object oriented relationships

Inheritance:

Inheritance is “IS-A” type of relationship. “IS-A” relationship is a totally based on Inheritance, which can be of two types Class Inheritance or Interface Inheritance. Inheritance is a parent-child relationship where we create a new class by using existing class code. It is just like saying that “A is type of B”. For example is “Apple is a fruit”, “Ferrari is a car”.

For better understanding let us take a real world scenario.

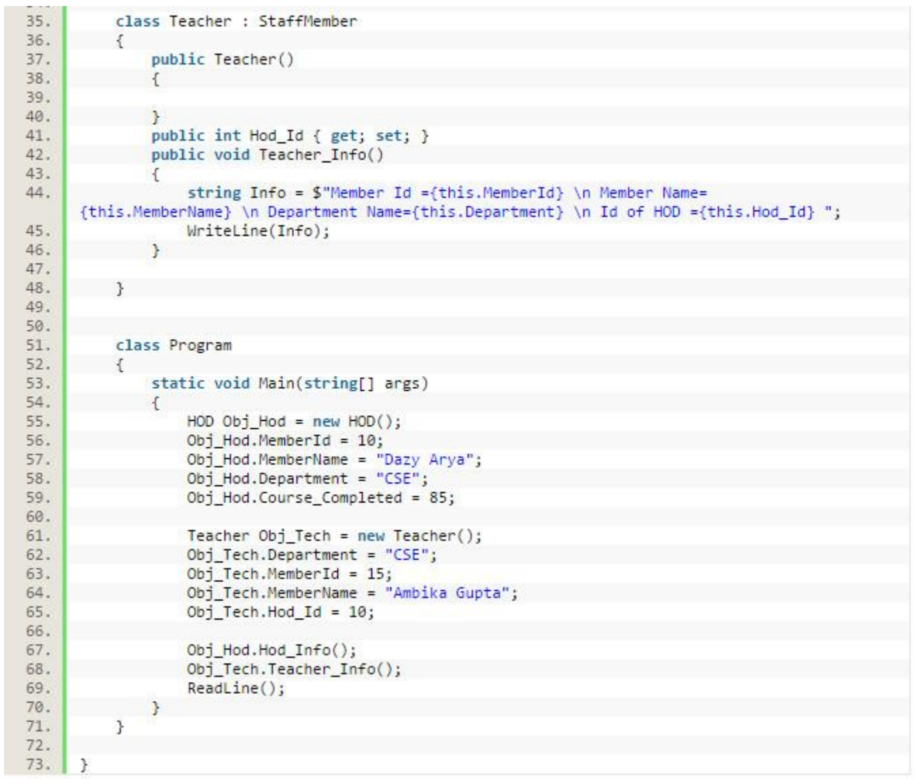
- HOD is a staff member of college.
- All teachers are staff member of college.
- HOD and teachers has id card to enter into college.
- HOD has a staff that work according the instruction of him.
- HOD has responsibility to undertake the works of teacher to cover the course in fixed time period.

Let us take first two assumptions , “HOD is a staff member of college” and “All teachers are staff member of college”. For this assumption we can create a “StaffMember” parent class and inherit this parent class in “HOD” and “Teacher” class.

```
class StaffMember
    {
        public StaffMember()
        {

        }
    }

    class HOD : StaffMember
    {
        public HOD()
        {

        }
    }

    class Teacher : StaffMember
    {
        public Teacher()
        {

        }
    }
```

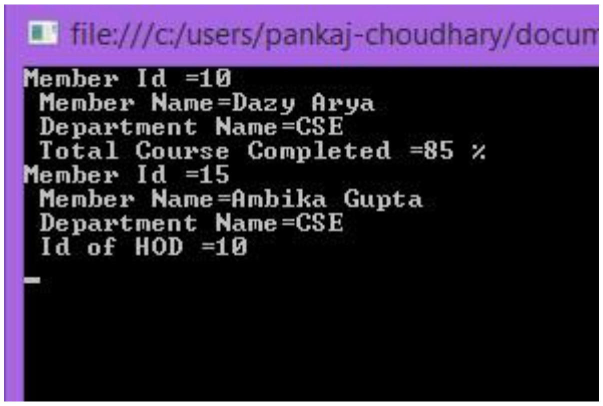
Let us take an example for better understanding.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using static System.Console;

namespace Entity2
{
   class StaffMember
    {
        public int MemberId {get; set;}
        public string MemberName { get; set;}
        public string Department { get; set; }
        public StaffMember()
        {

        }
    }

    class HOD : StaffMember
    {
        public HOD()
        {

        }
        public int Course_Completed { get; set; }

        public void Hod_Info()
        {
            string Info = $"Member Id ={this.MemberId} \n Member Name={this.MemberName} \n Department Name={this.Department} \n Total Course Completed = {this.Course_Completed} %";
            WriteLine(Info);
        }
    }

```



```
    class Teacher : StaffMember
    {
        public Teacher()
        {

        }
        public int Hod_Id { get; set; }
        public void Teacher_Info()
        {
            string Info = $"Member Id ={this.MemberId} \n Member Name={this.MemberName} \n Department Name={this.Department} \n Id of HOD ={this.Hod_Id} ";
            WriteLine(Info);
        }

    }


    class Program
    {
        static void Main(string[] args)
        {
            HOD Obj_Hod = new HOD();
            Obj_Hod.MemberId = 10;
            Obj_Hod.MemberName = "Dazy Arya";
            Obj_Hod.Department = "CSE";
            Obj_Hod.Course_Completed = 85;

            Teacher Obj_Tech = new Teacher();
            Obj_Tech.Department = "CSE";
            Obj_Tech.MemberId = 15;
            Obj_Tech.MemberName = "Ambika Gupta";
            Obj_Tech.Hod_Id = 10;

            Obj_Hod.Hod_Info();
            Obj_Tech.Teacher_Info();
            ReadLine();
        }
    }

}
```

Output

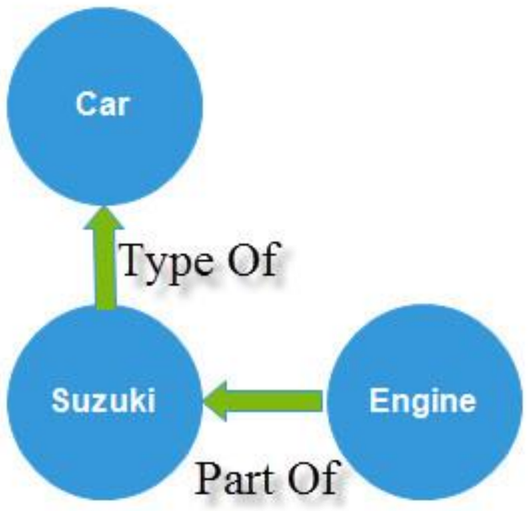


```
Member Id =10
 Member Name=Dazy Arya
 Department Name=CSE
 Total Course Completed =85 %
Member Id =15
 Member Name=Ambika Gupta
 Department Name=CSE
 Id of HOD =10
```

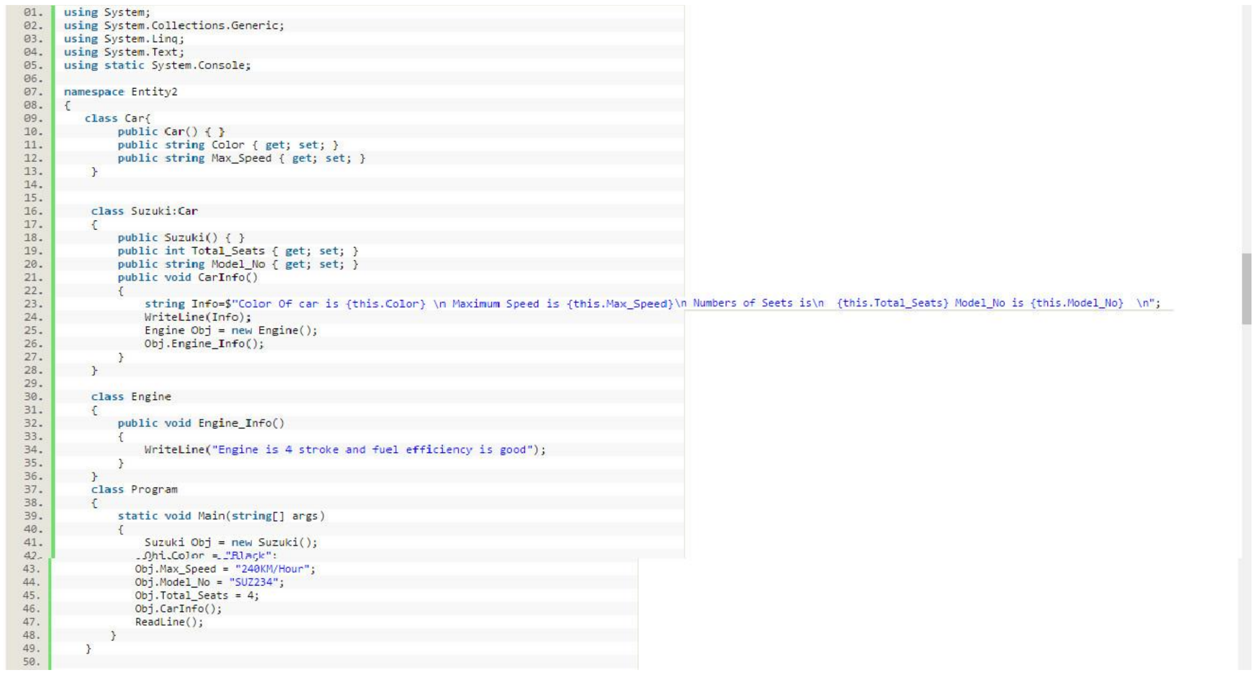
Composition:

Composition is a "part-of" relationship. Simply composition means mean use of instance variables that are references to other objects. In composition relationship both entities are interdependent of each other for example “engine is part of car”, ”heart is part of body”.

Let us take an example of car and engine. Engine is a part of each car and both are dependent on each other.

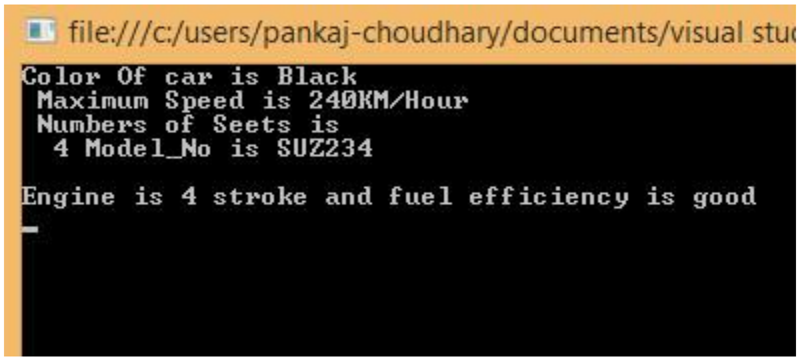


Example



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using static System.Console;

namespace Entity2
{
   class Car{
        public Car() { }
        public string Color { get; set; }
        public string Max_Speed { get; set; }
    }


    class Suzuki:Car
    {
        public Suzuki() { }
        public int Total_Seats { get; set; }
        public string Model_No { get; set; }
        public void CarInfo()
        {
            string Info=$"Color Of car is {this.Color} \n Maximum Speed is {this.Max_Speed}\n Numbers of Seets is\n  {this.Total_Seats} Model_No is {this.Model_No}  \n";
            WriteLine(Info);
            Engine Obj = new Engine();
            Obj.Engine_Info();
        }
    }

    class Engine
    {
        public void Engine_Info()
        {
            WriteLine("Engine is 4 stroke and fuel efficiency is good");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Suzuki Obj = new Suzuki();
           Obj.Color = "Black";
           Obj.Max_Speed = "240KM/Hour";
           Obj.Model_No = "SUZ234";
           Obj.Total_Seats = 4;
           Obj.CarInfo();
           ReadLine();
       }
    }

```

Output

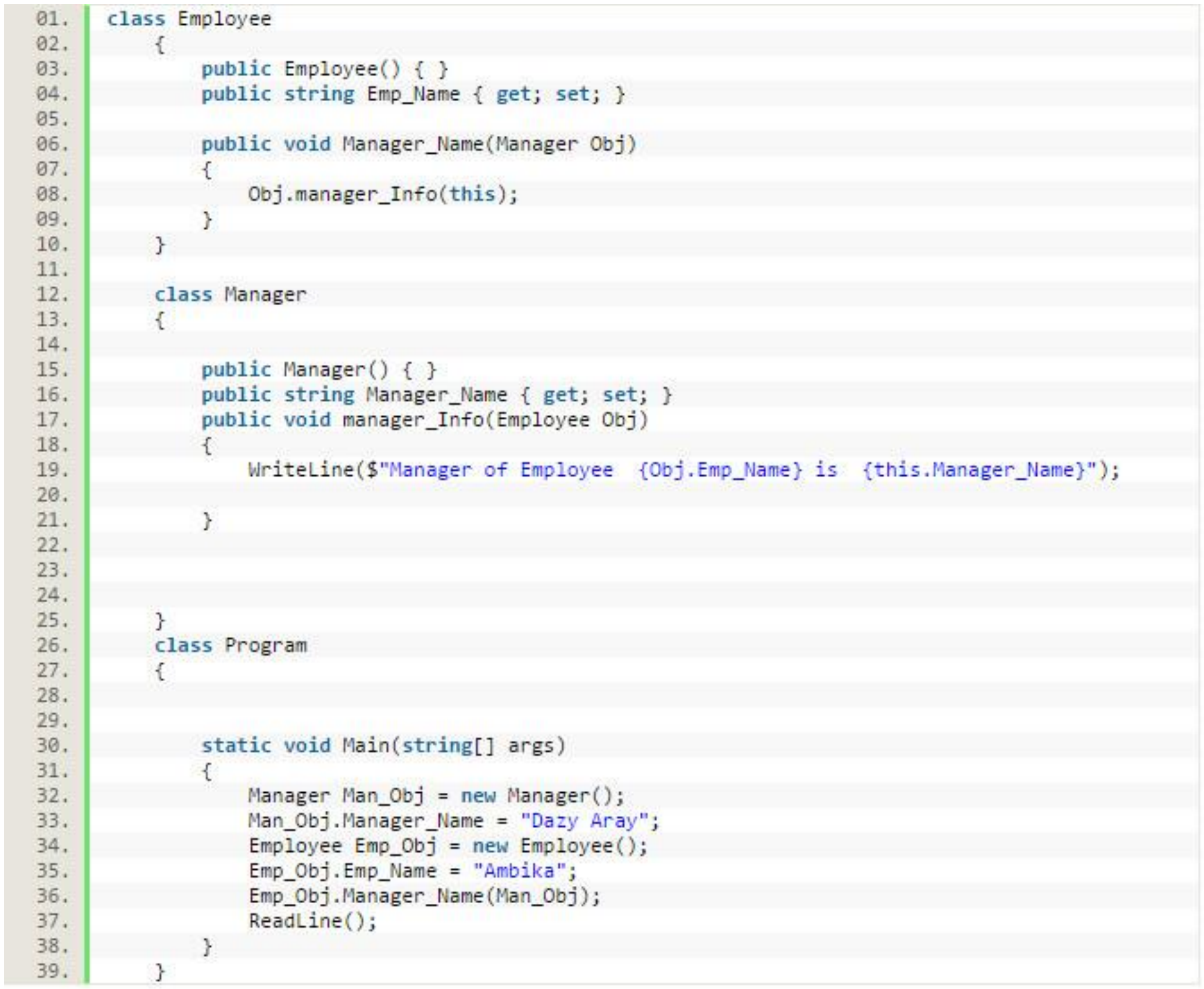


```
Color Of car is Black
 Maximum Speed is 240KM/Hour
 Numbers of Seets is
  4 Model_No is SUZ234

Engine is 4 stroke and fuel efficiency is good
```

Association:

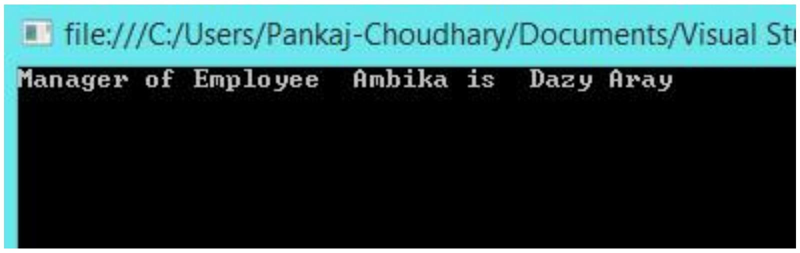
Association is a “has-a” type relationship. Association establish the relationship b/w two classes using through their objects. Association relationship can be one to one, One to many, many to one and many to many. For example suppose we have two classes then these two classes are said to be “has-a” relationship if both of these entities share each other’s object for some work and at the same time they can exists without each others dependency or both have their own life time.

Example



```
class Employee
    {
        public Employee() { }
        public string Emp_Name { get; set; }

        public void Manager_Name(Manager Obj)
        {
            Obj.manager_Info(this);
        }
    }

    class Manager
    {

        public Manager() { }
        public string Manager_Name { get; set; }
        public void manager_Info(Employee Obj)
        {
            WriteLine($"Manager of Employee  {Obj.Emp_Name} is  {this.Manager_Name}");

        }



    }
    class Program
    {


        static void Main(string[] args)
        {
            Manager Man_Obj = new Manager();
            Man_Obj.Manager_Name = "Dazy Aray";
            Employee Emp_Obj = new Employee();
            Emp_Obj.Emp_Name = "Ambika";
            Emp_Obj.Manager_Name(Man_Obj);
            ReadLine();
        }
    }
```

Output



```
Manager of Employee  Ambika is  Dazy Aray
```

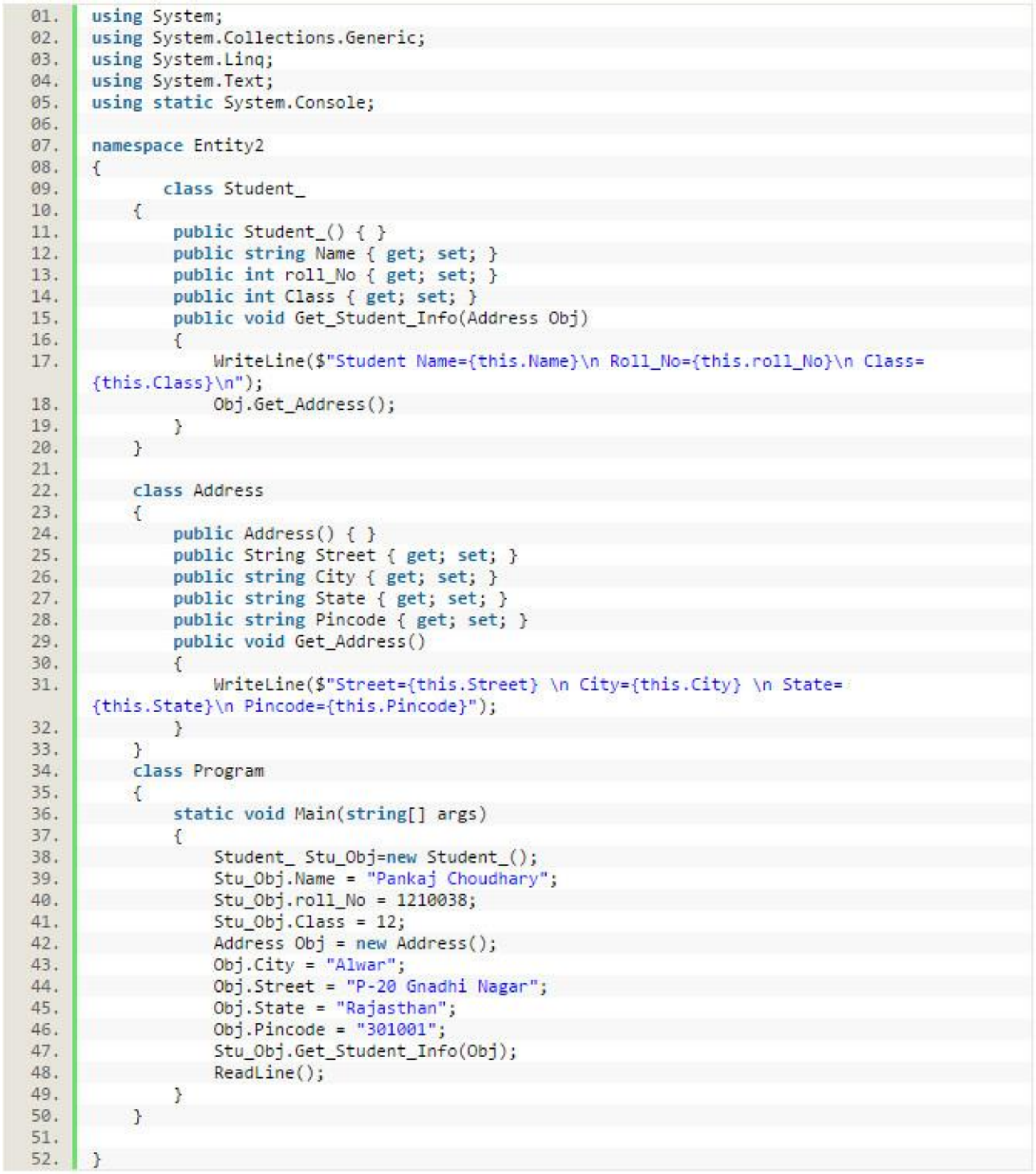
Above example showing an association relationship because both Employee and Manager class using the object of each other and both their own independent life cycle.

Aggregation

Aggregation is based is on "has-a" relationship. Aggregation is a special form of association. In association there is not any classes (entity) work as owner but in aggregation one entity work as owner. In aggregation both entities meet for some work and then get separated. Aggregation is a one way association.

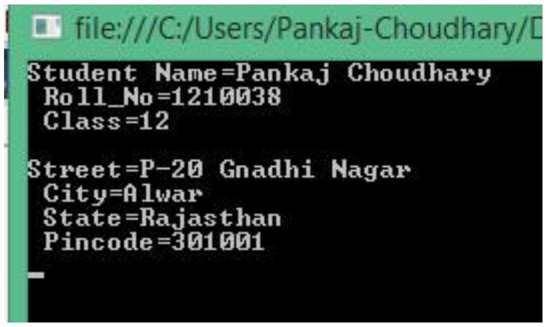
Example

Let us take an example of “Student” and “address”. Each student must have an address so relationship b/w Student class and Address class will be “Has-A” type relationship but vice versa is not true(it is not necessary that each address contain by any student). So Student work as owner entity. This will be a aggregation relationship.



```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using static System.Console;

namespace Entity2
{
       class Student_
    {
        public Student_() { }
        public string Name { get; set; }
        public int roll_No { get; set; }
        public int Class { get; set; }
        public void Get_Student_Info(Address Obj)
        {
            WriteLine($"Student Name={this.Name}\n Roll_No={this.roll_No}\n Class={this.Class}\n");
            Obj.Get_Address();
        }
    }

    class Address
    {
        public Address() { }
        public String Street { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string Pincode { get; set; }
        public void Get_Address()
        {
            WriteLine($"Street={this.Street} \n City={this.City} \n State={this.State}\n Pincode={this.Pincode}");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Student_ Stu_Obj=new Student_();
            Stu_Obj.Name = "Pankaj Choudhary";
            Stu_Obj.roll_No = 1210038;
            Stu_Obj.Class = 12;
            Address Obj = new Address();
            Obj.City = "Alwar";
            Obj.Street = "P-20 Gnadhi Nagar";
            Obj.State = "Rajasthan";
            Obj.Pincode = "301001";
            Stu_Obj.Get_Student_Info(Obj);
            ReadLine();
        }
    }

}
```

Output



```
Student Name=Pankaj Choudhary
 Roll_No=1210038
 Class=12

Street=P-20 Gnadhi Nagar
 City=Alwar
 State=Rajasthan
 Pincode=301001
_
```

OOA

Object-Oriented Analysis (OOA) and Object-Oriented Design (OOD)

Object-Oriented Analysis (OOA) and *Object-Oriented Design* (OOD) are a software design methodology that takes the concept of objects to a higher, more conceptual, level than OOP. The two terms are sometimes combined as Object-Oriented Analysis and Design (OOAD).

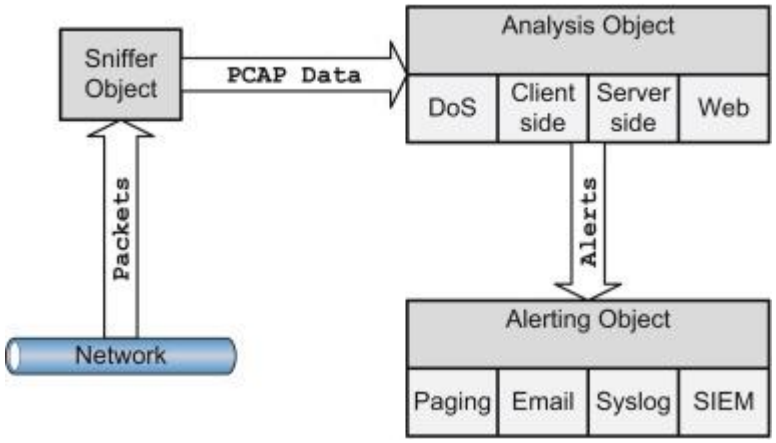
It is like drawing a flowchart on a whiteboard that shows how a program should conceptually operate. The way data in a program flows and is manipulated is visualized as a series of messages and objects. Once the software design is complete, the code may be programmed in an OOP language such as Ruby.

Object-Oriented Analysis (OOA) seeks to understand (analyze) a *problem domain* (the challenge you are trying to address) and identifies all objects and their interaction. Object-Oriented Design (OOD) then develops (designs) the solution.

We will use Object-Oriented Analysis and Design to design a network intrusion detection system (NIDS). As we learned in Chapter 8, Domain 7: Security Operations, a NIDS performs the following actions:

1. Sniffs packets from a network and converts them into pcap (packet capture) format;
2. Analyzes the packets for signs of attacks, which could include Denial of Service, client-side attacks, server-side attacks, web application attacks, and others;
3. If a malicious attack is found, the NIDS sends an alert. NIDS may send alerts via email, paging, syslog, or security information and event managers (SIEMs).

The previous steps serve as the basis for our Object-Oriented Analysis. A sniffer object receives messages from the network in the form of packets. The sniffer converts the packets to pcap (packet capture) data, which it sends to the analysis object. The analysis object performs a number of functions (methods), including detecting denial of service, client-side, server-side, or web application attacks. If any are detected, it sends an alert message to the alerting object. The alerting object may also perform a number of functions, including alerting via email, paging, syslog, or SIEM. The NIDS Object-Oriented Design is shown in Figure 9.10.



This NIDS design addresses the problem domain of alerting when malicious traffic is sent on the network.

OOD

What is Object Oriented Design?

In the object-oriented design method, the system is considered a collection of objects (i.e., entities). The state is shared among the objects, and each object is responsible for its own state data. Tasks designed for a specific purpose cannot refer to or update data from other objects. Objects have internal data that represents their current state. Similar objects form a class. In other words, every object belongs to a class.

OOP

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.