Extending Capabilities of a Class

Problem Scenario

Let's say we have to store information of students that include

name, session, isDayScholar, EntryTestMarks, HSMarks,

RoomNumber, isFridgeAvailable, isInternetAvailable,

isBusCardIssued, PickUpPoint, BusNo, PickupDistance

Problem Scenario

Also, there are three functions that can be applied on these data.
These functions are

1. getHostelFee()
2. getBusFees()
3. calculateMerit()

Problem Scenario: Activity

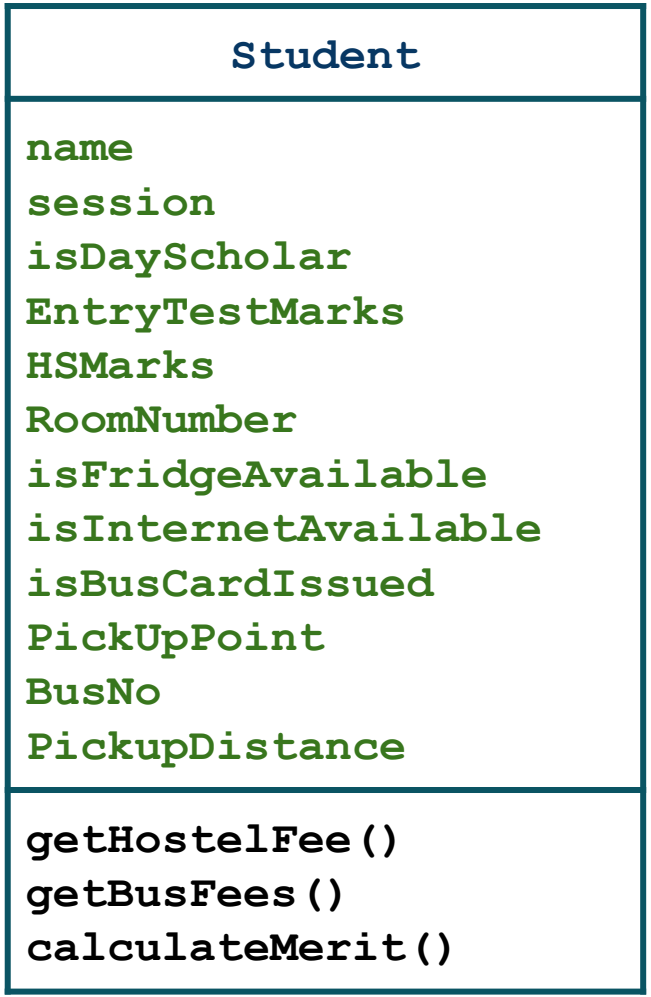
Model the class diagram for this scenario.

Problem Scenario:

One Possible Solution is to design a single class and put all data and functions within that class.

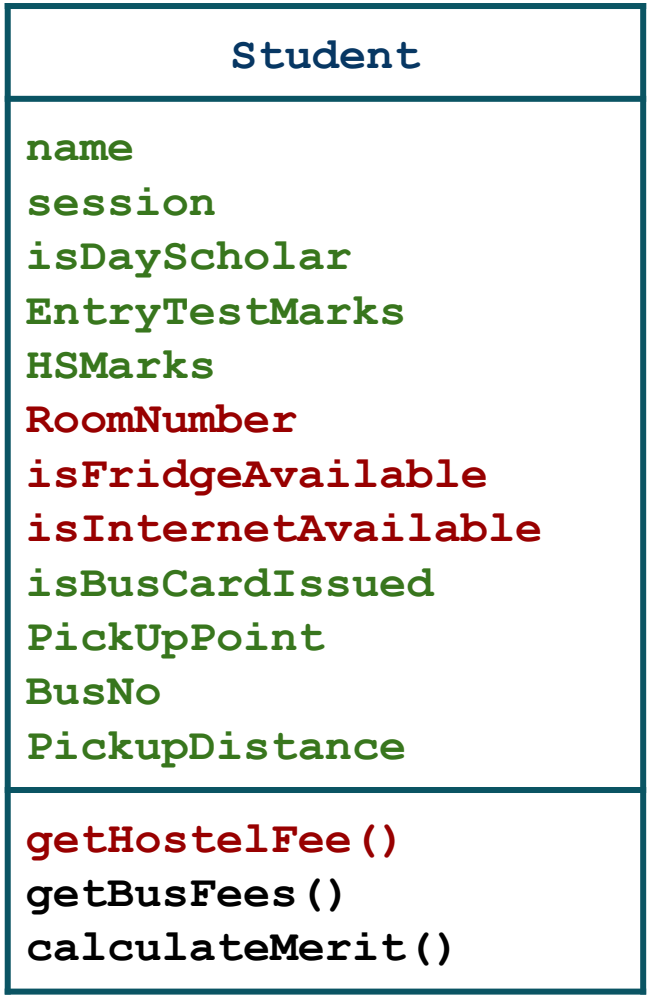
Problem Scenario:

One Possible Solution is to design a single class and put all data and functions within that class.



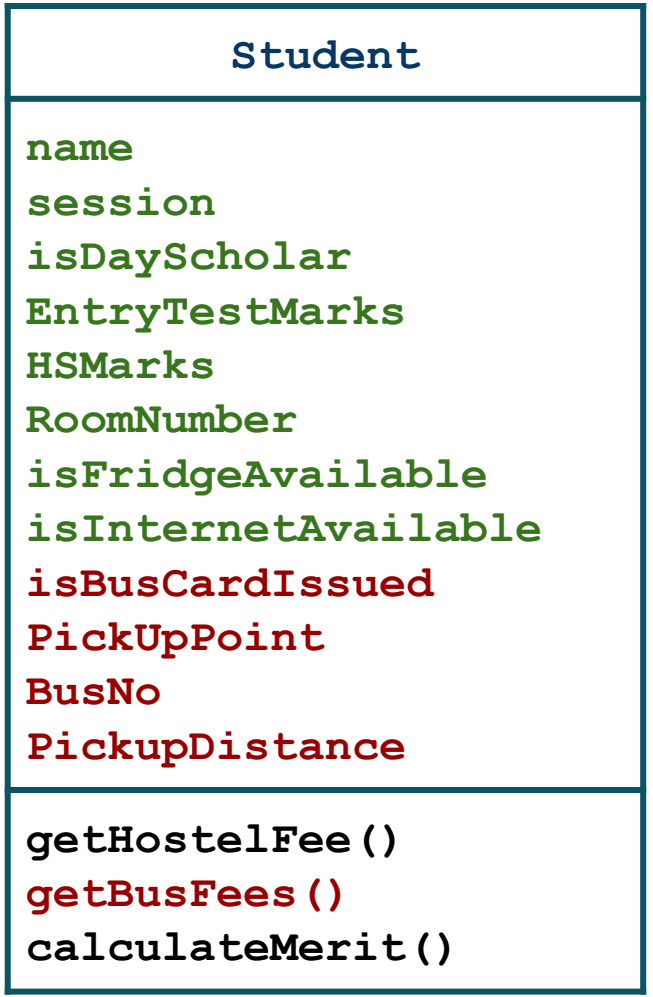
What's Wrong

Some functions and data are not interrelated, such as the highlighted attributes and functions are only related to hostelite student but not with the day scholars.



What's Wrong

Some functions and data are not interrelated, such as the highlighted attributes and functions are only related to day scholar student but not with the day scholars.



Problem Scenario: Activity

Any other (better) solution ?

Problem Scenario: Activity

Another Possible Solution is made two classes one for hostelite students and other for day scholar.

Problem Scenario:

Another Possible Solution is made two classes one for hostelite students and other for day scholar.

Problem Scenario:

In this solution we made two classes.

Hostelite
name session isDayScholar EntryTestMarks HSMarks RoomNumber isFridgeAvailable isInternetAvailable
getHostelFee() calculateMerit()

DayScholar
name session isDayScholar EntryTestMarks HSMarks PickUpPoint BusNo PickupDistance
getBusFees() calculateMerit()

Problem Scenario:

Any Problem with this Approach ?

Hostelite
name session isDayScholar EntryTestMarks HSMarks RoomNumber isFridgeAvailable isInternetAvailable
getHostelFee() calculateMerit()

DayScholar
name session isDayScholar EntryTestMarks HSMarks PickUpPoint BusNo PickupDistance
getBusFees() calculateMerit()

Problem Scenario:

Here some of the information is Repeating.

Hostelite
name session isDayScholar EntryTestMarks HSMarks RoomNumber isFridgeAvailable isInternetAvailable
getHostelFee() calculateMerit()

DayScholar
name session isDayScholar EntryTestMarks HSMarks PickUpPoint BusNo PickupDistance
getBusFees() calculateMerit()

Problem Scenario:

Before moving towards the solution, let's first define the problem.

Problem Scenario:

Some times, two objects have common functionality as well as some functionality that is different from each other.
How to model such objects ?

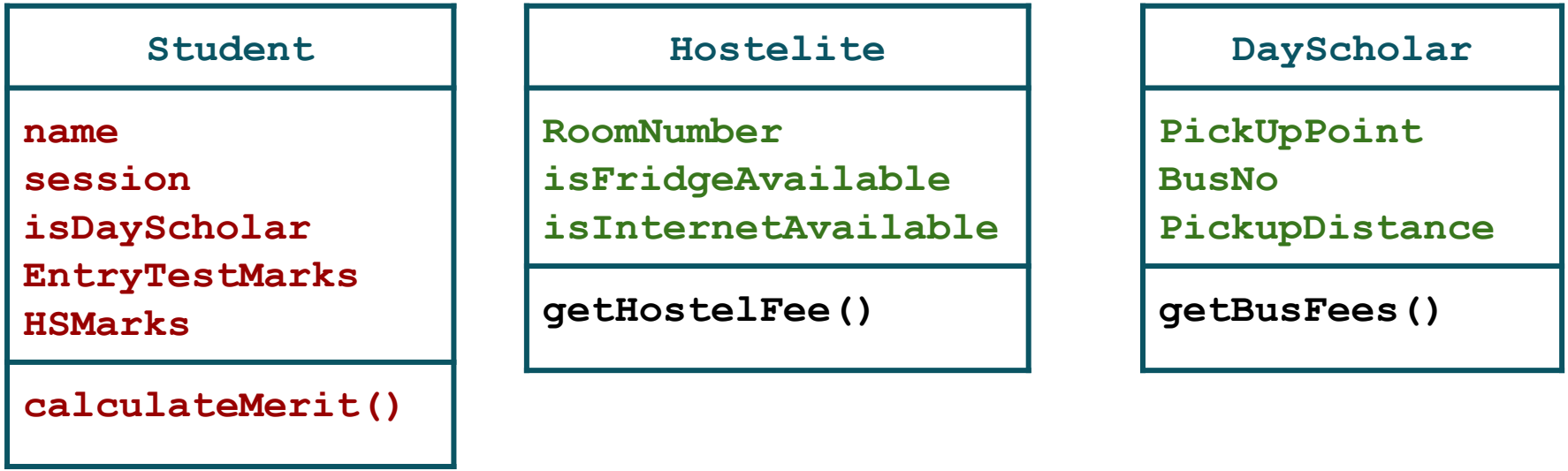
Solution

We create three classes:

1. Student: with common information
2. Hostelite: specific to hostelite students
3. DayScholar: specific to Day Scholar.

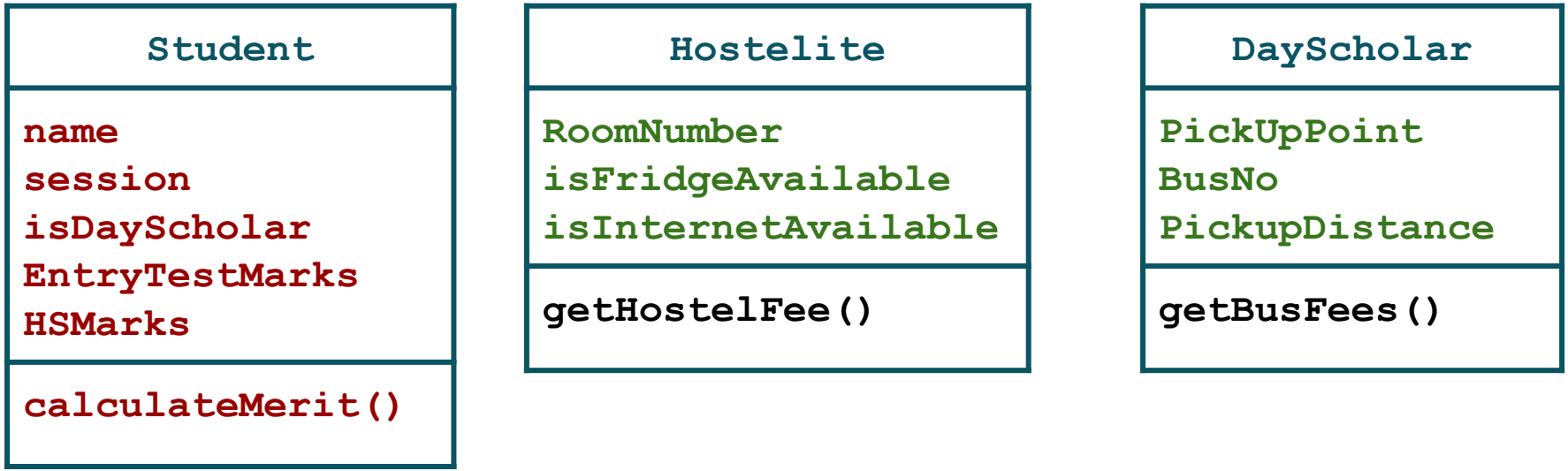
Solution

We create three classes:



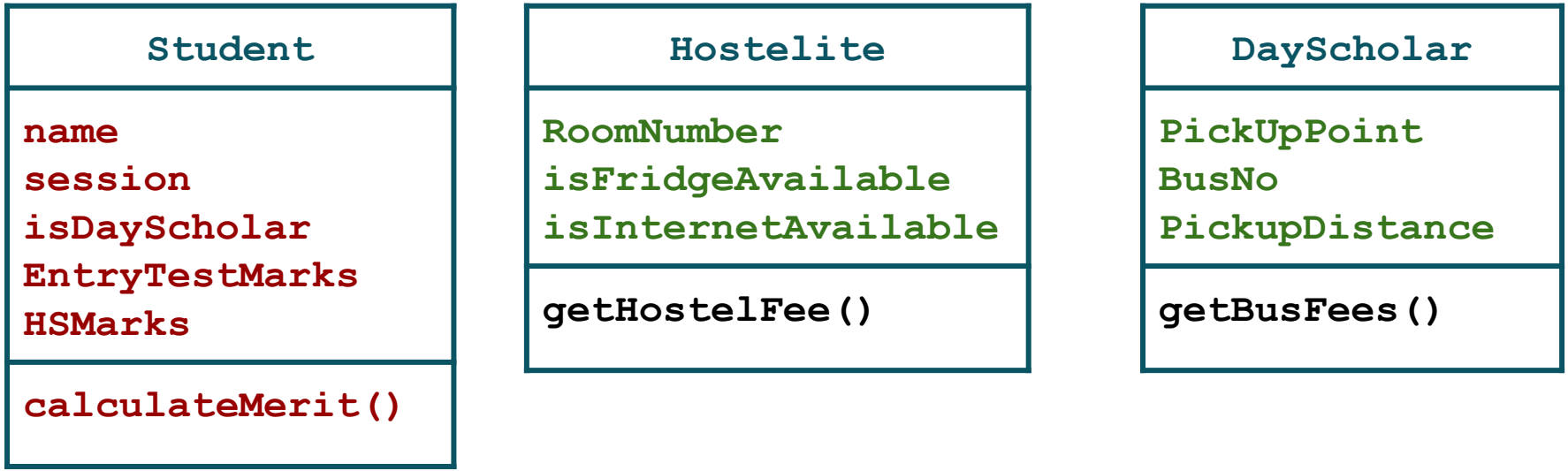
Solution

Is it correct solution ?



Solution

Should we create two objects for a student ?



Solution

Again, Disjoint Data not the single unit

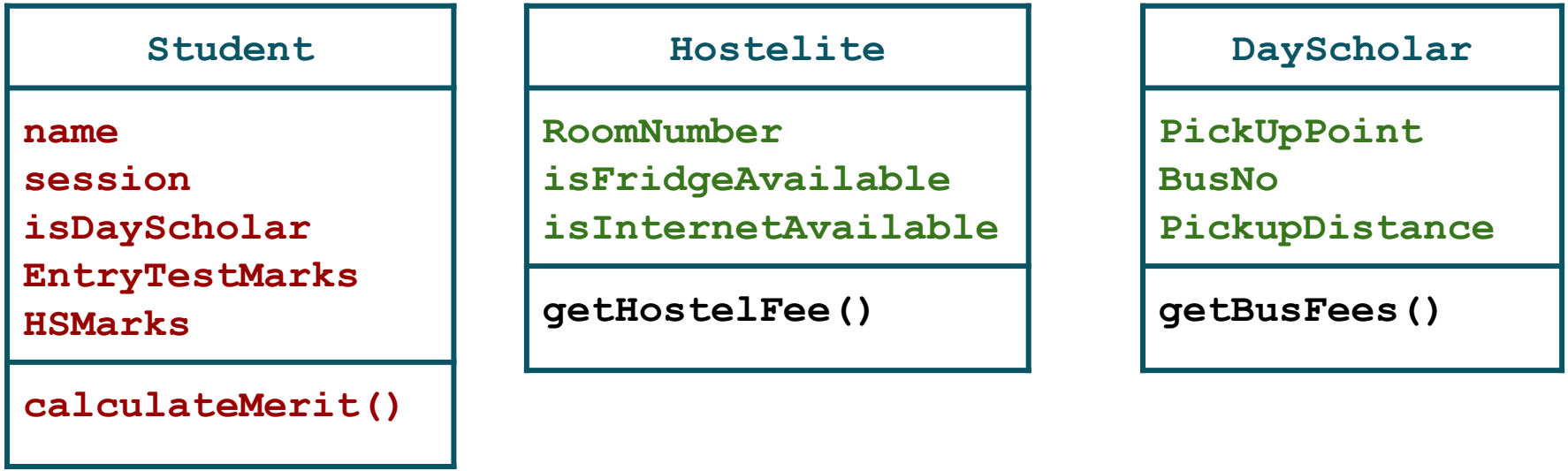
Student
name session isDayScholar EntryTestMarks HSMarks
calculateMerit()

Hostelite
RoomNumber isFridgeAvailable isInternetAvailable
getHostelFee()

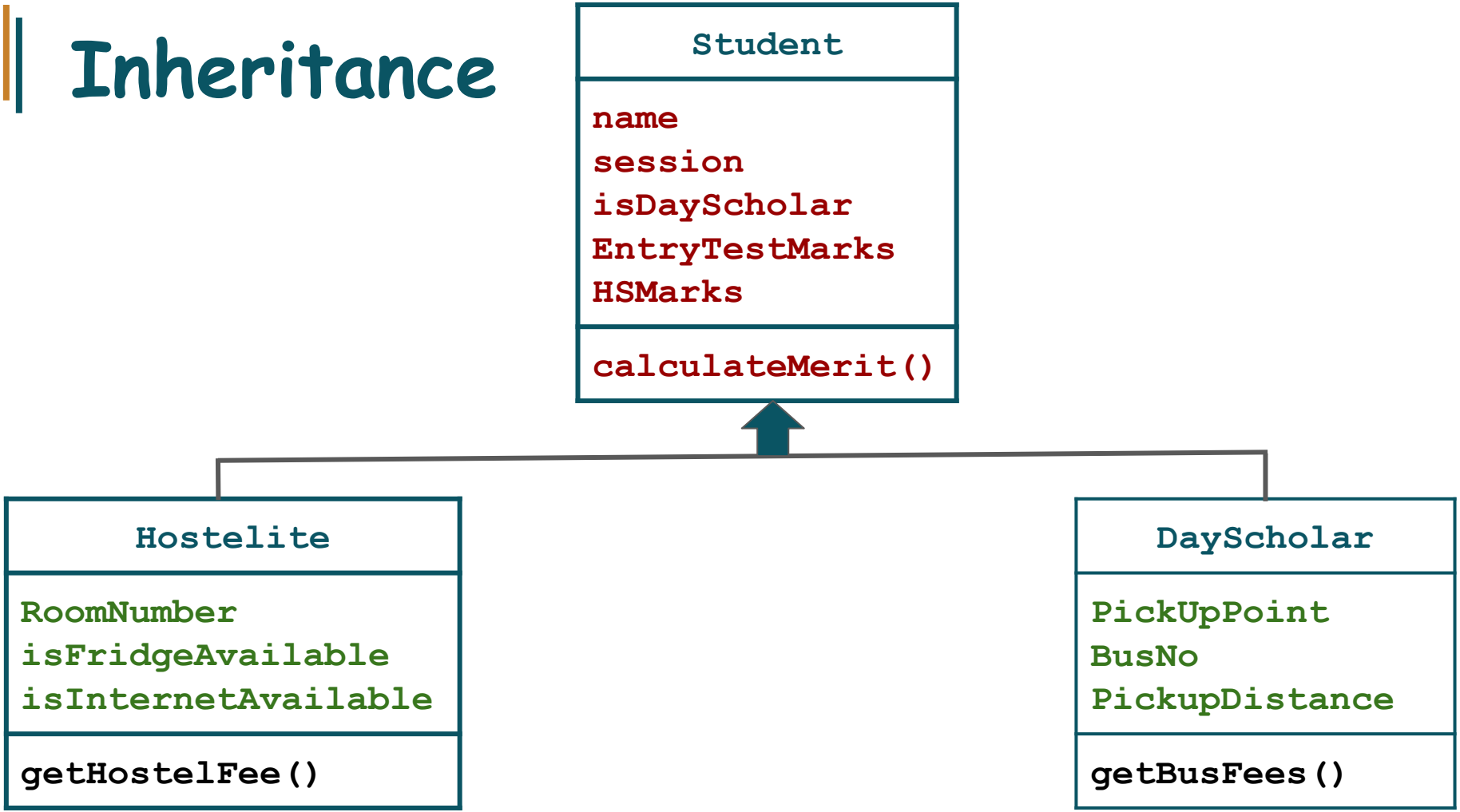
DayScholar
PickUpPoint BusNo PickupDistance
getBusFees()

Solution: Inheritance

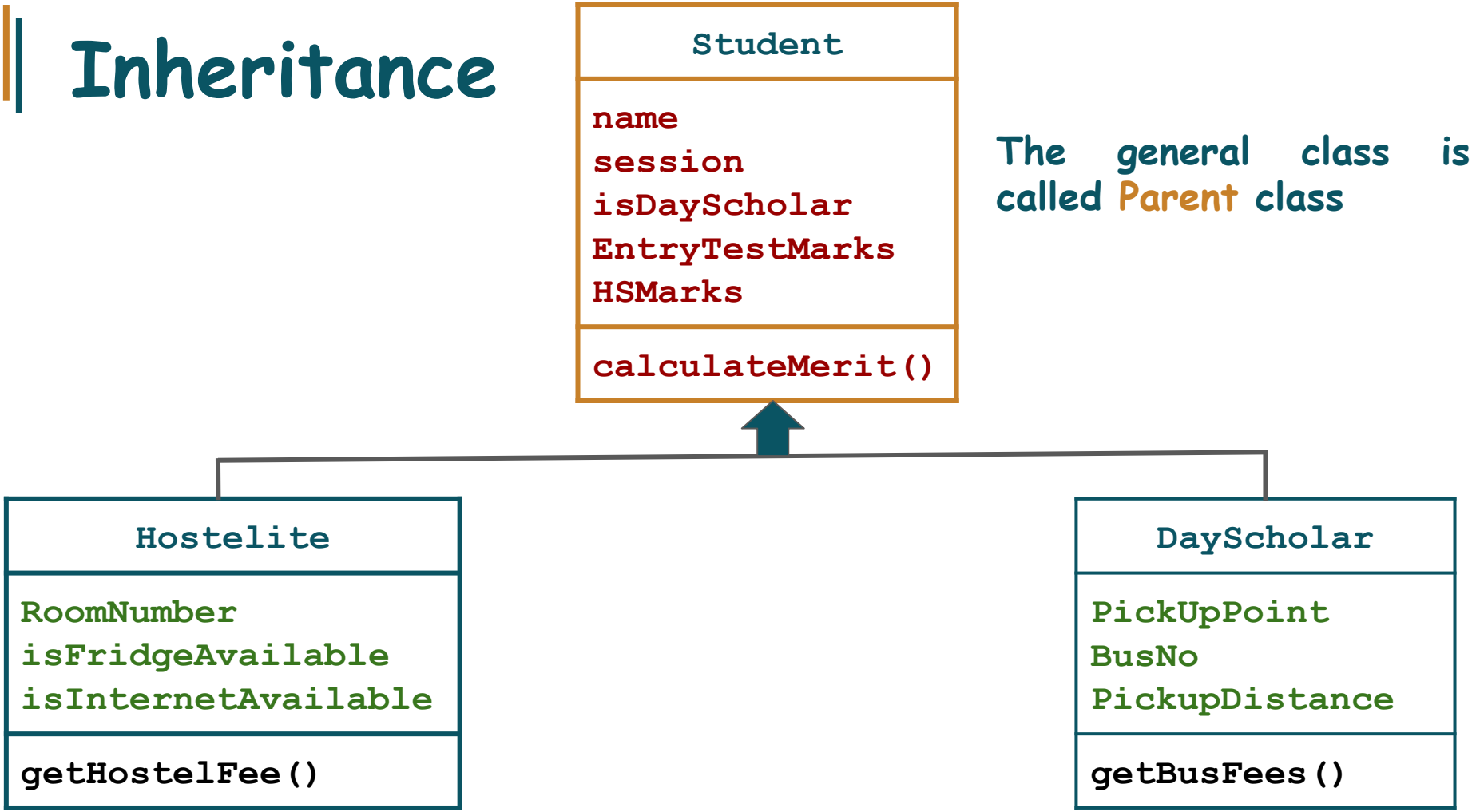
However, we can link these classes through Inheritance



Inheritance

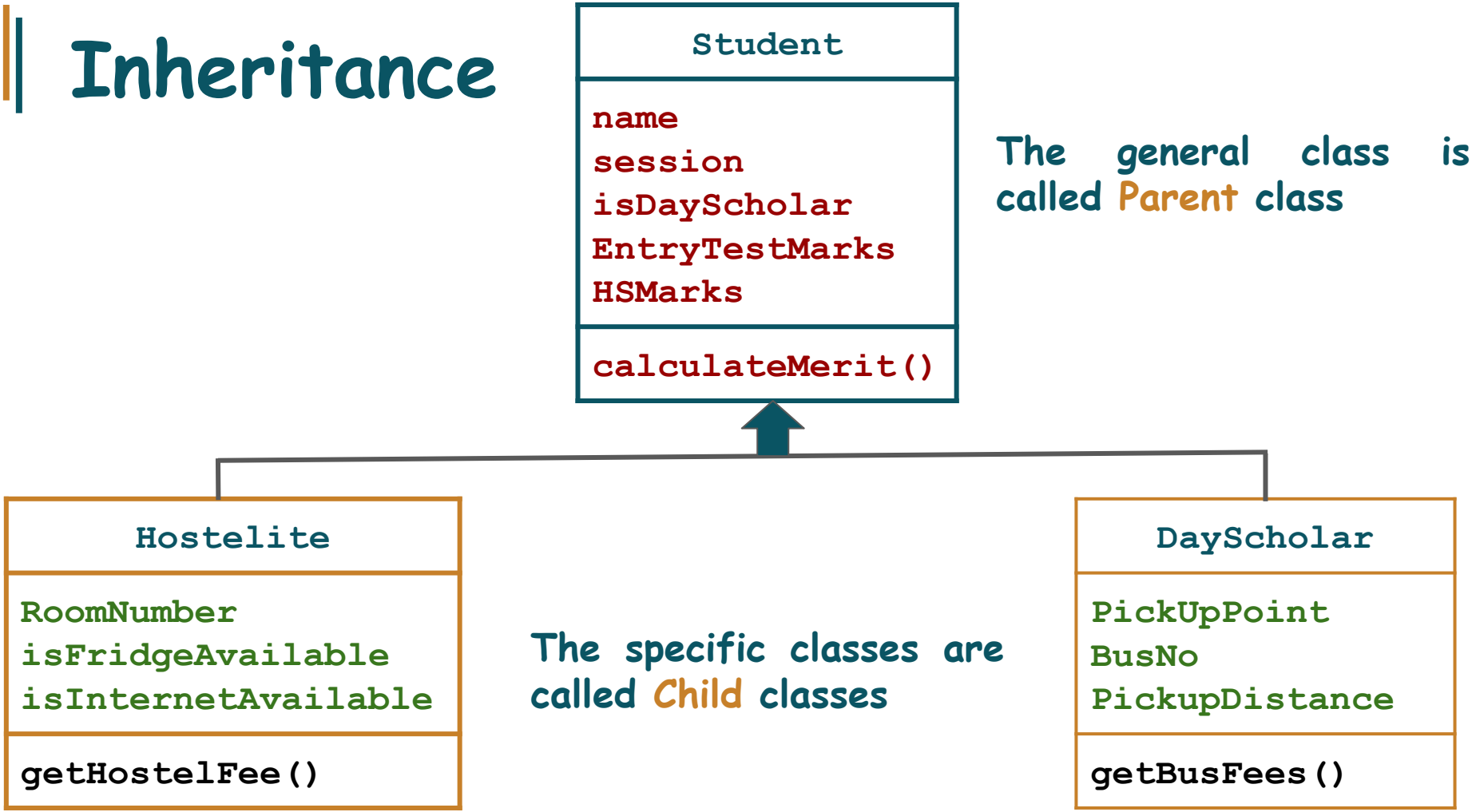


Inheritance



The general class is called Parent class

Inheritance



The general class is called Parent class

The specific classes are called Child classes

How to Convert in C# Code?

Now, lets see how to inherit a class using C#.

Working Example

We have defined the Student Class.

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

Working Example

Now, let's see how to inherit Hostelite Class from the Student Class.

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

Hostelite Class

Now, let's see how to inherit Hostelite Class from the Student Class.

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

```
class Hostelite : Student
{
    public int RoomNumber;
    public bool isFridgeAvailable;
    public bool isInternetAvailable;

    public int getHostelFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

Hostelite Class

To inherit from a class, use the : symbol

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

```
class Hostelite : Student
{
    public int RoomNumber;
    public bool isFridgeAvailable;
    public bool isInternetAvailable;

    public int getHostelFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

Hostelite Class

We have created a new Hostelite class from an existing Student class

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

```
class Hostelite : Student
{
    public int RoomNumber;
    public bool isFridgeAvailable;
    public bool isInternetAvailable;

    public int getHostelFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

Hostelite Class

We have created a new Hostelite class from an existing Student class

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

Parent Class

```
class Hostelite : Student
{
    public int RoomNumber;
    public bool isFridgeAvailable;
    public bool isInternetAvailable;

    public int getHostelFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

Child Class

Hostelite Class

Hostelite Class can Access all the attributes and functions of the Parent Class.

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

Parent Class

```
class Hostelite : Student
{
    public int RoomNumber;
    public bool isFridgeAvailable;
    public bool isInternetAvailable;

    public int getHostelFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

Child Class

Hostelite Class

Hostelite Class can Access all the attributes and functions of the Parent Class.

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

Parent Class

```
class Hostelite : Student
{
    public int RoomNumber;
    public bool isFridgeAvailable;
    public bool isInternetAvailable;

    public int getHostelFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

Child Class

Hostelite Class

```
static void Main(string[] args)
{
    Hostelite std = new Hostelite();
    std.name = "Ahmad";
    std.RoomNumber = 12;
    Console.WriteLine(std.name + " is Allocated
Room " + std.RoomNumber);
    Console.ReadKey();
}
```

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

Parent Class

```
class Hostelite : Student
{
    public int RoomNumber;
    public bool isFridgeAvailable;
    public bool isInternetAvailable;

    public int getHostelFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

Child Class

Hostelite Class

Hostelite Class can access the parent attributes

```
static void Main(string[] args)
{
    Hostelite std = new Hostelite();
    std.name = "Ahmad";
    std.RoomNumber = 12;
    Console.WriteLine(std.name + " is Allocated
Room " + std.RoomNumber);
    Console.ReadKey();
}
```

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

Parent Class

```
class Hostelite : Student
{
    public int RoomNumber;
    public bool isFridgeAvailable;
    public bool isInternetAvailable;

    public int getHostelFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

Child Class

Hostelite Class

Output is:

```
static void Main(string[] args)
{
    Hostelite std = new Hostelite();
    std.name = "Ahmad";
    std.RoomNumber = 12;
    Console.WriteLine(std.name + " is Allocated
Room " + std.RoomNumber);
    Console.ReadKey();
}
```

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

Parent Class

```
class Hostelite : Student
{
    public int RoomNumber;
    public bool isFridgeAvailable;
    public bool isInternetAvailable;

    public int getHostelFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

Child Class

```
Ahmad is Allocated Room 12
```

DayScholar Class

Now, let's inherit DayScholar Class from the Student Class also.

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

DayScholar Class

Now, let's inherit DayScholar Class from the Student Class also.

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

```
class DayScholar : Student
{
    public string pickUpPoint;
    public int busNo;
    public int pickUpDistance;

    public int getBusFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

DayScholar Class

DayScholar Class can also access the parent attributes

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

```
static void Main(string[] args)
{
    DayScholar std = new DayScholar();
    std.name = "Ahmad";
    std.busNo = 1;
    Console.WriteLine(std.name + " is Allocated
Bus " + std.busNo);
    Console.ReadKey();
}
```

```
class DayScholar : Student
{
    public string pickUpPoint;
    public int busNo;
    public int pickUpDistance;

    public int getBusFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

DayScholar Class

DayScholar Class can also access the parent attributes

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

```
class DayScholar : Student
{
    public string pickUpPoint;
    public int busNo;
    public int pickUpDistance;

    public int getBusFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

```
static void Main(string[] args)
{
    DayScholar std = new DayScholar();
    std.name = "Ahmad";
    std.busNo = 1;
    Console.WriteLine(std.name + " is Allocated
Bus " + std.busNo);
    Console.ReadKey();
}
```

```
Ahmad is Allocated Bus 1
```

Inheritance

Now, our Classes contain highly correlated data but with no repetition of code.

```
class Student
{
    public string name;
    public string session;
    public bool isDayScholar;
    public int EntryTestMarks;
    public int HSMarks;

    public double calculateMerit()
    {
        double merit = 0.0;
        // Code to calculate merit
        return merit;
    }
}
```

```
class Hostelite : Student
{
    public int RoomNumber;
    public bool isFridgeAvailable;
    public bool isInternetAvailable;

    public int getHostelFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

```
class DayScholar : Student
{
    public string pickUpPoint;
    public int busNo;
    public int pickUpDistance;

    public int getBusFee()
    {
        int fee = 0;
        // Code to calculate fee
        return fee;
    }
}
```

Inheritance: Advantages

Inheritance promotes reusability. When a class inherits or derives another class, it can access all the functionality of Parent class.

Conclusion

- Inheritance allows programmers to create classes that are built upon existing classes, to specify a new implementation while maintaining the same behaviors
- We create a general class that has common functionality. This class is also called the parent class (Base Class / Super Class).
- Then we create specific classes (Child Classes / Derived Classes / Sub Classes) that have different functionality to each other and inherit these classes from the common class.

Learning Objective

Write a class that extends the capabilities of any other class

Self Assessment: Case Study

Fire Department has hired you to make a training and simulation system for them.
In this system they have Fire Trucks. Where each Fire Truck contains a Ladder and a Hose Pipe. Ladder has a specific length and colour and they are built right into the truck (i.e., they cannot be separated from the truck).
Hose pipes are detachable from the truck. Hose pipes are either made of synthetic rubber or soft plastic and they can be either be cylindrical or circular in shape. They have specific diameter and water flow rate.
Each FireTruck has a Firefighter as its Driver. FireFighter has a name. He can drive the fire truck and can extinguish fire as well.
They have a Fire Chief as well. The fire chief is just another firefighter. He can drive a truck. He can put out fires. But he can also delegate responsibility for putting out a fire to another firefighter.

Self Assessment:

1. Identify the Classes from the Case Study.
2. Identify the relationship and multiplicity.
3. Draw the Class Diagram.
4. Convert the Class Diagram into C# Code.