Nov 09, 2015

Computer Programming

LAB 9

ABDUL AZIZ

Lab Instructor	**Abdul Aziz**
Course	**Computer Programming Lab**
Duration	**2hrs**

Objectives:

In this lab, following topics will be covered:

- Association
- Aggregation
- Composition

1. Association

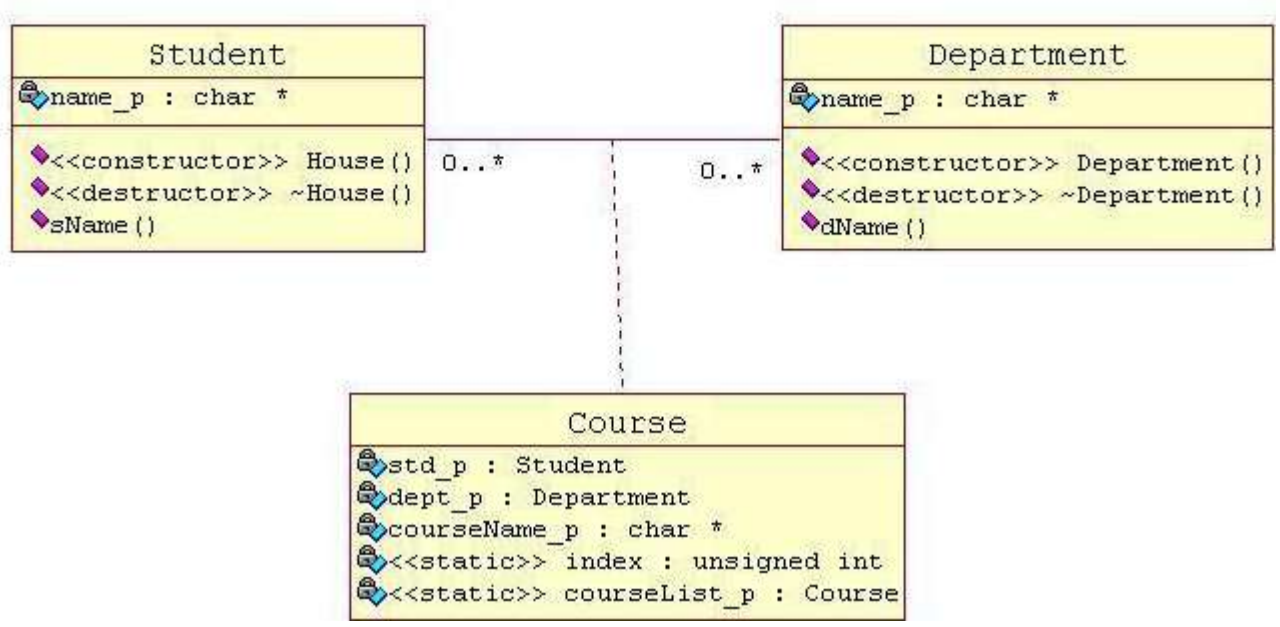
Association is a simple structural connection or channel between classes and is a relationship where all objects have their own lifecycle and there is no owner.

Let's take an example of Department and Student.

Multiple students can associate with a single Department and single student can associate with multiple Departments, but there is no ownership between the objects and both have their own lifecycle. Both can create and delete independently.

Here is respective Model and Code for the above example.

Course class Associates Student and Department classes



```
#include<iostream>
#include<string.h>
using namespace std;
class Student;
class Department
{
    char* name_p;
  public:
    Department(char *dName)
    {
      cout<<"Department::constructor\n";
      name_p = new char(sizeof(strlen(dName)));
      name_p = dName;
    }
    char* dName() const
    {
      return(name_p);
    }
    ~Department()
    {
      cout<<"Department::destructor\n";
      delete(name_p);
    }
};
class Student
{
  char* name_p;
  public:
    Student(char *sName)
    {
      cout<<"Student::constructor\n";
      name_p = new char(sizeof(strlen(sName)));
      name_p = sName;
    }
    char* sName()const
    {
      return(name_p);
    }
    ~Student()
    {
      cout<<"Student::destructor\n";
      delete(name_p);
    };
};
class Course
{
    Student * std_p;
    Department * dept_p;
    char * courseName_p;

    static unsigned int index;
    static Course *courseList[4];
  public:
      Course(char* crseName, Student* student, Department* dept):
      courseName_p(0), std_p(student), dept_p(dept)
    {
      cout<<"Course:constructor\n";
      if (index < 4)
      {
        courseName_p = new char(sizeof(strlen(crseName)));
        courseName_p = crseName;
               //insert this Course in courseList
        courseList[index] = this;
        ++index;
      }
      else
      {
        cout<<"Cannot accomodate any more Course\n";
      }
    };
    ~Course()
    {
      cout<<"Course:destructor\n";
      delete (courseName_p);
    };
    static char* findStudent(char *crseName, char* deptName)
    {
      for(int i=0; i<index; i++)
      {
        if ( (courseList[i]->getCourseName() == crseName) &&
             (courseList[i]->getDeptName() == deptName) )
        {
          return(courseList[i]->getStdName());
        }
      }
    }
    char * getStdName()const {return(std_p->sName());};
    char * getDeptName() const {return(dept_p->dName());};
    char * getCourseName()const {return(courseName_p);};
};
unsigned int Course::index =0;
Course* Course::courseList[4] = {0,0,0,0};
```

```
int main()
{
  int i;
  cout<<"\nExample of Association class...\n";
  cout<<"-----------------------------------\n\n";
  cout<<"We have got 4 students ...\n";
  Student *studentNames[4] = {new Student("Meera"), new Student("Saima"),
new Student("waqas"), new Student("mansoor")} ;
  cout<<"\n";
  cout<<"We have got 2 Departments...\n";
  Department *departNames[2] = {new Department("Mathemetics"), new
Department("ComputerSceince")} ;
  cout<<"\n";

  cout<<"Here class Course Associates Student and Department, with a Course
name ...\n";
  Course course1("DataStructure",studentNames[0], departNames[1]);
  Course course2("Maths",studentNames[3], departNames[0]);
  Course course3("Geometry",studentNames[2], departNames[0]);
  Course course4("CA",studentNames[1], departNames[1]);
  cout<<"\n";

  cout<<"Finding a Student using Course and Department...\n";
  cout<<"Student who has taken Maths Course in Mathemetics Department
is:"<<Course::findStudent("Maths", "Mathemetics")<<endl;
  cout<<"\n";
  cout<<"Deletion of objects...\n\n";
  for(i=0; i<4; ++i)
  {
    delete studentNames[i];
  }
  cout<<"\n";
  for(i=0; i<2; ++i)
  {
    delete departNames[i];
  }
  cout<<"\n";
  return(0);
}
```

2. <u>Aggregation</u>

Aggregation is a specialize form of Association where all object have their own lifecycle but there is a ownership like parent and child. Child object cannot belong to another parent object at the same time. We can think of it as "has-a" relationship.

Implementation details:

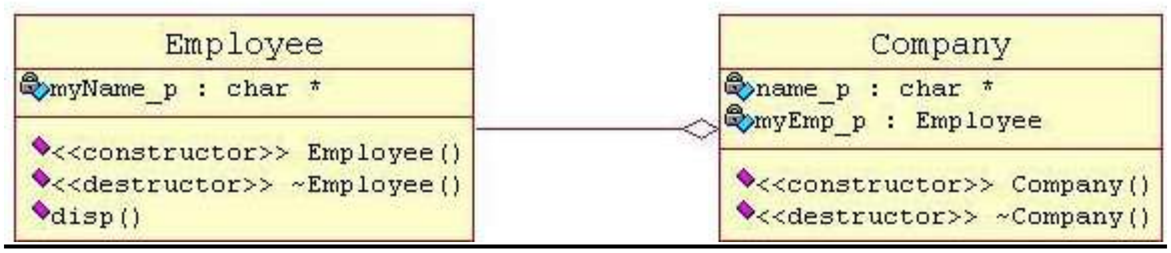
1. Typically we use pointer variables that point to an object that lives outside the scope of the aggregate class
2. Can use reference values that point to an object that lives outside the scope of the aggregate class
3. Not responsible for creating/destroying subclasses

<u>Let's take an example of Employee and Company.</u>

A single Employee can not belong to multiple Companies (legally!!), but if we delete the Company, Employee object will not destroy.

Here is respective Model and Code for the above example.

Employee class has Agregation Relatioship with Company class


Employee
myName_p : char *
<<constructor>> Employee()
<<destructor>> ~Employee()
disp()
Company
name_p : char *
myEmp_p : Employee
<<constructor>> Company()
<<destructor>> ~Company()

```
#include<iostream>
#include<string.h>
using namespace std;
class Employee
{
  public:
    Employee(char *name){
      cout<<"Employee::constructor\n";
      myName_p = new char(sizeof(strlen(name)));
      myName_p = name;
    }
    char* disp(){return(myName_p);};
    ~Employee()
    {
      cout<<"Employee:destructor\n\n";
      delete (myName_p);
    }
  private:
    char *myName_p;
};
class Company
{
  public:
    Company(char * compName, Employee* emp){
      cout<<"Company::constructor\n";
      name = new char(sizeof(strlen(compName)));
      name = compName;
      myEmp_p = emp;
    };
    ~Company()
    {
      cout<<"Company:destructor\n\n";
      myEmp_p = NULL;
    };
    private:
    char *name;
    Employee *myEmp_p;
};

int main()
{
  cout<<"\nExample of Aggregation Relationship \n";
  cout<<"-----------------------------------------\n\n";
    {
    cout<<"Here, an Employee-Waseem works for Company-MS \n";
    Employee emp("Waseem");
            {
                    Company comp("MS", &emp);
            } // here Company object will be deleted, whereas Employee
object is still there

    cout<<"At this point Company gets deleted...\n";
    cout<<"\nBut Employee-"<<emp.disp();
    cout<<" is still there\n";
   } //here Employee object will be deleted
  return(0);
}
```

3. <u>Composition</u>

Composition is again specialize form of Aggregation. It is a strong type of Aggregation. Here the Parent and Child objects have coincident lifetimes. Child object does not have its own lifecycle and if parent object gets deleted, then all of its child objects will also be deleted.

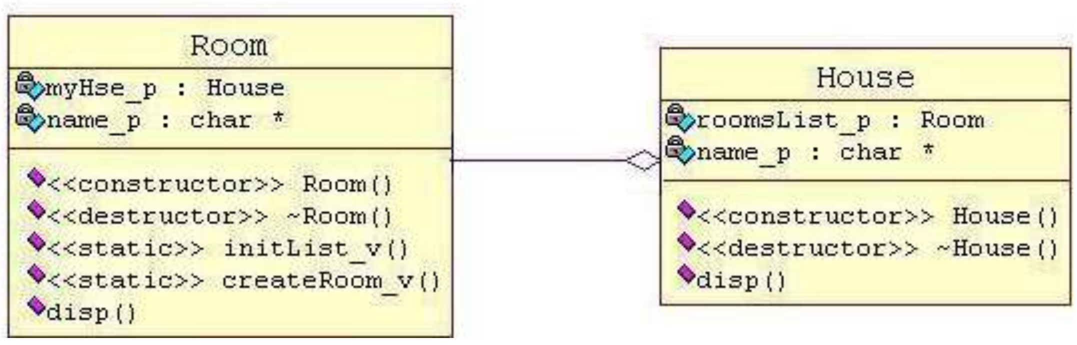
Implantation details:
1. Typically we use normal member variables
2. Can use pointer values if the composition class automatically handles allocation/deallocation
3. Responsible for creation/destruction of subclasses

Let’s take an example of a relationship between House and it's Rooms.

House can contain multiple rooms there is no independent life for room and any room cannot belong to two different house. If we delete the house room will also be automatically deleted.

Here is respective Model and Code for the above example.

Room class has Composition Relationship with House class



```
#include<iostream>
#include<string.h>
using namespace std;
class House;
class Room
{
  public:
   Room()
    {
    };
    static void createRoom_v(Room* (&room), House* hse, char* name)
    {
      room = new Room(hse, name);
    }
    Room(House* hse, char* myName)
    {
      cout<<"Room::constructor\n";
      myHse_p = hse;

      if(NULL != myHse_p)
      {
        name_p = new char(sizeof(strlen(myName)));
        name_p = myName;
      }
      else
      {
        cout<<"Oops House itself is not Created Yet ...\n";
      }
    };
    ~Room()
    {
      cout<<"Room:destructor\n";
      myHse_p = NULL;
      delete (name_p);
    };
    void disp()
    {
      cout<< name_p;
      cout<<"\n";
    }
    static void initList_v(Room *(& roomsList_p)[3])
    {
      roomsList_p[3] = new Room[3];
    }
  private:
    House * myHse_p;
    char * name_p;
};
class House
{
  public:
    House(char *myName)
    {
      cout<<"House::constructor\n";
      name_p = new char(sizeof(strlen(myName)));;
      name_p = myName;
      Room::initList_v(roomsList_p);
      Room* myRoom;
      Room::createRoom_v(myRoom, this, "Kitchen");
      roomsList_p[0] = myRoom;
      Room::createRoom_v(myRoom, this, "BedRoom");
      roomsList_p[1] = myRoom;
      Room::createRoom_v(myRoom, this, "Drwaing Room");
      roomsList_p[2] = myRoom;
    }
    ~House()
    {
      cout<<"House:destructor\n";
      unsigned int i;
      cout<<"Delete all the Rooms ...\n";
      for(i=0; i<3; ++i)
      {
        if(roomsList_p[i] != NULL)
        {
          delete (roomsList_p[i]);
        }
      }
      delete [] roomsList_p;
```

```
          delete (name_p);
        }
        void disp()
        {
          cout<<"\n\nName of the House :"<<name_p;
          if(roomsList_p != NULL)
          {
            unsigned int i;
            cout<<"\n\nRooms details...\n";
            for(i=0; i<3; ++i)
            {
              if(NULL != roomsList_p[i])
              {
                roomsList_p[i]->disp();
              }
            }
            cout<<"\n\n";
          }
        }
      private:
        char* name_p;
        Room* roomsList_p[3];
            };

    int main()
    {
      cout<<"\nExample of Composition Relationship\n";
      cout<<"-----------------------------------------\n\n";
      House hse("Bilawal House");
      cout<<"\n\nHouse details...\n";
      hse.disp();
      cout<<"Here House itself creates the Rooms and Deletes as well, before
it gets deletd...\n";
      return(0);
    }
```

Exercise