

CS-1004 Object Oriented programming

Week 5

Instructor:

ATIYA

LECTURER

(COMPUTER SCIENCE DEPARTMENT)

NATIONAL UNIVERSITY- FAST (KHI CAMPUS)

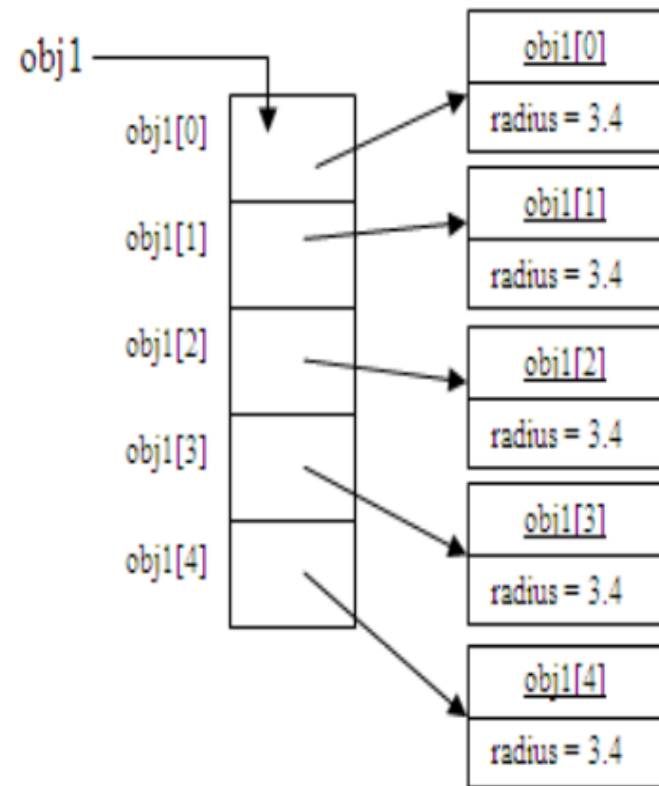
EMAIL: ATIYA.JOKHIO@NU.EDU.PK

ROOM: (LECTURER ROOM#19, CS DEPT BASEMENT-2)

Array of Objects

- We can create an array of objects.
 - By combining the advantage of array, and objects, we can build program more clean and better logical organization
 - Assume we need to store collection of 50 students, for example, we need to use several different arrays (one for names, one for address, one for matrix number and so forth.) which is very cumbersome and prone to error
 - The better way is by creating an array of student's object. This will result a more concise and easier-to understand code

Memory layout



Example:

```
// C++ program to implement
// the above approach
#include<iostream>
using namespace std;

class Employee
{
    int id;
    char name[30];
public:

    // Declaration of function
    void getdata();

    // Declaration of function
    void putdata();
};

// Defining the function outside
// the class
void Employee::getdata()
{
    cout << "Enter Id : ";
    cin >> id;
    cout << "Enter Name : ";
    cin >> name;
}
```

```
// Defining the function outside
// the class
void Employee::putdata()
{
    cout << id << " ";
    cout << name << " ";
    cout << endl;
}

// Driver code
int main()
{
    // This is an array of objects having
    Employee emp[30];
    int n, i;
    cout << "Enter Number of Employees - ";
    cin >> n;

    // Accessing the function
    for(i = 0; i < n; i++)
        emp[i].getdata();

    cout << "Employee Data - " << endl;

    // Accessing the function
    for(i = 0; i < n; i++)
        emp[i].putdata();
}
```

```
Enter Number of Employees - 3
Enter Id : 101
Enter Name : Mahesh
Enter Id : 102
Enter Name : Suresh
Enter Id : 103
Enter Name : Magesh
Employee Data -
101 Mahesh
102 Suresh
103 Magesh

-----
Process exited after 24.74 seconds with return value 0
Press any key to continue . . .
```

Objects	Employee Id	Employee name
<u>emp[0]</u> →		
<u>emp[1]</u> →		
<u>emp[2]</u> →		
<u>emp[3]</u> →		

Array of Objects

Advantages:

- The array of objects represent storing multiple objects in a single name.
- In an array of objects, the data can be accessed randomly by using the index number.
- Reduce the time and memory by storing the data in a single variable.

Object Relationship

- One of the advantages of an Object-Oriented programming language is code reuse.
- There are two ways we can do code reuse either by the implementation of inheritance (IS-A relationship), or object composition (HAS-A relationship).

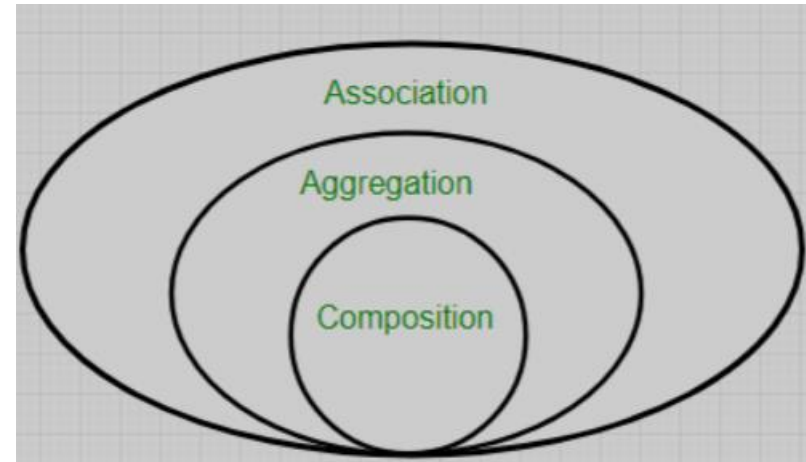
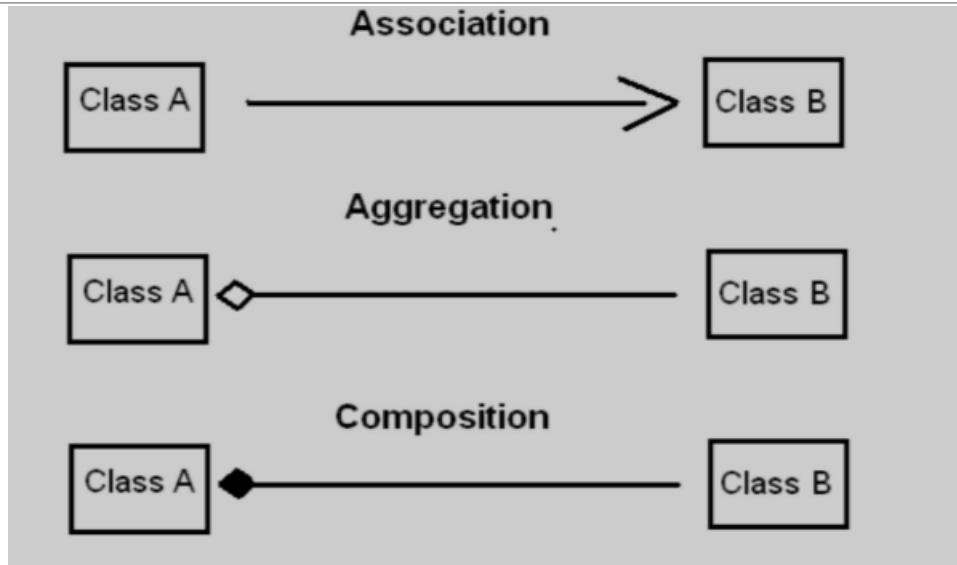
Object Relationship

Often, classes do not have to be created “from scratch.” Rather, they can include objects of other classes as members or they may be **derived** from other classes that provide attributes and behaviors the new classes can use.

Such software reuse can greatly enhance productivity and simplify code maintenance.

Including class objects as members of other classes is called **composition** (or **aggregation**). Deriving new classes from existing classes is called **inheritance**

UML notations



Object Relationship

is-a relationship (Achieved by inheritance)

has-a relationship (Achieved by composition)

has-a Relationship

- In a *has-a* relationship, an object contains one or more objects of other classes as members
- A technique in which a class contains an object of another class as its member is called containership. This type of relationship between classes is also called has-a relationship. It means that one class has an object of another class as its member.
- **“Has-A”** — An object **has a** subset of object
- For example is :-
- A rose has a leaf
- A dog has a legs

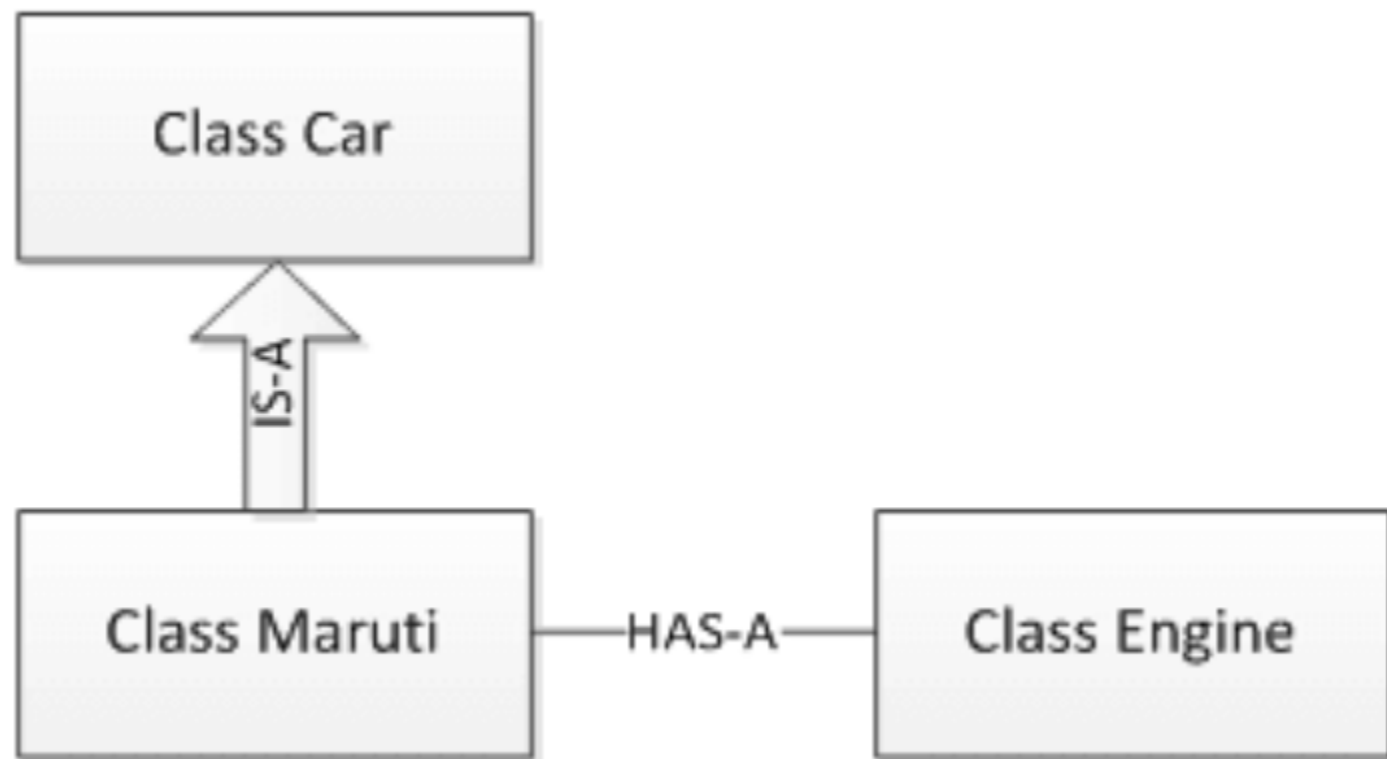
has-a Relationship

Composition(HAS-A) simply mean the use of instance variables that are references to other objects.

For example: Maruti has Engine, or House has Bathroom.

has-a Relationship

Use a car as an example. You could logically define a car as being a subclass of vehicle, so it inherits the properties of other vehicles. At the same time, a car has a motor. If you buy a car, you can logically assume that you are buying a motor as well.



Relationship

Association:

Explanation: Association represents a relationship between two classes where one class is aware of the other, but they are independent entities. It is a more general form of relationship.

Example: Consider a Teacher class and a Student class. They are associated because a teacher interacts with students, but neither owns the other. Both can exist independently.

Aggregation:

Explanation: Aggregation represents a "has-a" relationship where one class is a part of another class, but it can exist independently. It implies a weaker relationship compared to composition, and the part can be shared among multiple owners

.

Example: Consider a Library class and a Book class. Books are part of the library, but they can exist independently of the library, and multiple libraries can have the same book.

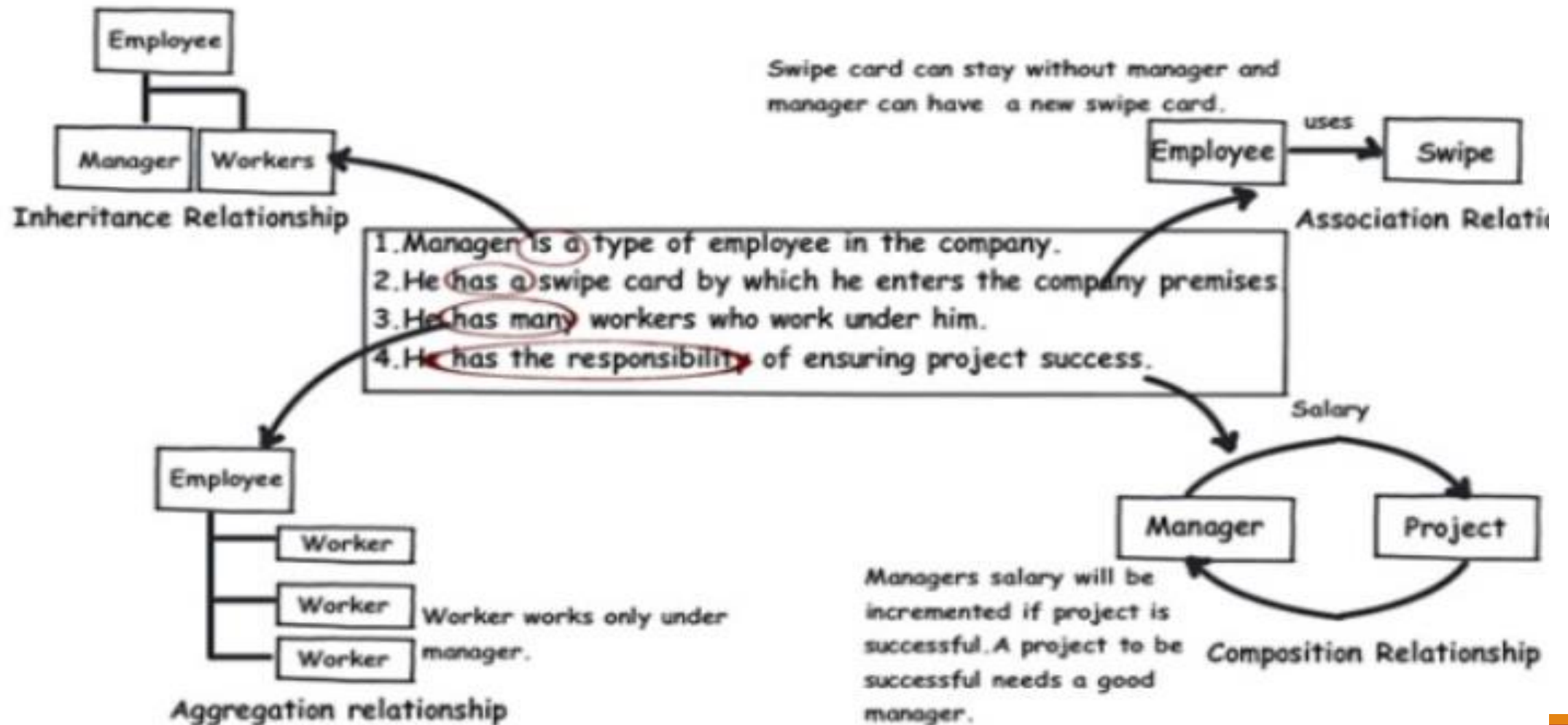
Composition:

Explanation: Composition represents a strong "owns-a" relationship where one class is a part of another class, and the lifetime of the part is controlled by the whole. If the whole is destroyed, all its parts are also destroyed.

Example: Consider a Car class composed of Engine, Wheel, and Body classes. If the car is destroyed, its components (engine, wheels, and body) are also destroyed.

Putting things together

Below is a visual representation of how the relationships have emerged from the



Extracting real world relationships from a requirement

1. Manager is an employee of XYZ limited corporation.
2. Manager uses a swipe card to enter XYZ premises.
3. Manager has workers who work under him.
4. Manager has the responsibility of ensuring that the project is successful.
5. Manager's salary will be judged based on project success.

Requirement 1: The IS A relationship

- If you look at the first requirement (Manager is an employee of XYZ limited corporation), it's a parent child relationship or inheritance relationship.
- The sentence above specifies that Manager is a type of employee , in other words we will have two classes: parent class Employee, and a child class Manager which will inherit from the Employee class.

Requirement 2: The Using relationship: Association

Requirement 2 is an interesting requirement (Manager uses a swipe card to enter XYZ premises). In this requirement, the manager object and the swipe card object use each other but they have their own object lifetime.

In other words, they can exist without each other. The most important point in this relationship is that there is no single owner.

Requirement 3: The Using relationship with Parent: Aggregation

The third requirement from our list (Manager has workers who work under him) denotes the same type of relationship like association but with a difference that one of them is an owner.

So as per the requirement, the Manager object will own worker objects

But... the Worker object can have its own lifetime which is completely disconnected from the Manager object. Looking from a different perspective, it means that if the Manager object is deleted, the Worker object does not die.

Requirements 4 and 5: The Death relationship: Composition

The last two requirements are actually logically one. If you read closely, the requirements are as follows:

1. Manager has the responsibility of ensuring that the project is successful.
2. Manager's salary will be judged based on project success.

Below is the conclusion from analyzing the above requirements:

1. Manager and the project objects are dependent on each other.
2. The lifetimes of both the objects are the same. In other words, the project will not be successful if the manager is not good, and the manager will not get good increments if the project has issues.

	Association	Aggregation	Composition
Owner	No owner	Single owner	Single owner
Life time	Have their own lifetime	Have their own lifetime	Owner's life time
Child object	Child objects all are independent	Child objects belong to a single parent	Child objects belong to a single parent

Example

```
#include <iostream>
using namespace std;

class first {
public:
    void showf()
    {
        cout << "Hello from first class\n";
    }
};

// Container class
class second {
    // creating object of first
    first f;

public:
    // constructor
    second()
    {
        // calling function of first class
        f.showf();
    }
};

int main()
{
    // creating object of second
    second s;
}
```

Example

```
#include <iostream>
using namespace std;

class first {
public:
    void showf()
    {
        cout << "Hello from first class\n";
    }
};

// Container class
class second {
    // creating object of first
    first f;

public:
    // constructor
    second()
    {
        // calling function of first class
        f.showf();
    }
};

int main()
{
    // creating object of second
    second s;
}
```

```
#include <iostream>
using namespace std;

class first {
public:
    first()
    {
        cout << "Hello from first class\n";
    }
};

// Container class
class second {
    // creating object of first
    first f;

public:
    // constructor
    second()
    {
        cout << "Hello from second class\n";
    }
};

int main()
{
    // creating object of second
    second s;
}
```

Example

Employee class has the reference of Address class as data member

Aggregation

```
#include <iostream>
using namespace std;
class Address {
public:
    string city;
    string street;

    Address(string c, string s) {
        city=c;
        street =s; }
    void display() {
        cout << "Address: " << street <<
            ", " << city << endl;
    }
};
```

```
class Employee {
    string name;
    Address* address; // Aggregation: Address is
                     //not owned by Employee
public:
    Employee(string n, Address* addr) {
        name=n;
        address=addr;}

    void display() {
        cout << "Employee: " << name << " lives at ";
        address->display();
    }
};
```

```
int main() {
    Address addr("New York", "5th Avenue");
    Employee emp1("Alice", &addr);
    Employee emp2("Bob", &addr);
    emp1.display();
    emp2.display();

    return 0;
}
```

Aggregation

```
D:\SPRING 2025\OOP\Lecture x + v
Employee: Alice lives at Address: 5th Avenue, New York
Employee: Bob lives at Address: 5th Avenue, New York

-----
Process exited after 0.2994 seconds with return value 0
Press any key to continue . . .
```

Modified in Composition

```
class Employee {  
    string name;  
    Address address; // Composition: Address is  
    owned by Employee  
public:  
    Employee(string n, string city, string street)  
        : name(n), address(city, street) {} // Initialize  
    Address in constructor  
  
    void display() {  
        cout << "Employee: " << name << " lives at ";  
        address.display();  
    }  
};
```

Engine composition

In the case of composition, the Engine is completely encapsulated by the Car. There is no way for the outside world to get a reference to the Engine. The Engine lives and dies with the car.

Composition

```
#include <iostream>
using namespace std;
class Engine {
public:
    Engine() {
        cout << "Engine created." << endl;
    }
    void start() {
        cout << "Engine started." << endl;
    }
    ~Engine() {
        cout << "Engine destroyed." << endl;
    }
};
```

```
class Car {
    Engine engine; // Composition: Car "owns" Engine
public:
    Car() {
        cout << "Car created." << endl;
    }
    void drive() {
        engine.start();
        cout << "Car is driving." << endl; }
    ~Car() {
        cout << "Car destroyed." << endl;
    }
};
```

```
int main() {
    Car car;
    car.drive();
    return 0;}
```

Composition

```
D:\SPRING 2025\OOP\Lecture x + v
Engine created.
Car created.
Engine started.
Car is driving.
Car destroyed.
Engine destroyed.

-----
Process exited after 0.2625 seconds with return value 0
Press any key to continue . . .
```

Feature	Composition	Aggregation
Object Ownership	The container class owns the contained object	The contained object exists independently
Lifetime	Contained object is created/destroyed with the container	Contained object can outlive the container
Example	<code>Car has-a Engine</code> (Engine is inside <code>Car</code>)	<code>Employee has-a Address</code> (Address exists separately)

Electric Cars

With aggregation, the Car also performs its functions through an Engine, but the Engine is not always an internal part of the Car. Engines may be swapped, or even completely removed. Not only that, but the outside world can still have a reference to the Engine

Exercise

Imagine you are designing a student course management system for a university. In this system:

Each **student** can enroll in multiple **courses**.

Each **course** has a unique course ID, name, and number of credits.

A student must be able to view the list of their enrolled courses along with the total credits they have earned from these courses.

Task:

Design a simple class structure in which the **Student** class aggregates **Course** objects.

Write a method in the **Student** class to:

- Add a new course.

- Calculate and return the total number of credits from the enrolled courses.

```
#include <iostream>
#include <string>
using namespace std;

#define MAX_COURSES 10
```

```
class Course {
private:
    string courseName;
    int courseID;
    int credits;
public:
    Course(string name = "", int id = 0, int cr = 0) {
        courseName = name;
        courseID = id;
        credits = cr;
    }
};
```

```
    string getCourseName() {
        return courseName;
    }

    int getCredits() {
        return credits;
    }

    void displayCourseInfo() {
        cout << "Course ID: " << courseID
              << ", Name: " << courseName
              << ", Credits: " << credits << endl;
    }
};
```

```

class Student {
private:
    string studentName;
    Course courses[MAX_COURSES];
    int courseCount;

public:
    Student(string name) {
        studentName = name;
        courseCount = 0;
    }

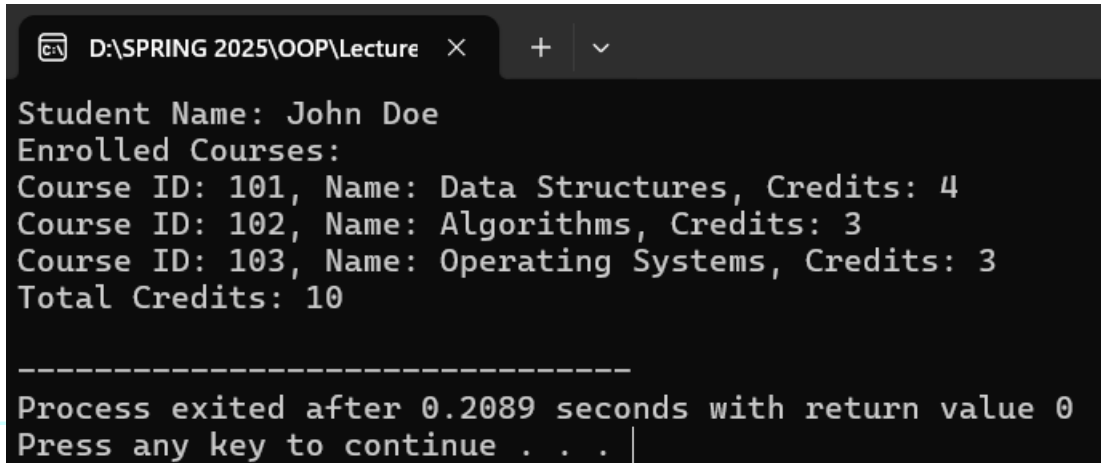
    void addCourse(Course &course) {
        if (courseCount < MAX_COURSES) {
            courses[courseCount] = course;
            courseCount++;
        } else {
            cout << "Cannot add more courses. Maximum limit reached." << endl;
        }
    }

    int calculateTotalCredits() {
        int totalCredits = 0;
        for (int i = 0; i < courseCount; i++) {
            totalCredits += courses[i].getCredits();
        }
        return totalCredits;
    }

    void displayStudentInfo() {
        cout << "Student Name: " << studentName << endl;
        cout << "Enrolled Courses:" << endl;
        for (int i = 0; i < courseCount; i++) {
            courses[i].displayCourseInfo();
        }
        cout << "Total Credits: " << calculateTotalCredits() << endl;
    }
};

```

```
int main() {  
    Course course1("Data Structures", 101, 4);  
    Course course2("Algorithms", 102, 3);  
    Course course3("Operating Systems", 103, 3);  
  
    Student student("John Doe");  
  
    student.addCourse(course1);  
    student.addCourse(course2);  
    student.addCourse(course3);  
  
    student.displayStudentInfo();  
  
    return 0;  
}
```



The screenshot shows a C++ IDE window titled "D:\SPRING 2025\OOP\Lecture". The output console displays the following text:

```
Student Name: John Doe  
Enrolled Courses:  
Course ID: 101, Name: Data Structures, Credits: 4  
Course ID: 102, Name: Algorithms, Credits: 3  
Course ID: 103, Name: Operating Systems, Credits: 3  
Total Credits: 10  
  
-----  
Process exited after 0.2089 seconds with return value 0  
Press any key to continue . . . |
```

Exercise

A university needs to maintain student academic records. Each student has a **Transcript** that contains detailed information about their academic progress, such as completed courses and grades. The **Transcript** is uniquely owned by the **Student** and cannot exist independently.

Task:

1. Create a **Student** object with the name "Jane Doe."
2. Update Jane's transcript to indicate completion of the course "**Data Structures**" with a grade of **A**.
3. Display the student's information along with their updated transcript record.

```

#include <iostream>
#include <string>
using namespace std;
class Transcript {
private:
    string record;
public:
    Transcript() {
        record = "No records available";
    }
    void setRecord(string rec) {
        record = rec;
    }
    void displayRecord() {
        cout << "Transcript Record: " << record << endl;
    }
};

```

```

int main() {
    Student student("Jane Doe");
    student.setTranscriptRecord("Completed Data Structures with A grade.");
    student.displayStudentInfo();
    return 0;
}

```

```

class Student {
private:
    string studentName;
    Transcript transcript;

public:
    Student(string name) {
        studentName = name;
    }
    void setTranscriptRecord(string record) {
        transcript.setRecord(record);
    }
    void displayStudentInfo() {
        cout << "Student Name: " << studentName << endl;
        transcript.displayRecord();
    }
};

```

Example TASK

Example. The BusStation class contains arrays of instances of the Bus and Car classes

In the example, the BusStation class (Bus Station) contains arrays of instances of the Bus and Car classes.

The number of elements in arrays of the Bus and Car classes can vary. Even if there is not a single bus or car at the bus station at a given time, the bus station will function.

This is?

Explained in Dev c++ file

Storage Classes

- To fully define a variable one needs to mention not only its ‘type’ but also its ‘storage class’.
- In other words, not only do all variables have a data type, they also have a ‘storage class’.
- storage classes have defaults, If we don’t specify the storage class of a variable in its declaration,
- There are basically two kinds of locations in a computer where such a value may be kept—
- Memory and CPU registers.
- It is the variable’s storage class that determines in which of these two locations the value is stored.

Storage Classes

Moreover, a variable's storage class tells us:

- (a) Where the variable would be stored.
- (b) What will be the initial value of the variable, if initial value is not specifically assigned.(i.e. the default initial value).
- (c) What is the scope of the variable; i.e. in which functions the value of the variable would be available.
- (d) What is the life of the variable; i.e. how long would the variable exist.

Storage-class Specifiers

1. Auto
2. Register
3. Extern
4. Static
5. Mutable

static storage class

Storage – Memory.

Default initial value – Zero.

Scope – Local to the block in which the variable is defined.

Life – Value of the variable persists between different function calls.

A simple function call

```
void staticDemo()
{
    int val = 0;

    ++val;

    cout << "val = " << val << endl;
}

int main()
{
    staticDemo();           // prints val = 1
    staticDemo();           // prints val = 1
    staticDemo();           // prints val = 1
}
```

Static Local Variables

```
void staticDemo()
{
    static int val = 0;

    ++val;

    cout << "val = " << val << endl;
}

int main()
{
    staticDemo();           // prints val = 1
    staticDemo();           // prints val = 2
    staticDemo();           // prints val = 3
}
```

Static Members of a C++ Class (Why?)

- We can define class members static using **static** keyword.
- When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.
- A static member is shared by all objects of the class.

Static Members of a C++ Class

It can be initialized outside the class using the scope resolution operator `::` to identify which class it belongs to.

All static data is initialized to zero when the first object is created, if no other initialization is present.

3 Objects of Cube Class with their individual instance variable

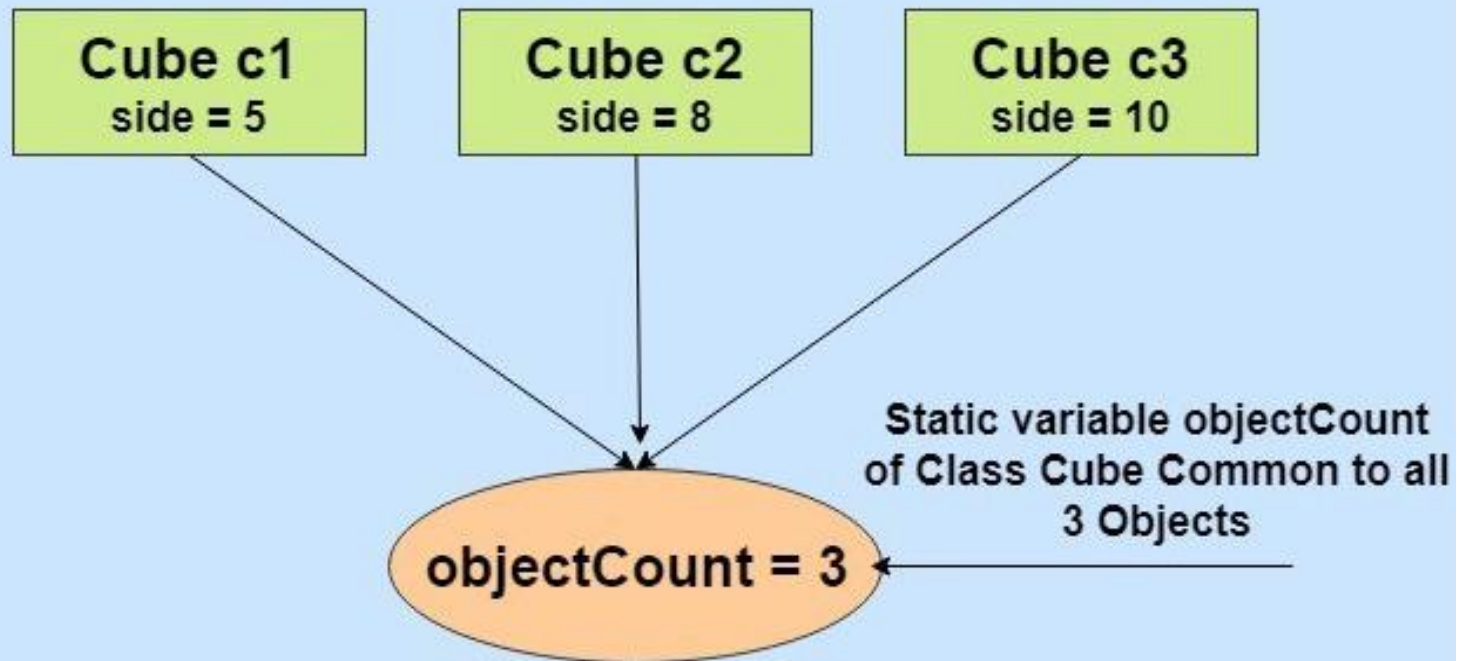
Cube c1
side = 5

Cube c2
side = 8

Cube c3
side = 10

Static variable objectCount
of Class Cube Common to all
3 Objects

objectCount = 3



```
graph TD; c1["Cube c1<br/>side = 5"] --> oc(["objectCount = 3"]); c2["Cube c2<br/>side = 8"] --> oc; c3["Cube c3<br/>side = 10"] --> oc; text["Static variable objectCount<br/>of Class Cube Common to all<br/>3 Objects"] --> oc;
```


Declaration

```
static data_type member_name;
```

Defining the static data member

It should be defined outside of the class following this syntax:

- **data_type class_name :: member_name =value;**

Concept: Static Members can be initialized outside the class

```
1 #include <iostream>
2 using namespace std;
3
4 class Student {
5     public:
6         int rollnumber, age;
7         static int m = 1; //Error
8 };
9 int main() {
10     Student S1;
11     return 1;
12 }
```

```
1 #include <iostream>
2 using namespace std;
3
4 class Student {
5     public:
6         int rollnumber, age;
7         static int m;
8 };
9
10 int Student::m=1; //Correct Initialization
11
12 int main() {
13     Student S1;
14     return 1;
15 }
```

Concept: A static member is shared by all objects of the class

```
56 #include <iostream>
57 using namespace std;
58 class student
59 {
60     public:
61         int roo, age;
62         static int m;
63
64 };
65 int student :: m=1;
66 int main (){
67
68     student s1;
69     s1.m = 5;
70     student s2;
71     cout <<s1.m<<endl<<s2.m<<endl;
72     s2.m=10;
73     cout <<s1.m<<endl<<s2.m<<endl;
74     student s3;
75     cout<<s3.m;
76     return 1;
77 }
```

```
5
5
10
10
10
-----
Process exited after 0.01155 seconds with return value 1
Press any key to continue . . .
```

Static Member Functions of a C++ Class

By declaring a function member as static, you make it independent of any particular object of the class

A static member function can be called even if no objects of the class exist and the **static** functions are accessed using only the class name and the scope resolution operator ::.

Static Member Functions of a C++ Class

A static member function can only access static data member, other static member functions and any other functions from outside the class.

Concept: A static member function can only access static data member

```
1 #include <iostream>
2 using namespace std;
3
4 class Student {
5     public:
6         int rollnumber, age;
7         static int m;
8
9         static modifyAge () //Error
10        {
11            age = 10;
12        }
13};
```

```
1 #include <iostream>
2 using namespace std;
3
4 class Student {
5     public:
6         int rollnumber, age;
7         static int m;
8
9         static modifyAge () //Not an Error
10        {
11            m = 10;
12        }
13};
```

Concept: A static member function can only access other static member functions

```
1 #include <iostream>
2 using namespace std;
3
4 class Student {
5     public:
6         int rollnumber, age;
7         static int m;
8         void print()
9     {
10         cout<<"Hello"<<endl;
11     }
12     static modifyAge ()
13     {
14         m = 10;
15         print(); //Error
16     }
17 };
```

```
1 #include <iostream>
2 using namespace std;
3
4 class Student {
5     public:
6         int rollnumber, age;
7         static int m;
8         static void print()
9     {
10         cout<<"Hello"<<endl;
11     }
12     static modifyAge ()
13     {
14         m = 10;
15         print(); //Not an Error
16     }
17 };
```

Difference between a static variable and global variable

- All global variables are static, but all static variables are not necessarily global.
- static variables have visibility limited to the function in which declared.
- What all static variables (including global) have in common is that they have a lifetime that lasts the duration of the program. They are initialized just once, when the program begins.