Chapter 2

Relationships between classes Using UML

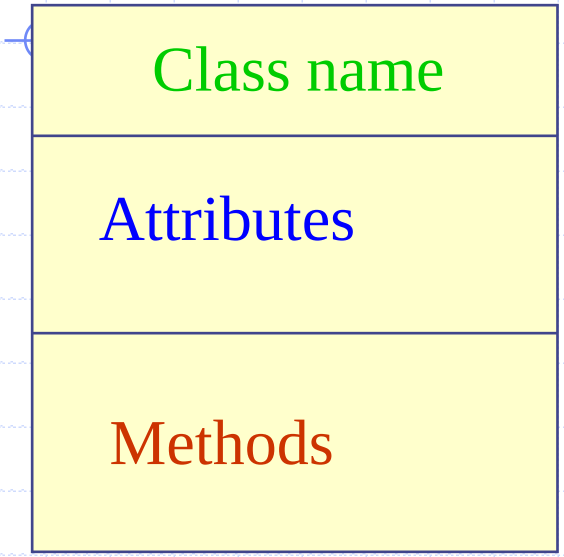
Objectives: What is UML?

- *``UML is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct and document the artifacts of a software intensive system"*
- Defined semantics for each of the graphical symbols
- Allows for unambiguous specification and for inspection of requirements and designs
- Allows tools to directly generate code from diagrams - but programmers still has to do some work
- Provides documentation of products, so allowing auditing and facilitating management

OUTLINE

1. UML Object Models: Classes
2. Association
3. Composition
4. Aggregation
5. Examples

1. UML Object Models: Classes



Class Name

Should be descriptive of the class and capitalized in the first letter

Attributes

The named properties of the class. Can be typed, possibly with default values

Methods

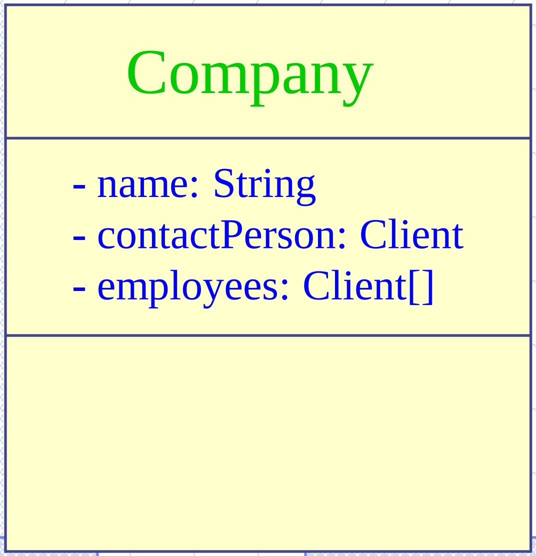
Services offered by the class. Methods can be typed e.g. parameter types and return types specified.

2. UML Object Models: Association, Aggregation, and Composition

Associations

UML diagrams show a collection of named boxes - indicating classes or types of object. The boxes have lines connecting them called links. Each link is called an *association* and should model some relationship or connection between the classes. Associations also play roles in classes that are often given special names.

Example:
Classes can contain references to each other. The *Company* class has two attributes that reference the Client class.

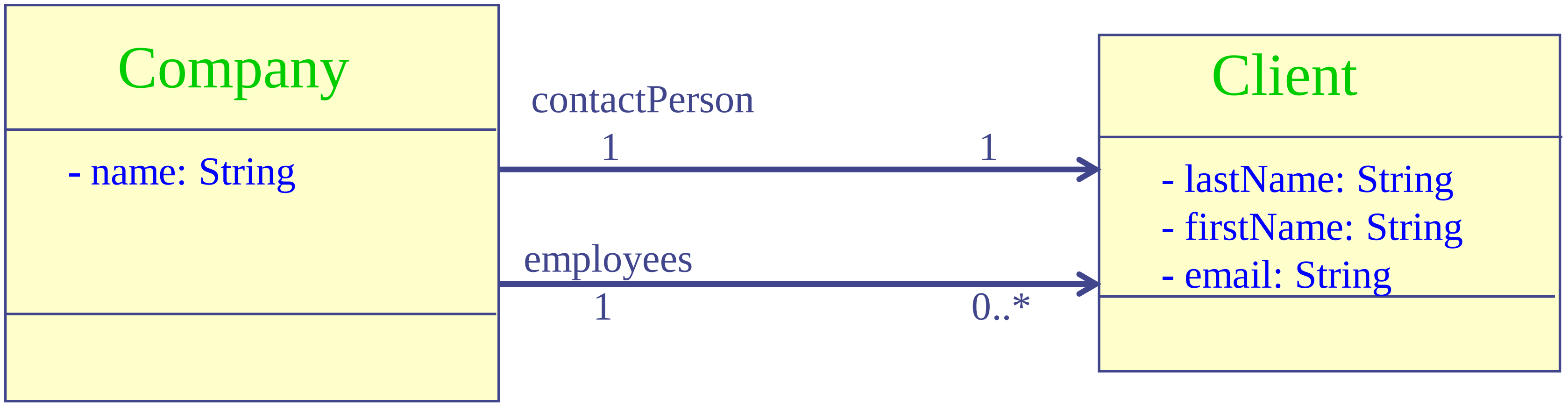


UML Object Models:

Association, Aggregation, and Composition

Associations

Although this is perfectly correct, it is sometimes more expressive to show the attributes as associations.



UML Object Models:

Association, Aggregation, and Composition

Associations

The above two associations have the same meaning as the attributes in the old version of the *Contact* class.

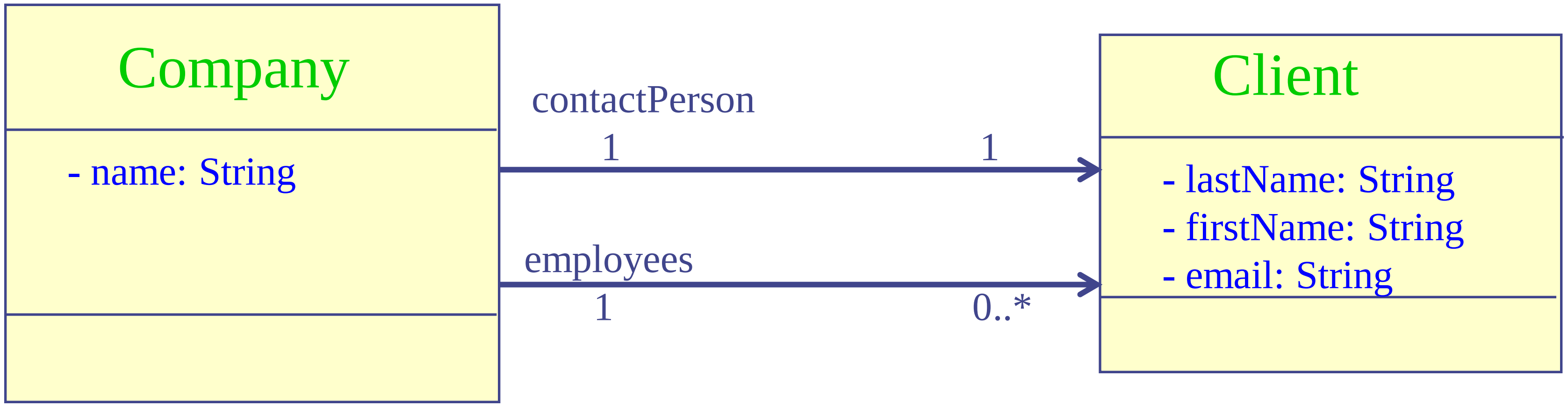
The first association (the top one) represents the old *contactPerson* attribute. There is one contact person in a single Company.

The ***multiplicity*** of the association is one to one meaning that for every *Companythere* is one and only one *contactPerson* and for each *contactPerson* there is one Company.

UML Object Models:

Association, Aggregation, and Composition

Associations



The first association (the top one) represents the old *contactPerson* attribute. There is one contact person in a single Company.

The ***multiplicity*** of the association is one to one meaning that for every *Companythere* is one and only one *contactPerson* and for each *contactPerson* there is one Company.

In the bottom association there are zero or many employees for each company.

UML Object Models:

Association, Aggregation, and Composition

Associations

Multiplicities can be anything you specify. Some examples are shown:

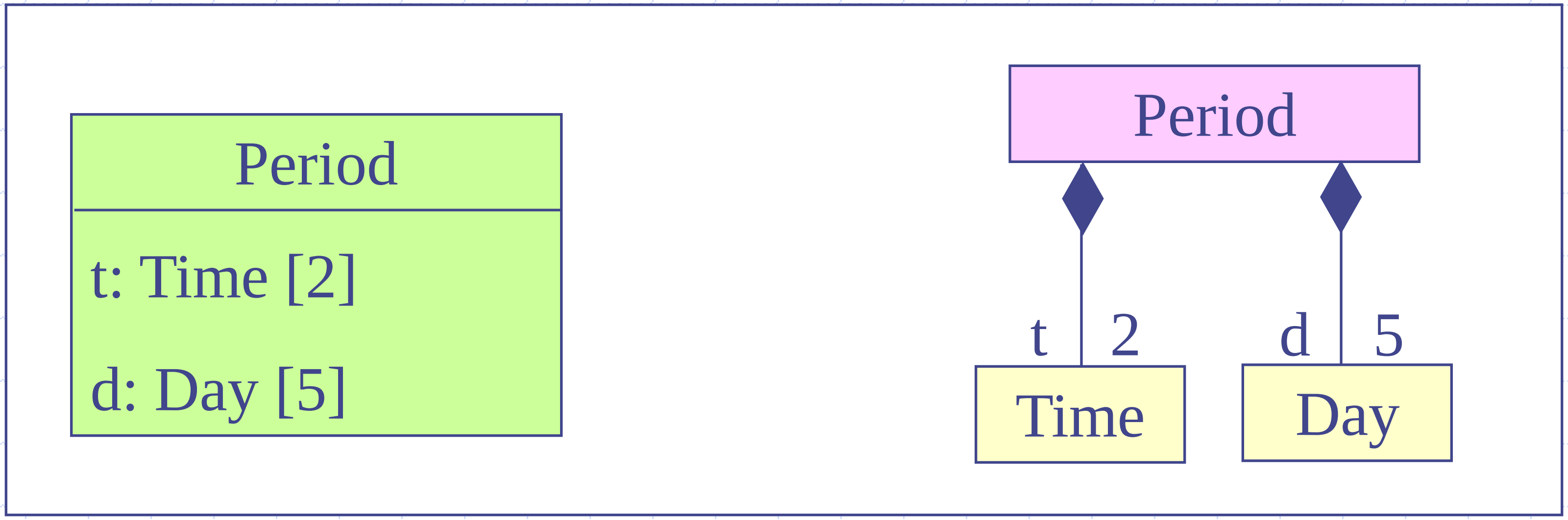
0	zero
1	one
1..*	one or many
1..2, 10..*	one, two or ten and above but **not** three through nine

3. UML Object Models: Aggregation, and Composition

Composition

- *UML* provides several notations that can express the physical construction of a class. The filled in diamond is often used when a class contain other objects within them as parts or components. **The *composition* association is represented by the solid diamond.**

Here are two examples: **Period is *composed* of *Time and Day*.**



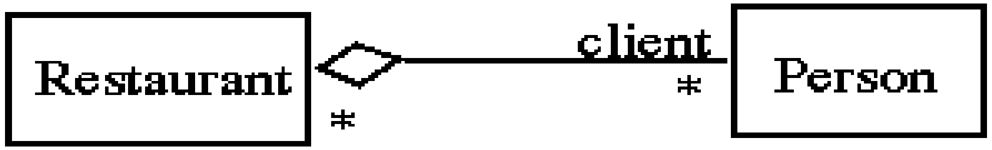
We can use the dark diamond to indicate that the class possesses the components in the sense of controlling whether they exist of not. The filled in diamond indicates that the deletion of an object may delete its components as well.

4. UML Object Models: Aggregation, and Composition

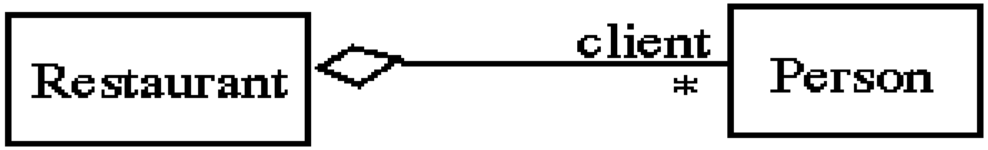
Aggregation

We can also show that a class *has* some parts and yet they have an independent existence. Example: In the computer world a page on the world Wide Web can use a hypertext reference to point to another resource -- deleting the page does not effect the other page. This association is called aggregation. **is represented by the hollow diamond.**

Here is an example showing that a Restaurant will have a number of clients who are People and the the clients exist whether or not they are clients of the Restaurant:



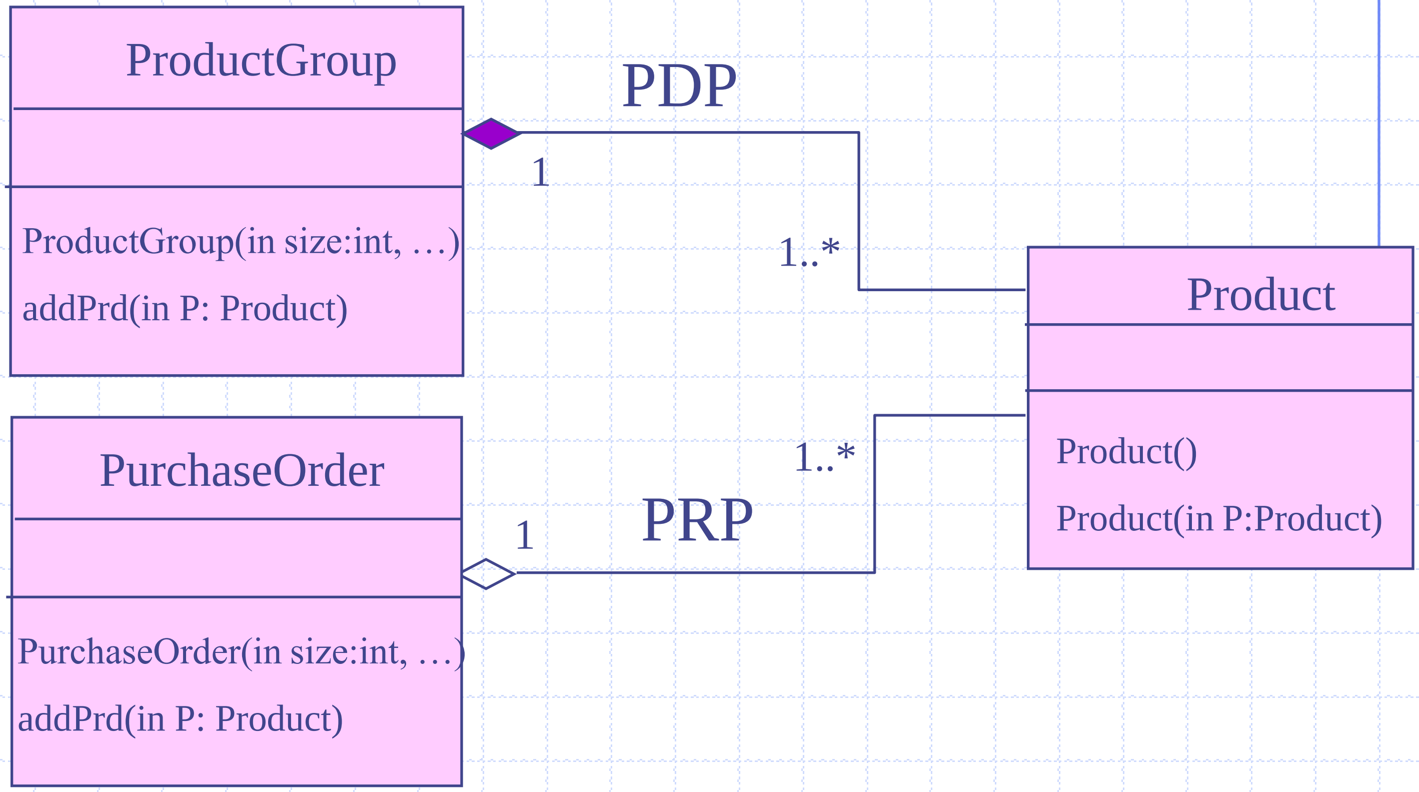
Each Person eats at any number of Restaurents



Each Person eats at one Restaurent.

UML Object Models: Aggregation, and Composition

Example 1:



ProductGroup **is** ***composed*** **of** ***Products*****. This means that if a** ***ProductGroup*** **is destroyed, the** ***Products*** **within the group are destroyed as well.**

PurchaseOrder **is an** ***aggregate*** **of** ***Products*****. If a** ***PurchaseOrder*** **is destroyed, the** ***Products*** **still exist.**

If you have trouble remembering the difference between composition and aggregation, just think of the alphabet. Composition means destroy and the letters 'c' and 'd' are next to each other.

UML Object Models: Aggregation, and Composition

How to Implement Aggregation?

```
public class PurchaseOrder
{
  private Product PRP [];
  private int nprp; // number of current
                                product in the array.
  …..

  public PurchaseOrder (int size, …)
  {
     PRP = new Product[size];
     nprp=0;
     ….
   }

   ……
}
```

How to Implement Composition?

```
public class ProductGroup
{
  private Product PDP [];
  private int npdp; // number of current
                                product in the array.
  …..

  public ProductGroup (int size, …)
  {
     PDP = new Product[size];
     npdp=0;
     ….
   }

   ……
}
```

UML Object Models: Aggregation, and Composition

Aggregation: How to add a new product

```
public class PurchaseOrder
{
  private Product PRP [];
  private int nprp; // number of current
                              product in the array.
  …..

  public void addPrd (Product P)
  {
     PRP[nprp] = P;
     nprp++;
   }

   ……
}
```

Composition: How to add a new product

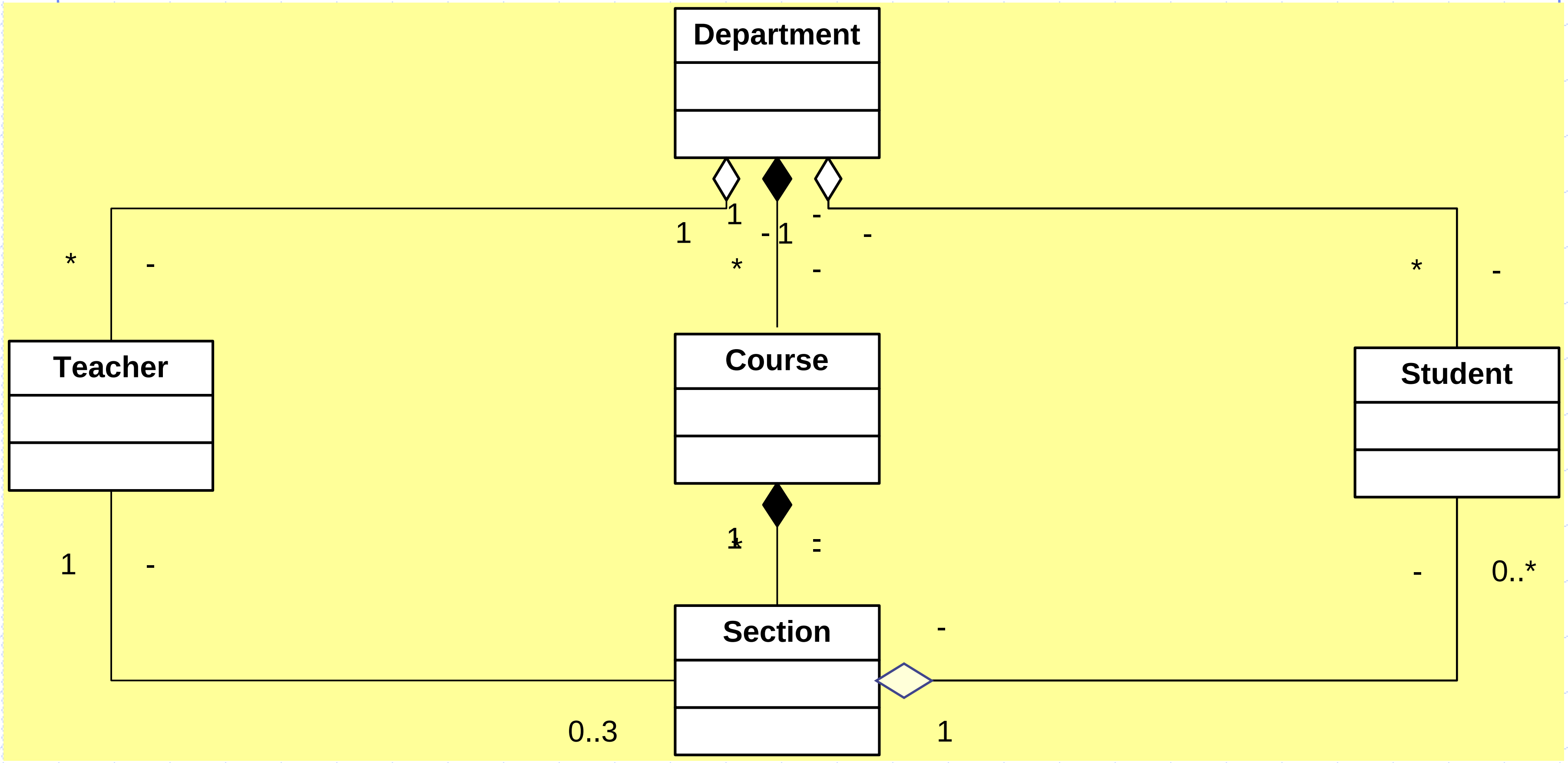
```
public class ProductGroup
{
  private Product PDP [];
  private int npdp; // number of current
                              product in the array.
  …..

  public void addPrd (Product P)
  {
     PDP[npdp] = new Product(P);
     nprp++;
   }

   ……
}
```

UML Object Models: Aggregation, and Composition

Example 2:



UML Object Models: Aggregation, and Composition

Example 2:

The following Java code shows just how the links between the different objects can be implemented in Java. Note that this code just shows the links. It does not show constructors, or any other methods what would be required to actually use these objects.

```
/*
* Student.java -
*/
public class Student
{
  private String name;
  private String id;

  public void copyStudent(Student st)
  {
    name= st.name;
    id= st.id ;
  }
// ...
}
```

```
/*
* Section.java -
 */
public class Section
{
  private String sectionName;
  private int capacity;
  private int currentNbStudents;
  private Student[ ] stud;
 ….
  public void addStudent(Student s)
  {
    stud[currentNbStudents]=s;
    currentNbStudents ++;
  }
// ...
}
```

UML Object Models: Aggregation, and Composition

Example 2:

```
/*
* Course.java -

*/
public class Course
{
  private String
  courseName;
  private int nbSection;
  private Section[ ] sect;
// ...

}
```

```
/*
* Teacher.java -

*/
public class Teacher
{
  private String
teacherName;
  private String Id;
  private Section[3] sect;
// ...

}
```

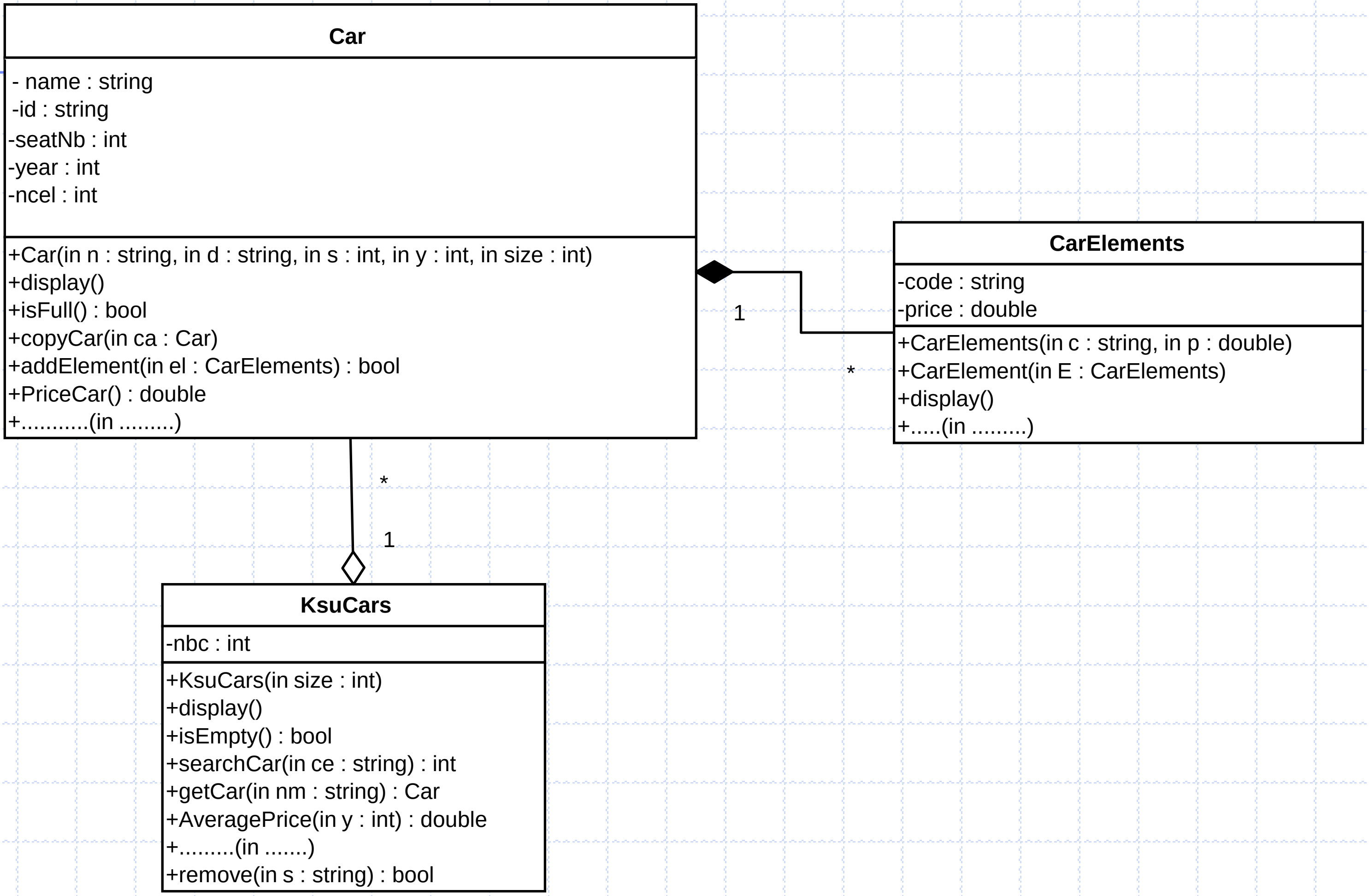
```
/*
* Teacher.java -

*/
public class Department
{
  private String
departName;
  private Student[ ] stud;
  private Course[ ] csc;
  private Teacher[ ] teach;

// ...
}
```

UML Object Models: Aggregation, and Composition

Example 3:



Car
- name : string
-id : string
-seatNb : int
-year : int
-ncel : int
+Car(in n : string, in d : string, in s : int, in y : int, in size : int)
+display()
+isFull() : bool
+copyCar(in ca : Car)
+addElement(in el : CarElements) : bool
+PriceCar() : double
+...........(in ....)
CarElements
-code : string
-price : double
+CarElements(in c : string, in p : double)
+CarElement(in E : CarElements)
+display()
+.....(in ....)
1
*
*
1
KsuCars
-nbc : int
+KsuCars(in size : int)
+display()
+isEmpty() : bool
+searchCar(in ce : string) : int
+getCar(in nm : string) : Car
+AveragePrice(in y : int) : double
+.........(in ....)
+remove(in s : string) : bool

Question: Implement all the classes with all their methods using the following descriptions.

Description of the different classes:

-

Class CarElements:

- ✓ *The method* **display ()** displays the code and the price.
- ✓ **+ …….. (in ……..)** : if you need an other methods in this class you can add it.

You can't add another constructor.

Class Car:

- name
- id
- seatNb : *Number of seats*
- year : *Production year of car*
- ncel : *number of CarElements object currently in an object of the class Car.*
- ***And other attribute(s) deduced from the UML diagram.***

- ✓ **display ():** Displays all the attributes of an object Car.
- ✓ **addElement (CarElements el)**: This method receives a CarElements object and adds it to the Car object.
- ✓ **priceCar()**: Returns the sum of the CarElements price in an object of the class Car.

+ …….. (in ……..) : if you *need an other methods in this class you can add it.*

Class KsuCars:

- nbc : *number of Car currently in an object of the class KsuCar.*
- ***And other attribute(s) deduced from the UML diagram.***

- ✓ **display ():** Displays all the attributes of an object KsuCars.
- ✓ **search (String ce):** This method receives a String representing the *name* of a Car object and returns the array index of the car object.
- ✓ **getCar (String nm):** This method receives a String representing the *id* of a Car object and returns the Car object if it's exist.
- ✓ **removeCar (String s)**: Removes a Car according to its name. It will return a value *true* if the operation has been completed successfully, or *false* if not.
- ✓ **AveragePrice(int y):** Calculates the average price of all car in an object of class KsuCars that produced after the year **y**.
- ✓ *+ …….. (in ……..) : if* you *need an other methods in this class you can add it.*