



Association, Aggregation, and Composition

Department of Software Engineering

National University of Computer & Emerging Sciences,
Islamabad Campus



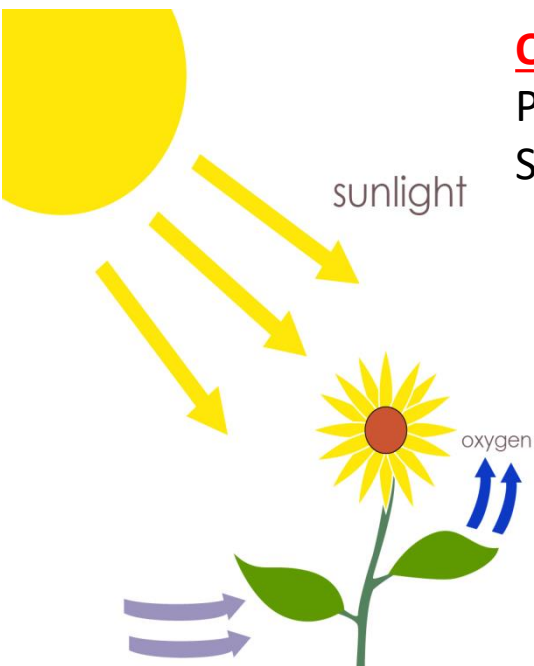
Part 1

Identifying relationships and their types



Identifying objects

- **Identifying** objects / classes in real life
- How is this identification possible?



Objects

Plant (Flower, Leaves)
Sunlight



Objects

Car (Body, Steering wheel, tires, engine)
Driver



Interaction between objects

Plant Example

- Plant **is composed** of leaves and flowers
- Leaves **interact** with the sunlight (even if you don't know how, exactly),
- Flower's **existence depends on** the plant.

Car Example

- Car **is composed** of body, steering wheel, tires, engine etc.
- Driver **interacts** with the car



Relationship b/w objects

- **What** is a **relationship** between **objects**?
 - Whenever an object **interacts** with another...
- In real life
 - **Object** in **real life** have **relationships** with each other
 - Only by understanding these we **understand** the **behavior** of these **objects**
- In programming
 - **Objects** also have **relationship**, **hierarchies** among them
 - Understanding these relationships help in writing **reusable** and **extensible** code – **HOW?**



Identifying relationships b/w objects

- In case of real life we identify these relationships with experience and years of training
- How can we identify relationships between programming objects?
 - Same as in real life!
 - Class Responsibility Collaborator (CRC) Cards (CRC)



CRC Cards

- CRC card is divided into three sections: **Class Name**, **Responsibilities**, and **Collaborators**
 - Class** represents collection of similar objects
 - Object** is a person, thing, place, event, concept, screen or report
 - Responsibility** – is anything that **a class knows** or **does**
 - Collaborator** – when **a class doesn't have information** to fulfill a responsibility it needs to collaborate

Sample CRC Card

Class Name	
Responsibilities	Collaborators



CRC Modelling - example

Inventory Item	
Item number	
Name	
Description	
Unit Price	
Give price	

Order	
Order number	Order Item Customer
Date ordered	
Date shipped	
Order items	
Calculate order total	
Print invoice	
Cancel	

Order Item	
Quantity	Inventory item
Inventory item	
Calculate total	

Customer	
Name	Order Surface address
Phone number	
Customer number	
Make order	
Cancel order	
Make payment	

Surface Address	
Street	
City	
State	
Zip	
Print label	



CRC Cards - Example

- CRC cards helped in identifying relationships:
 - Order item – Inventory item
 - Order – Order item
 - Order – Customer
 - Customer – Street address
 - Customer – Order
- How are these converted into interactions between programming objects?



Relationship type words

- There are special “**relationship type**” words to describe these relationships. These are:

- part-of
- has-a
- uses-a
- depends-on
- member-of
- is-a

Can we use these words to describe relationships we identified?

How are these words useful in context of C++ classes?

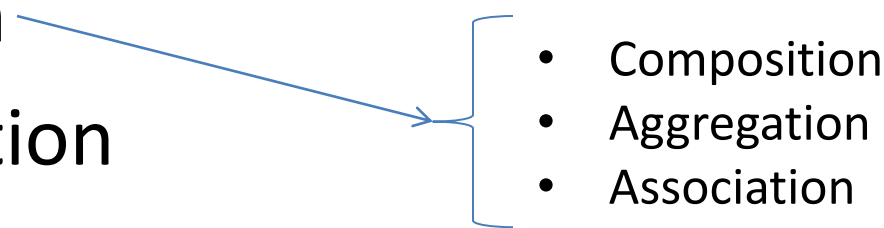


Other examples of relationship words

- For example:
 - a square “**is-a**” shape
 - a car “**has-a**” steering wheel
 - a computer programmer “**uses-a**” keyboard
 - a flower “**depends-on**” a bee for pollination
 - a student is a “**member-of**” a class
 - Your brain exists as “**part-of**” you
- All of these **relation types** have useful analogies in **C++**.



Types of Relationships

- Association
 - Generalization
- 
- Composition
 - Aggregation
 - Association

and

In these two objects are not really related!

- Dependency
- Realization

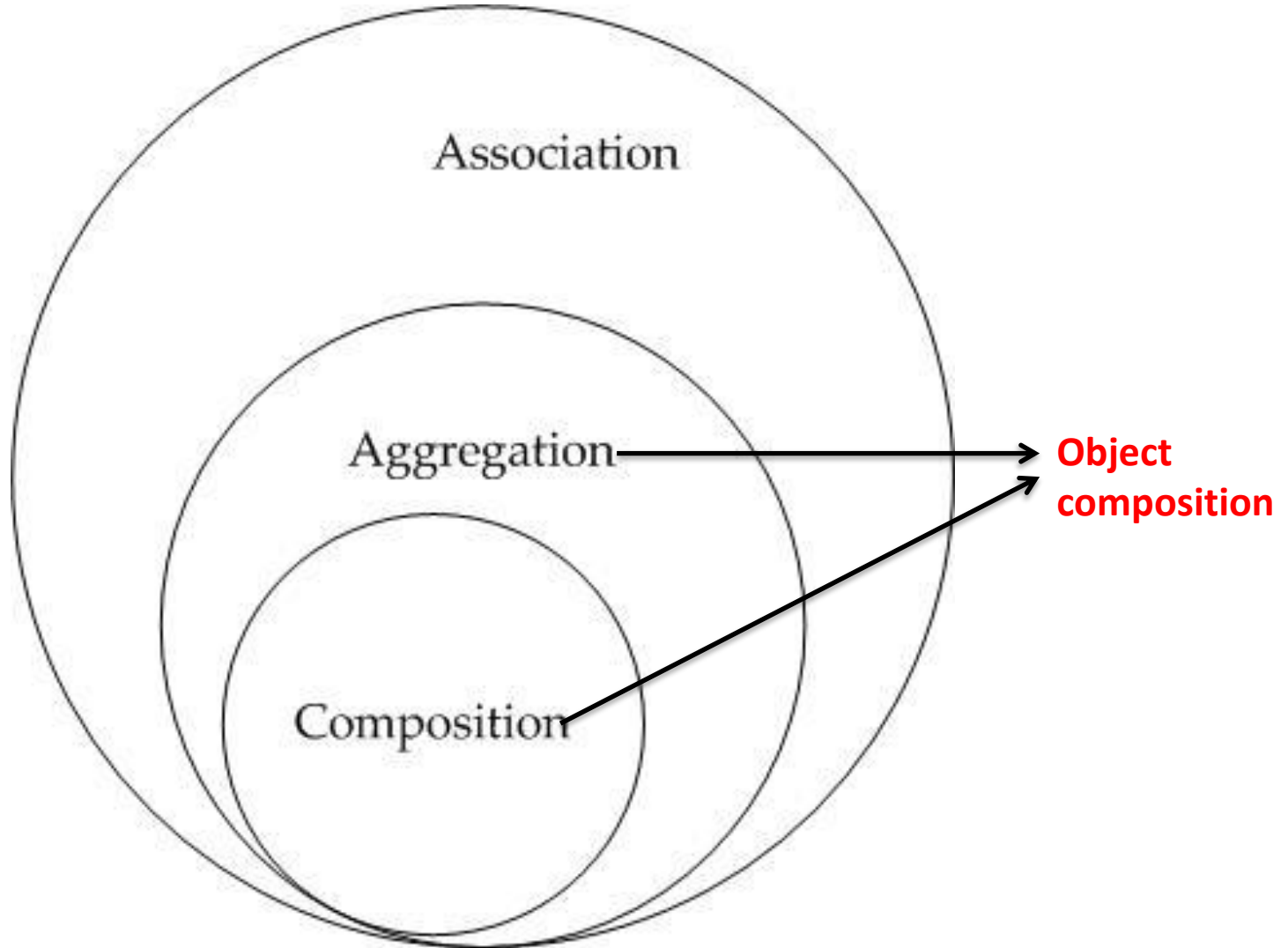


Relationships between Objects

Weak

Strength of relationship

Strong





Part 2

Composition



Object composition

- Process of building **complex objects** from **simpler ones** is called **object composition**
- This relationship is described using “**has-a**” word
 - Car – engine, steering wheel, frame etc.
 - Computer – CPU, motherboard, memory etc.
- **Complex part** is called the **whole**
- **Simpler object** is called the **part**



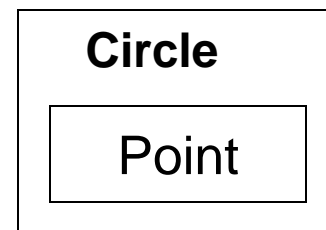
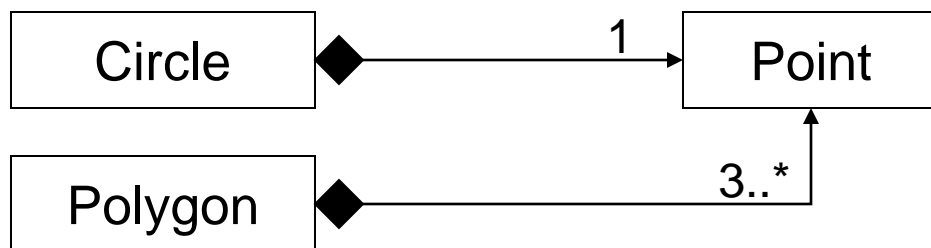
Types of object composition

- Two types
 - Composition
 - Aggregation



Composition

- Composition models “**part-of**” relationships
- These relationships are **part-whole** relationships
- Composition is often used to model **physical relationships**, where one object is physically contained inside another.
 - Heart is **part-of** body
 - Fish are **part-of** pond





Composition tests

- The **part** (member) is part of the object (class)
- The **part** (member) can only belong to **one object** (class) at a time
- The **part** (member) has its **existence** managed **by the object** (class)
- The **part** (member) does not know about the **existence of the object** (class) - **Directional**



Fraction Class (Example)

```
class Fraction
{
    private:
        int c_num;
        int c_den;
    public:
        Fraction(int n=0, int d=1): c_num{n}, c_den{d}
        {}
};
```



Engine is a part-of Car (Example)

```
class Car
{
public:
    Car(char* e_No){
        cout << "Car created" << endl;
        ptr_engine = new Engine(e_No); //Engine created
    }
    void disp(){
        cout << ptr_engine->getEngineNumber() << endl;
    }
    ~Car() {
        cout << "\nCar destroyed" << endl;
        delete ptr_engine; //engine destroyed/deleted
    }
private:
    Engine* ptr_engine;
};
```

Car

Engine



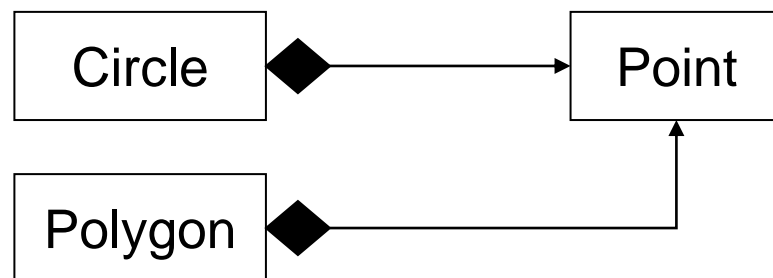
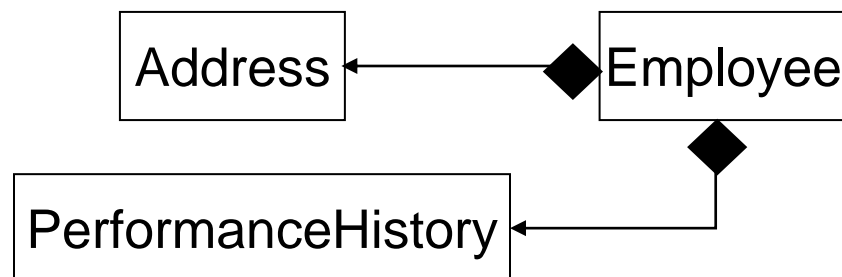
Composition variants

- Whole **creates** the parts and **destroy** them **BUT** it can do it **indirectly** as well
 - **Deferring creation** of parts For example, a string class may not create a dynamic array of characters
 - Instead of creating part, whole can opt to **use a part** that has been **given to it as input**
 - Whole can **delegate destruction** of its parts (e.g. to a garbage collection routine).
- The key point here is that the composition should manage its parts.



Composition and subclasses

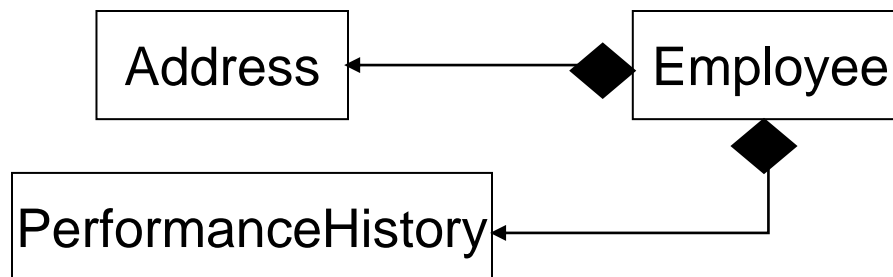
- When/why create a subclass instead of direct implementation of a feature?
 - Car (whole) Engine (part) example
- Composition → subclass
 - **Each individual class** should be focused on performing **one task** (simple and straight forward)
 - Each subclass can be **self-contained**, which makes them **reusable**.
 - The **parent class** can focus only on **coordinating the data flow** between the subclasses.





Composition and subclasses

- **Subclass** or **direct implementation**?
 - One class one task
 - Task can be
 - storage and manipulation
 - coordination





Composition - recap

- Relationship between objects
 - Association
 - Object composition (**Composition** and **Aggregation**)
- **Object composition** is the process of **creating complex objects** from **simpler one**.
- Composition (models **part-of** relationship)
 - Whole is responsible for **existence** of part



Part 3

Aggregation

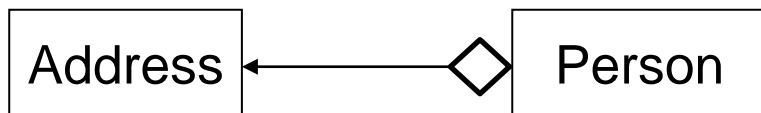


Aggregation

- An **aggregation** is also a **part-whole** relationship
- It models **has-a** relationship
- Similar to composition
 - The **parts** are **contained** within the **whole**
 - It is also a **unidirectional** relationship
- Unlike composition
 - Parts **can belong to more than one** object at a time
 - Whole **is not responsible for the existence** and lifespan of the parts

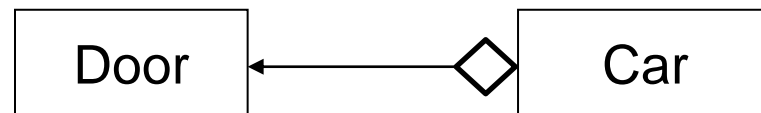


Aggregation



Singular part

- Every person has an address.
- One address can belong to more than one person at a time
- Address existed before the person starting living at the address
- Whole knows of existence (person knows)
- Part doesn't know about the whole



Multiplicative parts

- A car door is part of the car.
- Door belongs to the car,
- It can belong to other things as well, like the body of the car.
- The car is not responsible for the creation or destruction of the door.
- Whole knows about existence
- Part doesn't know about the whole



Aggregation tests

- The part (member) **is part of** the object (class)
- The part (member) **can belong to more than one object** (class) at a time
- The part (member) **does not** have its **existence** managed by the object (class)
- The part (member) **does not know** about the existence of the object (class)



Implementing aggregation VS composition

- Aggregation
 - **Parts** are **added** as **references** or **pointers**
 - **Whole** is **not responsible** for creation and deletion
 - Whole takes the objects it is going to point to as: 1) **constructor parameters**; 2) **parts are added later via access functions**
 - Parts exists **outside the scope** of whole
- Composition
 - **Parts** are **added** as **normal variables** (or pointers)
 - **Whole** is **responsible** for **creation** and **deletion**



Examples

Composition

```
class Part{  
    //class implementation  
};
```

```
class Whole {  
    private:  
        Part* p; //can be normal variable  
    public:  
        Whole() {  
            this->p = new Part();  
        }  
        ~Whole(){  
            delete p;  
        }  
};
```

```
int main()  
{  
    Whole w;  
}
```

Aggregation

```
class Part{  
    //class implementation  
};
```

```
class Whole {  
    private:  
        Part* p;  
    public:  
        Whole(Part *p) {  
            this->p = p;  
        }  
};
```

```
int main()  
{  
    Part* p = new Part();  
    Whole w(p);  
}
```



Person has an Address - Example

```
class Address
{
    private:
        int h_No; //house no
        int st_No; //street no
        string sector; //sector
        string city; //store city
    public:
        //parameterized constructor
        Address(int h, int s, const string& sec, const string& c)
        { }
};
```



Person has an Address - Example

```
class Person
{
    private:
        string p_name; //person name

        //it will get reference to address object (part)
        const Address& p_address; // A person can live at only one address (here)

    public:
        //parameterized constructor
        Person(const string& s, const Address& address) : p_name{s}, p_address{ address }
        { }

        //display person details
        void disp_Person() const{
            cout << "Name: " << p_name << "; ";
            p_address.disp_Address();
        }
};
```




Person has an Address - Example

```
int main()
{
    //part object created
    Address part_Object( 12, 3, "G-20", "Islamabad" );

    //whole object created
    Person whole_Object("Random person", part_Object );

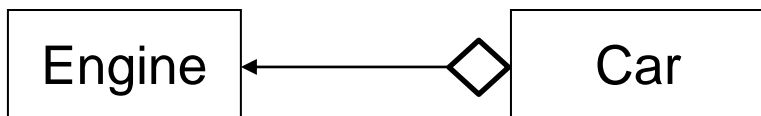
    whole_Object.disp_Person();

    return 0;
}
```

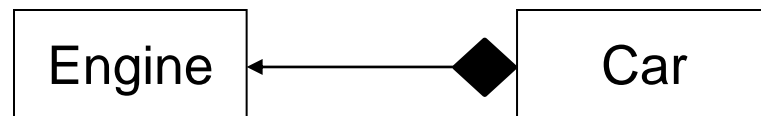


Aggregation or Composition

- When to do what?



Repair shop software



Car performance software

Implement the simplest relationship that meets your needs!!!

Not the one that seems like it would fit best in a real-life context.



Aggregation/Composition - recap

- Object composition
 - Composition
 - Aggregation
- Used to model relationships where a whole is built from one or more parts