# CSL304- Object Oriented Programming

**Lab Assignment – 2 Batch 1**

**Submission Date: Monday 17 October, 2016 Time 2.00 PM**

**Marks : 10**

Case study: A company wants to implement a C++ program that performs its payroll calculations polymorphically. A company pays its employees weekly. The employees are of four types: Salaried employees are paid a fixed weekly salary regardless of the number of hours worked, hourly employees are paid by the hour and receive overtime pay for all hours worked in excess of 40 hours, commission employees are paid a percentage of their sales and base-salary-plus-commission employees receive a base salary plus a percentage of their sales. For the current pay period, the company has decided to reward base-salary-plus-commission employees by adding 10 percent to their base salaries.

Abstract class Employee represents the general concept of an employee. The classes that derive directly from Employee are: SalariedEmployee, CommissionEmployee and HourlyEmployee. Class BasePlusCommissionEmployee derived from CommissionEmployee represents the last employee type.

Each employee, regardless of the way his or her earnings are calculated, has a first name, a last name and a social security number, so private data members firstName, lastName and socialSecurityNumber appear in abstract base class Employee.

| | Earnings | Print |
|---|---|---|
| Employee | 0 | fName,lName,SSN |
| Salaried | Weekly Salary | fName,lName,SSN, weekly salary |
| Hourly | If hours <= 40<br>Wage * hours<br>If hours >40<br>(40* wage) + ((hours-40)*wage*1.5) | fName,lName,SSN, hourly wage, hours worked |
| Commission | commissionRate * grossSales | fName,lName,SSN, gross sales, commission rate |
| Base Plus commission | baseSalary+ (commissionRate*grossSales) | fName,lName, SSN,grossSales, commissionRate, baseSalary |

```
class Employee {

  private:

            string firstName;

            string lastName;

            string socialSecurityNumber;

 public:

  Employee(  string &,  string &,  string & );

  void setFirstName(  string & ); // set first name

  string getFirstName() ; // return first name

  void setLastName(  string & ); // set last name

  string getLastName() ; // return last name

  void setSocialSecurityNumber(  string & ); // set SSN

  string getSocialSecurityNumber() ; // return SSN

  virtual double earnings()  = 0; // pure virtual

  virtual void print() ; // virtual

}; // end class Employee
```

For class SalariedEmployee the public member functions include a constructor that takes a first name, a last name, a social security number and a weekly salary as arguments; a set function to assign a new nonnegative value to data member weeklySalary ; a get function to return weeklySalary's value ; a virtual function earnings that calculates a SalariedEmployee's earnings and a virtual function print that outputs the employee's type, namely, "salaried employee:" followed by employee-specific information produced by base class Employee's print function.

```
class SalariedEmployee : public Employee

{

   private:

                double weeklySalary; // salary per week
```

```
  public:

          SalariedEmployee(  string &,  string &,  string &, double = 0.0 );

           void setWeeklySalary( double ); // set weekly salary

           double getWeeklySalary() ; // return weekly salary

           // keyword virtual signals intent to override

           virtual double earnings() ; // calculate earnings

           virtual void print() ; // print SalariedEmployee object

}; // end class SalariedEmployee
```

The class's constructor passes the first name, last name and social security number to the Employee constructor to initialize the private data members that are inherited from the base class, but not accessible in the derived class.

For the class HourlyEmployee the public member functions include a constructor that takes as arguments a first name, a last name, a social security number, an hourly wage and the number of hours worked; set functions that assign new values to data members wage and hours, respectively; get functions to return the values of wage and hours, respectively ; a virtual function earnings that calculates an HourlyEmployee's earnings and a virtual function print that outputs the employee's type, namely, "hourly employee:" and employee-specific information produced by base class Employee's print function.

```
class HourlyEmployee : public Employee

{

 private:

                  double wage; // wage per hour

                   double hours; // hours worked for week

  public:

     HourlyEmployee(  string &,  string &,  string &, double = 0.0, double = 0.0 );

           void setWage( double ); // set hourly wage

           double getWage() ; // return hourly wage
```

```
        void setHours( double ); // set hours worked
        double getHours() ; // return hours worked
        // keyword virtual signals intent to override
        virtual double earnings() ; // calculate earnings
        virtual void print() ; // print HourlyEmployee object
}; // end class HourlyEmployee
```

The HourlyEmployee constructor passes the first name, last name and social security number to the base class Employee constructor to initialize the inherited private data members declared in the base class.

For the class CommissionEmployee the member-function implementations include a constructor that takes a first name, a last name, a social security number, a sales amount and a commission rate; set functions to assign new values to data members commissionRate and grossSales, respectively; get functions that retrieve the values of these data members; function earnings to calculate a CommissionEmployee's earnings; and function print , which outputs the employee's type, namely, "commission employee:" and employee-specific information produced by base class.

```
class CommissionEmployee :  public Employee
{
  private:
        double grossSales; // gross weekly sales
        double commissionRate; // commission percentage
  public:
        CommissionEmployee(  string &,  string &,  string &, double = 0.0, double = 0.0 );
        void setCommissionRate( double ); // set commission rate
       double getCommissionRate() ; // return commission rate
        void setGrossSales( double ); // set gross sales amount
        double getGrossSales() ; // return gross sales amount

        // keyword virtual signals intent to override

        virtual double earnings() ; // calculate earnings

        virtual void print() ; // print CommissionEmployee object

}; // end class CommissionEmployee
```

Class BasePlusCommissionEmployee directly inherits from class CommissionEmployee. It’s member-function implementations include a constructor that takes as arguments a first name, a last name, a social security number, a sales amount, a commission rate and a base salary. It then passes the first name, last name, social security number, sales amount and commission rate to the CommissionEmployee constructor to initialize the inherited members. BasePlusCommissionEmployee also contains a set function to assign a new value to data member baseSalary and a get function to return baseSalary's value. Function earnings calculates a BasePlusCommissionEmployee's earnings. Function earnings calls CommissionEmployee's earnings function to calculate the commission-based portion of the employee's earnings. BasePlusCommissionEmployee's print  outputs "base-salaried", followed by the output of CommissionEmployee's print function.

```
class BasePlusCommissionEmployee : public CommissionEmployee

{

    private:

                double baseSalary; // base salary per week

    public:

          BasePlusCommissionEmployee(string &, string &, string &, double = 0.0, double =
0.0, double = 0.0 );

                void setBaseSalary( double ); // set base salary

                double getBaseSalary(); // return base salary

                // keyword virtual signals intent to override

                virtual double earnings(); // calculate earnings
```

```
        virtual void print(); // print BasePlusCommissionEmployee object
}; // end class BasePlusCommissionEmployee
```

Create a menu driven program which would provide following options to user

1) Enter employee details

2) View Payroll of all employees and total payroll of the company

3) Exit

If user selects 1) Enter employee details

then it should prompt another options as follows

  1) Enter fName, lName and SSN

  2) Enter employee type

   a) Salary

   b) Commision

   c) Hourly

   d) BasePlus Commission

  If option 1) then enter salary

  option 2) enter sales and commission rate

  option 3) hourly wages and hours worked

  option 4) base salary, sales and commission rate.

If option 2) View Payroll of all employees and total payroll of the company

Then it should display all information (fName, lName, SSN and earnings) for all the employees in the company and total earnings of company.

If options 3) then exit the application

# Lab Assignment – 2 Batch 2

## Submission Date: 18 October 2016 2.00 PM

## Marks : 10

## College Management System:

**Case study:** School management system has 5 classes School, Department, Student, Course and Instructor and Each class have respective methods and data members shown in below diagram.

### Problem Statement

College Management System college can add or remove student also can add or remove department from the college. College can display student information by giving input as a student id and see the list of all students in the college. College can see the information of department and display the list of all departments. College department can add or remove instructor and also can see the list of instructor of each department. College student can register courses of department. Instructor can teaches 1 or more courses. Students only attend the registered courses.

College::addStudent() is used get add student into the college for that student has to fill the form(name , address , phone number)

College::removeStudnet(StudentId) is used to get deleted from college , department and if student id is not valid then it has display the some message.

College::getStudent(StudentId) by giving input as student id it will display student detail, Department, registered course. if student id is not valid then it has display the some message.

College::getAllStudent() it display the list of students with Department name.

College::addDepartmet it get added new department

College:: removeDepartmet it deleted department as well as corresponding Student

College:: getDepartmet By giving the input as Department name it will display list of course

College:: getAllDepartmet it display the list of department

Department::addInstructor() it add instructor name

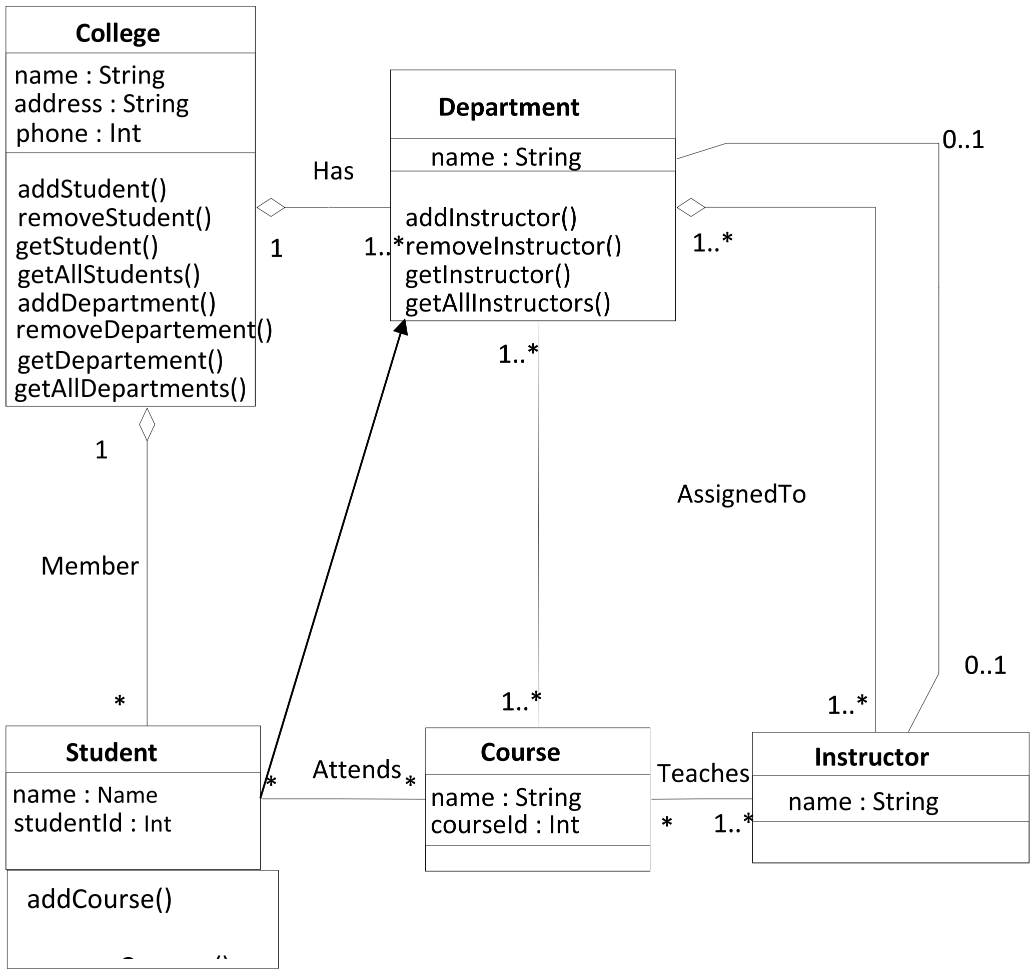
Department::removeInstructor() by giving input as an instructor name it should get deleted from department list.

Department::getInstructor(String name) It display the instructor detail (dept, Courses)

Department::getAllInstructor() It display the list of all instructor of the department.

Student::AddCourse() student can registered courses of same department.

Student::RemoveCourse() student can remove registered courses.



*In the given diagram, One instance of college can have *(Many) Students*

*1..*= one to many*

**Lab Assignment – 2 Batch 3**

**Submission Date: 19 October 2016 2.00 PM**

**Marks : 10**

## Online Railway Ticket reservation System

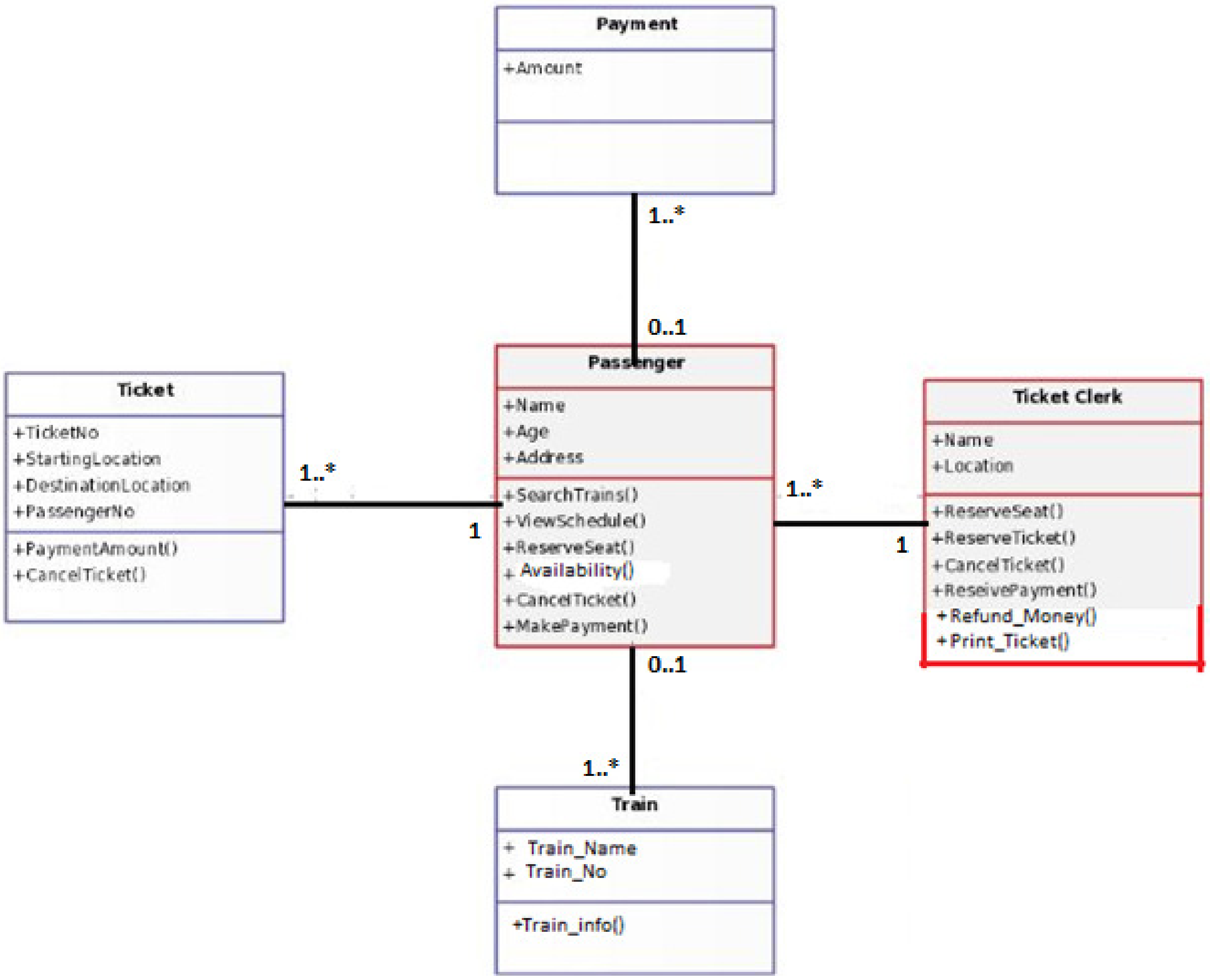
**Case Study**: Railway ticket reservation system has many classes: Passenger, Train, Ticket, Payment and Ticket_Clerk. The class Passenger has three data members Name, Age, Address and five data function SearchTrain(), viewschedule(), ReserveSeat(), Availability(), CancelTicket() and MakePayment().The class Train has 2 data member Trainname, Trainid. and one data function Train_info(). All classes have some data member and data function shown in below figure.

Railway Reservation System is a system used for booking tickets, cancel ticket and search Train. Any Passenger Can book tickets for different trains. Passenger can book a ticket only if the tickets are available. If the ticket is not available Passenger cannot book the ticket. Passenger searches for the availability of tickets by taking the input as Train_No.

While booking the ticket passenger has fill the personal information Name, Age and Address after that passenger has to filled information StaringLocation, DestinationLocation and number of passenger. Ticket:PaymentAmount() function tells the final amount. If passenger ready to pay final amount that calculated by Ticket:PaymentAmount() function. Passenger will go for MakePayment(), if passenger account don't have enough money to reserve ticket it has to print message “unsufficient balance”. If ticket payment is successful then seat reserve by Ticket_clerk:reserveseat() function. And printed to the Passenger By Print_Ticket() Function of Ticket_clerk class.

If the passenger want to cancel ticket by Passenger:CancelTicket() by taking input as a TicketNo. Then Ticket_clerk cancel the ticket and refund is transferred to passenger account. Availability is increased by number person ticket get canceled.

For the above Railway Ticket Reservation system, create respective classes and their methods.
Note: you can add more data members;



*[In the given diagram, One instance of passanger can have 1 or *(Many) Tickets. 1..*= one to many]*

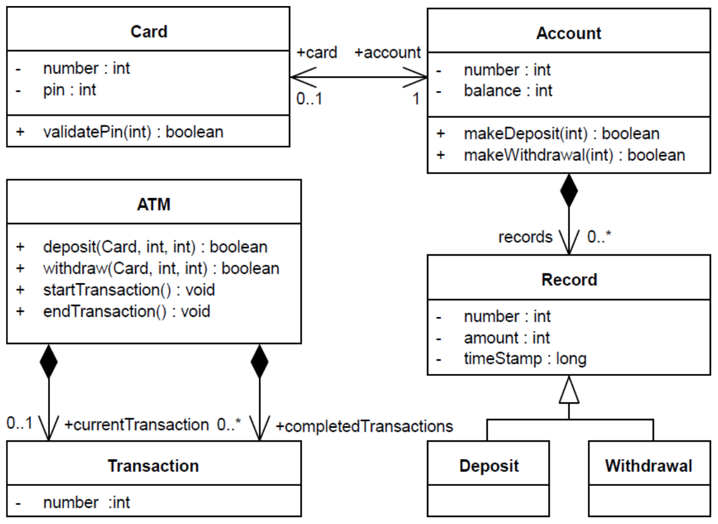
- Activity Passenger::SearchTrains() gets the input as Train_No and Train_info get called and it display the Train information by Train::Train_info
- Availability(Train_No, Train_Name)
- CancelTicket(TicketNO)
- ReserveSeat(Train_NO)
- ViewSchedule(Train _No) it display the departure time, arrival time and Date and journey hour.

# Lab Assignment – 2 Batch 4

## Submission Date: 20 October, 2016 Time 2.00 PM

## Marks : 10

Case study:- ATM system defines several classes: *Card, Account, ATM, Transaction, Record, Deposit,* and *Withdrawal*. The class *Card* has its 2 Integer attributes *number* and *pin* and is associated with the class *Account,* which has its own 2 Integer attributes *number* and *balance*.



In the given diagram, one instance of class Account can have 0 or 1 card. One account instance can have 0 or more than 0 records where both classes 'Deposit' and 'Withdrawal' inherits all the properties of class 'Record'. ' - ' represents private attributes and ' + ' represents public operations. ATM can have 0 or 1 current (running) Transaction and 0 or more completed transactions.

The withdrawal functionality is defined by the operation *withdraw()* of the class *ATM*, the operation *validatePin()* of the class *Card*, as well as the operation *makeWithdrawal()* of the class *Account*.

The activity *ATM.Withdraw* defining the behavior of the operation *ATM::withdraw()* gets as input the card, the pin, and the amount of money to be withdrawn entered by the client. It first starts a new transaction by calling the operation *ATM::startTransaction()*, then calls the operation *Card::validatePin()* for validating the pin entered by the client. If the pin is valid, the account associated with the provided card is obtained, and the operation *Account::makeWithdrawal()* is called to perform the withdrawal of the provided amount of money from the account. Finally, the operation *ATM::endTransaction()* is called to end the current transaction, and add it to the completed transactions of the ATM.

The activity *Card.validatePin* defines the behavior of operation *Card::validatePin()* and checks whether the pin stored on the card matches the provided pin. If this is the case, *true* is provided as output, otherwise *false* is provided. If the call of the operation *Card::validatePin()* provides *false* as output, the activity *ATM.withdraw* also provides *false* as output indicating an unsuccessful withdrawal. Otherwise, the withdrawal is carried out by calling the operation *Account::makeWithdrawal()* for the account associated with the provided card whose behavior is defined by the activity *Account.makeWithdrawal*.

The activity *Account.makeWithdrawal* first checks whether the account's balance exceeds the amount of money to be withdrawn. If this is the case, the balance is reduced by this amount, a new withdrawal record is created, and the activity provides *true* as output indicating the successful update of the account's balance. Otherwise *false* is provided as output. If the activity *Account.makeWithdrawal* provides *true* as output, the activity *ATM.withdraw* also provides *true* as output indicating the successful withdrawal, otherwise *false* is provided.

For the above ATM system, create respective classes and their methods.
Also write a driver program to test the above functionality.