NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Object Oriented Analysis & Design (CL-309)

Muhammad Nadeem || Awais Ahmed || Tooba Ali

Muhammad.nadeem@nu.edu.pk || awais.ahmed@nu.edu.pk || tooba.ali@nu.edu.pk

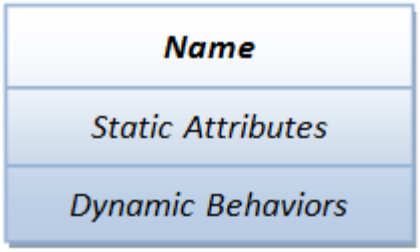
---

Lab Session # 05

Objective: To Understand Class Diagram

Introduction:

A Class is a 3-Compartment Box Encapsulating Data and Operations



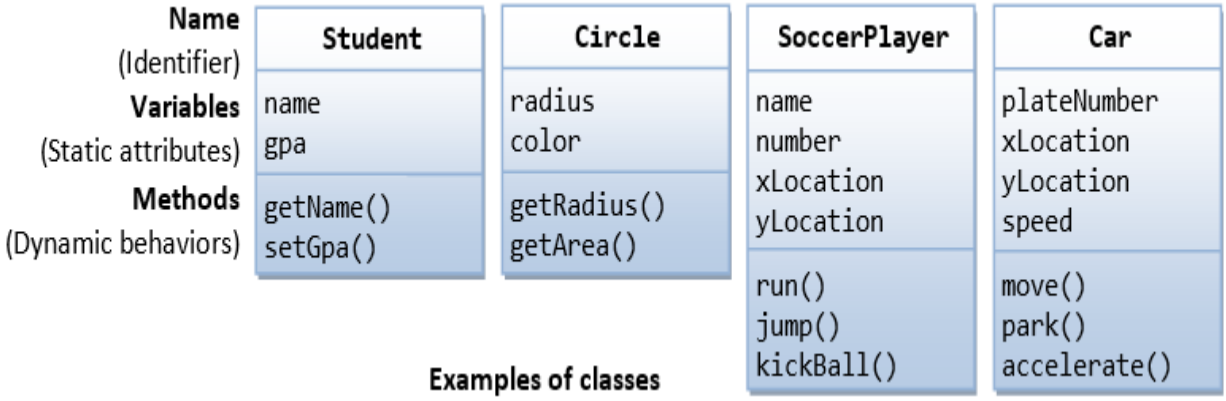
A class is a 3-compartment box

A class can be visualized as a three-compartment box, as illustrated:

1. *Name* (or identity): identifies the class.
2. *Variables* (or attribute, state, field): contains the *static attributes* of the class.
3. *Methods* (or behaviors, function, operation): contains the *dynamic behaviors* of the class.

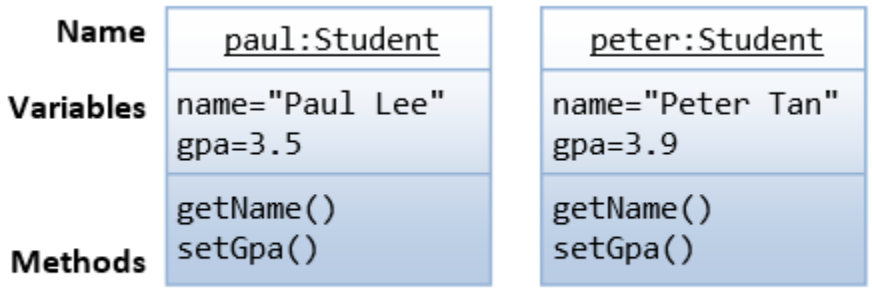
In other words, a class encapsulates the static attributes (data) and dynamic behaviors (operations that operate on the data) in a box.

The followings figure shows a few examples of classes:



Examples of classes

The following figure shows two instances of the class Student, identified as "paul" and "peter".



Two instances - paul and peter - of the class Student

Class Definition in Java

In Java, we use the keyword `class` to define a class. For examples:

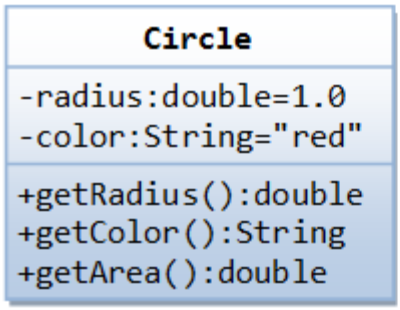
```
public class Circle {        // class name
   double radius;            // variables
   String color;

   double getRadius() { ..... }  // methods
   double getArea() { ..... }
}
public class SoccerPlayer {  // class name
   int number;               // variables
   String name;
   int x, y;

   void run() { ..... }      // methods
   void kickBall() { ..... }
}
```

The syntax for class definition in Java is:

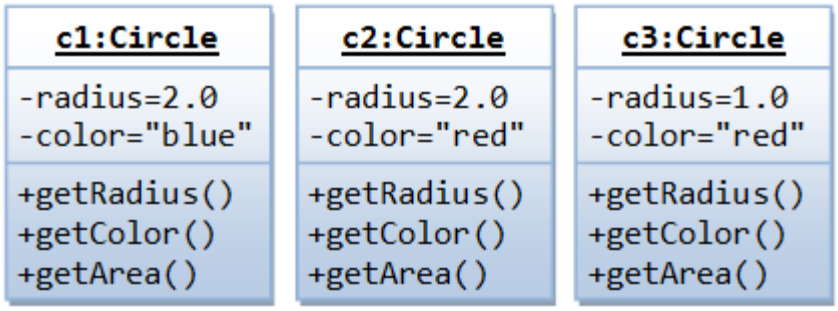
```
[AccessControlModifier] class ClassName {
   // Class body contains members (variables and methods)
   .....
}
```

Putting them Together: An OOP Example

Class Definition



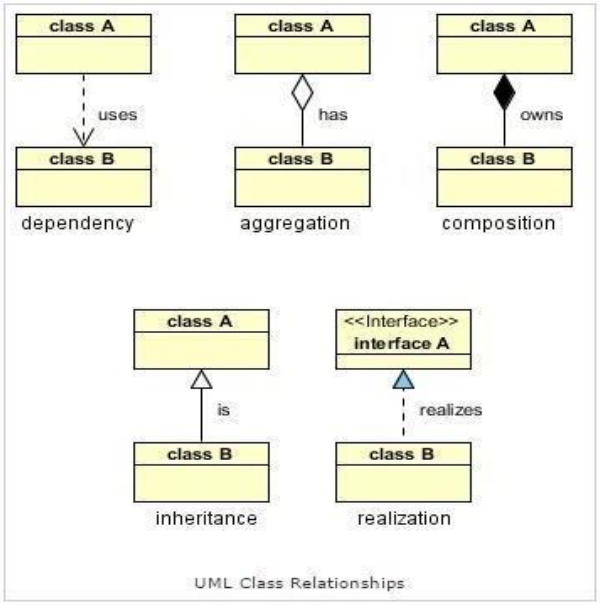
Instances



A class called Circle is defined as shown in the class diagram. It contains two private member variables: radius (of type double) and color (of type String); and three public member methods: getRadius(), getColor(), and getArea(). Three instances of Circles, called c1, c2, and c3, shall be constructed with their respective data members, as shown in the instance diagrams.

UML Class Diagram Relationships

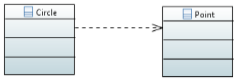
There are following key relationships between classes in a UML class diagram: dependency, aggregation, composition, inheritance and realization. These five relationships are depicted in the following diagram:



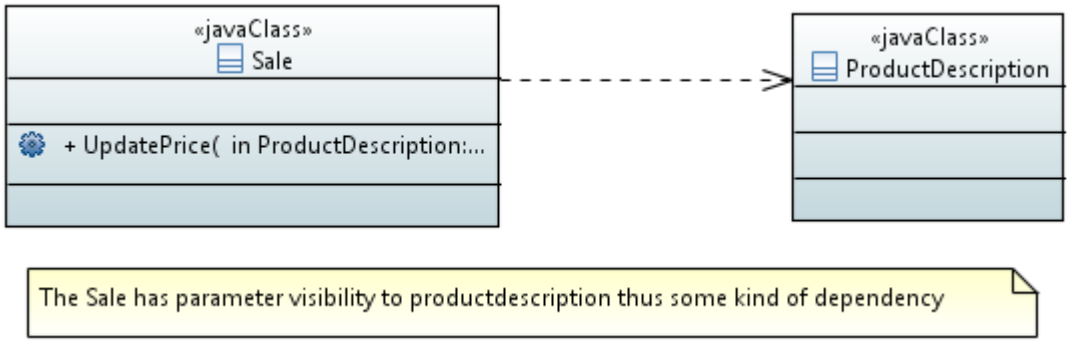
Dependency Relationship:

Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time.

Example 01

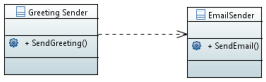


Example 02



Dependency Relationship Implementation Example

Dependency can develop between objects while they are associated, aggregated or composed. This kind of relationship develops while one object invokes another object's functionality in order to accomplish some task. Any change in the called object may break the functionality of the caller.



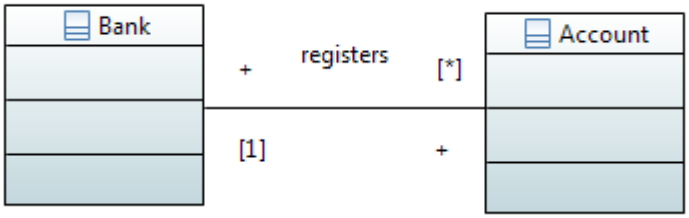

```
class GreetingSender
{
    EmailSender _emailSender;
    void SendGreetings(EmailSender emailSender)
    {
      _emailSender = emailSender;
      //Send Greeting through Email
```

```
        _emailSender.SendEmail();
    }
}
class EmailSender
{
    public void SendEmail()
    {
      //Send Email
    }
}
```

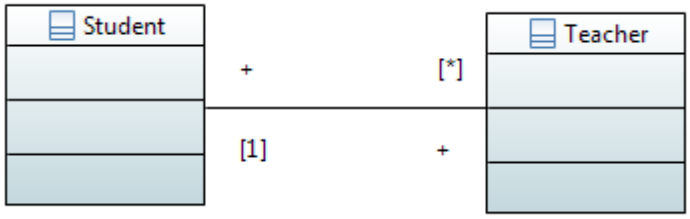
In the above example, Greetings are being sent through Email. The GreetingSender object is using the SendEmail() method of EmailSender object to accomplish its task. Now, if any modification (e.g introduction of a parameter) is to the SendEmail() method.
The GreetingsSender() method of GreetingSender class will break also as per theimplementation, we cannot send greetings by any other mode but only through Email. That way, my GreetingSender object's functionality is dependent on EmailSender object's functionality. This kind of relationship is termed as dependency.

Association Relationship:

An association relation is established, when two classes are connected to each other in any way A "bank registers account" association can be shown as follows



Association Relationship Implementation Example



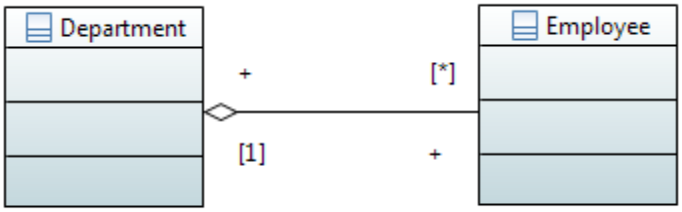

```
class Teacher
{

}
```

```
class Student
{
List&lt;Teacher> teachers;
}
```

In the above example, a Teacher list is referred in the student class. The system interprets a student has multiple teachers. In this student teacher relationship, one is being used by another. But there is no part-whole relationship between them. This means neither of them is the part of another. Hence they have their own lifetime.

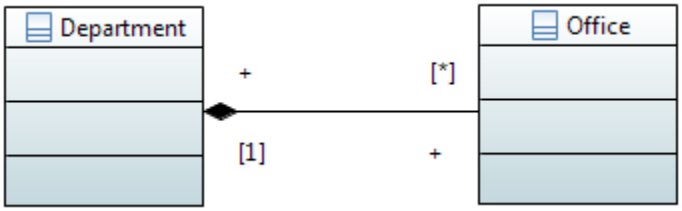
Aggregation Relationship:

For example: Department-Employee. A department contains several employees. Employee is not a composition relation because the employees can easily quit their jobs; retire without the company being affected. Also the company may bust and the employees can still live on.



Composition Relationship:

For example: Office-Department. The office cannot exist without its departments and keeping the departments without their office doesn't make sense either



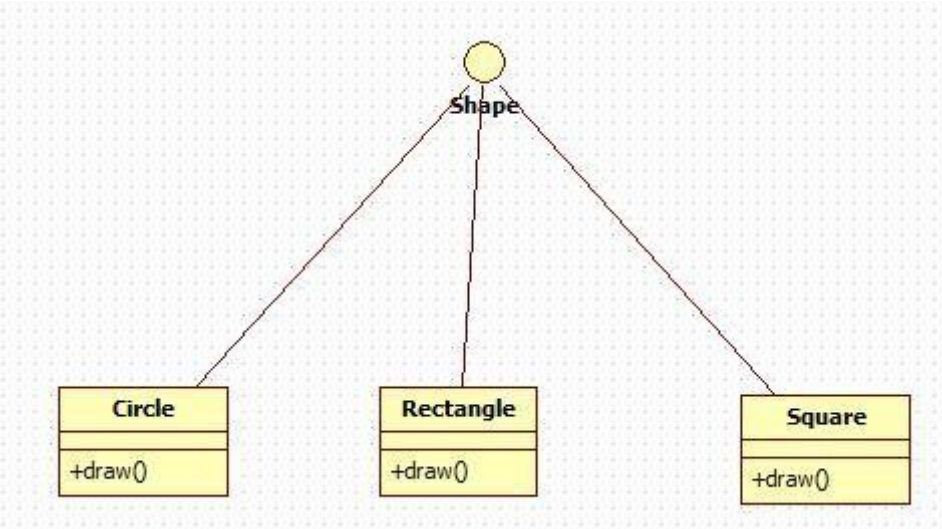
Realization:

A realization relationship indicates that one class implements a behavior specified by another class (an interface or protocol).

An interface can be realized by many classes.

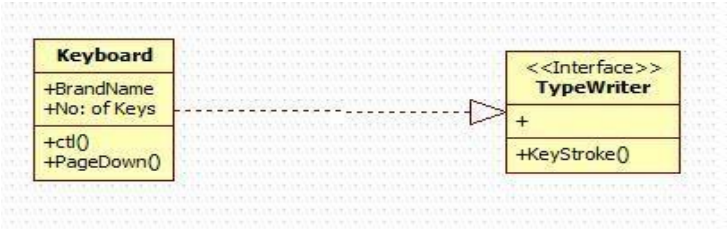
A class may realize many interfaces.

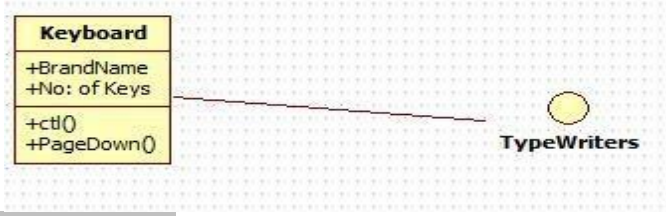
Example



Realization- Interface Implementation Example

OR






```
public interface TypeWriter {
void keyStroke()
}
public class KeyBoard implements TypeWriter
{ public void keyStroke(){
……
}
}
```

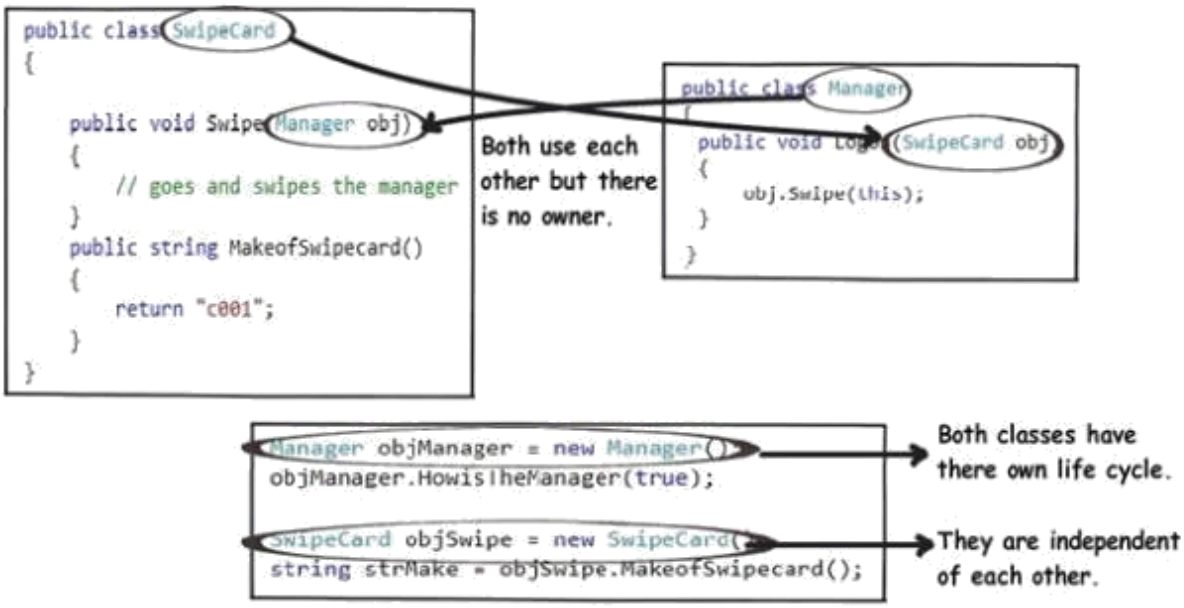
Understand Association, Aggregation and Composition relationships with code from a requirement

1. Manager is an employee of XYZ limited corporation.
2. Manager uses a swipe card to enter XYZ premises.
3. Manager has workers who work under him.
4. Manager has the responsibility of ensuring that the project is successful.
5. Manager's salary will be judged based on project success.

Let's understand them one by one.

Association Relationship:

Requirement 2 is an interesting requirement (Manager uses a swipe card to enter XYZ premises). In this requirement, the manager object and the swipe card object use each other but they have their own object life time. In other words, they can exist without each other. The most important point in this relationship is that there is no single owner.



The above diagram shows how the SwipeCard class uses the Manager class and the Manager class uses the SwipeCard class. You can also see how we can create objects of the Manager class and SwipeCard class independently and they can have their own object life time.
This relationship is called the "Association" relationship.

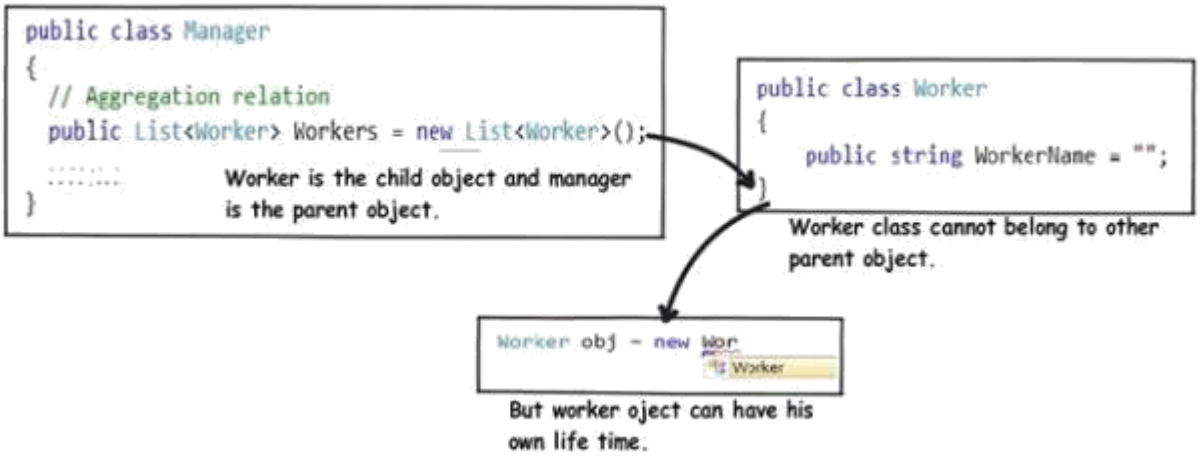
Aggregation Relationship:

The third requirement from our list (Manager has workers who work under him) denotes the same type of relationship like association but with a difference that one of them is an owner. So as per the requirement, the Manager object will own Worker objects.

The child Worker objects can not belong to any other object. For instance, a Worker object cannot belong to a SwipeCard object.

But… the Worker object can have its own life time which is completely disconnected from the Manager object. Looking from a different perspective, it means that if the Manager object is deleted, the Worker object does not die.

This relationship is termed as an “Aggregation” relationship.



Composition Relationship:

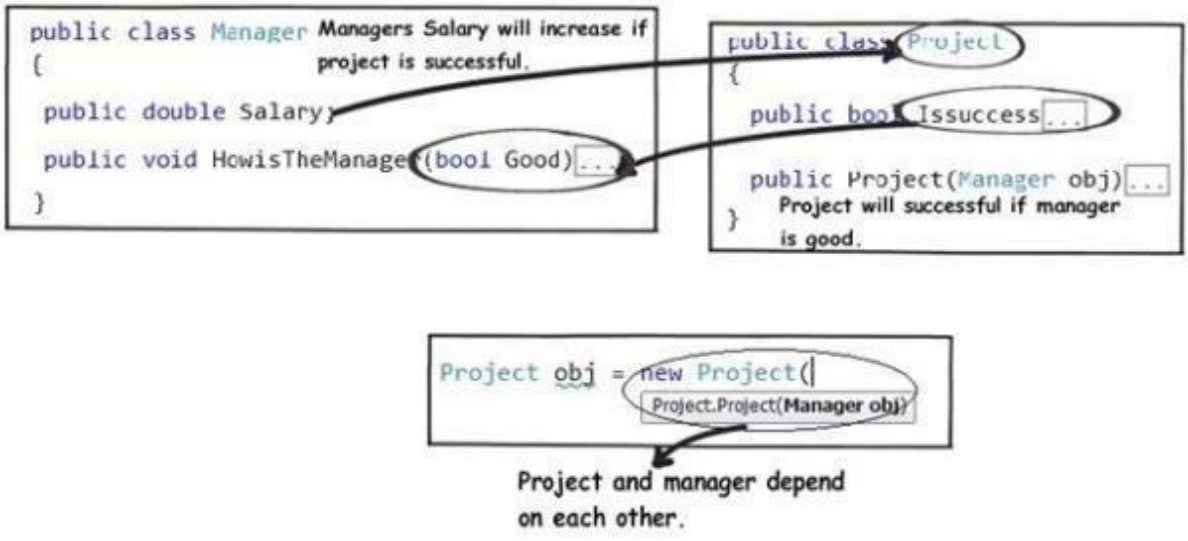
The last two requirements are actually logically one. If you read closely, the requirements are as follows:

1. Manager has the responsibility of ensuring that the project is successful.
2. Manager's salary will be judged based on project success. Below is the conclusion from analyzing the above requirements:

1. Manager and the project objects are dependent on each other.
2. The lifetimes of both the objects are the same. In other words, the project will not be successful if the manager is not good, and the manager will not get good increments if the project has issues.

Below is how the class formation will look like. You can also see that when I go to create the project object, it needs the manager object.

This relationship is termed as the composition relationship. In this relationship, both objects are heavily dependent on each other. In other words, if one goes for garbage collection the other also has to be garbage collected, or putting from a different perspective, the lifetime of the objects are the same. That's why I have put in the heading "Death" relationship.



Very Important Note

The scenario which I am considering for implementation is basically a teacher-course where both are independent of each other and can exist independently.

Yes, I know you might be thinking that I had taught in class dependency of teacher on course. That is also possible if I rephrase scenario. But, currently I'm considering both entities independent and there is no owner.

Another Implementation Example

Teacher.java

```
public class Teacher {
    private String teacher_name;
    private int teacher_cnic;

    public Teacher(String teacher_name, int teacher_cnic)
    {
        this.teacher_name = teacher_name;
        this.teacher_cnic = teacher_cnic;
    }
    public String getTeacher_name() {
        return teacher_name;
    }
    public void setTeacher_name(String teacher_name) {
        this.teacher_name = teacher_name;
    }
    public int getTeacher_cnic() {
        return teacher_cnic;
    }
    public void setTeacher_cnic(int teacher_cnic) {
        this.teacher_cnic = teacher_cnic;
    }
    @Override
    public String toString()
    {
        return teacher_name;
    }

}
```

Course.java

```java
public class Course {
    private String course_name;
    private int course_code;

    public Course(String course_name, int course_code)
    {
        this.course_name = course_name;
        this.course_code = course_code;
    }
    public String getCourse_name() {
        return course_name;
    }
    public void setCourse_name(String course_name) {
        this.course_name = course_name;
    }
    public int getCourse_code() {
        return course_code;
    }
    public void setCourse_code(int course_code) {
        this.course_code = course_code;
    }

    @Override
    public String toString()
    {
        return course_name;
    }
}
```

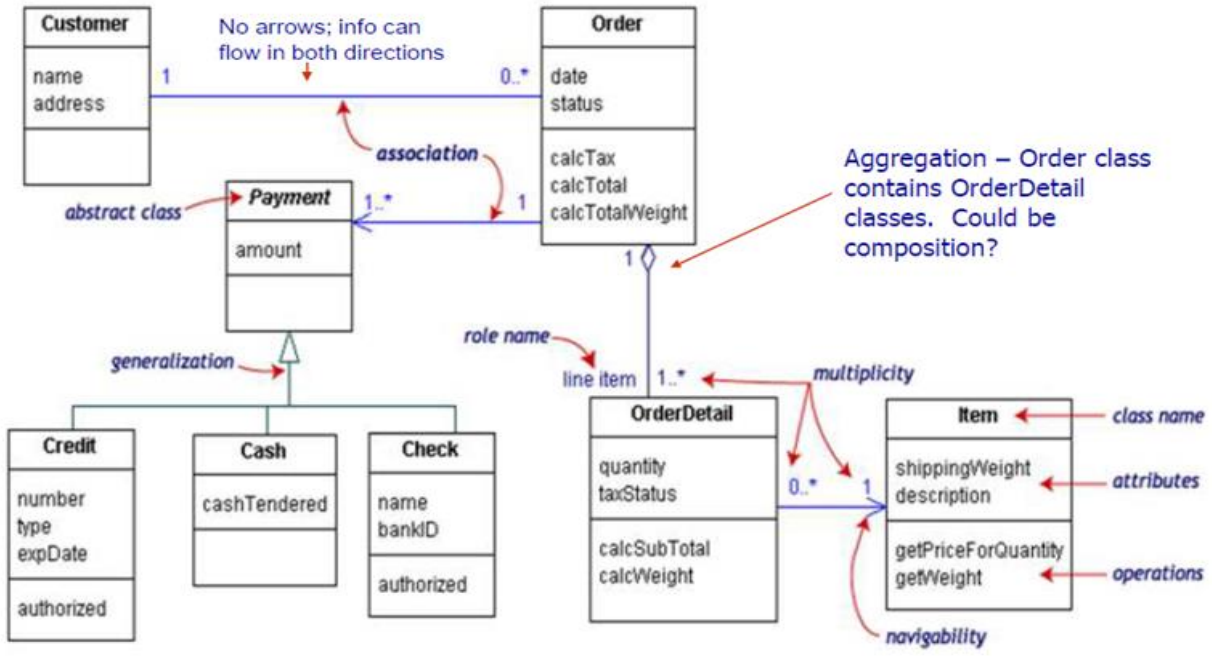
TeacherCourse.java

```
package teachercourse;

/**
 *
 * @author Neelam
 */
public class TeacherCourse {
    Teacher teacher_ob;
    Course course_ob;
    public TeacherCourse(Teacher teacher_ob, Course course_ob)
    {
        this.teacher_ob = teacher_ob;
        this.course_ob = course_ob;
    }
    public static void main(String[] args) {
        Teacher t1 = new Teacher("Neelam Shah",123);
        Teacher t2 = new Teacher("Mahrukh Khan",456);
        Teacher t3 = new Teacher("Muhammad Nadeem",789);

        Course c1 = new Course("OOAD", 123);
        Course c2 = new Course("ITC", 101);

        TeacherCourse tc1 = new TeacherCourse(t2, c2);
        TeacherCourse tc2 = new TeacherCourse(t1, c1);
        TeacherCourse tc3 = new TeacherCourse(t1, c2);
        TeacherCourse tc4 = new TeacherCourse(t3, c1);

        System.out.println("1st data");
        System.out.println("Teacher name: "+tc1.teacher_ob.getTeacher_name());
        System.out.println("Course name: "+tc1.course_ob.getCourse_name());

        System.out.println("\n\n\n");
        System.out.println("2nd data");
        System.out.println("Teacher name: "+tc2.teacher_ob.getTeacher_name());
        System.out.println("Course name: "+tc2.course_ob.getCourse_name());

        System.out.println("\n\n\n");
        System.out.println("3rd data");
        System.out.println("Teacher name: "+tc3.teacher_ob.getTeacher_name());
        System.out.println("Course name: "+tc3.course_ob.getCourse_name());

        System.out.println("\n\n\n");
        System.out.println("4th data");
        System.out.println("Teacher name: "+tc4.teacher_ob.getTeacher_name());
        System.out.println("Course name: "+tc4.course_ob.getCourse_name());

    }

}
```

OUTPUT

```
Output - TeacherCourse (run)
run:
1st data
Teacher nameMahrukh Khan
Course nameITC


2nd data
Teacher nameNeelam Shah
Course nameOOAD


3rd data
Teacher nameNeelam Shah
Course nameITC


4th data
Teacher nameMuhammad Nadeem
Course nameOOAD
BUILD SUCCESSFUL (total time: 0 seconds)
```

Class Diagram Example



FUNCTIONAL REQUIREMENT FOR LIBRARY MANAGEMENT SYSTEM

R.1:Register

- Description : First the user will have to register/sign up. There are two different type of users.
- The library manager/head : The manager have to provide details about the name of library ,address, phone number, email id.
- Regular person/student : The user have to provide details about his/her name of address, phone number, email id.

R.1.1: Sign up

- Input: Detail about the user as mentioned in the description.
- Output: Confirmation of registration status and a membership number and password will be generated and mailed to the user.
- Processing: All details will be checked and if any error are found then an error message is displayed else a membership number and password will be generated.

R.1.2 : Login

- Input: Enter the membership number and password provided.
- Output : User will be able to use the features of software.

R.2 : Manage books by user.

R.2.1 : Books issued.

- Description : List of books will be displaced along with data of return.

R.2.2 : Search

- Input : Enter the name of author's name of the books to be issued.
- Output : List of books related to the keyword.

R.2.3 : Issues book

- State : Searched the book user wants to issues.
- Input : click the book user wants.
- Output : conformation for book issue and apology for failure in issue.
- Processing : if selected book is available then book will be issued else error will be displayed.

R.2.4 : Renew book

- State : Book is issued and is about to reach the date of return.
- Input : Select the book to be renewed.
- Output : conformation message.
- Processing : If the issued book is already reserved by another user then error message will be send and if not then conformation message will be displayed.

R.2.5 : Return

- Input ; Return the book to the library.
- Output : The issued list will be updated and the returned book will be listed out.

R.2.6 ; Reserve book

- Input ; Enter the details of the book.
- Output : Book successfully reserved.
- Description : If a book is issued by someone then the user can reserve it ,so that later the user can issue it.

R.3 Manage book by librarian

- **R.3.1 Update details of books**

- **R.3.1.1 Add books**
  - Input : Enter the details of the books such as names ,author ,edition, quantity.
  - Output : confirmation of addition.

- **R.3.1.2 Remove books**
  - Input : Enter the name of the book and quantity of books.
  - Output : Update the list of the books available.

USER CHARACTERSTICS

We have 3 levels of users :

- User module: In the user module, user will check the availability of the books.

  - Issue book
  - Reserve book
  - Return book
  - Fine details

- Library module:

  - Add new book
  - Remove books
  - Update details of book

- Administration module:

The following are the sub module in the administration module :

- Register user
- Entry book details
- Book issue

(Sample Solution) CLASS DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM