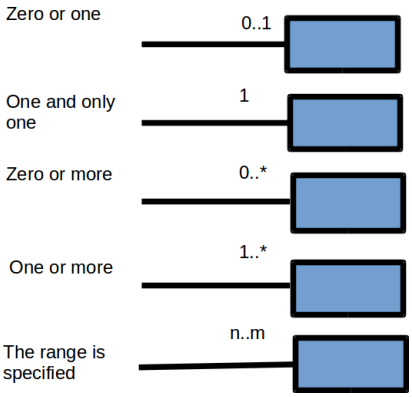
Vežba 1 (radi se dve nedelje)

Dijagrami i veze između entiteta

Prvi deo: Osnovne veze između entiteta







1. Mehaničar koristi alat (Dependency)

- **Zavisnost** je **slabija veza** između dva entiteta, gde jedan objekat (mehaničar) zavisi od drugog objekta (alat) da bi izvršio neku operaciju. U zavisničkoj vezi, objekat A koristi objekat B, ali objekat B nije dugoročno povezan sa A. Promene u objektu B mogu uticati na rad objekta A, ali njihov životni ciklus nije direktno povezan. Ova veza se često koristi u situacijama kada jedan objekat traži pomoć drugog objekta samo privremeno, kao što je slučaj sa alatom koji mehaničar koristi za popravke.

2. Zaposleni radi u kompaniji (Association)

- **Asocijacija** je **dvosmerna veza** koja opisuje vezu između dva nezavisna entiteta koji međusobno komuniciraju. U ovom primeru, zaposleni i kompanija su povezani vezom rada, ali mogu postojati nezavisno jedan od drugog. Ova veza može biti jednostrana (gde je samo jedan entitet svestan drugog) ili dvosmerna (gde su oba entiteta svesna postojanja i zavisnosti od drugog). Asocijacija je fleksibilna i omogućava modeliranje svakodnevnih odnosa između objekata u stvarnom svetu.

3. Predavanje ima studente (Aggregation)

- **Agregacija (Aggregation)** je **odnos "deo-celina"** gde je jedan entitet (npr. Predavanje) vlasnik ili domaćin drugih entiteta (npr. Studenti), ali ti entiteti mogu postojati nezavisno. Studenti mogu postojati i ako predavanje više ne postoji, što znači da ne postoji stroga zavisnost između njih. Ovo je **slabiji oblik kompozicije**, gde celina poseduje delove, ali ti delovi ne zavise potpuno od celine. Ova veza se koristi kada objekti mogu postojati sami, ali su privremeno grupisani u veću celinu.

4. Porudžbina se sastoji od stavki porudžbine (Composition)

- **Kompozicija** je **jači oblik odnosa "deo-celina"** gde su delovi potpuno zavisni od celine. U ovom primeru, Stavke porudžbine ne mogu postojati bez Porudžbine — ako porudžbina bude obrisana, sve njene stavke isto nestaju. Kompozicija podrazumeva strogu zavisnost između objekata, gde jedan objekat kontroliše životni ciklus drugog objekta. Ovo je **najjači oblik asocijacije** i koristi se kada jedan objekat nije samostalan bez drugog.

5. Pas je životinja (Generalization / Inheritance)

- **Generalizacija ili nasleđivanje** označava **hijerarhijski odnos** između klasa gde podklasa (Pas) nasleđuje karakteristike i metode nadklase (Životinja). Ovaj odnos omogućava ponovnu upotrebu koda i **polimorfizam**, jer podklasa može dodavati specifične funkcionalnosti ili redefinisati ponašanje nadklase. Ova veza stvara strukturu koja modelira pojmove iz stvarnog sveta na način da specifične klase proširuju opšte. Nasleđivanje takođe podržava **apstrakciju**, jer se zajedničke karakteristike izdvajaju u više apstraktne klase.

6. Ptica može da leti (Realization / Implementation)

- **Realizacija / Implementacija** je **veza između interfejsa i klase** gde klasa implementira funkcionalnost definisanu u interfejsu. U ovom primeru, Ptica implementira interfejs Fly, što znači da mora definisati ponašanje letanja. Interfejs definiše **"šta"** objekat može da radi, dok klasa implementira **"kako"** to radi. Realizacija omogućava fleksibilnost jer različite klase mogu implementirati isti interfejs na različite načine, bez brige o tome kako se funkcionalnost izvršava.

Kardinalnost

Kardinalnost u modeliranju odnosa između entiteta opisuje broj instanci jednog entiteta koji mogu biti povezani sa instancom drugog entiteta. Neke vrste kardinalnosti su:

- **Zero or one (0..1)**: Instanca nekog entiteta može biti povezana sa **nijednom ili jednom** instancom drugog entiteta. Ovaj odnos se koristi kada veza može postojati, ali nije obavezna (npr. osoba može ili ne mora imati vozačku dozvolu).
- **One and only one (1..1)**: Svaka instanca jednog entiteta **mora** biti povezana sa **tačno jednom** instancom drugog entiteta (npr. svaka osoba ima jedinstveni broj lične karte).
- **Zero or more (0..*)**: Instanca entiteta može biti povezana sa **nijednom ili više** instanci drugog entiteta. Veza je opcionalna i može biti mnogostruka (npr. kompanija može imati nula ili više projekata).
- **One or more (1..*)**: Svaka instanca jednog entiteta mora biti povezana sa **barem jednom, ali može i sa više** instanci drugog entiteta (npr. profesor predaje na jednom ili više predmeta, ali ne može predavati na nula).
- **Many-to-many (N:M)**: **Više instanci** jednog entiteta može biti povezano sa **više instanci** drugog entiteta (npr. studenti pohađaju više kurseva, a svaki kurs ima više studenata). Ova veza obično zahteva još jedan entitet između za implementaciju.

Drugi deo: UML klasni dijagrami

UML klasni dijagrami (Class diagram) su jedan od najvažnijih alata u razvoju softverskih sistema. Oni prikazuju statičku strukturu sistema, pomažući da se identifikuju ključni entiteti, njihova svojstva i odnosi u okviru softverskog rešenja.

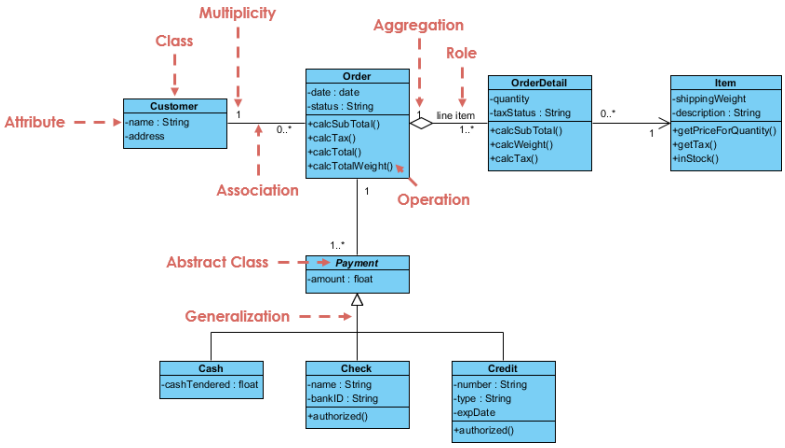
Svaka klasa definiše podatke (atribute) i ponašanja (metode) koji opisuju objekte iz stvarnog sveta ili konceptualne entitete sistema. UML dijagrami pružaju jasan i pregledan način za vizualizaciju ovih elemenata, što olakšava dizajn i razvoj softvera.

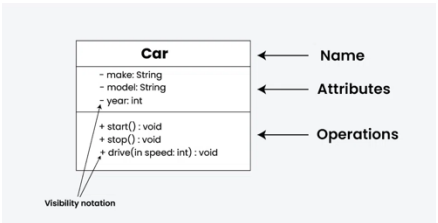
Klasni dijagrami služe kao alat za:

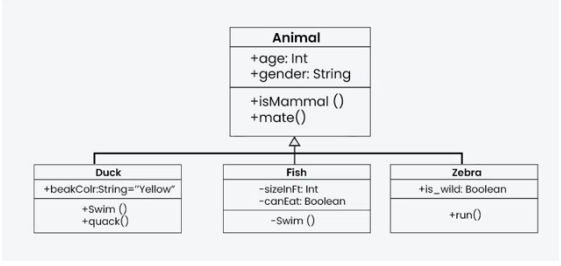
1. **Dizajn softverske arhitekture** – Omogućavaju arhitektama i programerima da definišu strukturu sistema, prepoznajući kako su entiteti međusobno povezani.
2. **Dokumentovanje sistema** – Ovi dijagrami nude jasnu dokumentaciju koja može biti korisna tokom razvoja i održavanja softverskog rešenja.
3. **Razumevanje sistema** – Klasni dijagrami pomažu članovima tima, interesnim grupama i klijentima da lakše razumeju koncept i strukturu sistema.
4. **Komunikacija među timovima** – Olakšavaju diskusije među programerima, dizajnerima, testerima i klijentima, jer svi mogu jasno videti kako je sistem organizovan.

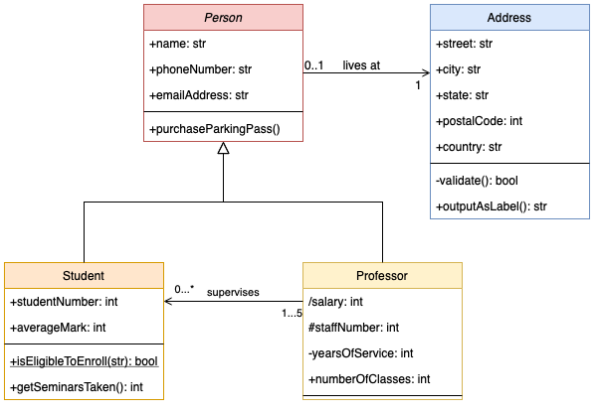
Klasni dijagrami prikazuju sledeće ključne elemente:

- **Klase**: Predstavljaju osnovne gradivne jedinice sistema, sa atributima i metodama.
- **Atribute**: Polja unutar klasa koja definišu stanje objekta (npr. ime, prezime, uzrast, plata).
- **Metode**: Operacije ili funkcionalnosti koje klasa može da izvrši.
- **Veze između klasa**: Kao što su asocijacije, nasleđivanje, agregacija i kompozicija, koji definišu odnose između klasa i način njihove interakcije.
- **Kardinalnost**: Oznake koje definišu koliko instanci jedne klase može biti povezano sa instancom druge klase (1:1, 1-many, itd).









Treći deo: Use Case dijagrami

Dijagrami Slučajeva Korišćenja (Use Case) su UML dijagrami koji prikazuju interakciju korisnika sa sistemom, pomažući da se identifikuju funkcionalnosti sistema i način na koji različiti korisnici (tzv. **akteri**) komuniciraju sa tim funkcionalnostima.

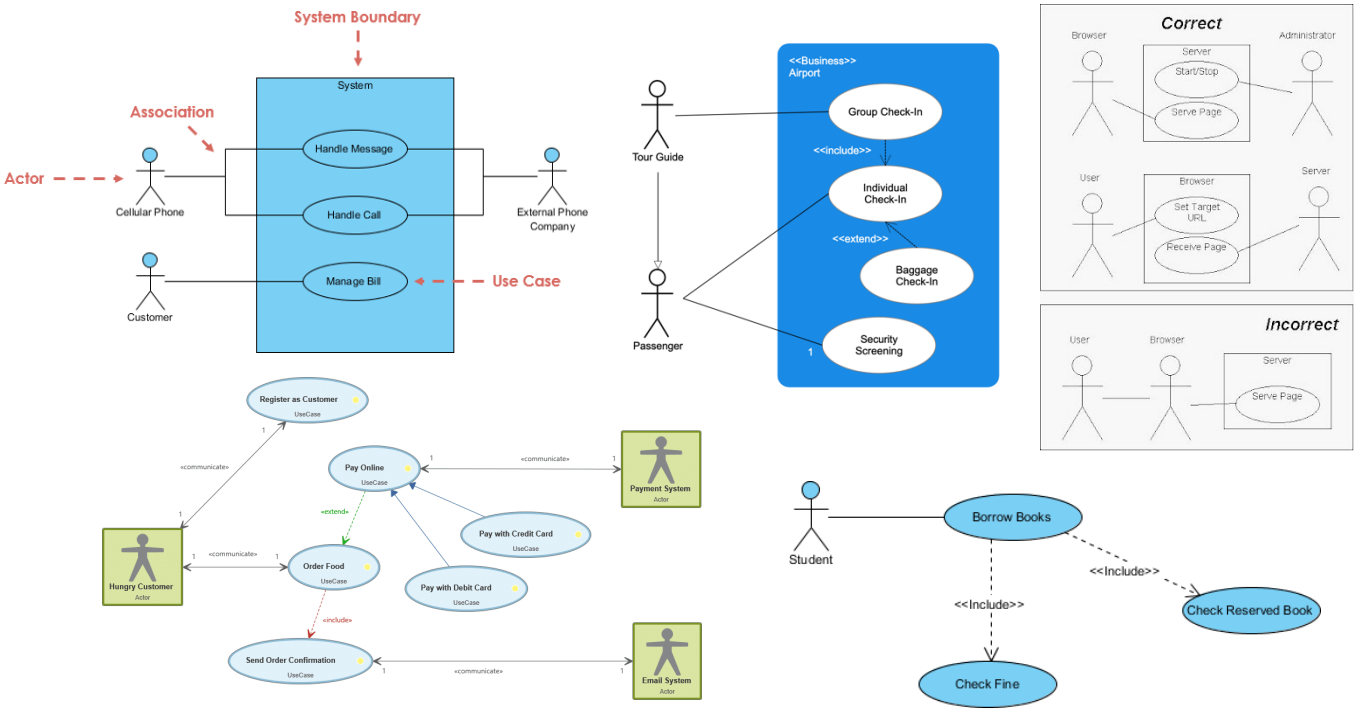
Use case dijagrami predstavljaju vizuelni prikaz zahteva sistema, opisujući **šta** sistem treba da radi, a ne **kako** će to postići. Fokusiraju se primarno na prikazivanje scenarija upotrebe sistema kroz različite funkcionalnosti (use cases) koje omogućavaju korisnicima ili drugim sistemima (akterima) da postignu određene ciljeve.

Use case dijagrami služe kao alat za:

1. **Razumevanje zahteva** – Jasno definišu zahteve sistema na visokom nivou i pomažu u identifikaciji svih funkcionalnosti koje su potrebne korisnicima.
2. **Komunikaciju sa klijentima i korisnicima** – Use case dijagrami su odličan način da se objasni kako će sistem funkcionisati iz perspektive korisnika, što olakšava komunikaciju sa klijentima i interesnim stranama.
3. **Dokumentovanje funkcionalnosti** – Dijagrami pružaju pregled funkcionalnosti sistema i pomažu u dokumentovanju njegovih mogućnosti.
4. **Identifikaciju aktera** – Ovi dijagrami olakšavaju identifikaciju različitih tipova korisnika i njihovih interakcija sa sistemom, što pomaže u planiranju korisničkog iskustva.

Use case dijagrami prikazuju sledeće ključne elemente:

- **Akteri**: Korisnici ili drugi sistemi koji komuniciraju sa sistemom. Mogu biti ljudski korisnici (npr. klijenti, administratori) ili drugi softverski sistemi.
- **Slučajevi Korišćenja (Upotrebe)**: Funkcionalnosti koje sistem pruža korisnicima, grafički predstavljene ovalnim oblicima. Svaki slučaj predstavlja određeni zadatak (task) ili cilj (goal) koji korisnik želi da postigne (npr. „Prijava na sistem“ ili „Kreiranje naloga“).
- **Veze**: Oznake koje pokazuju interakciju između aktera i slučajeva korišćenja, definišući koje funkcionalnosti su dostupne kojim korisnicima.
- **Generalizacija use case-ova i aktera**: U nekim situacijama, akteri ili use case-ovi mogu biti generalizovani kako bi se pokazalo da određeni entitet nasleđuje osobine drugih.



Četvrti deo: Korisni alati i AI generatori za dijagrame

U nastavku su navedeni neki od popularnih **alata** i **AI generatora** za kreiranje UML dijagrama, uključujući detalje o njihovoj upotrebi i mogućnostima integracije u razvojne okruženja kao što su Eclipse i Visual Studio Code.

Draw.io

Draw.io je besplatan alat za kreiranje dijagrama koji se može koristiti direktno u pretraživaču bez potrebe za instalacijom. Omogućava korisnicima da brzo i lako kreiraju razne vrste dijagrama, uključujući UML dijagrame, koristeći jednostavan i intuitivan interfejs.

Karakteristike:

- Podržava različite vrste dijagrama, uključujući UML, ER, poslovne procese i mnogo drugih.
- Nudi široku paletu oblika i simbola specifičnih za UML.
- Omogućava integraciju sa Google Drive-om, Dropbox-om i drugim sličnim platformama za čuvanje podataka.

Kako koristiti:

1. Posetite Drawio.com
2. Odaberite da radite u pretraživaču ili preuzmite aplikaciju.
3. Počnite sa praznim dijagramom ili izaberite željeni šablon UML dijagrama.
4. Koristite alate sa leve strane da dodate oblike i veze između njih.
5. Ako izaberete prazan dijagram možete ići i na **Arrange > Insert > Advanced** i onda dodati adekvatni PlantUML ili Mermaid kod na osnovu kog će sistem generatisati dijagram. Kada budete radili sa bazama, možete dodavati i SQL kod na ovaj način i dobijati dijagrame.
6. Poboljšajte svoje dijagrame paletama boja, fontovima, boljim pozicioniranjem, sve to je dostupno sa desne strane u web pretraživaču, u okviru Style, Text i Arrange odeljaka.
7. Spremite dijagram za željenu platformu. Možete ga sačuvati kao png ili jpg sliku, pdf ili snimiti na računar, Google Drive, Dropbox, itd. **File > Export as** ili Publish se koristi za ovo.

Sličan alat je i **Lucidchart** (lucidchart.com). On nudi širok spektar šablona i grafičkih oblika, kao i mogućnost integracije s raznim aplikacijama poput Google Drive-a, Microsoft Office-a i Slack-a, što ga čini idealnim alatom za profesionalce koji žele brzo i efikasno vizualizovati svoje ideje i procese. Možete koristiti besplatnu verziju ili platiti određenu godišnju sumu za svoj tim.

UMLet

UMLet je open-source alat za izradu UML dijagrama koji omogućava brzu i jednostavnu izradu različitih vrsta dijagrama, uključujući dijagrame klasa, sekvenci, aktivnosti i još dosta toga. Njegov korisnički interfejs je jednostavan, sa podrškom za povlačenje i ispuštanje, što olakšava kreiranje dijagrama bez potrebe za složenim podešavanjima.

UMLet takođe omogućava korisnicima da generišu dijagrame putem teksta, što može biti korisno za brzo kreiranje prototipa i dokumentacije. Pored toga, UMLet podržava integraciju s različitim IDE-ima, kao što su **Eclipse** i **Visual Studio Code,** što ga čini izuzetno praktičnim alatom za programere koji žele vizualizovati svoje dizajne direktno unutar okruženja za razvoj softvera.

Kako dodati UMLet u Eclipse

a. Otvorite Eclipse.
b. Idite na **Help > Eclipse Marketplace**.
c. U pretrazi upišite "UMLet" i odaberite UMLet iz rezultata.
d. Kliknite na **Go** i pratite uputstva za instalaciju.
e. Nakon instalacije, UMLet možete otvoriti putem **Window > Show View > Other** i odabrati UMLet. Novi dijagram možete kreirati na **New > Other > Umlet Diagram**
f. Možete kreirati nove dijagrame povlačenjem oblika iz palete i prilagođavanjem atributa.

Kako dodati UMLet u Visual Studio Code

a. Otvorite Visual Studio Code.
b. Idite na **Extensions** (Ctrl+Shift+X).
c. U pretragu upišite "UMLet" i instalirajte ekstenziju.
d. Nakon instalacije, napravite novi **.uxf** fajl u Visual Studio Code okruženju.
e. Započnite kreiranje i stilizovanje dijagrama tako što ćete prevlačiti komponente izlistane sa desne strane iređati ih u željenom redosledu na radnoj površini.
f. U donjem desnom uglu imate i kod onoga što je vizualno prikazano, tako da možete i tu unositi željene modifikacije ili nove elemente.

Svemoćni ChatGPT

ChatGPT može biti izuzetno koristan alat za generisanje koda za UML dijagrame, kao što su **PlantUML, yUML ili Mermaid**. Ovakav proces omogućava korisnicima da lako kreiraju vizuelne prikaze svojih ideja i dizajna. Kada vam ChatGPT da kod, možete otići na sajt poput PlantUML-a (https://www.plantuml.com/plantuml/uml/SyfFKj2rKt3CoKnELR1Io4ZDoSa70000), iskopirati taj kod i dobiti rezultujući grafikon. Isto tako možete uraditi i sa yUML i sa Mermaid kodovima.

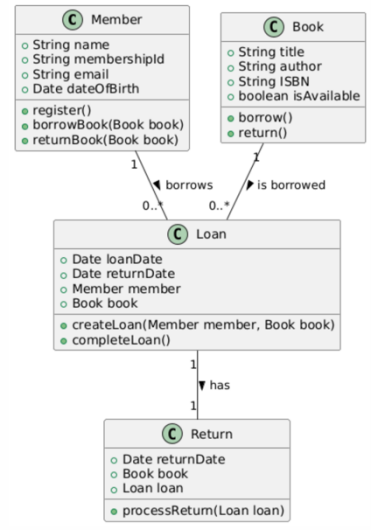
yUMl je dostupan na ovoj adresi: https://yuml.me/diagram/scruffy/class/draw

Mermaid je dostupan ovde: https://mermaid.live/edit

Dakle, postavite upit ChatGPT-u, navodeći detalje o dijagramu koji želite da kreirate. Na primer:

„Generiši PlantUML kod za moderni sistem za izdavanje knjiga. Treba da ima klase za članove (sa sledećim atributima i metodama ...), klase za knjige (ponovo dolaze atributi i metode), kao i klase za samu pozajmicu i vraćanje knjige (pa opet navedemo šta sve imaju ove klase)”.

Rezultujući kod onda prekopirajte na PlantUML sajtu, i možete dobiti ovako neki rezultat:



Naravno, koliko god da je ChatGPT moderan i moćan alat, ne treba se baš u potpunosti oslanjati na njega. I dalje treba razmišljati analitički i racionalno, a problemima (naročito u softverskom inženjerstvu) pristupati temeljno, uzimajući u obzir sve aspekte i potencijalne ishode.

Peti deo: Praktična primena

Hajde da u sklopu ove vežbe dizajniramo softverski sistem koji upravlja restoranom, omogućava konobarima da unose narudžbine, vodi evidenciju o dostupnim stolovima, pruža funkcionalnost za upravljanje jelovnikom, i generiše račune. Sistem takođe omogućava administratoru restorana da upravlja osobljem, da im dodeljuje smene, bonuse za platu, umanjuje platu zbog određenih prestupa, i eventualno zapošljava i otpušta radnike.

1. Use Case dijagram

Prvo ćemo napraviti **dijagram slučajeva korišćenja** koji prikazuje funkcionalnosti sistema i interakciju korisnika sa sistemom. U nastavku je jedan primer uloga, vaš može biti i drugačiji.

Glavne uloge (akteri):

1. **Konobar** – unosi narudžbine i dodeljuje stolove gostima.
2. **Kuvar** – koristi sistem da vidi narudžbine koje treba pripremiti.
3. **Administrator** – upravlja osobljem, jelovnikom, platama, smenama i stolovima.
4. **Gost** – koristi sistem da primi račun na kraju obroka.

Funkcionalnosti (Use Cases):

- **Konobar:**
    - Unos narudžbina za stolove.
    - Pregled slobodnih i zauzetih stolova.
    - Izdavanje računa gostu.
- **Kuvar:**
    - Pregled narudžbina koje treba pripremiti.
- **Administrator:**
    - Dodavanje, brisanje i izmena jela na jelovniku.
    - Upravljanje zaposlenima:
        - Dodela smena zaposlenima.
        - Dodavanje bonusa za platu.
        - Umanjivanje plate zbog prestupa.
        - Zapošljavanje novih radnika.
        - Otpuštanje radnika.
    - Upravljanje statusima stolova (slobodan/zauzet).
- **Gost:**
    - Naručivanje jela i pića.
    - Prijem računa.
    - Upisivanje u knjigu utisaka.

2. Klasni dijagram

Zatim ćemo napraviti **klasni dijagram** koji prikazuje strukturu sistema kroz implementirane klase, atribute, metode i veze među klasama. Vaš kod može sadržati i drugačije metode i atribute, pa i same klase, ovo je samo jedno potencijalno idejno rešenje.

Glavne klase:

1. **Restoran**: Glavna klasa koja sadrži listu stolova, narudžbina i osoblja.
    - **Atributi**: naziv_restorana, lista_stolova, lista_osoblja
    - **Metode**: dodajSto(), dodajOsoblje(), generišiIzveštaj()
2. **Sto**: Predstavlja sto u restoranu. Može biti slobodan ili zauzet.
    - **Atributi**: broj_stola, status (slobodan/zauzet)
    - **Metode**: promeniStatus()
3. **Narudžbina**: Predstavlja narudžbinu koju gost pravi.
    - **Atributi**: broj_narudžbine, lista_jela, ukupna_cena
    - **Metode**: dodajJelo(), izračunajCenu(), završiNarudžbinu()
4. **Jelo**: Predstavlja jelo sa jelovnika. Možete sličnu klasu i za pića i dezerte da napravite.
    - **Atributi**: naziv, cena
    - **Metode**: prikažiDetalje()
5. **Račun**: Generiše račun na osnovu narudžbine.
    - **Atributi**: broj_računa, ukupan_iznos, datum
    - **Metode**: generišiRačun()
6. **Osoblje** (Apstraktna klasa): Generalna klasa za sve zaposlene u restoranu.
    - **Atributi**: ime, pozicija, plata, smena
    - **Metode**: prikažiDetalje(), dodajBonus(), umanjiPlatu(), postaviSmenu()
7. **Konobar** (nasleđuje Osoblje): Klasa koja predstavlja konobara.
    - **Atributi**: lista_narudžbina
    - **Metode**: dodajNarudžbinu()
8. **Kuvar** (nasleđuje Osoblje): Klasa koja predstavlja kuvara.
    - **Atributi**: lista_priprema
    - **Metode**: pripremiNarudžbinu()
9. **Administrator** (nasleđuje Osoblje): Klasa koja predstavlja administratora restorana.
    - **Atributi**: lista_zaposlenih
    - **Metode**: dodajZaposlenog(), otpustiZaposlenog(), dodeliSmenu(), dodajBonus(), umanjiPlatu()
10. **Gost:** Klasa koja predstavlja gosta restorana.
    - **Atributi**: ime_gosta, broj_stola, narudžbina (lista jela, pića, dezerta...), račun
    - **Metode**: napraviNarudzbinu(), platiRacun(), oceniUslugu()

Primer veza između klasa:

- **Restoran** je u agregaciji sa klasom **Sto**, jer restoran sadrži više stolova.
- **Sto** je povezan sa klasom **Narudžbina** (kompozicija), jer se narudžbina veže za neki sto.
- **Narudžbina** je u agregaciji sa klasom **Jelo** jer jedna narudžbina može sadržati više jela.
- **Račun** je povezan sa **Narudžbinom**, jer račun prikazuje ukupnu cenu narudžbine.
- **Osoblje** je roditeljska klasa za izvedene klase **Konobar**, **Kuvar** i **Administrator**.

3. Implementacija u Javi ili Pythonu

Nakon pripreme dijagrama, vreme je za implementaciju koda.

Za implementaciju koda u Javi, možemo koristiti **Eclipse IDE** ili **IntelliJ IDEA**, jer su pogodni za rad sa OOP jezicima kao što je Java. Sledeći koraci će nas voditi kroz implementaciju klase Gost, a slično se kreiraju i ostale klase.

Kreiranje projekta u Eclipse-u:

- Otvorite Eclipse i kreirajte novi Java projekt. Nazovite ga, na primer, "RestoranSistem". Unutar tog projekta kreirajte paket sa nazivom "restoran", gde će se nalaziti sve klase.

Definisanje klase Gost:

- U okviru paketa "restoran" kreirajte novu klasu Gost. Ova klasa će imati atribute kao što su ime gosta, broj stola, narudžbina (koja može biti lista ili neki kolekcijski tip u Javi), i račun koji može biti tipa double.

Za implementaciju u **Pythonu**, koristićemo **PyCharm** okruženje. To je jedno od najpopularnijih i najmoćnijih razvojnih okruženja (IDE) za Python, koje pruža sveobuhvatan skup alata za efikasan razvoj aplikacija. Koristićemo objektno orijentisan pristup da definišemo klasu **Gost** sa opisanim atributima i metodama. Slično se razvijaju i ostale klase.

1. Pokrenite PyCharm i na početnom ekranu izaberite **Create New Project**.
2. U novom prozoru odaberite lokaciju gde želite da sačuvate projekat.
3. Unesite naziv projekta, na primer, "RestoranSistem", i kliknite na **Create**.
4. Ako je potrebno, izaberite Python interpreter (možete koristiti sistemski interpreter ili kreirati novo virtuelno okruženje).
5. Kada se otvori prozor za uređivanje, desnim klikom na projekat (koji će se prikazati sa leve strane) izaberite **New -> Python File**.
6. Nazovite fajl, npr, **gost.py**. Ovaj fajl će sadržati klasu **Gost** sa atributima i metodama.
7. Sada sledi dodavanje tih atributa i metoda. Evo može izgledati konstruktor:

```
class Gost:
    def __init__(self, ime, broj_stola):
        self.ime = ime
        self.broj_stola = broj_stola
        self.narudzbine = []
        self.racun = 0
```