Object Relationships

Composition and Beyond…

Composition

- Imagine that we want to build a **Car** class
  - We also decide to implement a **SteeringWheel** class

```
Class Car {
  private:
    SteeringWheel sw;
    Engine engine;
    string paintColor;
}
```

- The **SteeringWheel** is "part-of" the **Car**
- Note that no other car has this specific steering wheel!

Composition

- Member's existence managed by object (class)
  - Member variable's lifetime is bound to the lifetime of its owner
  - e.g. When the **Car** is destructed, the **SteeringWheel** is also destroyed
- Member can only belong to one object (class) at a time
- Member doesn't know about the object class
  - Unidirectional relationship
  - E.g. The **SteeringWheel** object does not know anything about the **Car**

Aggregation

- Consider a **Person** class
  - How could we utilize an **Address** class?

```
Class Person {
  private:
    string name;
    Address* addr;
}
```

- Multiple people might reside at the same address
  - By utilizing a pointer, we can share a single address between instances
  - Each person “has an” address

Aggregation

- Member's existence is **not** managed by object
  - If a person relocates to a different address, the old address still exists
  - The member is created outside the scope of the class
- Member can belong to multiple objects at a time
  - Typically implemented with pointers
- Member doesn't know about the object class
  - Unidirectional relationship
  - An **Address** object does not know about a **Person** object

Association

- This is what happens when two objects are aware of each other, (and may call functions) but are otherwise unrelated
- "**uses-a**" relationship
- Member can belong to multiple objects at a time
  - Typically implemented with pointers
- Member may or may not know about the object class
  - Unidirectional or bidirectional relationship

Introduction to Inheritance

- Suppose that we implement two C++ classes with the following member variables:
  - Student
    - ID number
    - Email address
    - Phone number
    - Major of Study
    - GPA
  - Teacher
    - ID number
    - Email address
    - Phone number
    - Office number
    - Office hours
    - Salary

Is there a better way to handle this?

Inheritance

- “**is-a**” relationship

- A student **is a** person
- A teacher **is a** person
- Student and Teacher class can both inherit from the Person class

Basics of Inheritance

- Classes that inherit properties are “derived classes”
  - Also known as “child” class
- The “parent” class is referred to as a “base class”
- Helps us avoid re-inventing the wheel
  - If a Student and an Instructor are both derived classes, we don’t need to write the same code twice
  - **Person** class could hold any redundant information

Inheritance cont…

- Inheritance is not limited to a single level
  - Let’s add an Employee class to the hierarchy