Object Oriented & Programming Language

Lab#08

Table of Contents

Inheritance & Composition

In this lab, we will solve a problem using inheritance & composition.

- Inheritance is an "is-a" relation.
- Composition (aggregation) is a "has-a" relation
    - In composition (aggregation), one or more members of a class are objects of another class type. Every person **has a** date of birth

Constructor & Destructors:
When initializing the object of a derived class, the **constructor** of the base class is executed first and vice versa for **Destructors.**

How to find classes from any case study:

> *An easy way to identify classes, objects, and operations is to describe the problem in English and then identify all of the nouns and verbs. Choose your classes (objects) from the list of nouns and operations from the list of verbs.*

Computerize the Billing System of a Hospital

Problem: In this exercise, you will design various classes and write a program **to computerize the billing system of a hospital**.

a) Design the class **doctorType**, inherited from the class **personType**, defined in at the end ofproblem, with an additional data member to store a ***doctor's specialty***. Add appropriate constructors and member functions to initialize, access, and manipulate the data members.

b) Design the class **billType** with data members to store *a patient's ID* and a ***patient's hospital charges***, such as ***pharmacy charges for medicine***, ***doctor's fee***, **and** ***room charges***. Add appropriate constructors and member functions to initialize and access and manipulate the data members.

c) Design the class **patientType**, inherited from the class personType, with additional data members to store a ***patient's ID***, ***age***, ***date of birth***, **attending physician's name**, the ***date*** when the patient was ***admitted*** in

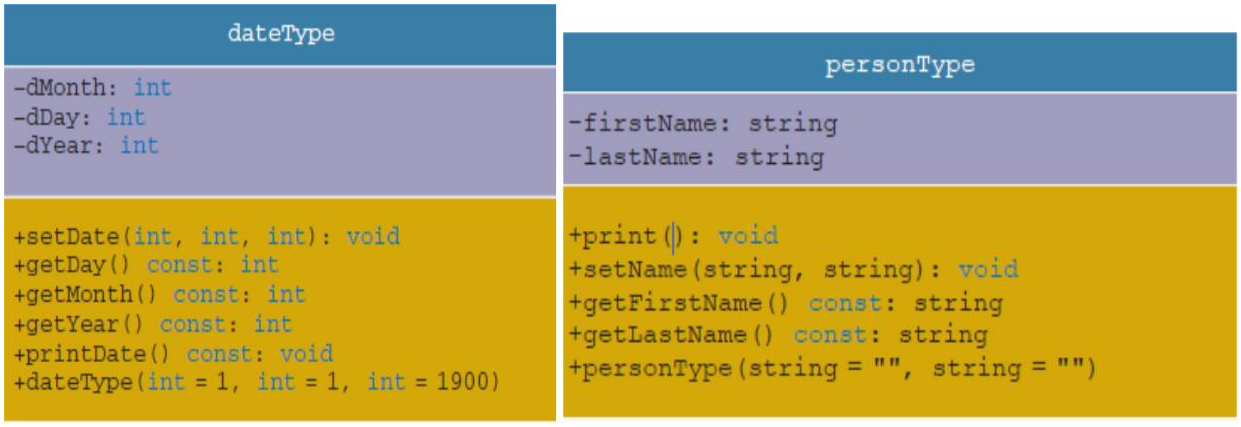
the hospital, and the ***date*** when the patient was ***discharged*** from the hospital. (Use the class **dateType** to store the date of birth, admit date, discharge date, and the class ***doctorType*** to store the attending physician's name.)

d) Add appropriate constructors and member functions to initialize, access, and manipulate the data members.

e) Write a program to test your classes.

UML Class Diagram

Class Diagram will be here…




```
//------------------------Title: Billing system of a hospital-------------------
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;
```

Base Class, Constructor & Destructor

```
class personType
{
public:
void print() const;

//Function to output the first name and last name //in the form firstName lastName.

void setName(string first, string last); //Function to set firstName and lastName according //to the parameters.
//Postcondition: firstName = first; lastName = last
```

```
string getFirstName() const; //Function to return the first name. //Postcondition: The value of firstName is returned.

string getLastName() const; //Function to return the last name. //Postcondition: The value of lastName is returned.

personType(string first = "", string last = ""); //Constructor

//Sets firstName and lastName according to the parameters. //The default values of the parameters are null strings. //Postcondition: firstName = first; lastName = last

~personType();

private:

string firstName; //variable to store the first name string lastName; //variable to store the last name
};

void personType::print() const
{
cout << firstName << " " << lastName<<endl;
}

void personType::setName(string first, string last)
{
firstName = first;
lastName = last;
}
string personType::getFirstName() const
{
return firstName;
}

string personType::getLastName() const

{
return lastName;
}

//constructor

personType::personType(string first, string last)
```

```
{

cout<<"\nPerson Type Constructor Invoked"<<endl;
firstName = first;
lastName = last;
}
//destructor
personType::~personType()
{
cout<<"\nPerson Type "<< getFirstName() <<" ~Destructor Invoked\n";

}
```

Function Overriding

Print function of parent class personType is overridden by doctorType class

```
class doctorType: public personType // doctorType inherited personType
{
private:
string speciality;
public:
doctorType()
{

cout<<"\tDoctor Type Default Contructor Invoked"<<endl; speciality = "NA";
}

doctorType(string spl)
{

cout<<"\tDoctor Type Contructor with 1-parameter Invoked"<<endl; setSpeciality(spl);
}

doctorType(string fname, string lname, string spl) :personType(fname,lname) // calling
personType construcotr
{

cout<<"\tDoctor Type Parametrized Contructor Invoked"<<endl; setSpeciality(spl);

}

void print()
{
```

```
cout<<"Printing doctor information\nDoctor Name: "; personType::print(); // calling
personType print() function explicitly cout<<"\tSpecialist: "<<speciality<<endl<<endl<<endl;

}
void printSpecility()

{
cout<<"Specialist: "<<speciality<<endl<<endl<<endl;
}
void setSpeciality(string str)
{
speciality = str;
}
string getSpeciality()
{
return speciality;
}
~doctorType()
{

cout<<"\nDoctor Type"<<getFirstName()<<" - "<<getSpeciality() <<" ~Destructor Invoked \n";
}

};
```

Parent Class with Default Constructor

```
class billType

{
private:
int patientId;
double pharmacyCharges;
int doctorFee;
double roomCharges;
double totalChares;
public:

billType(int pid=0, double pCharges = 0.0, int docFee = 0, double rcharges = 0.0) {

cout<<"\nBill Type Constructor Invoked\n";
setPatientId(pid);
setPharmacyCharges(pCharges);
```

```
setDoctorFee(docFee);
setRoomCharges(rcharges);
totalChares = 0;
}
void setPatientId(int pid)
{
patientId = pid;
}

void setPharmacyCharges( double t)
{
pharmacyCharges = t;
}
void setDoctorFee(int dFee)
{
doctorFee = dFee;
}
void setRoomCharges(double rchareges)
{
roomCharges = rchareges;
}
int getPatientId() { return patientId;}

double getPharmacyCharges() {
return pharmacyCharges;
}
 int getDoctorFee()
 {
return doctorFee;
          }
double getRoomCharges() {return roomCharges;}
void printBill()
{
cout<<right;
cout<<"Patient ID:"<<setw(19)<<getPatientId()<<"\n";
cout<<"Pharmacy Charges:"<<setw(13)<<getPharmacyCharges()<<"\n";
cout<<"Doctor Fee:"<<setw(19)<<getDoctorFee()<<"\n";
cout<<"Room Charges:"<<setw(17)<<getRoomCharges()<<"\n";
cout<<setfill('_');
cout<<setw(31)<<"\n";
cout<<"Total Charges:"<<setfill('.')<<setw(16)<<totalChares<<"\n";
}
void billAmount(int noDays=1)
{
```

```
totalChares = (pharmacyCharges+doctorFee+roomCharges*noDays);
}
~billType()
{
cout<<"\nBill Type of Patient ID "<< getPatientId()<<" ~Destructor
Invoked\n";
}

};
```

Base Class with Parameterized Constructor

```
class dateType
{
public:
void setDate(int month, int day, int year);
//Function to set the date.

//The member variables dMonth, dDay, and dYear are set //according to the parameters.
//Postcondition: dMonth = month; dDay = day;
          // dYear = year

int getDay() const;
Function to return the day

int getMonth() const;
//Function to return the month
//post condition: the value of dmonth is returned.

int getYear() const;
//Function to return the year
//post condition: the value of dYear is returned.

void printDate() const;
//Function that outputs the date in the form mm-dd-yyyy.

dateType(int month =1, int day = 1, int year = 1900);

//Constructor to set the date
//The member variables dMonth, dDay, and dYear are set
//according to the parameters.
//Postcondition: dMonth = month; dDay = day; dYear = year;
```

```
//        If no values are specified, the default
// values are used to initialize the member
//variables
~dateType();
private:
int dMonth; //variable to store month
int dDay; //variable to store the day
int dYear; //variable to store the year

void dateType::setDate(int month, int day, int year)
{
dMonth = month;
dDay = day;
dYear = year;
}

int dateType::getDay() const
{
return dDay;
}

int dateType::getMonth() const
{
return dMonth;
}

int dateType::getYear() const
{
return dYear;
}

void dateType::printDate() const

{
cout << dMonth << "-" << dDay << "-" << dYear;
}

//Constructor with parameters dateType::dateType(int month, int day, int year) {

dMonth = month;
dDay = day;
dYear = year;
}
```

```
//desctructor
dateType::~dateType()
{
cout<<"\nDate Type ";
printDate();
cout<<" ~Destructor Invoked "<<endl;

}
```

Composition (Aggregation)

Aggregation shows **has a relation**. Patient Type class implements this concept while creating three objects of date Type class as well one object of bill Type Class and one object of doctor Type.

- Patient class **has**
  - **three** objects of date type class
    - dob
    - admitDate
    - dischargeDate
  - **one** object of bill type class
    - patientBill
  - **one** object of doctor type class
    - physician

```
class patientType: public personType // inheritance
{
private:
int id;
int age;
//----------------------Composition
dateType dob; // dob object will be created in patient's object
dateType admitDate; //admitDate object .....
dateType dischargeDate; // dischageDate object....


int calculateDays()
{
int years,months,days;

years = (dischargeDate.getYear() - admitDate.getYear())*365; months = (dischargeDate.getMonth() -
admitDate.getMonth())*30; days = (dischargeDate.getDay() - admitDate.getDay());

return (years + months + days);
}
```

```
public:
//--------------------------------composition
billType patientBill; // patientBill object will be created inside patient object
//--------------------------------composition

doctorType physician; // creation of doctoryType object //----------------------------------------

patientType()

{
setId(0);
setAge(0);
patientBill.setPatientId(0);
}
patientType(int i, int a)
{
setId(i);
setAge(a);
patientBill.setPatientId(i);

}
patientType(int i, int a, string fname, string lname):personType(fname,lname)
{

setId(i);
patientBill.setPatientId(i); //to print on bill

setAge(a);
}

void setId(int a)
{
id = a;
}
void setAge(int b)
{
age = b;
}
int getAge()
{
return age;
}
int getId()
{
return id;
}
void admitPatient(dateType d1, dateType d2)
{

dob.setDate(d1.getMonth(),d1.getDay(),d1.getYear()); admitDate.setDate(d2.getMonth(),d2.getDay(),d2.getYear());
```

```
}
void dischargPatient(dateType d)
{
dischargeDate.setDate(d.getMonth(),d.getDay(),d.getYear());
}
void calculateBill()
{
cout<<" Calculated days"<<calculateDays();
patientBill.billAmount(calculateDays());
}
void printPatientBill()
{
cout<<setfill('-');
cout<<setw(31)<<"\n";
cout<<"Patient Bill Details\n";
cout<<setw(31)<<"\n";
cout<<setfill('.');
//---------------
cout<<"\nPhysician: \t";
// calling person type's print through doctorType object
physician.personType::print();
cout<<"\t\t"<<physician.getSpeciality()<<endl;
//----------------

cout<<"Admit Date:            ";
admitDate.printDate();
cout<<"\nDischarge Date:   ";
dischargeDate.printDate();
cout<<"\n";
patientBill.printBill();
}

void print()
{

cout<<"\n\nPrinting patient information... \n"; personType::print();
cout<<"\tID: "<<getId()<<"\t Age: "<<getAge()<<endl<<endl<<endl;
cout<<"\nDOB: ";

dob.printDate();
cout<<"\n";


}
~patientType()
{
cout<<"\n Patient Type "<<getId()<<" ~ Destructor Invoked" ;
}

};
```

Execution Order of Constructors & Destructors in Inheritance & Composition

```
void main()
{

	{// This block of curly braces {} is added intent all
	//to clarify the order of destructors invocation
	 //---------------------------------------
	cout<<"\n Playing with personType Objects\n";
	personType pt1("Nauman","Asgar");
	pt1.print();
	//---------------------------------------
	cout<<"\n Playing with doctorType Objects\n";
	doctorType d("Asfand","Ali","Ear Specialist");
	d.print();
	//---------------------------------------
	cout<<"\n Playing with billType Objects\n";
	billType b1;
	b1.printBill();
	//---------------------------------------
	cout<<"\n Playing with dateType Objects\n";
	dateType d1;
	dateType d2(02,15,2012);
	dateType d3(03,25,2013);
	d1.printDate();
	cout<<endl;
	d2.printDate();
	cout<<endl;
	d3.printDate();
	cout<<endl;
	//---------------------------------------
	cout<<"\n Playing with patientType Objects\n";
	patientType p(111,20,"Rabah","Khan");
	p.admitPatient(d1,d2);
	p.physician.setName("Basit","Ashraf");
	p.physician.setSpeciality("Eye Specialist");
	p.patientBill.setDoctorFee(120);
	p.patientBill.setPharmacyCharges(50);
	p.patientBill.setRoomCharges(19);
	p.dischargePatient(d3);
	p.calculateBill();
	p.print();
	p.printPatientBill();
	//-----------------------------------------------
	cout<<"\n\n\n Many Destructors will be invoked now ...\n";
```

```
}
cout<<endl;
system("PAUSE");
}
```

```
/*
```

```
Playing with personType Objects

Person Type Constructor Invoked

Nauman Asgar
```

```
Playing with doctorType Objects

Person Type Constructor Invoked

       Doctor Type Parametrized Contructor Invoked

Printing doctor information

Doctor Name: Asfand Ali

       Specialist: Ear Specialist
```

```
Playing with billType Objects

Bill Type Constructor Invoked

Patient ID:                        0

Pharmacy Charges:                  0

Doctor Fee:                        0

Room Charges:                      0

_____________________________

Total Charges:.............      0
```

```
Playing with dateType Objects

1-1-1900

2-15-2012

3-25-2013
```

```
Playing with patientType
ObjectsPerson Type Constructor Invoked
Bill Type Constructor Invoked
Person Type Constructor Invoked
Doctor Type Default Contructor Invoked
Date Type 1-1-1900 ~Destructor Invoked
Date Type 2-15-2012 ~Destructor Invoked
Date Type 3-25-2013 ~Destructor Invoked
 Calculated days405
Printing patient information...
Rabah Khan
        ID: 111   Age: 20


DOB: 1-1-1900
 ------------------------------
Patient Bill Details
 ------------------------------
Physician:      Basit Ashraf
                Eye Specialist
Admit Date:            2-15-2012
Discharge Date:        3-25-2013
Patient ID:.............        111
Pharmacy Charges:.......          50
Doctor Fee:.............        120
Room Charges:...........          19

Total Charges                  7865
```

```
 Many Destructors will be invoked now ...
 Patient Type 111 ~ Destructor Invoked

Doctor TypeBasit - Eye Specialist ~Destructor Invoked
Person Type Basit ~Destructor Invoked
Bill Type of Patient ID 111 ~Destructor Invoked
Date Type 3-25-2013 ~Destructor Invoked
Date Type 2-15-2012 ~Destructor Invoked
Date Type 1-1-1900 ~Destructor Invoked
Person Type Rabah ~Destructor Invoked
Date Type 3-25-2013 ~Destructor Invoked
Date Type 2-15-2012 ~Destructor Invoked
Date Type 1-1-1900 ~Destructor Invoked
Bill Type of Patient ID 0 ~Destructor Invoked
Doctor TypeAsfand - Ear Specialist ~Destructor Invoked
Person Type Asfand ~Destructor Invoked
Person Type Nauman ~Destructor Invoked
Press any key to continue . . .

*/
```