



Aggregation and Composition

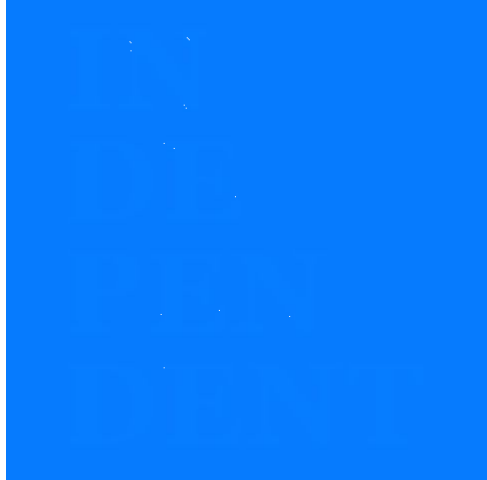


| Problem Scenario

In Real World Scenarios, when objects communicate with each other. There could be **two modes** of the communications.

Problem Scenario 01

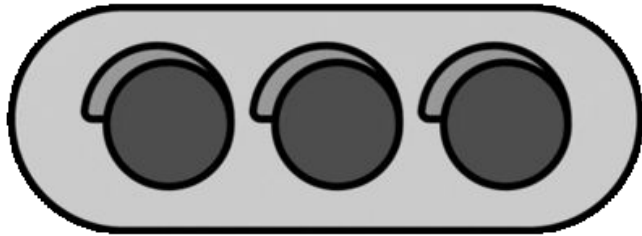
Sometimes **two objects** (instances) of two different classes **exists on their own** and the lifetime of an object does not depend on the lifetime of another object.



Problem Scenario 01: Example

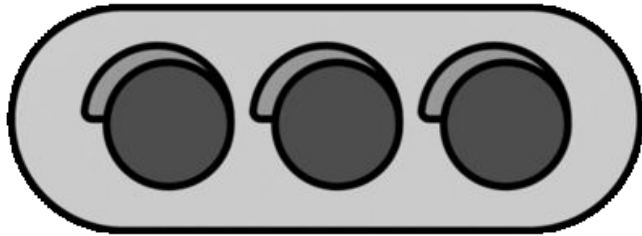
For example, At Road, Traffic Signal and Car can be seen as **two independent Classes**.

Individual signals and cars can be represented through objects of these classes.



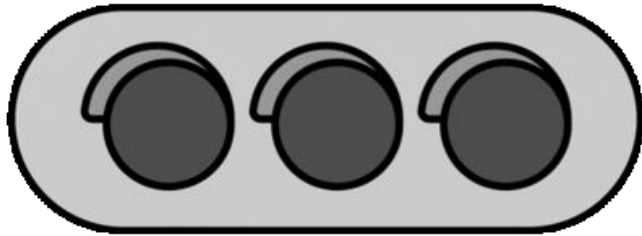
Problem Scenario 01: Example

These instances, **Car** and **Traffic Signal** need to communicate with each other but the life time of both instances can be different from each other.



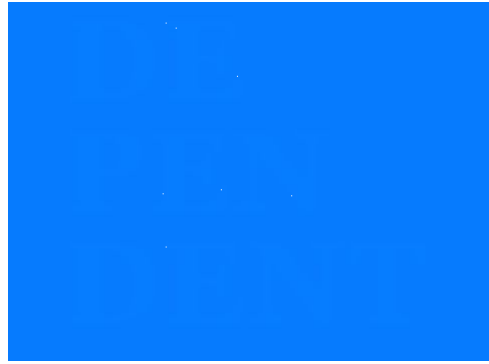
Problem Scenario 01: Example

Existence of **Car** object does not depend on the existence of **Traffic Signal** object and Vice versa.



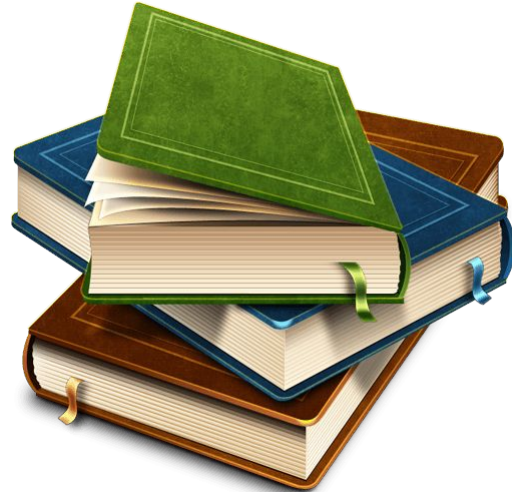
Problem Scenario 02

Similarly, sometimes in real life there are **some objects** whose existence **depends** on the existence of other objects.



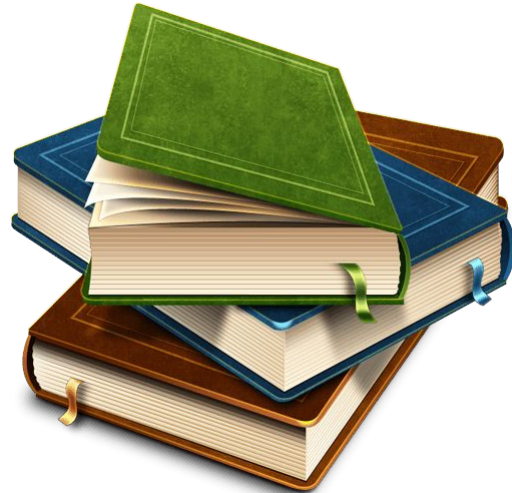
Problem Scenario 02: Example

For example, **TableofContents** and **Book** both have two different properties and qualified as for **two different classes**.



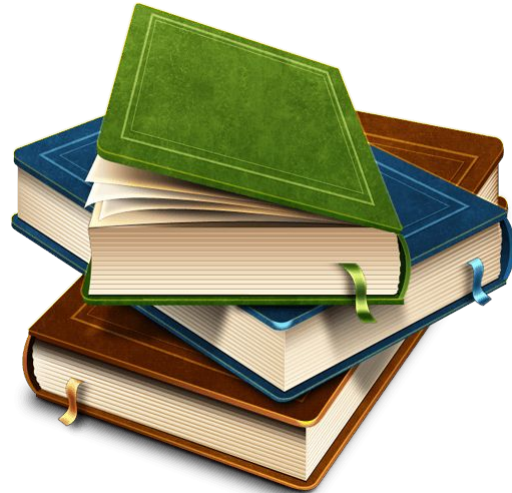
Problem Scenario 02: Example

An object of **TableofContents** class does not mean anything without its corresponding object of **Book** class.



Problem Scenario 02: Example

In other words, we can say the object **lifetime** of **TableofContents** class **depends** on the lifetime of the corresponding **Book** class



Problem Scenario

So when the classes collaborate with each other, there could be two possible scenarios

1. Lifetime of objects is independent of each other.
2. Lifetime of objects dependent on each other.

Types of Association

To handle these scenarios of Real Life, **Object Oriented Programming** offers **two** types of Associations.

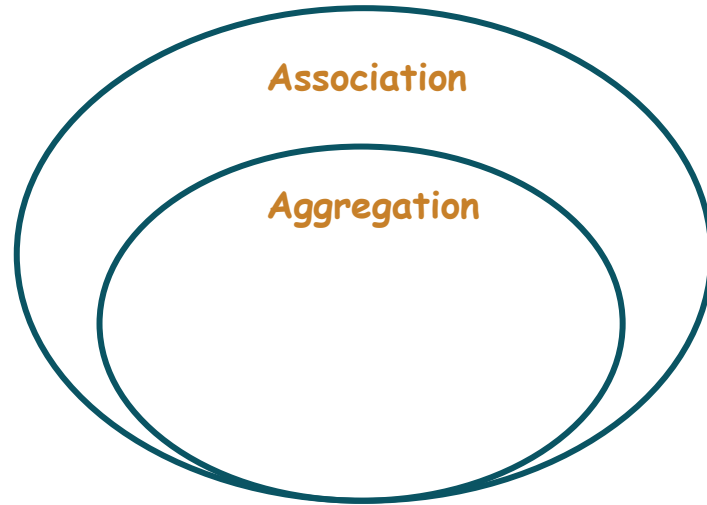
1. Aggregation.



2. Composition.

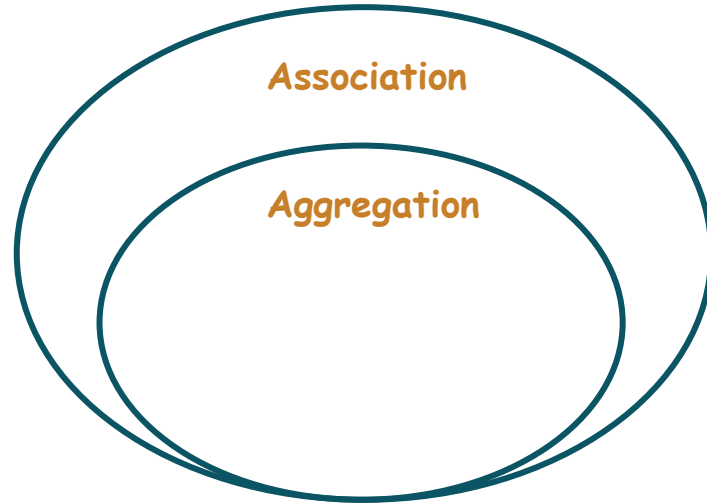
Types of Association: Aggregation

When two objects have **independent lifetime** but one object has an other object such type of association is called the **Aggregation**.



Types of Association: Aggregation

The relation between **Car** and **Traffic Signal** class is **Aggregation** relation because both object have their own lifetime and signal object contains the cars.



Aggregation: Working Example

The **Car** class provide **one method** that allow external world to communicate with it

```
class Car
{
    public string carName;           //Data Member
    public Car(string name)         //Parameterized Constructor
    {
        carName = name;
    }
    public void onSignalChange(string action) //Member Function/Behaviour
    {
        Console.WriteLine("I am " + carName + " and I am in " + action + " state");
    }
}
```

Aggregation: Working Example

The **TrafficSignal** class contains cars in **carList** that are present on the signal and new cars can be added into the list using **addCar** Function

```
class TrafficSignal
{
    //Data Members
    string state;
    List<Car> carList = new List<Car>();

    //Member Functions/Behaviours
    public void addCar(Car c)
    {
        carList.Add(c);
    }
}
```


Aggregation: Working Example

TrafficSignal class
also provides two
methods to set it
state Red and Green

```
class TrafficSignal
{
    //Data Members
    string state;
    List<Car> carList = new List<Car>();

    //Member Functions/Behaviours
    public void addCar(Car c)
    {
        carList.Add(c);
    }
    public void setRedState()
    {
        state = "Red";
        informCars();
    }
    public void setGreenState()
    {
        state = "Green";
        informCars();
    }
}
```

Aggregation:

Both of these methods, call **informCars** function to update the cars state.

```
class TrafficSignal
{
    //Data Members
    string state;
    List<Car> carList = new List<Car>();

    //Member Functions/Behaviours
    public void addCar(Car c)
    {
        carList.Add(c);
    }
    public void setRedState()
    {
        state = "Red";
        informCars();
    }
    public void setGreenState()
    {
        state = "Green";
        informCars();
    }
    public void informCars()
    {
        foreach (Car car in carList)
        {
            car.onSignalChange(state);
        }
    }
}
```

Aggregation:

Both of these methods, call **informCars** function to update the cars state. This function inform all cars through the available method of car that the signal has been changed.

```
class TrafficSignal
{
    //Data Members
    string state;
    List<Car> carList = new List<Car>();

    //Member Functions/Behaviours
    public void addCar(Car c)
    {
        carList.Add(c);
    }
    public void setRedState()
    {
        state = "Red";
        informCars();
    }
    public void setGreenState()
    {
        state = "Green";
        informCars();
    }
    public void informCars()
    {
        foreach (Car car in carList)
        {
            car.onSignalChange(state);
        }
    }
}
```

```

class TrafficSignal
{
    //Data Members
    string state;
    List<Car> carList = new List<Car>();

    //Member Functions/Behaviours
    public void addCar(Car c)
    {
        carList.Add(c);
    }
}

```

```

static void Main(string[] args)
{
    TrafficSignal signal1 = new TrafficSignal();
    Car car1 = new Car("Car1");
    Car car2 = new Car("Car2");
    signal1.addCar(car1);
    signal1.addCar(car2);
    signal1.setRedState();
    signal1.setGreenState();
    Console.ReadKey();
}

```

```

class TrafficSignal
{
    //Data Members
    string state;
    List<Car> carList = new List<Car>();

    //Member Functions/Behaviours
    public void addCar(Car c)
    {
        carList.Add(c);
    }
    public void setRedState()
    {
        state = "Red";
        informCars();
    }
    public void setGreenState()
    {
        state = "Green";
        informCars();
    }
    public void informCars()
    {
        foreach (Car car in carList)
        {
            car.onSignalChange(state);
        }
    }
}

```

}

```

class TrafficSignal
{
    //Data Members
    string state;
    List<Car> carList = new List<Car>();

    //Member Functions/Behaviours
    public void addCar(Car c)
    {
        carList.Add(c);
    }
}

```

I am Car1 and I am in Red state
 I am Car2 and I am in Red state
 I am Car1 and I am in Green state
 I am Car2 and I am in Green state

```

static void Main(string[] args)
{
    TrafficSignal signal1 = new TrafficSignal();
    Car car1 = new Car("Car1");
    Car car2 = new Car("Car2");
    signal1.addCar(car1);
    signal1.addCar(car2);
    signal1.setRedState();
    signal1.setGreenState();
    Console.ReadKey();
}

```

```

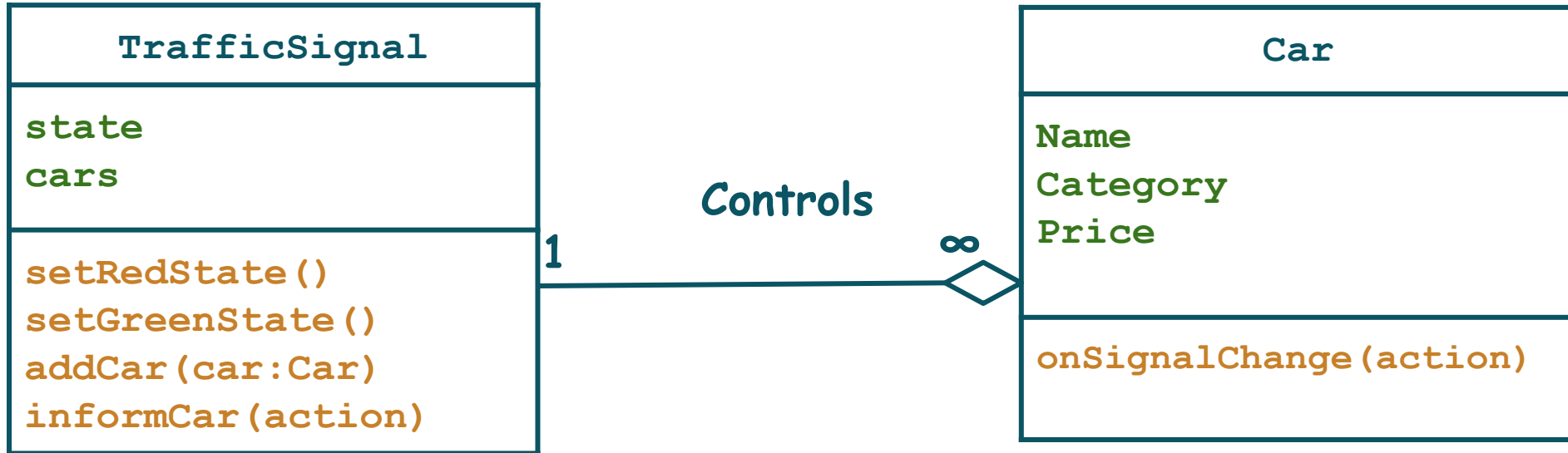
class TrafficSignal
{
    //Data Members
    string state;
    List<Car> carList = new List<Car>();

    //Member Functions/Behaviours
    public void addCar(Car c)
    {
        carList.Add(c);
    }
    public void setRedState()
    {
        state = "Red";
        informCars();
    }
    public void setGreenState()
    {
        state = "Green";
        informCars();
    }
    public void informCars()
    {
        foreach (Car car in carList)
        {
            car.onSignalChange(state);
        }
    }
}

```

Aggregation: Class Diagram

We use **empty diamond** symbol in class diagram to represent the **Aggregation** Relation between two classes.



Types of Association

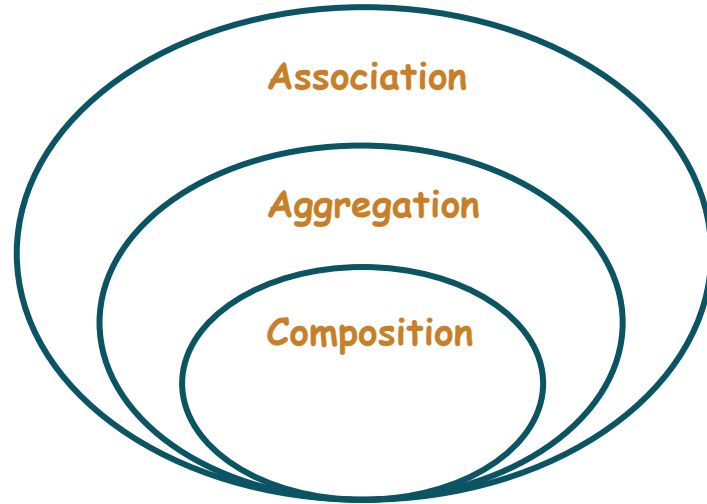
To handle these scenarios of Real Life, **Object Oriented Programming** offers **two** types of Associations.

1. Aggregation.
2. Composition.



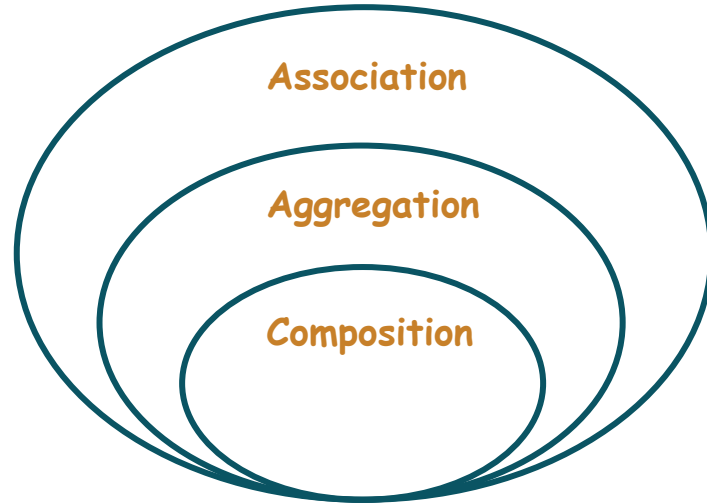
Types of Association: Composition

When life time of an object **depends** on the lifetime of another object this type of Association is called **Composition**.



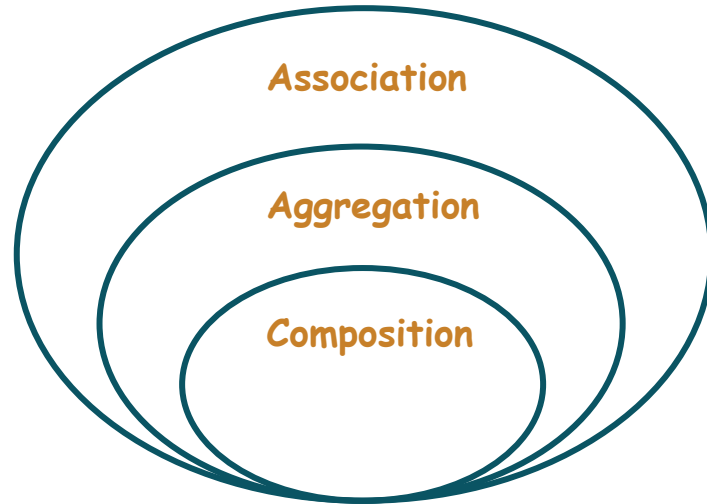
Types of Association: Composition

The Association between **Book** and **TableofContents** is type of composition.



Composition: Working Example

The Association between **Book** and **TableofContents** is type of composition.



Composition: Working Example

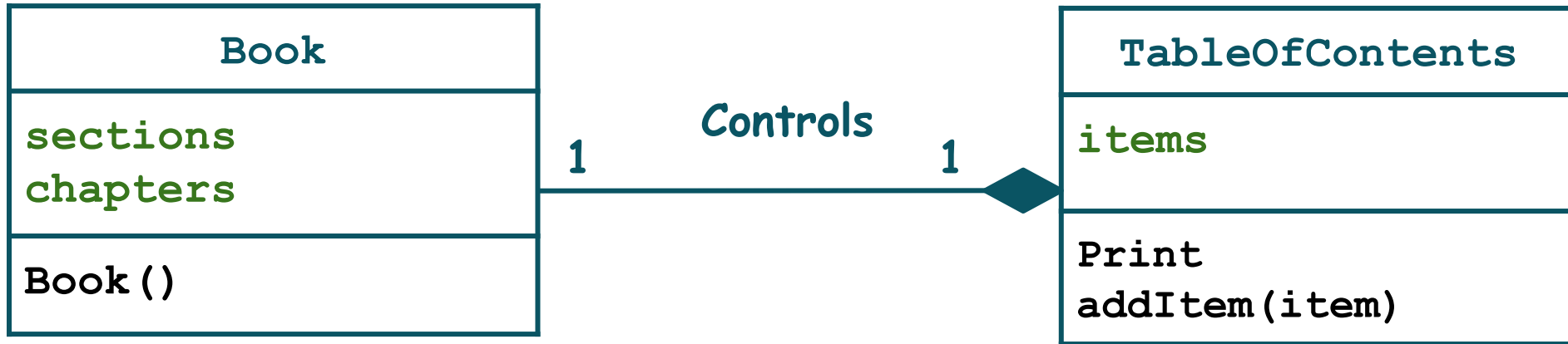
```
static void Main(string[] args)
{
    Book book1 = new Book();
    Console.ReadKey();
}
```

```
class TableofContents
{
    List<string> items;
    public void addItem(string item)
    {
        items.Add(item);
    }
}
```

```
class Book
{
    public Book()
    {
        toc = new TableofContents();
    }
    TableofContents toc;
    List<string> sections;
    List<string> chapters;
}
```

Composition: Working Example

We use **filled diamond** symbol in class diagram to represent the **Composition** Relation between two classes.



Conclusion

- Association has two types
 - 1) Aggregation
 - 2) Composition
- When life times of two separate objects are **independent** of each other, this type of association is called **Aggregation**
- When life time of two separate objects are **dependent** of each other, this type of association is called **Composition.**



Learning Objective

Identify and Write Code for
Aggregation and Composition
Scenarios.



Self Assessment: Association

1. A Football player has a football. Write two classes.
 1. Football
 2. FootballPlayer

Football class has type, size and weight attributes.

Write the default constructor as well as parameterized constructor of the class.

FootballPlayer has name and Football attributes.

Write the default as well as parameterized constructor of the class.



Self Assessment: Association

2. Write the code in C# for the following domain model.

