Relationships Among Classes

Relationship among classes

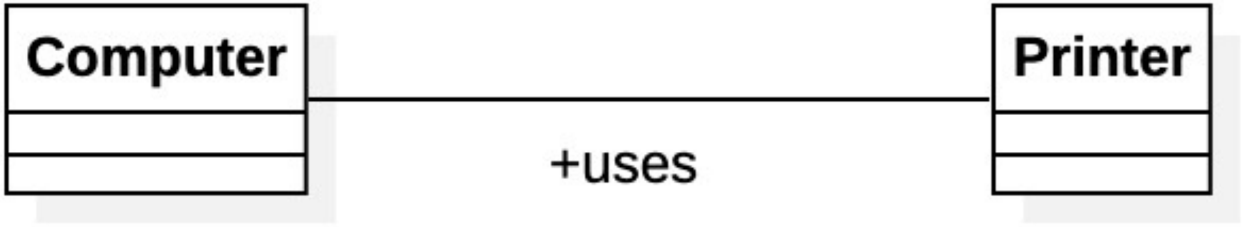
- There are three major type of relationships among classes that most of O.O. programming languages support them:
  - Association
  - Aggregation/Composition
  - Inheritance

Association

- The association relationship expresses a semantic connection between classes:
  - There is no hierarchy.
- It is a relationship where two classes are weakly connected; i.e. they are merely "associates."
  - All object have their own lifecycle;
  - There is no ownership.
  - There is no whole-part relationship.
- In another words, if inheritance and aggregation/composition doesn't apply, it is most likely a simple association

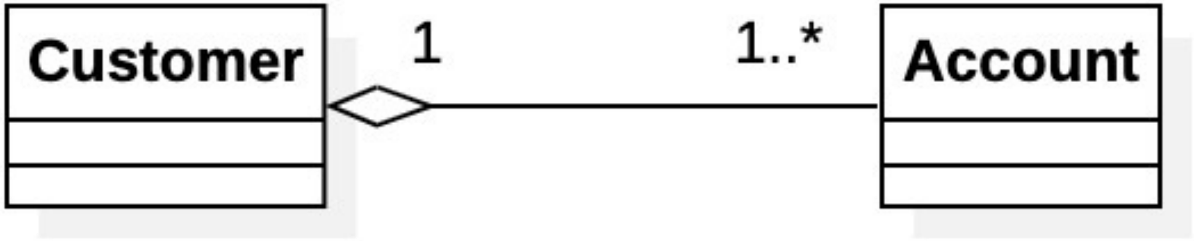
Association (Review)

- The association of two classes must be labeled.
- To improve the readability of diagrams, associations may be labeled in active or passive voice.



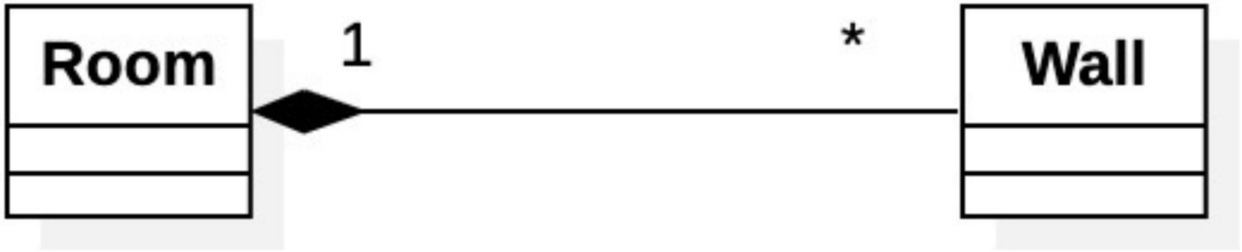
Aggregation (Review)

- An aggregation represents an asymmetric association, in which one of the ends plays a more important role than the other one.
- The following criteria imply an aggregation:
  - A class is part of another
  - The objects of one class are subordinates of the objects of another class
- Denotes a whole/part hierarchy with the ability to navigate from whole (aggregate) to its parts (attributes).
- The part is normally being referenced to either a pointer or by a reference in C++



Aggregation (Review)

- A strong type of aggregation, when deletion of the whole causes the deletion of the part, is called ***composition.***
- The “part” objects are usually created in the constructor of the container class.



Review of Implementation of Aggregation in Java

Composition and Aggregation in Java:

- **Consider the following definition of class Engine in Java:**

```
class Engine {
    private int size;
    public Engine (int size){
        this.size = size;
    }
    …
    // ASSUME MORE METHODS HERE
}
```

Now let's assume we need to design a class in Java, called Car to contain an object of class Engine.

- **How can we implement Composition?**
- **How can we implement Aggregation?**

Here is a possible solution to implement Composition in Java:

```
class Engine {
      private int size;
      public Engine (int size) {
                   this.size = size;
      }
     …
       // ASSUME MORE METHODS HERE
}

class Car {
      private Engine engine;
      public Car (int size) {
          engine = new Engine(size);

      }
     …
       // ASSUME MORE METHODS HERE
}
```

Notice that lifetime
of engine depends
on lifetime of car

Here is a possible solution to implement Aggregation in Java:

```
class Engine {
   private int size;
   public Engine (int size) {
          this.size = size;
   }
      ...
    // ASSUME MORE METHODS HERE
}

class Car {
   private Engine engine;
   public Car (Engine e){
       engine = e;

   }
      ...
    // ASSUME MORE METHODS HERE
}
```

Notice that lifetime of engine, doesn't depend on lifetime of car

Implementation of Aggregation and Composition in C++

Aggregation and Composition in C++

- In principle, the concepts of aggregation and composition in C++ is identical to Java.
- However, the main difference stems from the fact that objects in Java are always created dynamically on the “heap”, but in C++ an aggregated object can be statically allocated at the compilation time, or dynamically at the runtime.

```
// Java:
class Inner { …}
class outer {
   private Inner x;
   // notice that x is reference
   // similar to a pointer in C++
}
```

```
// C++:
class Inner { …};
class outer {
   private:
       Inner x;
       // x is an object,
       //not a pointer in C++
}
```

A C++ version of the Java program, in previous slides, will be discussed in details during lectures.