Discover Your Future

IT1050- Object Oriented Concepts

Lecture-13

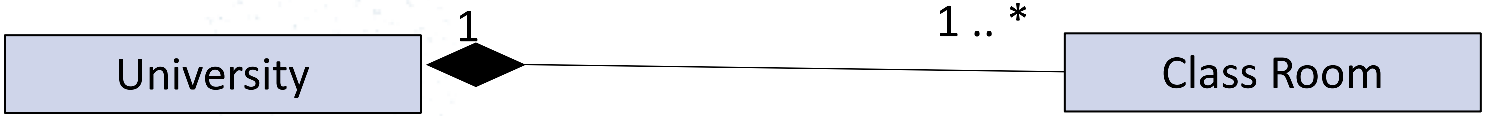
Implementation of relationships among classes using C++

SLIIT FACULTY OF COMPUTING

Learning Outcomes

- At the end of the lecture, students should be able to
  - Implement Composition, Aggregation, Association and Dependency

Composition



- Whole : University
- Part : Room
- A University is composed of at least one Room,
- If there are no rooms, there is no university
- Implies that the "part cannot exist without the whole"

Composition

- In composition, the objects have coincident lifetimes.
- if parent (whole) object gets deleted, then all of it’s child (part) objects will also be deleted.
- If the University object is deleted, the class room objects will get deleted automatically.
- Child (part) objects are created in the parent (whole) class.

Composition – C++ Implementation

```
class ClassRoom {
  private:
    int roomno;
  public:
    ClassRoom(){};
    ClassRoom(int no) {
       roomno = no;
    };
    void Display() {
       cout << "Class Room " << roomno << endl;
    };
    ~ClassRoom() {
      cout << "Deleting Room " << roomno << endl;
    }
};
```

Sample Code

Composition – C++ Implementation

```
class University {
   private:
      ClassRoom *room[SIZE];
   public:
      University(){
         room[0] = new ClassRoom(101);
         room[1] = new ClassRoom(102);
      };
      University(int no1, int no2) {
           room[0] = new ClassRoom(no1);
           room[1] = new ClassRoom(no2);
      };
      void DisplayClassRooms() {
          for (int i=0; i<SIZE; i++)
              room[i]->Display();      };
      ~University() {cout << "Univesity shutting down" << endl;
         for (int i=0; i <SIZE; i++)
             delete room[i];
         cout << "the End" << endl;
      }
};
```

Composition – C++ Implementation

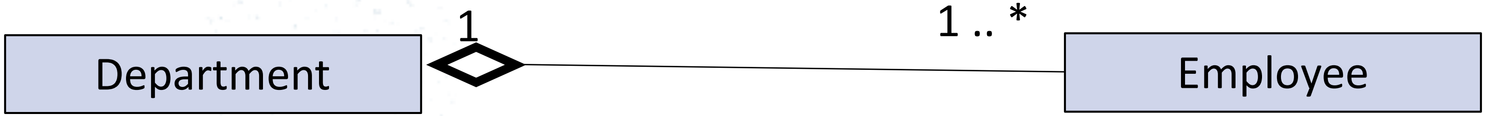
```
int main()
{

 University *myUniversity;
 myUniversity. = new University(501, 502);
 myUniversity >DisplayClassRooms();

  return 0;
}
```

Class room objects are created inside the university class and when the University destructor is called all the class room objects are deleted.

<u>Output</u>

```
Class Room 501
Class Room 502
University shutting down
Deleting Room 501
Deleting Room 502
the End
```

Aggregation



- Whole : Department
- Part : Employee
- A Department has one or more employees
- This implies that the Part can exist without the Whole.

Aggregation

- In aggregation the objects have their own life cycles, but there is a ownership.
- The Department and Employee objects have their own life cycles.
- If the Department object is deleted, still the Employee objects can exist.
- If the Employee objects is deleted, still the Department object can exist.

Aggregation – C++ implementation

```
class Employee
{
private :
        string empID;
        string name;
public :
        Employee(string pempID, string pname)
        {
                empID = pempID;
                name = pname;
        }
        void displayEmployee()
        {
                cout << "empID  = "  << empID << endl;
                cout << "name  = " << name << endl;
                cout << "***************************" << endl;
        }
        ~Employee(){cout << "Deleting Employee" << empID << endl;
        }
};
```

Sample Code

Aggregation - C++ implementation

```
class Department
{
private:
        Employee *emp[2];
public:
         Department(){};
         void addEmployee(Employee *emp1, Employee *emp2)
         {
                emp[0] = emp1;
                emp[1] = emp2;
          }
          void displayDepartment(){
                for(int i = 0; i < SIZE; i++)
                       emp[i]->displayEmployee();
           }

        ~Company(){cout << ”Department shutting down" << endl;                }
};
```

Aggregation - C++ implementation

```
int main()

{

  Department*ABC = new Department();

  Employee *e1 = new Employee("E001",
"Nimal");

   Employee *e2 = new Employee("E002",
"Jagath");

  ABC->addEmployee(e1, e2);

  ABC->displayDepartment();

  delete ABC;

  e1->displayEmployee();

  e2->displayEmployee();

  return 0;

}
```

After the company ABC is deleted the two employees exists.

Output

```
empID  = E001
name  = Nimal
*************************
empID  = E002
name  = Jagath
*************************
Company shutting down
empID  = E001
name  = Nimal
*************************
empID  = E002
name  = Jagath
*************************
```

Aggregation - C++ implementation

```
int main(){
   Department*ABC = new Department();
    Employee *e1 = new Employee("E001", "Nimal");

   Employee *e2 = new Employee("E002", "Jagath");

   ABC->addEmployee(e1, e2);
   delete e1;
   delete e2;
   Employee *e3 = new Employee("E003",  “Kamal");

   Employee *e4 = new Employee("E004", “Lal");
   ABC->addEmployee(e3, e4);
   ABC->displayDepartment();
  return 0;
}
```

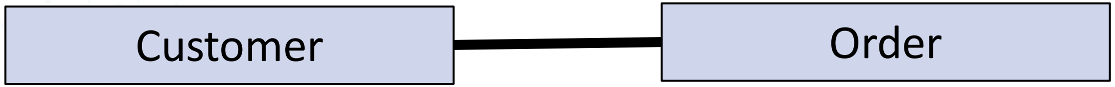
After E001 and E002 is deleted still the company exist and new employees can be added.

Output

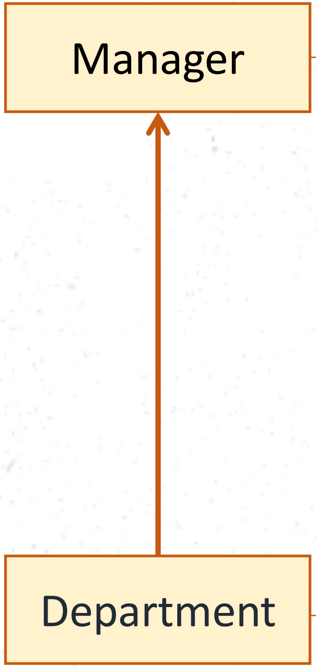
```
Deleting EmployeeE001
Deleting EmployeeE002
empID  = E003
name  = Kamal
*************************
empID  = E004
name  = Lal
*************************
```

Association

- An association between two classes indicates that objects at one end of an association “recognize” objects at the other end and may send messages to them.
- Example: “A Customer has many Orders”



Uni-directional association

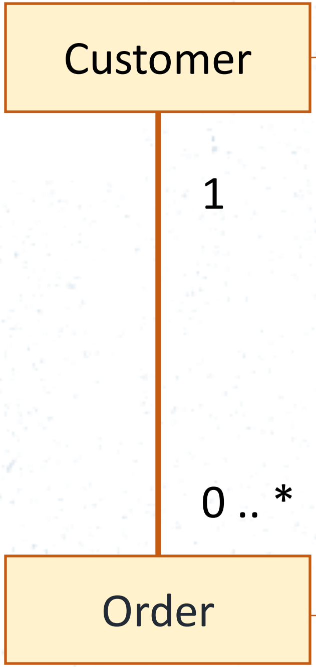


```
class  Manager
{
        public:
                Manager();
                ~Manager();
};
```

```
# include “Manager.h”
class  Department
{      private:
                Manager* mgr;

        public:
                Department();
                ~Department();
        };
```

Sample Code

Bi-directional association

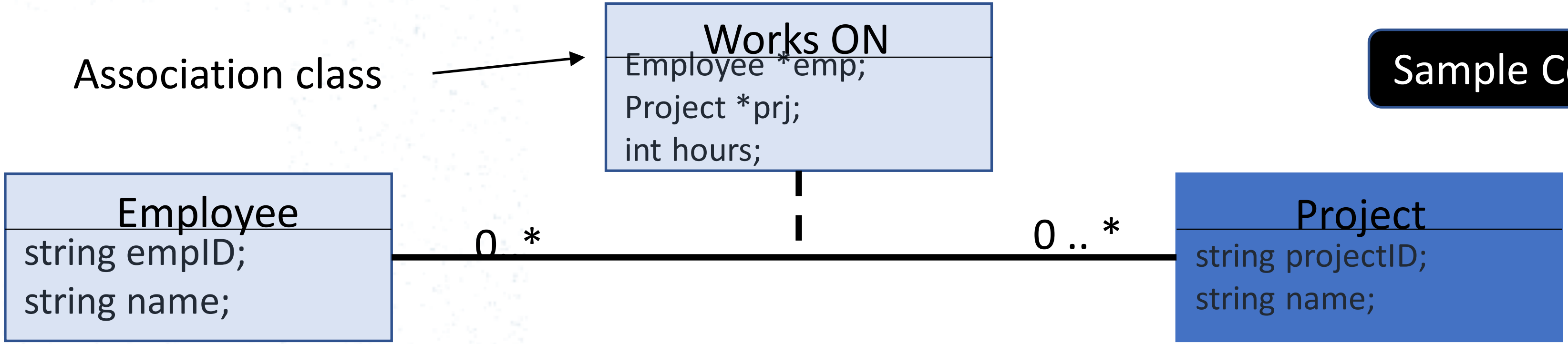


```
class Customer
{
            private:
                string name;
                string address;
                Order *order[SIZE];
                int noOfOrders;
            public:
                Customer();
                Customer( string pname, string
paddress);
                void addOrder(Order *O);
                void displayCustomer();

};
```

```
class Order
{
   private:
            string orderID;
            Customer *Cus;

   public:
            Order (string       porderID,  Customer
*pCus);
            void displayOrders();
};
```

Sample Code

Association class



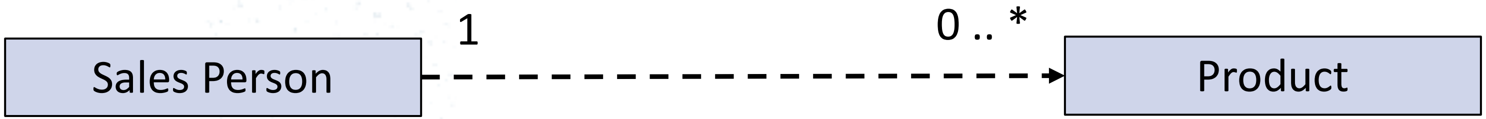
Sample Code

- An association class is a class that is part of an association relationship between two other classes.
- An association class provides additional information about the relationship.
- If an employee works for more than one project and if each project is assigned to more than one employee , the additional information (number of hours each employee spend on the a project) can be store in Works ON class.

Dependency

- Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time.
- It implies that a change to one class may affect the other but not vice versa.

Dependency



- The Sales Person class depends on a Product class because the Product class is used as a parameter for an add operation in the Sales Person class.

```
void SalesPerson::addSales(int qty , Product *P)
        {
            salesAmount = qty  * P->getPrice();
        }
```

Dependency

```
class Product
 {         private:
            string productID;
            string name;
            double price;
        public:
            Product(){}
            Product(string pID, string pname,double pPrice){
                productID = pID;
                name = pname;
                price = pPrice;
            }
            float getPrice(){
                    return price;
            }
            void display()
            {
                cout << "  Product ID =" << productID << endl;
                cout << " Product name =" << name << endl;
                cout << " Price = " << price << endl;
            }
 };
```

Sample Code

Dependency

```
class SalesPerson
 {
        private:
            string name;
            double salesAmount;
        public:
            SalesPerson(string pname){
                name = pname;
                salesAmount = 0;
            }
            void addSales(int qty , Product *P){
                salesAmount = qty  * P->getPrice();
            }
            void display()
            {
                 cout << "name = " << name << endl;
                 cout << "Sales Amount = " <<  salesAmount << endl;
            }
 };
```

Dependency

```
int main()
 {

      Product *P1 = new Product("P001","Mugs" , 200.00);
      SalesPerson *SP = new SalesPerson("Ajith");
      SP->addSales(10, P1);
      SP->display();

 }
```