PANEVROPSKI UNIVERZITET

APEIRON

za multidisciplinarne i virtuelne studije

Pan-European University for Multidiscipline & Virtual Studies

Banja Luka

FAKULTET INFORMACIONIH TEHNOLOGIJA

COLLEGE OF INFORMATION TECHNOLOGIES

Seminarski rad iz predmeta

OBJEKTNO ORJENTISANO PROGRAMIRANJE

Tema rada

RELACIJE U JAVA PROGRAMSKOM JEZIKU

Student:

Maja Milešević

Broj indeksa: 53-20/RITP-S

Mentor:

Prof. dr Saša Salapura

Banja Luka, 2022.godina

SADRŽAJ:

1 UVOD

Tema mog seminarskog rada i razmatranja iz predmeta objektno orjentisano programiranje su relacije u programskom jeziku Java. U ovom radu ću se potruditi da objasnim šta su relacije u Javi, koje vrste relacija postoje, da uporedim neke od njih i da dam praktičan primjer za svaku vrstu relacije. Najviše ću se fokusirati na relacije između različitih klasa.

Ukratko, relacije u programskom jeziku Java predstavljaju veze i odnose između pojedinih elemenata programa, npr. veze između određenih klasa, objekata, metoda…

Vrlo je važno da znamo odrediti odnose između klasa, kada pravimo velike i složene aplikacije i programe. Ako imamo u aplikaciji više različitih klasa koje imaju isto ponašanje tj. imaju iste metode, možemo značajno uštediti na prostoru u memoriji postavljanjem uobičajenih ponašanja (metoda) unutar superklase. Pretpostavimo da neke klase nisu povezane jedna s drugom, tada možemo obavezati različite programere da implementiraju svaku od njih, bez brige da će jedan od njih morati čekati drugog.

U tu svrhu, moramo naučiti različite tipove odnosa među klasama u Javi. Odnosi među klasama pomažu da se shvati kako objekti u programu rade zajedno i komuniciraju jedni s drugima. Tip relacije između klasa pomaže u ponovnom korišćenju funkcija jedne klase, u drugoj klasi. U Java programskom jeziku imamo 3 tipa relacija između klasa.

U objektno orijentisanom programiranju, klasa je nacrt za objekte koje želimo kreirati. Jedna klasa može biti povezana s drugom klasom ili ne. Generalno, postoje neki odnosi između klasa u oop-u, a to su: agregacija, kompozicija i nasljeđivanje, kao i mnoge druge o kojima će biti riječ u narednom dijelu rada.

2 RELACIJE U JAVA PROGRAMSKOM JEZIKU

Relacije u programskom jeziku Java se odnose na odnos između različitih elemenata u programskom kodu, kao što su klase, objekti, metode i polja.

U Java-i postoji nekoliko vrsta relacija:

- Nasljeđivanje: Jedna klasa može naslijediti sve metode i polja iz druge klase, što je poznato kao nasljeđivanje. Nasljeđivanje omogućava da se kod ponavlja i da se klase po potrebi proširuju. Nasleđivanje je mehanizam koji omogućava da se jedna klasa (podklasa) naslijedi od druge klase (nadklase). Podklasa može da koristi sve atribute i metode nadklase, a takođe može da ih proširi ili redefiniše. Nasleđivanje se označava pomoću ključne riječi "extends" u deklaraciji podklase. Cilj nasleđivanja je da se omogući kodeks ponovne upotrebe, tako da se ne mora pisati isti kod više puta.
- Agregacija: Agregacija se odnosi na vezu između dvije klase u kojoj jedna klasa (klasa agregata) sadrži objekat druge klase (klasa agregirana). Ovaj odnos se naziva i "klasa-u-klasi" jer se agregirana klasa može smatrati sastavnim dijelom agregata. Agregacija se često koristi da bi se modelovali odnosi "sastavljen od" između objekata.
- Asocijacija: Asocijacija je veza između dvije klase koja omogućava da jedna klasa poziva metode druge klase, ali ne nasleđuje njena svojstva i metode. Ovaj odnos se naziva i "klasa-sa-klasa" jer asocijacija ne predstavlja sastavni dio neke klase. Asocijacija se može definisati kao jednosmerna ili dvosmerna. U jednosmernoj asocijaciji, samo jedna klasa može da poziva metode druge klase, dok u dvosmernoj asocijaciji obe klase mogu da pozivaju metode jedna drugoj. Asocijacija se često koristi da bi se modelovali odnosi "koristi" između objekata.
- Kompozicija: Kompozicija je vrsta agregacije u kojoj jedna klasa (klasa kompozitora) ima kontrolu nad životom objekta druge klase (klase komponovana). Ovaj odnos se naziva i "klasa-u-klasi-i-odgovornost" jer kompozitor ima odgovornost za život objekata komponovane klase. Ovaj tip relacije je sličan agregaciji, ali je jače vezan. U kompoziciji, jedan objekat je potpuno zavisan od drugog objekta i ne može da postoji bez njega.
- Polimorfizam: Polimorfizam se odnosi na sposobnost jednog objekta da se pretvori u više oblika. Postoje dvije vrste polimorfizma u Java programskom jeziku: statički polimorfizam i dinamički polimorfizam. Statički polimorfizam se postiže korišćenjem višestruke definicije metoda, dok se dinamički polimorfizam postiže korišćenjem nasleđivanja i preklapanja metoda. Polimorfizam se postiže kroz nasleđivanje i može se ostvariti na više načina u Java programiranju, uključujući preopterećenje metoda i preklapanje metoda.
- Generalizacija/Specijalizacija: Ovaj tip relacije se odnosi na odnos između klase i njene nadklase ili podklase. Klasa se smatra "opštijom" od nadklase i nasleđuje

njena svojstva i metode. Podklasa se smatra "specijalnijom" od klase i dodaje svoja sopstvena svojstva i metode.

- Realizacija: Ovaj tip relacije se odnosi na odnos između klase i interfejsa. Klasa može da realizuje interfejs tako što će implementirati sve njegove metode. Realizacija se koristi da bi se modelovali odnosi "implementira" između objekata.
- Kolekcija: Ovaj tip relacije se odnosi na odnos između klase koja sadrži druge objekte i klase koja se nalazi u toj kolekciji. Kolekcija se često koristi da bi se modelovali odnosi "sadrži" između objekata.
- Enkapsulacija: Ovaj tip relacije se odnosi na odnos između klase i njenih svojstava i metoda. Enkapsulacija se koristi da bi se svojstva i metode klase skrili od ostatka sistema i omogućilo da se pristup njima vrši preko određenih mehanizama (poput gettera i settera).
- Dependency (Zavisnost): Ovaj tip relacije se odnosi na odnos između dve klase u kojem jedna klasa zavisi od druge klase da bi obavila svoj posao. Dependency se često koristi da bi se modelovali odnosi "zavisi od" između objekata.
- Asimetrična asocijacija: Ovaj tip relacije se odnosi na asocijaciju između dve klase u kojoj jedna klasa može da pozove metode druge klase, ali ne obrnuto. Ovo se često koristi da bi se modelovali odnosi "koristi" između objekata gde jedan objekat ima dominantnu ulogu u odnosu.
- Multipla asocijacija: Ovaj tip relacije se odnosi na asocijaciju između više od dve klase. Ovo se često koristi da bi se modelovali odnosi "koristi" između više objekata gde svi objekti imaju jednaka prava i obaveze u odnosu.
- Asocijacija sa kardinalitetom: Ovaj tip relacije se odnosi na asocijaciju između dve klase u kojoj se određuje koliko objekata jedne klase može da bude povezano sa objektima druge klase. Ovo se često koristi da bi se modelovali odnosi "koristi" između objekata gde je odnos ograničen na određeni broj objekata.
- Asocijacija sa rolama: Ovaj tip relacije se odnosi na asocijaciju između dve klase u kojoj se određuje uloga koju objekti jedne klase imaju u odnosu sa objektima druge klase. Ovo se često koristi da bi se modelovali odnosi "koristi" između objekata gde je uloga objekata značajna za tumačenje odnosa.
- Asocijacija sa grupisanjem: Ovaj tip relacije se odnosi na asocijaciju između dve klase u kojoj se određuje da li objekti jedne klase mogu da budu povezani sa više objekata druge klase ili sa samo jednim objektom druge klase. Ovo se često koristi da bi se modelovali odnosi "koristi" između objekata gde je važno da li se objekti jedne klase grupišu sa više objekata druge klase ili sa samo jednim objektom druge klase.
- Multipolna generalizacija/specijalizacija: Ovaj tip relacije se odnosi na odnos između klase i više od jedne nadklase ili podklase. Ovo se često koristi da bi se modelovali odnosi nasleđivanja između više klasa gde sve klase imaju jednaka prava i obaveze u odnosu.
- Realizacija sa apstrakcijom: Ovaj tip relacije se odnosi na odnos između klase koja realizuje interfejs i apstraktne klase koja implementira metode interfejsa. Ovo se

često koristi da bi se modelovali odnosi "implementira" između objekata gde postoji više nivoa apstrakcije između klase i interfejsa.

- Kompozicija sa kardinalitetom: Ovaj tip relacije se odnosi na odnos između dve klase u kojoj se određuje koliko objekata jedne klase može da se sastoji od objekata druge klase. Ovo se često koristi da bi se modelovali odnosi "sastavljen od" između objekata gde je odnos ograničen na određeni broj objekata.
- Kompozicija sa rolama: Ovaj tip relacije se odnosi na odnos između dve klase u kojoj se određuje uloga koju objekti jedne klase imaju u odnosu sa objektima druge klase. Ovo se često koristi da bi se modelovali odnosi "sastavljen od" između objekata gde je uloga objekata značajna za tumačenje odnosa.
- Kompozicija sa grupisanjem: Ovaj tip relacije se odnosi na odnos između dve klase u kojoj se određuje da li objekti jedne klase mogu da se sastoje od više objekata druge klase ili od samo jednog objekta druge klase. Ovo se često koristi da bi se modelovali odnosi "sastavljen od" između objekata gde je važno da li se objekti jedne klase grupišu sa više objekata druge klase ili samo jedan.
- Asocijacija sa apstrakcijom: Ovaj tip relacije se odnosi na asocijaciju između klase i apstraktne klase koja definiše metode interfejsa. Ovo se često koristi da bi se modelovali odnosi "koristi" između objekata gde postoji više nivoa apstrakcije između klase i interfejsa.
- Asocijacija sa kardinalitetom i rolama: Ovaj tip relacije se odnosi na asocijaciju između dve klase u kojoj se određuje koliko objekata jedne klase može da bude povezano sa objektima druge klase i uloga koju objekti jedne klase imaju u odnosu sa objektima druge klase. Ovo se često koristi da bi se modelovali odnosi "koristi" između objekata gde je odnos ograničen na određeni broj objekata i gde je uloga objekata značajna za tumačenje odnosa.
- Asocijacija sa kardinalitetom i grupisanjem: Ovaj tip relacije se odnosi na asocijaciju između dve klase u kojoj se određuje koliko objekata jedne klase može da bude povezano sa objektima druge klase i da li objekti jedne klase mogu da budu povezani sa više objekata druge klase ili sa samo jednim objektom druge klase. Ovo se često koristi da bi se modelovali odnosi "koristi" između objekata gde je odnos ograničen na određeni broj objekata i gde je važno da li se objekti jedne klase grupišu sa više objekata druge klase ili sa samo jednim objektom druge klase.
- Asocijacija sa rolama i grupisanjem: Ovaj tip relacije se odnosi na asocijaciju između dve klase u kojoj se određuje uloga koju objekti jedne klase imaju u odnosu sa objektima druge klase i da li objekti jedne klase mogu da budu povezani sa više objekata druge klase ili sa samo jednim objektom druge klase.
- Asocijacija sa svim navedenim atributima: Ovaj tip relacije se odnosi na asocijaciju između dve klase u kojoj se određuje koliko objekata jedne klase može da bude povezano sa objektima druge klase, uloga koju objekti jedne klase imaju u odnosu sa objektima druge klase i da li objekti jedne klase mogu da budu povezani sa više objekata druge klase ili sa samo jednim objektom druge klase. koristi" između objekata gde su svi navedeni atributi značajni za tumačenje odnosa.

- Asocijacija sa apstrakcijom i svim navedenim atributima: Ovaj tip relacije se odnosi na asocijaciju između klase i apstraktne klase koja definiše metode interfejsa, u kojoj se određuje koliko objekata jedne klase može da bude povezano sa objektima druge klase, uloga koju objekti jedne klase imaju u odnosu sa objektima druge klase i da li objekti jedne klase mogu da budu povezani sa više objekata druge klase ili sa samo jednim objektom druge klase. Ovo se često koristi da bi se modelovali odnosi "koristi" između objekata gde postoji više nivoa apstrakcije između klase i interfejsa i gde su svi navedeni atributi značajni za tumačenje odnosa.

Ovi tipovi relacija se mogu koristiti da bi se modelovali različiti odnosi između elemenata u Java programu i pomognu da se strukturira i organizuje kod. Važno je razumjeti kako se relacije koriste i kako se međusobno odnose, jer to može da pomogne u razvoju strukturiranog i dobro održivog Java koda.

Ove su samo neke od relacija koje postoje u Java-i. Postoji mnogo drugih vrsta relacija koje se mogu koristiti u programiranju u Java-i, kao što su anotacije, enumi i drugo.

3 RELACIJE IZMEDJU KLASA U JAVA PROGRAMSKOM JEZIKU

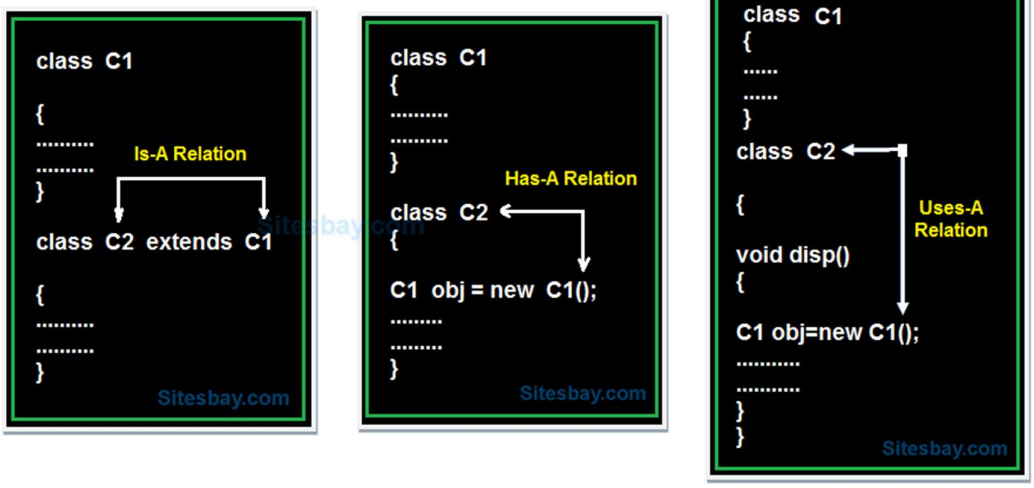
U java programskom jeziku postoje 3 tipa relacija između klasa. Ta tri tipa relacija su:

- Is-A Relationship (inheritance) - nasljeđivanje
- Has-A Relationship (association) - asocijacija
- Uses-A Relationship (dependence) – zavisnost

Podrazumijevani odnos u Javi je IS-A relacija iz razloga što za svaku klasu u Javi postoji unaprijed definisana implicitna super klasa java.lang.object.

Univerzalni primjer za HAS-A relaciju je System.out. Out je objekat klase printStream, kreiran kao statički član podatak u drugoj klasi sistema. Klasa printStream je poznata kao odnos HAS-A.

Svaki metod logike izvršavanja main() logike izvršenja, koristi objekat klase poslovne logike. Klasa poslovne logike poznata je kao odnos USES-A.



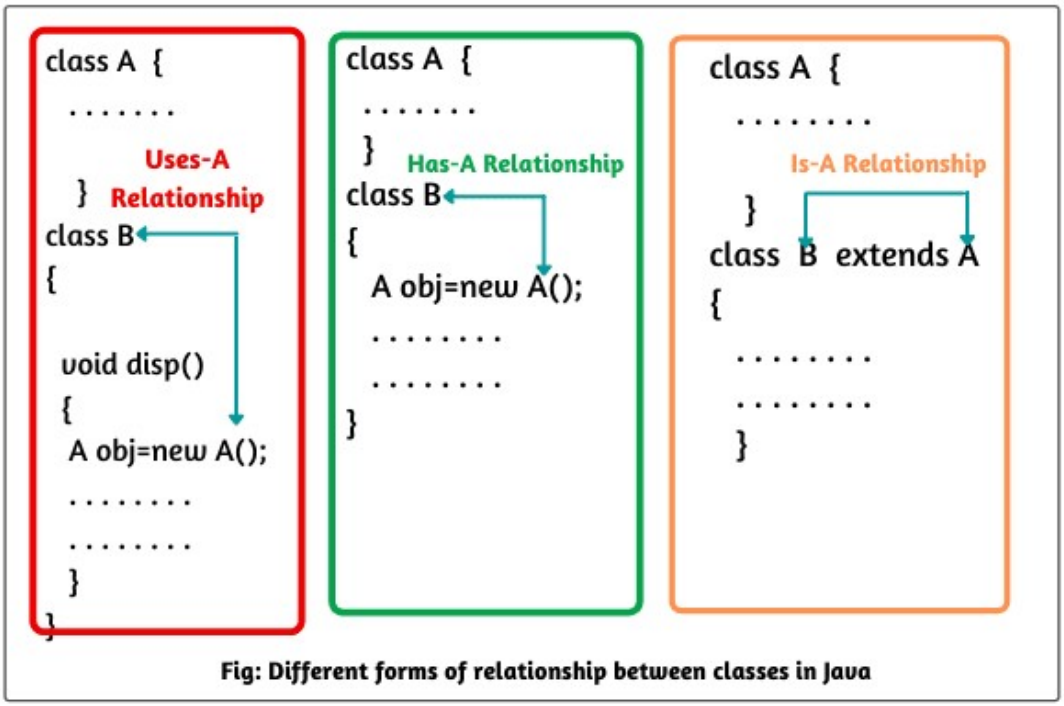
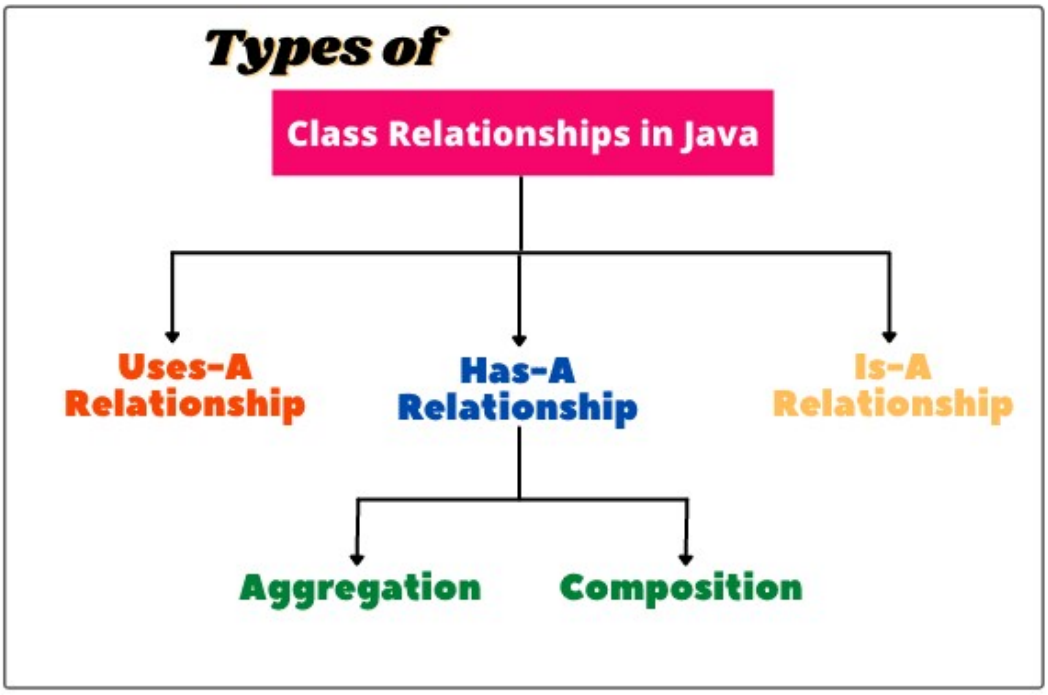


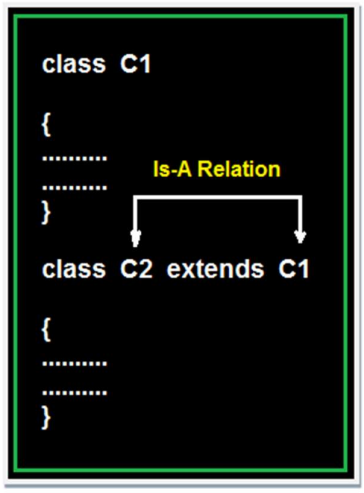
Fig: Different forms of relationship between classes in Java



4 IS-A RELATIONSHIP

Nasljeđivanje u Java programskom jeziku je vrsta odnosa IS-A koja ukazuje da je jedna klasa vrsta neke druge klase, korišćenjem koncepta nasljeđivanja sa proširenim ključnim riječima. Drugim riječima, IS-A odnos definiše odnos između dvije klase u kojem jedna klasa proširuje drugu klasu. Npr. Klasa automobil je vrsta klase vozilo - automobil je vrsta vozila.

Odnos nasljeđivanja uspostavlja odnos između opšte klase - poznate kao superklasa i specijalizovane klase - poznate kao podklasa. Ključna riječ extends služi za nasljeđivanje članova podataka iz superklase.



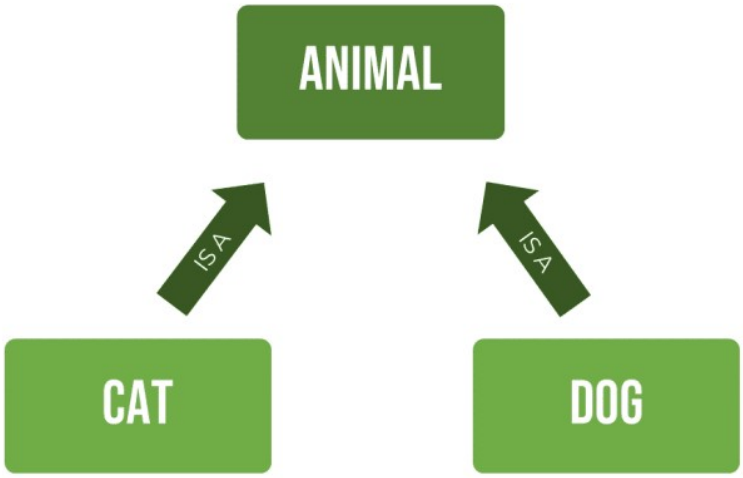
Nasljeđivanje je IS-A odnos između klasa gdje je roditeljska klasa opšta klasa, a podređena klasa posebna klasa.

Na primjer, odnos između mačke, psa i njegove opšte klase, koja je životinja. Životinja se može navesti kao mačka ili pas. Mačka i pas mogu se generalizirati kao životinje.

```
public class Animal {
   protected weight;
   protected color;

   public Animal () {}

   public void talk () {
      System.out.println("It is animal");
   }
}

public class Cat extend Animal {
```

```
    public Cat () {}

    public void talk () {
        System.out.println("meow");
    }
}

public class Dog extend Animal {
    public Dog () {}

    public void talk () {
        System.out.println("bark");
    }
}
```

Mačka i pas imaju svojstva težine i boje. Da bismo generalizirali ta svojstva Cat i Dog, možemo kreirati jednu roditeljsku klasu koja može predstavljati i mačku i psa sa tim svojstvima, a to je klasa Animal.

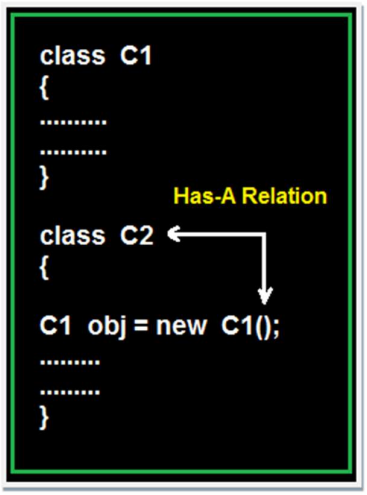


Još jedan primjer IS-A relacije:

```
class Faculty
{
float salary=30000;
}
class Science extends Faculty
{
float bonous=2000;
public static void main(String args[])
{
Science obj=new Science();
System.out.println("Salary is:"+obj.salary);
System.out.println("Bonous is:"+obj.bonous);
}
}
```

```
Salary is: 30000.0

Bonous is: 2000.0
```

5 HAS-A RELATIONSHIP



U odnosu HAS-A, objekat jedne klase se kreira kao član podatak u drugoj klasi i odnos između ove dve klase je HAS-A ili odnos **asocijacije**.
Asocijacija je fundamentalni odnos između klasa koji je neformalno poznat kao odnos HAS-A. Asocijacija je dalje klasifikovana na agregaciju i kompoziciju. Asocijacija može buspostaviti odnose: 1 na 1,1 prema više, više prema 1, više prema više.
Vrlo čest primjer odnosa jedan na jedan je „Osoba može imati samo jedan pasoš“.

Dobar primjer odnosa jedan-prema-više je “Fakultet i studenti”. Fakultet može imati mnogo studenata. Još jedan primjer odnosa jedan-prema-više je “Banka i zaposleni”. Banka može imati mnogo zaposlenih radnika.

Primjer odnosa više na jedan je “Država i gradovi”. Država može imati mnogo gradova, ali svi gradovi su povezani s tom državom.

Primjer odnosa više-prema-više je „Nastavnici i učenici“. Jedan učenik se može družiti sa više nastavnika, a mnogi učenici se mogu družiti i sa jednim nastavnikom.

Kada je objekat jedne klase kreiran kao član podataka unutar druge klase, to se zove Has-A odnos.
Drugim riječima, odnos u kojem objekt jedne klase ima referencu na objekt druge klase ili drugu instancu iste klase naziva se Has-A odnos u Javi.
Asocijacija u Javi je odnos koji definira odnos između dvije klase koje su nezavisne jedna od druge.

Asocijacija je jedan od osnovnih koncepata objektno orijentisanog programiranja u Javi. Ukazuje na to kako objekti klasa komuniciraju jedni s drugima i kako koriste funkcionalnost i usluge koje pruža taj komunicirani objekt.

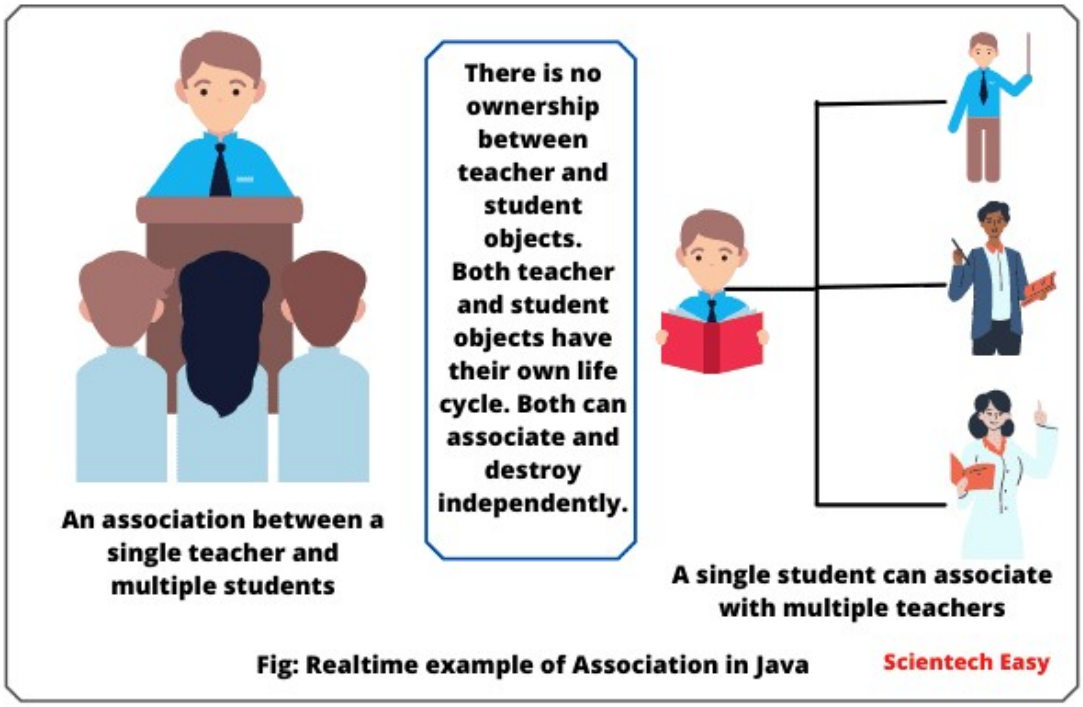
Java asocijacija uspostavlja relaciju višestrukosti između objekata gdje svi objekti imaju svoj životni ciklus i ne postoji vlasnik asocijacije.

Višestrukost označava broj koji određuje koliko je objekata klase uključeno u odnos.

Uzmimo primjer nastavnika i učenika u stvarnom vremenu da bismo razumjeli stvarno značenje asocijacije u Javi.

Više učenika se može povezati s jednim nastavnikom, a jedan učenik se može povezati s više nastavnika, ali ne postoji vlasništvo između objekata.

I nastavnik i učenik imaju svoj životni ciklus. Oba se mogu povezati i uništiti nezavisno.



Asocijacija je predstavljena punom linijom između dva objekta klase koji opisuju odnos.

Asocijacija je kombinacija agregacije i kompozicije.

Postoje sljedeće vrste odnosa kojima asocijacija može upravljati. One su sljedeće:

Jednosmjerna asocijacija: Ako se objekt povezuje s drugim objektom, naziva se jednosmjerna asocijacija. Naziva se i jednosmjernim odnosom.

Na primjer, odjel na fakultetu može imati mnogo studenata, ali obrnuto nije moguće. Ovo je jednosmjerne prirode. Teniser ima reket. To je jednosmjerna asocijacija jer reket ne može imati tenisera.

Dvosmjerna asocijacija: Kada objekti uključeni u asocijaciju komuniciraju jedni s drugima, to se naziva dvosmjerna asocijacija. Vrlo čest primjer dvosmjernog odnosa je “osoba i adresa”. Možemo imati dvosmjerni odnos više prema mnogo između osoba i objekata

adrese. Drugim riječima, osoba može biti povezana s više adresa i adresa može pripadati više ljudi. Ovo pokazuje odnos „mnogo prema mnogo“.

Klasa osoba ima varijablu instance tipa Adresa kao što je prikazano u kodu ispod. Instanca klase Address kreira se izvan klase Person.

```
public class Address {

  // Code goes here.

}

public class Person {

// Person has-a Address.

   Address addr = new Address();



  // Other codes go here.

}
```

Objekt klase Address kreiran kao član podataka unutar druge klase Person. Ovaj odnos je poznat kao Has-A odnos.

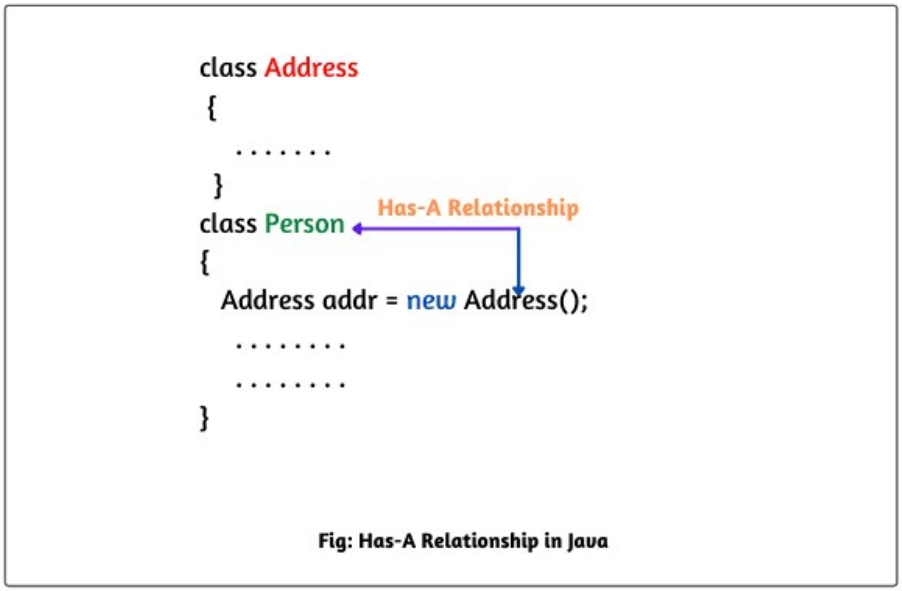


Fig: Has-A Relationship in Java

Još jedan primjer: Znamo da je CPU dio računara. Takođe možemo preformulisati ovaj odnos u „Računar ima CPU“. Da li postojanje CPU-a izvan računara ima smisla? Odgovor je ne. Postojanje CPU-a ima smisla samo unutar računara. Dakle, računar i CPU predstavljaju odnos celi deo.

```
public class CPU {

   // Code goes here.

}

public class Computer {



 // CPU part-of Computer.

    private CPU cpu = new CPU();



    // Other codes go here.

}
```

Još jedan primjer: Znamo da mozak ima misao. Misao ne može postojati bez postojanja mozga. Kod je dat u nastavku:

```
public class Thought {

  // Code goes here.

}

public class Brain {

  Thought thought = new Thought();



// Other codes go here.

}
```

5.1.1 VEZA ONE-TO-ONE RELATIONSHIP

```
public class Person {
 private String name;
 public String getName() {
     return name;
 }
 public void setName(String name) {
     this.name = name;
 }
}
public class Passport {
 private int passportNo;
 public int getPassportNo(){
  return passportNo;
 }
 public void setPassportNo(int passportNo){
  this.passportNo = passportNo;
 }
}
public class OneToOneTest {
public static void main(String[] args) {
 Person p1 = new Person();
```

```
 p1.setName("John");



Person p2 = new Person();

 p2.setName("Shubh");



Passport pp1 = new Passport();

 pp1.setPassportNo(1234567);



Passport pp2 = new Passport();

 pp2.setPassportNo(12398576);



// Association between two classes in the main method.

 System.out.println(p1.getName()+ " has a US passport whose passport no is: "
+pp1.getPassportNo());

 System.out.println(p2.getName()+ " has an Indian passport whose passport no is: "
+pp2.getPassportNo());

  }

}

Output:

    John has a US passport whose passport no is: 1234567

    Shubh has an Indian passport whose passport no is: 12398576
```

U prethodnom primjeru programa, asocijacija između osobe i pasoša se postiže glavnom metodom i pokazuje odnos jedan na jedan.

5.1.2 DVOSMJERNI ODNOS MNOGO-PREMA-VIŠE

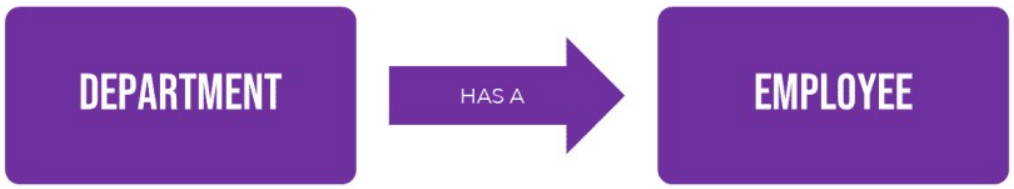
```
public class Person {
 private String name;
 public String getName() {
     return name;
 }
 public void setName(String name) {
     this.name = name;
 }
}
public class Address {
 private String state;
 private String city;
 private String zip;

public String getState(){
 return state;
 }
public String getCity(){
 return city;
 }
public String getZip(){
```

```
 return zip;
 }
public void setState(String state){
  this.state = state;
}
public void setCity(String city){
  this.city = city;
 }
public void setZip(String zip){
  this.zip = zip;
 }
}
public class OneToOneTest {
public static void main(String[] args) {
 Person p1 = new Person();
  p1.setName("John");


Person p2 = new Person();
 p2.setName("Shubh");


Address a1 = new Address();
 a1.setState("Jharkhand");
```

```
 a1.setCity("Dhanbad");

 a1.setZip("123524");



Address a2 = new Address();

 a2.setState("Maharashtra");

 a2.setCity("Mumbai");

 a2.setZip("123635");



// Association between two classes in the main method.

  System.out.println(p1.getName()+ " lives at address " +a1.getCity()+ "' " +a1.getState()+
", " +a1.getZip()+ " but he has also address at " +a2.getCity()+ ", " +a2.getState()+ ",
"+a2.getZip());

  System.out.println(p2.getName()+ " lives at address " +a2.getCity()+ "' " +a2.getState()+
", " +a2.getZip()+ " but she has also address at " +a1.getCity()+ ", " +a1.getState()+ ",
"+a1.getZip());



  }

}

Output:

     John lives at address Dhanbad' Jharkhand, 123524 but he has also address at Mumbai,
Maharashtra, 123635

     Shubh lives at address Mumbai' Maharashtra, 123635 but she has also address at
Dhanbad, Jharkhand, 123524
```

U ovom primjeru programa, asocijacija između osobe i adrese ostvaruje se u glavnoj metodi i pokazuje dvosmjerni odnos više prema mnogo.

5.1.3 AGREGACIJA



Agregacija je poseban oblik asocijacije. To je jednosmjerna asocijacija. Na primjer, odjeljenje može imati studente, ali obrnuti slučaj nije moguć. U agregaciji oba slučaja mogu opstati nezavisno jedan od drugog, pa završetak jednog entiteta neće uticati na drugi entitet.

Dva agregirana objekta imaju svoj vlastiti životni ciklus, ali jedan od objekata ima vlasnika Has-A odnosa i podređeni objekt ne može pripadati drugom roditeljskom objektu.

Drugim riječima, kada dva agregirana objekta imaju svoj vlastiti životni ciklus (tj. nezavisni životni vijek), ali je jedan od objekata vlasnik Has-A odnosa, to se u Javi naziva agregiranjem. Vlasnički objekt naziva se agregirajući objekt, a njegova klasa naziva se agregirajuća klasa. Klasa agregiranja ima referencu na drugu klasu i vlasnik je te klase.

Imati vlastiti odnos znači da uništavanje jednog objekta neće utjecati na drugi objekt. Agregacija se koristi za ponovnu upotrebu koda ako ne postoji Is-A odnos.

Na primjer, biblioteka ima učenike. Ako biblioteka bude uništena, učenici će postojati bez biblioteke.

```
public class Employee {
    private String name;
    private double salary;

    public Employee() {}
}

public class Department {
    public String name;
    public List<Employee> employees;
```

```
    public Department() {}
}
```

Na primjer, odnos između zaposlenika i odjela. Zaposleni može stajati sam bez odjela, pa tako i odjel. .Klasa Odeljenje ima odnos sa klasom Employee. Objekt odjela može imati listu zaposlenih ili uopšte ne imati. Objekt zaposlenika može pripadati objektu odjela ili ne. Nema ograničenja u odnosu agregacije.

Najčešći primjer agregiranja odnosa je „Učenik ima adresu“. Učenik ima mnogo informacija kao što su ime, broj liste, imejl id, itd. Takođe sadrži još jedan važan objekat pod nazivom "adresa" koji sadrži informacije kao što su grad, država, država, poštanski broj.

U ovom primjeru programa, klasa Student ima objekt klase Address gdje objekt adrese sadrži vlastite informacije kao što su grad, država, država, itd. Ovaj odnos je Student ima-A adresu i naziva se agregacija.

```
public class Address {

String city, state, country;

int pinCode;



public Address(String city, String state, String country, int pinCode) {

  this.city = city;

  this.state = state;

  this.country = country;

  this.pinCode = pinCode;

  }

}
```

A student Has-A an address. Therefore, the Student class must be able to receive address information as follows:

```
public class Student {

 String name;

 int rollNo;

 Address address;

 int pinCode;



public Student(String name, int rollNo, Address address) {

 this.rollNo = rollNo;

 this.name = name;

 this.address=address;

}

void display(){

 System.out.println("Name: " +name + ", "+"Roll no: " +rollNo);

 System.out.println("Address:");

 System.out.println(address.city+" "+address.state+" "+address.country+ " "
+address.pinCode);

 System.out.println("\n");

}

public static void main(String[] args)

{

 Address addr1 = new Address("Dhanbad,","Jharkhand,","India,", 826001);

 Address addr2 = new Address("Ranchi,","Jharkhand,","India,", 825001);
```

```
 Student st1 = new Student("Deep", 05, addr1);
 Student st2 = new Student("John", 02, addr2);


  st1.display();
  st2.display();
 }
}
```

Output:

```
    Name: Deep, Roll no: 5
    Address:
    Dhanbad, Jharkhand, India, 826001


    Name: John, Roll no: 2
    Address:
    Ranchi, Jharkhand, India, 825001
```

Kompletan aplikativni program se zove Agregacija. Klasa Student sadrži referencu na adresu klase čija instanca postoji i dostupna je izvan Studenta, možemo reći da je Student agregacija adrese. Ako je Student agregacija Address, možemo reći da je Studentski objekat “Has-A” Address objekat.

5.1.4 RAZLIKA IZMEDJU ASOCIJACIJE I AGREGACIJE

I asocijacija i agregacija su osnovni koncepti OOP-a i predstavljaju Has-A odnos u Javi. Ali postoje i određene važne razlike između povezivanja i agregacije koje su sljedeće:

1. Asocijacija uspostavlja odnos između dve klase koje su nezavisne jedna od druge, dok agregacija uspostavlja odnos vlasništva između dve klase.

2. Udruženje nema vlasnika, dok agregacija ima vlasništvo nad udruženjem.

3. Asocijacija može imati odnos kao što je jednosmjeran/dvosmjeran, jedan-prema-jedan, jedan-prema-više, više-prema-jedan i više-prema-mnogo, dok agregacija ima jednosmjeran odnos.

5.1.5 KOMPOZICIJA

Kompozicija je ograničeni oblik agregacije u kome su dva entiteta zavisna jedan od drugog i sastavljeni objekat ne može postojati bez drugog entiteta. Drugim riječima, to je restriktivniji i snažniji oblik agregacije. Dva kompozitna objekta ne mogu imati vlastiti životni ciklus. To jest, kompozitni objekat ne može postojati sam po sebi. Ako je jedan složeni objekat uništen, svi njegovi dijelovi se također brišu.

Java kompozicija se razlikuje od agregacije po tome što agregacija predstavlja Has-A odnos između dva objekta koji imaju svoj životni vijek, ali kompozicija predstavlja Has-A odnos koji sadrži objekt koji ne može postojati sam.

Drugim riječima, dječji objekat nema svoj vijek trajanja. Ako je nadređeni objekt uništen, potom će se uništiti i podređeni objekt. Podređeni objekat ne može postojati bez postojanja svog nadređenog objekta. Roditeljski objekt sadrži podređeni objekt i podređeni objekt ne može postojati sam bez postojanja nadređenog objekta, to se u Javi zove kompozicija.

To se može postići korištenjem varijable instance koja upućuje na drugi objekt.

Primjer kompozicije u realnom vremenu u Javi:

Živimo u kući. Kuća može imati više soba. Ali ne postoji samostalan život sobe i ne može se povezati s dvije različite kuće. Ako uništimo kuću, soba će biti automatski uništena. Dakle, možemo reći da je soba DIO kuće.

Na primjer, odnos između laptopa i njegovog procesora. Laptop ne može postojati bez svog procesora. Klasa Laptop ima odnos sa klasom Procesor. Kada se kreira laptop objekat, automatski se kreira objekt procesora koji pripada tom laptopu.

```
public class Processor {
    private String modelName;
    private int frequency;

    public Processor () {}
}

public class Laptop {
   private Processor processor = new Processor();

   public Laptop () {}
}
```

5.1.6 KARAKTERISTIKE KOMPOZICIJE U JAVI

Kompozicija predstavlja odnos ima-odnos u Javi. Drugim riječima, predstavlja odnos DIO-OD. To je restriktivniji oblik agregacije. U sastavu, oba entiteta su povezana jedan s drugim.

Kompozicija između dva entiteta se dešava kada objekat (drugim rečima, roditeljski objekat) sadrži sastavljeni objekat (drugim rečima, podređeni objekat), a sastavljeni objekat ne može postojati bez postojanja tog objekta.

Na primjer, biblioteka sadrži više knjiga o istim ili različitim temama. Ako se biblioteka iz bilo kojeg razloga uništi, sve knjige koje se nalaze u toj biblioteci bit će automatski uništene. Odnosno, knjige ne mogu postojati bez biblioteke škole ili fakulteta.

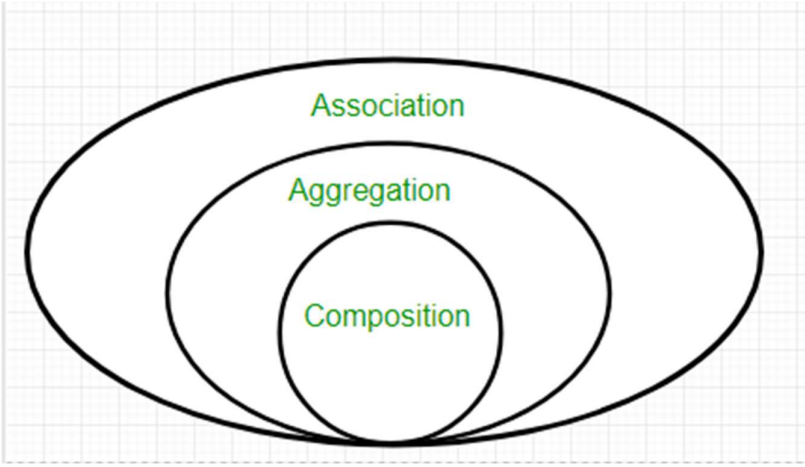
Može se postići korištenjem varijable instance koja se odnosi na druge objekte.

5.1.7 AGREGACIJA NASPRAM KOMPOZICIJE

Agregacija predstavlja odnos gdje dijete može postojati nezavisno od roditelja- banka i zaposleni. Ako izbrišemo banku, zaposleni i dalje postoji. Agregacija je slaba asocijacija.

Kompozicija predstavlja odnos gdje dijete ne može postojati nezavisno od roditelja. NPR. Čovjek i srce ne mogu postojati jedno bez drugog. Kompozicija je jaka asocijacija.

5.1.8 RAZLIKA IZMEDJU ASOCIJACIJE, AGREGACIJE I KOMPOZICIJE



Association	Aggregation	Composition
1. Association established the relation between two classes that is independent of each another.	1. Aggregation defines a special form of unidirectional association between two classes.	1. Composition represents a special and more restrictive form of aggregation where an object cannot exist on its own.
2. In association, there is no owner relationship.	2. In aggregation, one of the objects is the owner of the Has-A relationship.	2. In composition, both the entities are associated with each other and cannot exist on their own.
3. Association defines the relationships as one-to-one, one-to-many, many-to-one, and many-to-many.	3. It defines only a unidirectional relationship.	3. It represents an exclusive whole-part relationship.

1. Asocijacija je odnos između dvije klase koje su nezavisne jedna od druge. U ovoj relaciji ne postoji vlasnički odnos. Asocijacija definiše odnose kao jedan-prema-jedan, jedan-prema-više, mnogo-prema-jedan, i mnogo-prema-više.

2. Agregacija definiše poseban oblik jednosmjerne asocijacije između dvije klase. U agregaciji, jedan od objekata je vlasnik Has-A odnosa. Definiše samo jednosmjerni odnos.

1. Kompozicija predstavlja poseban i restriktivniji oblik agregacije gdje objekt ne može postojati sam.U sastavu, oba entiteta su povezana jedan s drugim i ne mogu postojati sami. Predstavlja ekskluzivni odnos cijeli dio.

6 USES-A RELATIONSHIP

U ovoj vrsti odnosa metoda jedne klase koristi objekat druge klase, a odnos između ove dve klase poznat je kao USES-A relacija. Kada kreiramo objekat klase unutar metode druge klase, ovaj odnos se u Javi naziva odnosom **zavisnosti**, ili jednostavno relacija Uses-A. To je najočitiji i najopštiji odnos u Javi. Ako nekoliko klasa aplikacijskog programa zavise jedna od druge, onda kažemo da je povezanost između klasa visoka. Dobra je programska praksa da se minimizira zavisnost između klasa (tj. spajanje) jer previše zavisnosti otežava upravljanje aplikacijskim programom. S druge strane, ako postoji malo zavisnosti između klasa, onda kažemo da je povezanost između klasa niska.

Ako klasa promijeni svoje ponašanje u sljedećem izdanju aplikacijskog programa, sve klase koje ovise o njoj također mogu biti pogođene. U ovoj situaciji, moramo ažurirati sve povezane klase.

Ako će veza između klasa biti niska, lako možemo upravljati njima. Zato moramo ukloniti nepotrebno spajanje između klasa.

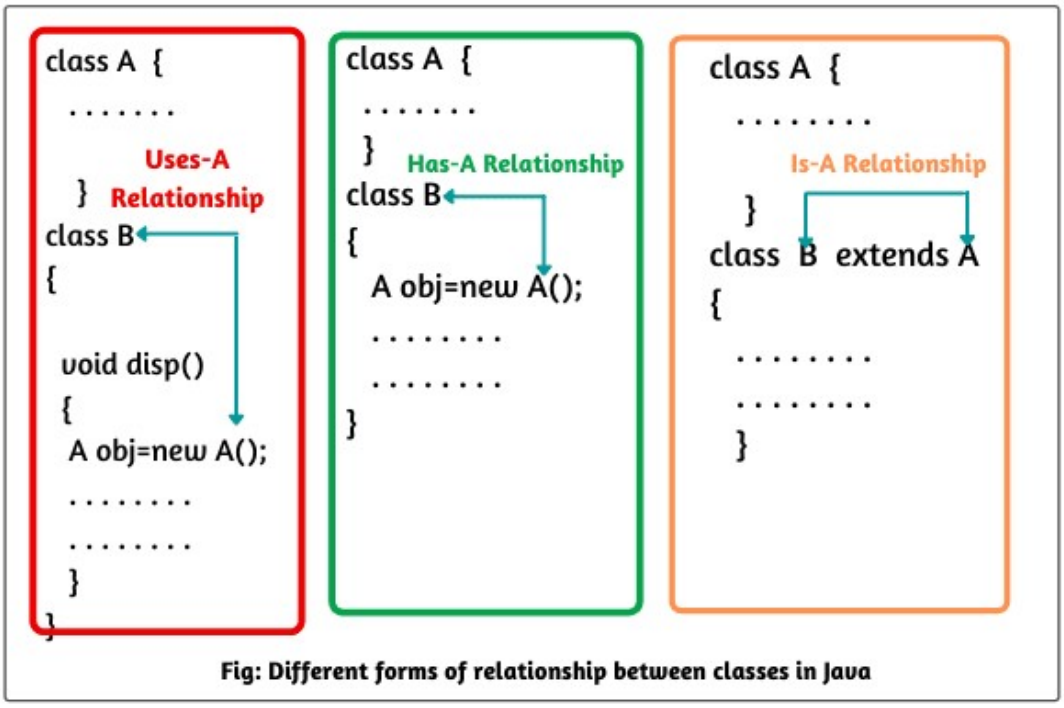
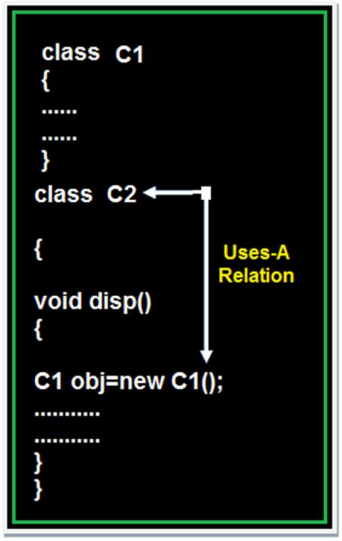


Fig: Different forms of relationship between classes in Java



Dependency relationship, ili odnos zavisnosti, ukazuje na to da jedna klasa ili metoda zavisi od druge klase ili metode. To znači da prva klasa ili metoda koristi drugu klasu ili metodu u svom kodu.

U Java programskom jeziku, postoji nekoliko različitih tipova odnosa zavisnosti:

- Zavisnost između dve klase: Ovo se odnosi na situaciju u kojoj jedna klasa koristi drugu klasu u svom kodu, kao što je pozivanje metode iz druge klase ili korišćenje atributa druge klase.
- Zavisnost između interfejsa i implementacije: U Java-i, interfejsi su skupovi metoda koje se trebaju implementirati u klasama koje ih implementiraju. Kada jedna klasa implementira interfejs, ona postaje zavisna od tog interfejsa i mora da implementira sve metode koje su navedene u interfejsu.
- Zavisnost između paketa: U Java-i, paketi su grupe klasa i interfejsa koje se koriste zajedno. Kada jedna klasa ili interfejs zavisi od drugog paketa, ona mora da navede ime tog paketa u svom import naredbenom redu kako bi mogla da koristi klase i interfejse u tom paketu.

Injekcija zavisnosti (DI) je koncept u kojem objekti dobijaju druge potrebne objekte izvana. DI se može implementirati u bilo kojem programskom jeziku. Opšti koncept koji stoji iza injekcije zavisnosti naziva se inverzija kontrole.

Java klasa zavisi od druge klase, ako koristi instancu ove klase. Ovo nazivamo zavisnošću od klase. U idealnom slučaju Java klase treba da budu što je moguće nezavisnije od drugih Java klasa. Ovo povećava mogućnost ponovne upotrebe ovih klasa i mogućnost testiranja nezavisno od drugih klasa.

Uses-a relacija, ili odnos korišćenja, ukazuje na to da jedna klasa ili metoda koristi neke od funkcionalnosti druge klase ili metode u svom kodu. To znači da prva klasa ili metoda zavisi od druge klase ili metode, ali samo u smislu da koristi neke od njenih funkcija, a ne da se potpuno oslanja na nju.

Uses-a relacija se razlikuje od odnosa zavisnosti, koji ukazuje na to da jedna klasa ili metoda zavisi od druge u potpunosti. Na primer, ako jedna klasa koristi neke od metoda druge klase samo da bi izvršila neke operacije, onda postoji uses-a relacija između njih, ali ne i odnos zavisnosti.

Uses-a relacija se često koristi u slučajevima kada je potrebno da se koriste neke od funkcionalnosti neke klase bez potrebe da se klasa u potpunosti instancira ili nasleđuje. To može biti korisno u situacijama u kojima je potrebno da se koriste samo neke od metoda neke klase, bez potrebe da se koristi cela klasa.

Ukupno, uses-a relacija je važan koncept u Java-i jer omogućava da se klase i metode povežu i koriste uzajamno u svom kodu, što je neophodno za razvoj složenih aplikacija.

Ako Java klasa kreira instancu druge klase preko operatora new, ona se ne može koristiti (i testirati) nezavisno od ove klase i to se zove čvrsta zavisnost.

Zamislimo da imamo dve klase - Fakultet i Student. Klasa Student ima atribut ime i metode getIme() i setIme(String ime). Klasa Fakultet ima atribut studenti koji predstavlja listu studenata i metodu dodajStudenta(Student student) koja dodaje studenta u listu. Međutim, klasa Fakultet ne zavisi od klase Student u potpunosti, već samo koristi neke od njenih metoda da bi postavila ili dobila ime studenta.

```
class Student {

  private String ime;


  public String getIme() {

    return ime;

  }


  public void setIme(String ime) {

    this.ime = ime;

  }

}
```

```
class Fakultet {
  private List<Student> studenti;
  public void dodajStudenta(Student student) {
    studenti.add(student);
  }
}
```

Ovde možemo videti da klasa Fakultet koristi metode getIme() i setIme(String ime) iz klase Student da bi postavila ili dobila ime studenta, ali ne zavisi od klase Student u potpunosti. To znači da postoji uses-a relacija između ovih dve klase.

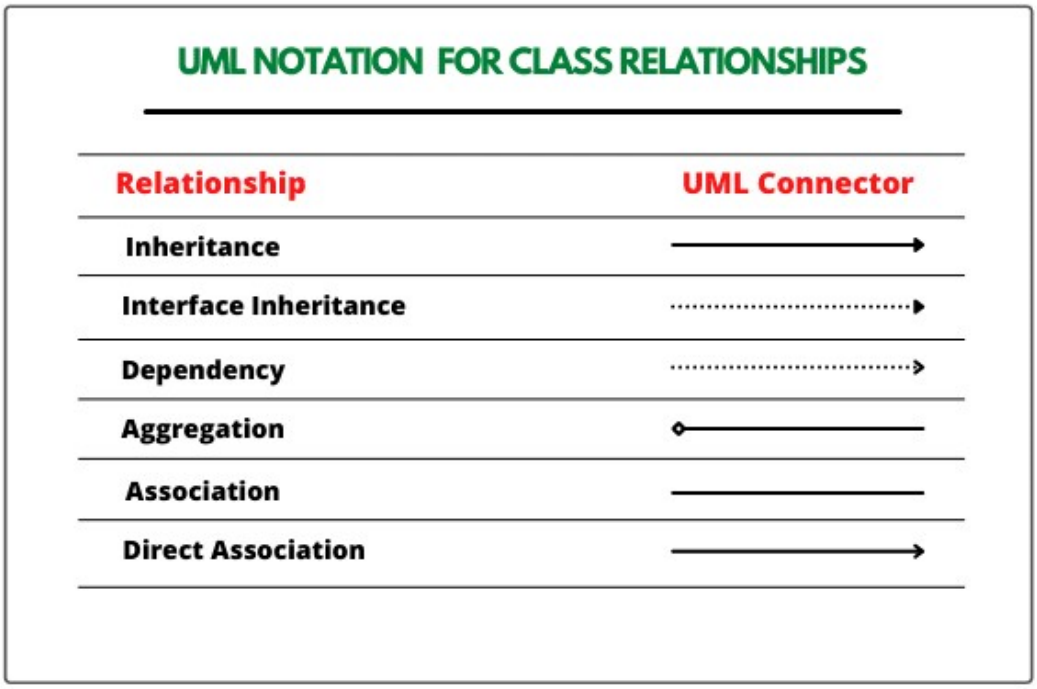
Primjer za USES-A relaciju:

```
class Employee
{
float salary=30000;
}
class Salary extends Employee
{
void disp()
{
float bonous=1000;
Employee obj=new Employee();
float Total=obj.salary+bonous;
System.out.println("Total Salary is:"+Total);
}
}
class Developer
{
public static void main(String args[])
{
Salary s=new Salary();
s.disp();
}
}
```

```
Total Salary is: 31000.0
```

7 UML NOTACIJA U JAVI

UML je skraćenica od Unified Modeling Language. To je međunarodna standardna notacija. Mnogi programeri koriste ovu notaciju za crtanje dijagrama klasa koji objašnjava odnos između klasa.



Postoji veliki broj dostupnih alata za crtanje UML dijagrama. Tabela na donjoj slici prikazuje UML notaciju za odnose klasa u Javi.

8 ZAKLJUČAK

Fokus mog rada bile su relacije, najviše relacije između klasa u Java programskom jeziku. Cilj je bio da objasnim šta su to relacije, koje vrste relacija postoje, način na koji se one implementiraju u programima i sve to da potvrdim kroz konkretne primjere.

Ono što smo mogli zaključiti je da postoji velik broj različitih relacija i da ne postoji najbolja i univerzalna relacija u svim situacijama. Tip relacije zavisi od toga kakve odnose želimo da modelujemo između elemenata u našem java programu, i koje informacije želimo da prikažemo. Ukoliko odaberemo pravu relaciju, velike su vjerovatnoće da ćemo razviti strukturiran i dobro održiv java kod.

Kako odlučiti koja nam je vrsta veze potrebna?

Znamo da odnosi između objekata čine sve razlike u objektno orijentiranom programiranju. Najvažniji odnosi su veza IS-A i HAS-A.

Najbolji način da odlučimo kakva nam je veza potrebna je sljedeći:

a. Ako naš problem sadrži frazu ” . . . . je . . . .”. tada bismo trebali koristiti Is-A odnos (nasljeđivanje). Na primjer, "Pas je kućni ljubimac". Ne možemo reći „Pas ima kućnog ljubimca“, jer nema smisla. Dakle, u ovom slučaju ćemo kreirati superklasu pod nazivom Pet i izvedenu podklasu pod nazivom Dog.

b. S druge strane, ako je naš problem sa frazom poput ove: “Kućni ljubimac ima ime” onda koristimo odnos HAS-A odnos. To je zato što ako koristimo odnos IS-A umjesto odnosa HAS-A, onda će izjava biti ovakva: “Pas je ime”. Ova izjava nema nikakvog smisla. Iz svih gore navedenih primjera možemo zaključiti koja nam je vrsta odnosa potrebna u našem kodu.

9 LITERATURA

- https://dev.to/fajarzuhrihadiyanto/types-of-relation-between-classes-in-object-oriented-programming-551m
- https://thecitadelpub.com/object-oriented-programming-in-java-relationships-between-classes-7b217fae9172
- https://www.scientecheasy.com/2021/03/association-vs-aggregation-vs-composition.html/
- https://hr.myservername.com/top-10-best-free-antivirus-software
- https://www.youtube.com/watch?v=rkQ7NNVc5pk
- https://www.vogella.com/tutorials/DependencyInjection/article.html