Object-Oriented Programming

Week 4

Object Oriented Programming

- Object-Oriented Programming (OOP) is a paradigm or pattern of programming whereby the solution to a programming problem is modeled as a collection of collaborating objects.
- OOP is an approach to problem solving where all computations are carried out using objects.
- The object-oriented paradigm allows us to organize software as a collection of objects that consist of both data and behavior.
- A class is a blueprint of an object. It is a concept, and the object is the embodiment of that concept.
- An object is an entity that possesses both state (or properties or attributes) and behavior.
- An Object is an instance of a class
- The data is usually hidden from other objects so that the only way to affect the data is through the object's functions (or methods)

Object Oriented Class

- Think of a real world object such as a Mobile phone.
- The Mobile phone has several features and properties
  - We do not need to open the case to be able to use it
  - It has controls such as buttons and screen
  - We can still understand the concept of a mobile phone even if its brand or shape change
  - It is complete and functional when we buy it
  - The mobile phone will not crash

Object Oriented Class

- A class should
  - Provide a well-defined interface - such as the keys and screen.
  - Represent a clear concept of a general mobile phone
  - Be complete and well documented
  - The code should be robust
- Classes have two components:
  - States - are the values that the object has.
  - Methods - are the ways in which the object can interact with its data, the actions.

Objects

- An object represents an entity in the real world that can be distinctly identified.
- For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.
- An object has a unique identity, state, and behaviors.
- The state of an object consists of a set of data fields (also known as properties) with their current values.
- The behavior of an object is defined by a set of methods.
- Classes are constructs that define objects of the same type.
- A C++ class uses variables to define data fields and methods to define behaviors.

Object Oriented Analysis and Design

- Object-Oriented Analysis: that phase of program development when the program functionality is determined from the requirements
- It includes
  - identification of classes and objects
  - definition of each class's attributes
  - definition of each class's behaviors
  - definition of the relationship between classes

Identify Classes and Objects

- Consider the major data elements and the operations on these elements
- Candidates include
  - user-interface components (menus, text boxes, etc.)
  - I/O devices
  - physical objects
  - historical data (employee records, transaction logs, etc.)
  - the roles of human participants
- Attributes are the data elements of an object of the class, they are necessary for the object to work in its role in the program
- Behavior, for each class, Identify what an object of a class should do in the program
- The behaviors determine some of the member functions of the class

Finding Classes

- Write a description of the problem domain (objects, events, etc. related to the problem)
- List the nouns, noun phrases, and pronouns.
- These are all candidate objects
- Refine the list to include only those objects that are relevant to the problem
- Problem Domain is the set of real-world objects, and major events related to the problem.

Problem Domain Description Example

Joe's Automotive Shop services foreign cars and specializes in servicing cars made by Mercedes, Porsche, and BMW. When a customer brings a car to the shop, the manager gets the customer's name, address, and telephone number. The manager then determines the make, model, and year of the car, and gives the customer a service quote. The service quote shows the estimated parts charges, estimated labor charges, sales tax, and the total estimated charges.

Identify all the Nouns

Joe's Automotive Shop services **foreign cars** and specializes in servicing **cars** made by **Mercedes**, **Porsche**, and **BMW**. When a **customer** brings a **car** to the **shop**, the **manager** gets the **customer's name**, **address**, and **telephone number**. The **manager** then determines the **make**, **model**, and **year** of the **car**, and gives the **customer** a **service quote**. The **service quote** shows the **estimated parts charges**, **estimated labor charges**, **sales tax**, and the **total estimated charges**.

Nouns (no duplicates)		
Address	Foreign cars	Porsche
BMW	Joe's Automotive shop	Sales tax
Car	Make	Service quote
Cars	Manager	Shop
Customer	Mercedes	Telephone number
Estimated labor charges	Model	Total estimated charges
Estimated parts charges	Name	year

Refine the List of Nouns

- Some of the nouns mean the same thing (Car, Cars, Foreign cars)

Nouns (no duplicates)		
Address	~~Foreign cars~~	Porsche
BMW	~~Joe's Automotive shop~~	Sales tax
~~Car~~	Make	Service quote
Cars	Manager	**Shop**
Customer	Mercedes	Telephone number
Estimated labor charges	Model	Total estimated charges
Estimated parts charges	Name	year

Refine the List of Nouns

- Some nouns might represent items thar we do not need to be concerned with when solving the problem

Nouns (no duplicates)		
Address		Porsche
BMW		Sales tax
	Make	Service quote
Cars	~~Manager~~	~~Shop~~
Customer	Mercedes	Telephone number
Estimated labor charges	Model	Total estimated charges
Estimated parts charges	Name	year

Refine the List of Nouns

- Some of the nouns might represent objects, not classes

Nouns (no duplicates)		
Address		~~Porsche~~
~~BMW~~		Sales tax
	Make	Service quote
Cars		
Customer	~~Mercedes~~	Telephone number
Estimated labor charges	Model	Total estimated charges
Estimated parts charges	Name	year

Refine the List of Nouns

- Some of the nouns might represent simple values that can be stored in a variable and do not require a class

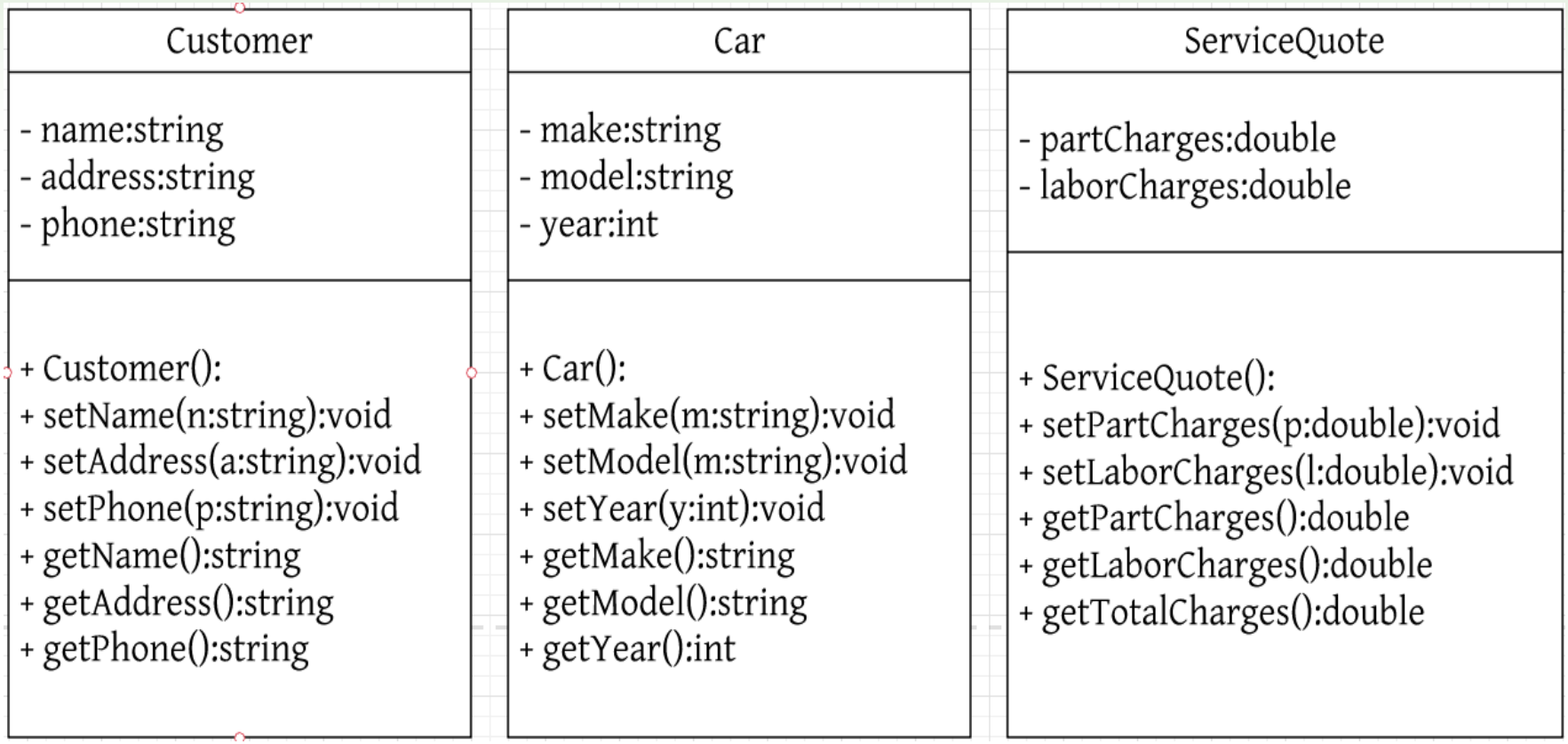
Nouns (no duplicates)		
~~Address~~		
		~~Sales tax~~
	~~Make~~	Service quote
Cars		
Customer		~~Telephone number~~
~~Estimated labor charges~~	~~Model~~	~~Total estimated charges~~
~~Estimated parts charges~~	~~Name~~	~~year~~

Identify Class Responsibilities

- Class responsibilities are
  - The things that the class is responsible for knowing (class attributes)
  - The actions that the class is responsible for doing (class methods)

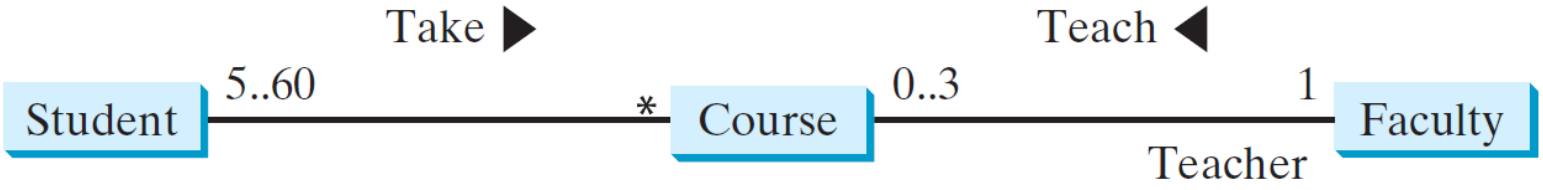
Classes
Service quote
Cars
Customer

The Classes: class UML Diagram



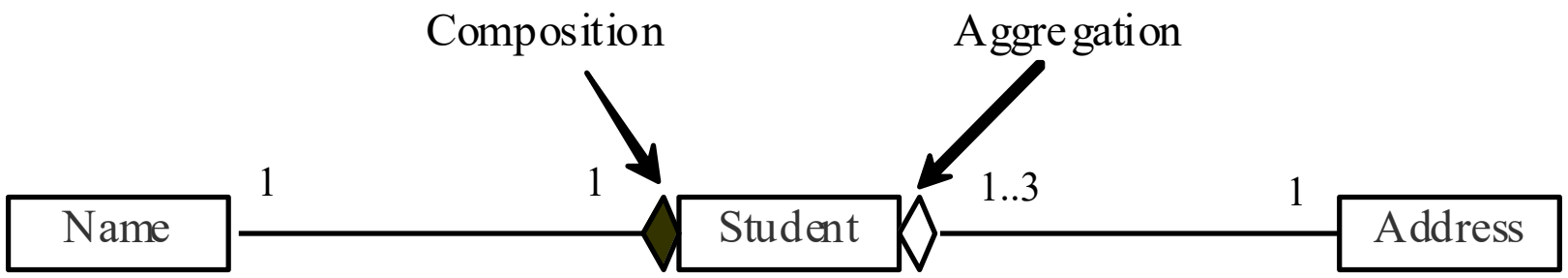
Class Relationships: Association

- Association: is a general binary relationship that describes an activity between two classes.
- It is a relationship where all objects have their own lifecycle and there is no owner.
- A student may take any number of courses,
- A faculty member may teach at most 3 courses,
- Course may have from 5 to 60 students, and
- A course is taught by only one faculty member.



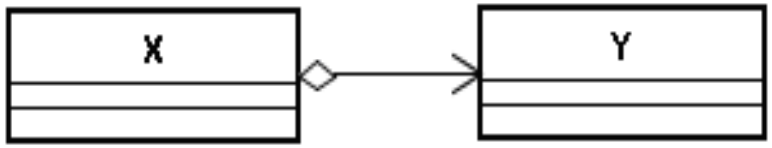
Class Relationships: Aggregation

- **Aggregation**: is a special form of association that represents an ownership relationship between two objects.
- Aggregation models has-a relationships.
- The owner object is called an **aggregating object** and its class an **aggregating class**.
- The subject object is called an **aggregated object** and its class an **aggregated class**.
- A Student object has an Address object, but an Address object can be shared by up to 3 Student objects (3 students rent the same house).
- In UML, an empty diamond is attached to an aggregating
- class (Student) to denote the aggregation relationship.



Class Relationships: Aggregation

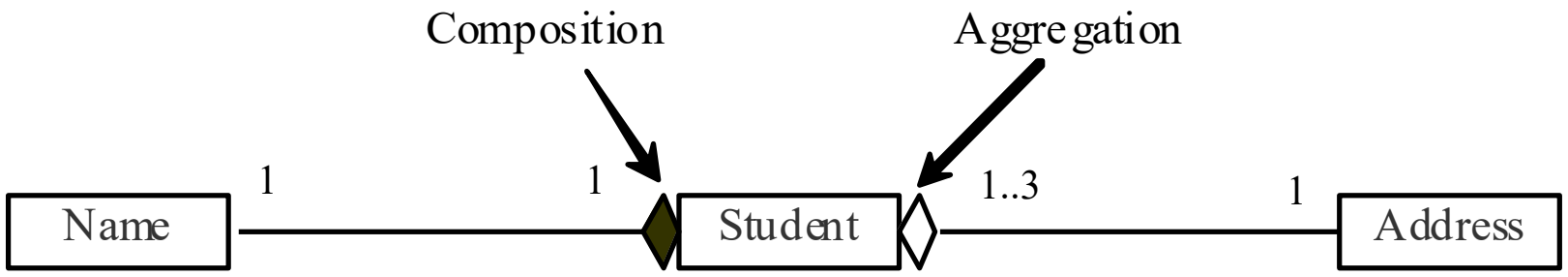
- If a C++ class declares a pointer to some type as a member or local to some member function, that pointer may, at any time, be attached to an instance of the pointed to type on the C++ heap by calling new.
- This must be done in some member function, and so the aggregating class will only own an instance of the aggregated type if that function is called.
- So aggregation is a weaker form of ownership - it does not guarantee that the aggregated possess that part.




```
class X {
 :
 void f1(){
  :
  Y p = new Y();
  :
  delete p;
  :
 }
 :
};
```

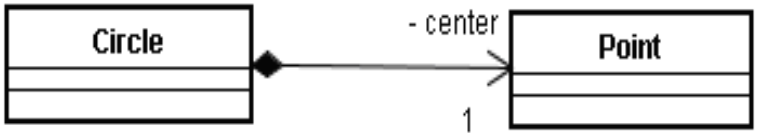
Class Relationships: Composition

- **Composition**: is actually a special case of the aggregation relationship whereby an object is exclusively owned by an aggregating object.
- A Name object is owned by one Student object only.
- In UML, a filled diamond is attached to an aggregating
- class (Student) to denote the composition relationship.



Class Relationships: Composition

- When class C composes an instance of class P, it acquires the capabilities of P to support the implementation of its own methods. Composition is a strong ownership relationship.
- The C++ language directly supports the composition of all types, both primitive language types and also user defined types.
- Being able to compose and efficiently copy instances of user defined types is a significant advantage for C++.




```
class Circle
{
private:
    Point center;
    :
    :
};
```

Class Representation

- An aggregation relationship is usually represented as a data field in the aggregating class.

```
public class Name {
  ...
}
```

Aggregated class

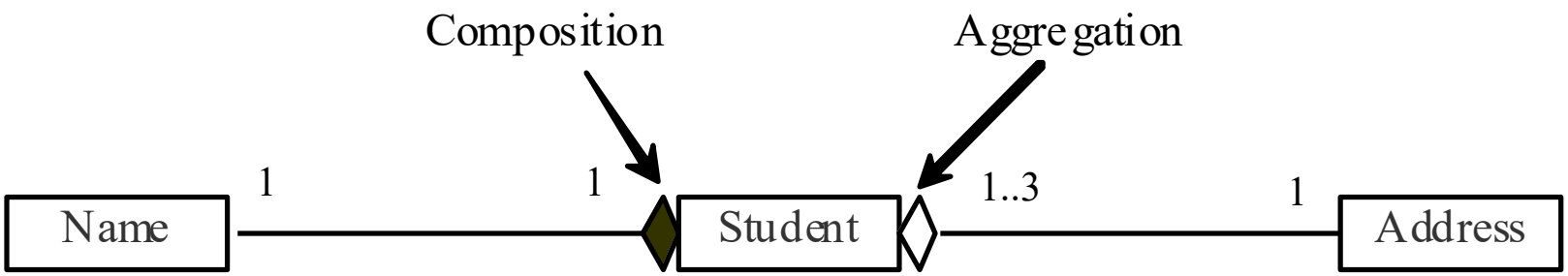
```
public class Student {
  private Name name;
  private Address address;

  ...
}
```

Aggregating class

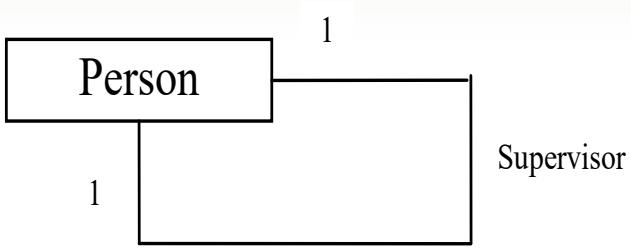
```
public class Address {
  ...
}
```

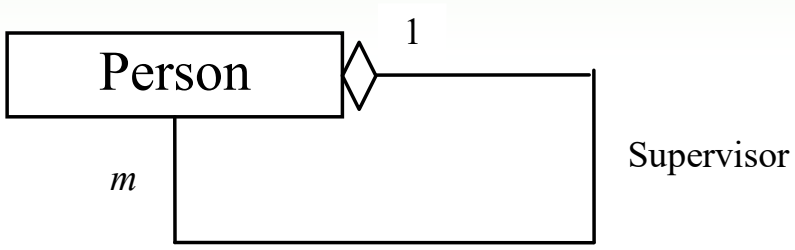
Aggregated class



Aggregation or Composition between the same class

- Aggregation may exist between objects of the same class. For example, a person may have a supervisor.



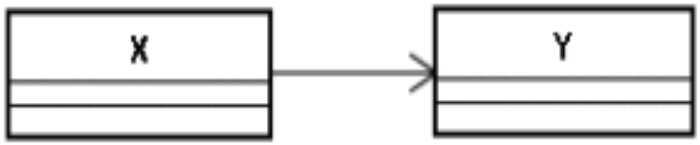



```
public class Person {
  // The type for the data is the class itself
  private Person supervisor;
  ...
}
```

```
public class Person {
   ...
   private Person[] supervisors;
}
```

Uses

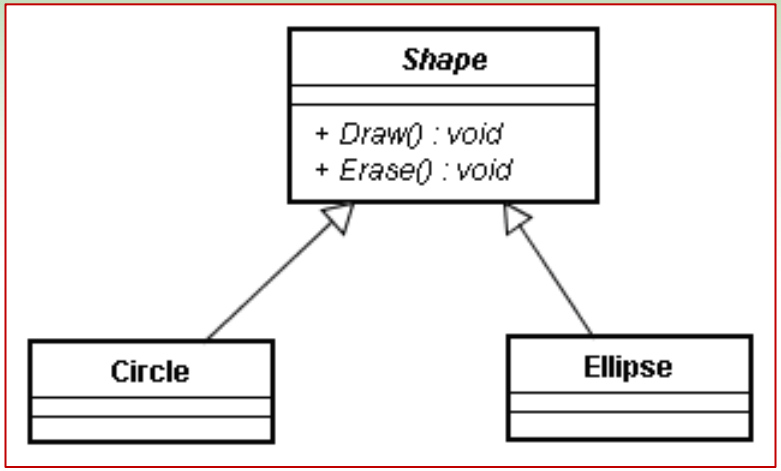
- If a member function of a C++ class is passed a pointer or reference to an existing instance of another type we say that an instance of the class is using the instance of the referenced type.
- It does not own the instance and should not destroy it when finished as the instance belongs to some other entity in the program's code.




```
class X {
 ...
 void f1(Y &y){
    :
    y.Foo();
    :
 }
 :
};
```

Inheritance

- When class D inherits from class B.
- it acquires the public interface of B.
- All of the public non-constructor/destructor methods of B become methods of D.
- Any code that uses a pointer or reference to a B instance will continue to build and function if we replace the pointer or reference with one bound to the derived D instance.
- This is know as Liskov substitution, and is one of the most useful properties of inheritance.
- It enables us to build very flexible applications that can easily be modified without breaking a lot of code.
- The Parser application is a good example.




```
class Circle: public Shape
{
    :
};

class Ellipse: public Shape
{
    :
};
```