(27) Has Relationship and dependency injection

↳ Nithin Sir
↳ 16 Nov live class.

① Relationship between 2 classes

ⓐ is-A relationship
ⓑ Has-A relationship.

ⓐ **is-A relationship**:- Whenever one class inherits another class, it is called an **is-A relationship**.

ⓑ **Has-A relationship**:- Whenever an instance of one class is used in another class, it is called **Has-A Relationship**.

Eg①:- Car has Engine

```
Class Engine { // Dependent object

}
Class Car {   // target object.

   // Has relationship
   // Engine Engine = new Engine();

   Engine engine; // Instance Variable

}
```

↳ Another class reference should be part of this class reference.

→ Engine object created and it should be injected to Car object.

→ ② Dependency Injection

The process of Injecting dependent object into target object technically we call it as **Dependency Injection**.

Eg:-
```
Class Engine  → Dependent object
{

}
Class Car     → Target object
{
   Engine Engine; // Instance Variable   → Has-A relationship!
}                 ↳ Variable declaration.
```

→ I need Engine object in target object (Car)

Engine - Dependent object
Car → Target object

Eg②:- class address { // Dependent object

}

class Student { // Target object.

// Has-A relationship

Address address; // Instance Variable

}

Eg③:- Class Account { // Dependent object

}

Class Empolyee { // Target object

// Has-A relationship

Account account; // Instance Variable

}

→ We can initialize Instance Variable in inside the Constructor and Setter.

→ ~~We can achieve~~

→ We can achieve dependency Injection in 2-ways

ⓐ Constructor dependency Injection

ⓑ Setter dependency Injection

ⓐ Constructor dependency Injection :-

Injecting dependent object into target object through Constructor is called as "Constructor dependency injection".

ⓑ Setter dependency Injection :-

Injecting dependent object into target object through a Setter is called "Setter-dependency Injection".

Notes → given in class.

Relationships in Java

① As Part of Java application development, we have to use Entities (class) a per the application requirements.

② In Java application development, if we wan to Provide optimization over memory utilization, Code, Reusability, Execution Time, Sharability then we have to define relationships between Entities.

→ There are three types of relationships between Entities (classes)

① Has-A Relationship (Extensively used in Project)

② Is-A relationship (achieve using extends keyword).

③ USE-A relationship (not popular).

① What is the difference between HAS-A Relationship and is-A Relationship.

(i) Has-A relationship will define associations between Entities in Java applications, here associations between Entities will improve Communication between Entities and data navigation between Entities.

(ii) Is-A Relationship is able to define inheritance between any Entity class, it will improve "Code Reusability" in Java applications.

Associations in Java

There are four types of association between Entities (class)

① one - To - one association (1:1)
② One - To - Many Association (1:M)
③ Many - To - one Association (M:1)
④ Many - To - Many Association (M:M)

Association: it means relating two classes in Has-A style we call it as Association.

→ Establishing relationship between two classes through Has-A relationship is called Association.

→ To achieve association between Entities (class), We have to declare either single reference or array of reference variable of an Entity (class) in another Entity class.

```
class Address {

}
class Account {

}
class Employee {
    Address [] addr;  // it will Establish One-Two Many Association (1:M)
    Account account;  // One-to-One Association (1:1)
}
```

→ Employee has one Salary account.
→ Employe has Multiple addresses. So it is array.
↳ one to many association.

① Primitive Value Code Injection :-

```
Package in.ashish.main
Public class student {
  // Instance Variables
  Private String name;
  Rivate Integer age;
  Rivate Integer sid;
```

→ here datatypes are object-type

// two ways we can set values for Instance Variable.

① Constructor to set a value → Right click → Source → generate Constructor using fields

```
Public Student(String name, integer age,
                integer sid) {
  super();
  this.name = name;
  this.age = age;
  this.sid = sid;
}
```

② Setter and Getter Injection → Right-click → source → generate Setter & getter

```
Public string getName() {
    return name;
}
Public string SetName(string name)
{
    this.name = name;
}
```

```
Public Integer getAge() {
  return age;
}
Public Void setAge(IntegerAge)
{
   this.age = age;
}
Public Integer getsid() {
    return sid;
}
Public Void setsid(Integer sid)
{
   this.sid = sid;
}
```

// we have to override to get info from to string Method. → Right click → Source → generate to String

```
@overide
Public String toString() {
  return "Student [name=" "+name+",
          age = "+age+",
          sid = "+sid + "]";
}
```

```
Package in.ashish.main;
Public Static Void main(String[] args) {
    Student student = new student("ashish",50,10);
    S.o.P(student);
} }
```

→ ~~object~~

→ Java is object oriented programming language
So better to use object type.

→ if you don't ~~Call~~ overite to string Method it Calles object to string Method.

→ object to string Method will never print value associated with instance variables.

Code :-

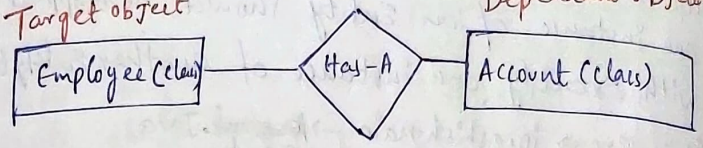
```
Class Address{

}
Class Account{

}
class Employee{
Address[] add;
Account account;
}
```

→ Every Empolyee should have only account
↳ one-to-one association.
→ Empolyee has multiple addresses

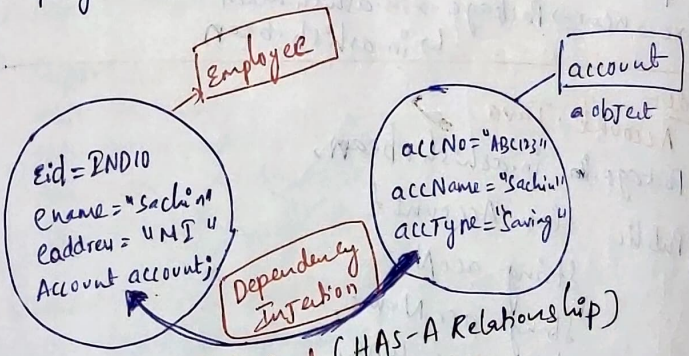
Memory Map :- One-To-One-Association (1:1) Code ⇒

#2 Target object

#1 Dependant object



→ dependent object should Coded before Target object



↳ Linked (HAS-A Relationship)
↳ here account & Employee are linked.
↳ Two ways we can link classes by using
① Constructors (In Code we use constructor)
② Setter and getters

→ Account object injected into Employee object (Target object).

→ Try using Setter and getter.

→ Data-navigation :- being in Employee object we are getting ~~dat~~ account data.

① One-To-one Association :-

It is a relation between Entities, where one instance of an Entity should be mapped with Exactly one instance of another Entity (Cla

Eg :- Src → in.ashish.main → ~~Account~~ Test.Java
↳ in.ashish.beam → Account.Java
→ Employee.Java

→ Src → new → Package → in.ashish.main
↳ in.ashish.beam

Code :-

Account.Java

```
Package in.ashish.beam;
Public class Account{
        String accNo;
        String accName;
        String accType;
  //Injecting Values through Constructor
  Public Account(String accNo, String accName,
                                String accType){

    Super();
    this.accNo = accNo;
    this.accName = accName;
    this.accType = accType;
  }
}
```

Data Navigation :- getting data from account object.
In this Example.

Employee.java

```
Package in.ashish.beam;
Public class Employee{
  Private String eid;
  Private String eName;
  Private String eaddr;
  // Has-A-Relationship
  Account account;
  // Account is injected into Constructor.
  Public Employee(String eid, String eName, String eaddr, Account
  {
    Super();
    this.eid = eid;
    this.eName = eName;
    this.eaddr = eaddr;
    this.account = account;
  }
  //Method to get Employe details
  Public Void getEmployeeDetails()
  {
    S.o.p(eid);
    S.o.p(eName);
    S.o.p(eaddr);
    S.o.p(account.accNo);
    S.o.p(account.accName);
    S.o.p(account.accType);
  }
}
```

→ being in employee Class we are getting data from account Object.

↳ Data Navigation

Constructor Injection

Test.Java

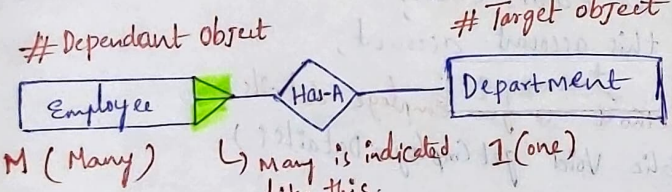
```
Package in.ashish.Main;
Public class TestApp{
  Public static Void Main(String[] args){

    Account account = new Account("ABCD12", "Ashish",
                                  "Saving");
    Employee employee = new Employee("123", "Ashish", "Hyd", account);
    employe.getEmployeeDetails();
  }
}
```

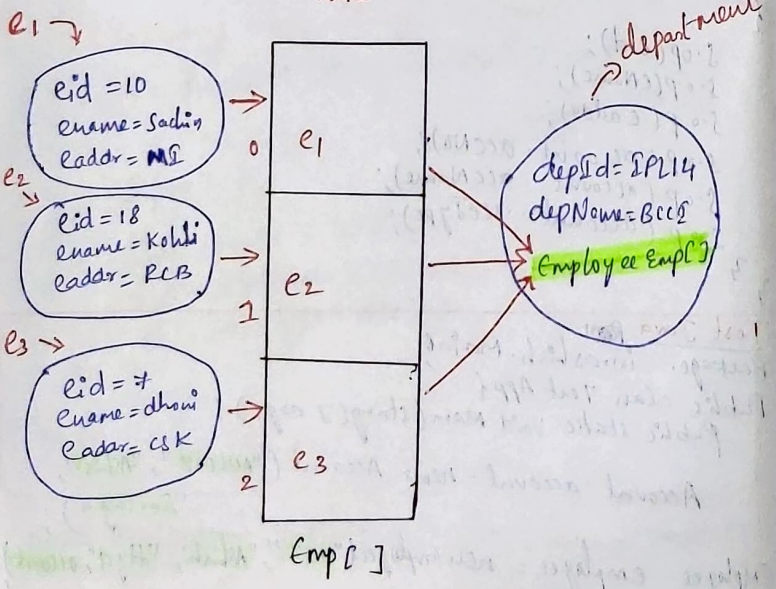
② One-To-Many Association :-

It is relationship between entity classes, where one instance of an entity should be Mapped with Multiple instances of another Entity.

Eg:- Single Department has Multiple Employee

Memory Map :- Employee first they work later on they join in department





Code :- Refer github repo (one-to-Many-association)

one-to-Many → src → in.ashish.bean
↳ Employe.Java
↳ Department.Java
in.ashish.Main
↳ TestApp.Java

① Employe.Java

```
Package in.ashish.bean;
//Dependant object
Public class Employe {
    String eid;
    String ename;
    String eaddr;
// Injecting Values through Constructor.
Public Employee ( String eid, String ename, String addr) {
    Super();
    this.eid = eid;
    this.ename = ename;
    this.eaddr = eaddr;
}
}
```

② Department.Java

```
Package. in.ashish.bean;
Public class Department {
    String did;
    String dname;
// Many Employees Mapped to one department
Private Employee [ ] emps;
```

⇒ Continuation.

// Constructor Injection

```
Public Department (String did, String dname, Employee[] emps) {
    Super ();
    this.did = did;
    this.dname = dname;
    this.emps = emps;
}
// getting Employee details.
Public Void getDepartmentDetails () {
    S.o.p ("Department details");
    S.o.p ("Department id:" + did);
    S.o.p ("Department name:" + dname);
    S.o.p ();
    S.o.p ("Employee Details are");
    for (Employee employee : emps) {
        S.o.p ("Employee id" + employee.eid);
        S.o.p ("Employee Name" + employee.ename);
        S.o.p ("Employee Address" + employee.eaddr);
    }
}
```

③ TestApp.Java

```
Package in.ashish.main;
import in.ashish.bean.Employee;
import in.ashish.bean.Department;
Public class TestApp {
    Public static Void main (String[] args) {
        Employee e1 = new Employee ("10", "Sachin", "MI");
        Employee e2 = new Employee ("18", "Kohli", "RCB");
        Employee e3 = new Employee ("7", "Dhoni", "CSK");
        Employee[] Emps = new Employee[3];
        emps[0] = e1;
        emps[1] = e2;
        emps[2] = e3;
        Department department = new Department ("IPL14", "BCCI", emps);
        department.getDepartmentDetails ();
    }
}
```

output:-

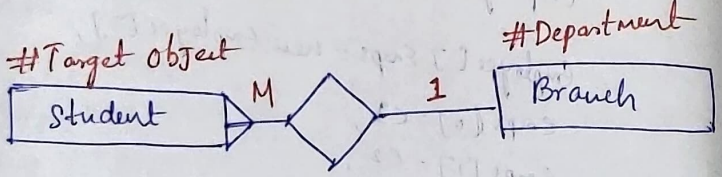
```
Department Details
Department id : IPL14
Depatret name : BCCI
Employ details
Employee Id : 10
    name = Sachin
    addr = MI
    ||
    ||
    :
    :
```

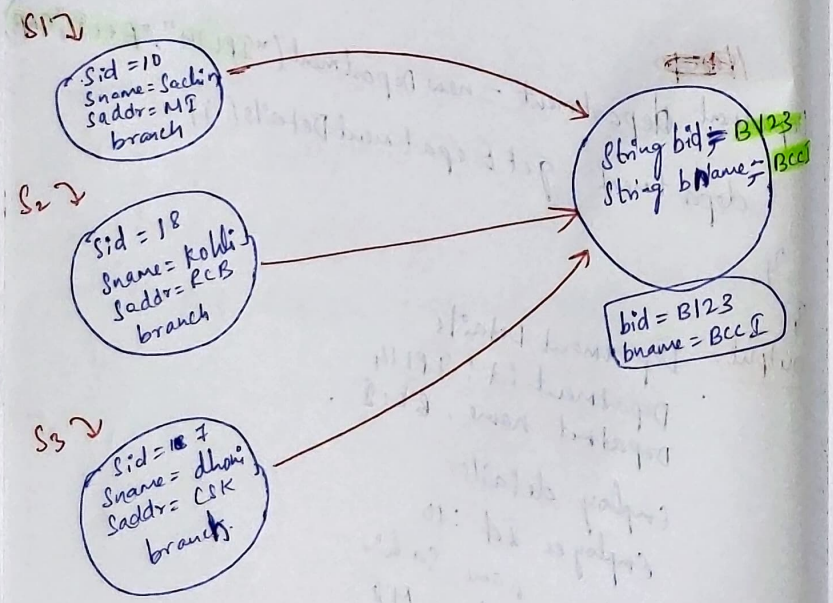
③ Many-To-one Association :-

It is a relationship between entities, where Multiple instances of an entity should be Mapped with Exactly one instance of another Entity.

Eg: Multiple students have joined with a single branch.

→ First branch should be ready bfr then students join in Branch.





Many-to-one Association → src →
- in.ashish.bean
  - Branch.java
  - Student.java
- in.ashish.main
  - TestApp.java

Code: refer github repo

» Branch.java

```
Package in.ashish.bean

Public class Branch {  // Dependent object
        String bId;
        String bname;

// Injecting Values through getters and Setters
  Public String getBid(){
      return bid;
  }
  Public String setBid(String bid){
      this.bid=bid;
  }
  Public string getbName(){
        return bName;
  }
}
```

Student.java (target object)

```
Package. in.ashish.java

Public class Student{
    Private String sId;
    Private String sName;
    Private String sAddr;

// Student are many joined in one branch
    Branch branch;

// injecting through setters and getters
Public String getsId()
{
    return sId;
}
Public Void setsId(String sId)
{
    this.sId = sId;
}
Public String getsName(){
    return sName;
}
Public Voids setsName(String sName){
    this.sName = sName;
}
Public String getsAddr(){
    return sAddr;
}
Public Void setsAddr(String sName){
    this.sAddr = sAddr;
}
Public Branch getBranch(){
    return branch;
}
Public Void setBranch(Brach branch)
{
    this.branch = branch;
}
}
```

TestApp.java (refer git repo)

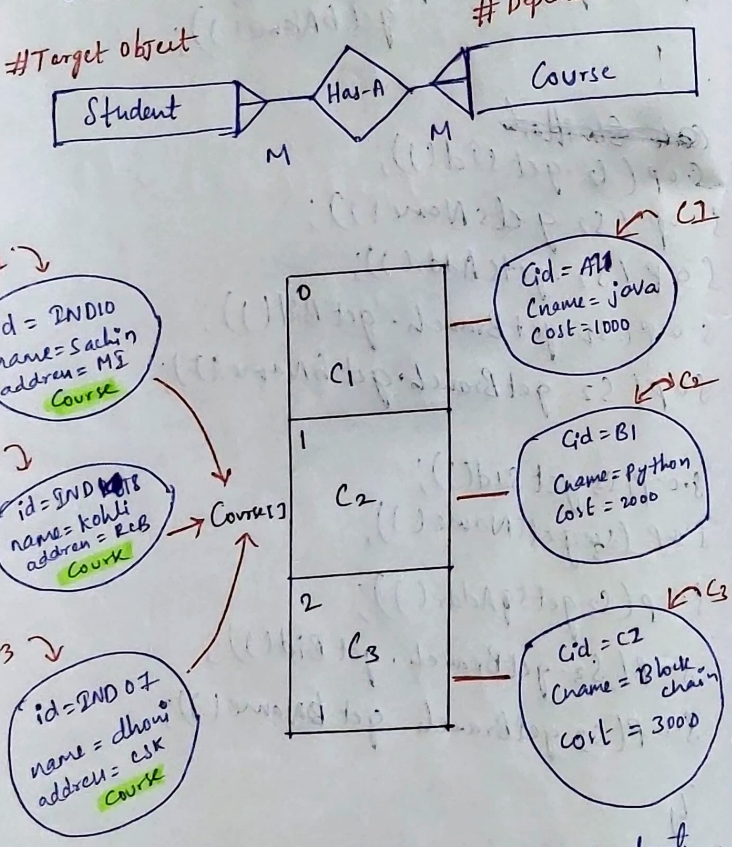
```
Package in.ashish.Main;
Public class TestApp {
    Public static void main(String[] args) {

// Creating branch object
Branch branch = new Branch();
branch.setBid("B123");
branch.getBid("BccI");
// Student 1
Student S1 = new Student();
S1.setId('10');
S1.setsName("ashish");
S1.setsAdar("MI");
S1.SetsBranch(branch);
// Student 2
Student S2 = new student();
S2.setId('18');
S2.setsName("kohli");
S2.setAddr("@RCB");
S2.setsBranch(branch);
// Student 3
Student S3 = new student();
S3.setId("7");
S3.setsName("Dhoni");
S3.setAddr("CSK");
```

```
// get student details
S.o.p(S1.getsId());
S.o.p(S1.getsName());
S.o.p(S1.getsAddr());
S.o.p(S1.getBranch.getBid());
S.o.p(S2.getBranch.getBName());

S.o.p(S2.getsId());
S.o.p(S2.getsName());
S.o.p(S2.getsAddr());
S.o.p(S2.getBranch.getBid());
S.o.p(S2.getBranch.getBName());

S.o.p(S3.getsId());
S.o.p(S3.getsName());
S.o.p(S3.getsAddr());
S.o.p(S3.getBranch.getBid());
S.o.p(S3.getBranch.getBName());

    }
}
```

(4) Many-To-Many Association

It is relationship between Entities, where multiple instances of an Entity should be mapped with multiple instances of another Entity.

Eg: Multiple Students have joined in Multiple Courses.



→ All three Courses given to each Student

→ S1, S2, S3 get Three Courses Each.

→ one-to-one Association is used commonly.

Course.java

```
Package in.ashish.bean;

Public class Course {
    String Cid;
    String Cname;
    int Cost;
    Public Course (String Cid, String Cname, int Cost) {

     Super();
     this.Cid = Cid;
     this.Cname = Cname;
     this.cost = Cost;
    }
}
```

Student.java :

```
Package in.ashish.bean
Public class Student {
    Private String Sid;
    Private String Sname;
    Private String Saddr;
// Has-A-Relationship
    Course [] Course;

// Constructor Injection
// Injecting Values using Constructor
Public Student (String Sid, String Sname, String Saddr,
                                    Course[] Course) {

    Super();
    this.Sid = Sid;
    this.Sname = Sname;
    this.Saddr = Saddr;
    this.Course = Course;

}
```

Continuation ⇒

```
Public Void getStudentDetails( ){
    S.o.p("SID"+ Sid)
    S.o.p( Sname)
    S.o.p( Saddr)
    S.o.p("Course details");
    for (Course C : Course)
    {
      S.o.p(" C.Cid);
      S.o.p ( C.name);
      S.o.p ( C.Cost);
    }
}

}
```

Test App.java

```
Package in.ashish.main

Public class Test App {

        Public static Void main(String[] args){

Course C1 = new Course("A1","Java","1000");
Course C2 = new Course("B1","Python","2000");
Course C3 = new Course("C1","Blockchain","3000");
Course[] Course = new Course[3];
Course[0] = C1;
Course[1] = C2;
Course[2] = C3;

Student S1 = new Student("10","Sachin","MI");
Student S2 = new Student("18","Kohli","RCB");
Student S3 = new Student("7","dhoni","CSK");

// getting all students details
  S1. getStudentDetails( );
  S2.      "          ( );
  S3.      "          ;
}
}
```