A Summary on Class Relationships

The relationship among classes can be mainly classified as:

- Association ("uses-a")
- Aggregation/Composition ("has-a")
- Inheritance ("is-a")

In this section we will discuss the first three relationships, and the last one, inheritance, will be discussed in the next lab.

Association:

A typical example of this relationship is that one class uses another class. For example, the relationship between class Person and class Computer is a good example of association relationship. We say that a "Person uses a Car". Note that two classes with association relationship don't have a strong relationship such as one object being part-of or kind-of the other object.

Now let's assume we want to implement the relationship between a car and a person. In this case our intention is to implement a simple association relationship, where a person doesn't necessarily own a car but can drive or use it:

```
class Person {
       private String name;
       private Car aCar;
       public Person (int id, String name) {
              this.id = id;
              this.name = name;
       }
       public void setDriver (Car anyCar) {
              aCar = anyCar;
      }
       //  ASSUME MORE CODE HERE
}

class Car{
       private Engine engine;
       public Car(int size, char type){
              engine = new Engine(size, type)
       }
       // ASSUME MORE METHODS HERE
}
```

Aggregation/Composition:

This relationship that is also called a "part of" or "has-a" represents a stronger relationship, compared to association. The relationship between an order and items in the order is an aggregation relationship. An "`Order`" object contains "`Item`" objects.

Conceptually there is a difference between aggregation and composition. Composition is a stronger type of aggregation; which means the part is strongly dependent on the container. In other words, the part’s lifetime depends on its container. Or, we can say that the ‘whole’ is responsible for creation and destruction of the ‘part’. As an example the relationship between a Room and its Walls is considered a composition relationship. The lifetimes of the walls are strongly dependent on the lifetime of the room. When we destroy the room the wall will be automatically destroyed.

Let see how the implementation of the two relationships are different in Java: The relationship between a person and his bank account can be definitely modelled as an aggregation, since the lifetime of a Person object is not depending on the person’s accounts and vice versa.

```
class BankAccount {
       private int accountNumber;
       private double amount;
       public BankAccount (int num, double amnt) {
       accountNumber = num;
            amount = amnt;
       }
       ... //  ASSUME MORE METHODS HERE
}

class Person{
       private BankAccount [] manyAccounts;
       public Person(BankAccount [] accnts, ...){
              manyAccounts = accnts;
              ...
       }
       ... //  ASSUME MORE METHODS HERE
}
```

The above Java code shows a person owns many accounts but accounts are created elsewhere. Also it is possible that the same account to be shared between two or more people.

Now let’s assume we want to model and implement the relationship between Room and Wall in Java:

```
class Wall {
      private double width;
      private double height;
      public Wall (double wid, double hi) {
            width = wid;
            height = hi;
      }
      ... //  ASSUME MORE METHODS HERE
}

class Room{
      private int numberOfWalls;
      private final Wall [] walls;
      public Room(int num){
            numberOfWalls = num;
            walls = new Wall[num];
            for(int i = 0; i < num; i++){
               double width = readWidth();//assume readWith() is defined
               double height=readHeight();//assume readHeight() is defined
               walls[i] = new Wall(width, height);
         }  // END OF FOR LOOP
      }

}
```

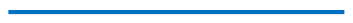
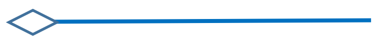
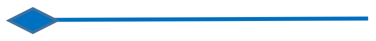
**** Please notice how the constructor of class Room is responsible to call the constructor of class Wall.**

There are more details about relationships among classes (we have discussed during lectures). For example a class might be associated with another class as a bi-directional, reflective, or dependency linkages. For further details please refer to your lecture notes.

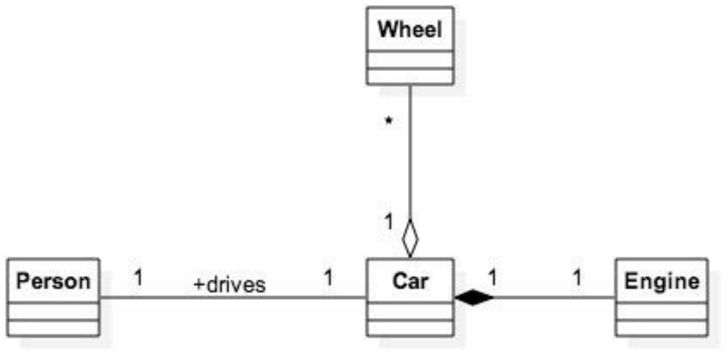
Modeling Notations:

To illustrate the relationship between classes, there are several modeling methodologies and languages (notations). Three of the most well-known methods are: OMT, Booch, and UML notations. In fact the last one (UML), has been introduced by the developers of the all three methods, James Rumbaugh, Grady Booch and Ivar Jacobson. In this course, we are going to learn and use the UML notation.

UML notation for class relationships are shown below:

Relationship	UML Connector
Association	
Aggregation	
Composition	

The following figure shows an example a class diagram, using UML notations.


Wheel
*
1
Person
1
+drives
1
Car
1
1
Engine