# Experiment No. 9

# Inheritance

## Objectives:

Things that will be covered in today's lab:

- Inheritance

## Theory:

**Inheritance:** The process by which a class incorporates the attributes and behaviors of a previously defined class. The new classes that we create from the existing classes are called *derived* classes and the existing classes are called *base* classes. Derived classes inherit the properties of base classes. Therefore, rather than creating completely new classes from scratch, we can take advantage of inheritance and reduce software complexity. Each derived class, in turn, becomes a base class for a future derived class. Inheritance can be either single inheritance or multiple inheritances. In single inheritance, the derived class is derived from a single base class; in multiple inheritances, the derived class is derived from more than one base class.

```
class derived-class: access-Specifier base-class
{
//member list
};
```

Where *access-specifier* is one of **public, protected,** or **private**, and base-class is the name of a previously defined class. If the *access-specifier* is not used, then it is private by default.

When deriving a class from a base class, the base class may be inherited through *public***,** *protected* or *private* inheritance. The type of inheritance is specified by the *access-specifier* as explained above.

We hardly use *protected* or *private* inheritance, but public inheritance is commonly used. While using different types of inheritance, the following rules are applied:

**Public Inheritance:** When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class but can be accessed through calls to the public and protected members of the base class

**Protected Inheritance:** When deriving from a protected base class, public and protected members of thebase class become protected members of the derived class.

**Private Inheritance:** When deriving from a private base class, public and protected members of thebase class become private members of the derived class.

Where *access* is one of **public, protected,** or **private** and would be given for every base classand they will be separated by comma as shown above. Let us see the following

Suppose that we have defined a class called **shape**. The following statements specify that the **class circle** is derived from **shape**, and it is a **public** inheritance.

```
class circle: public shape
        {
        .
        .
        .
        };
```

On the other hand, consider the following definition of the **class circle**:

```
class circle: private shape
        {
        .
        .
        .
        };
```

This is a **private** inheritance. In this definition, the **public** members of **shape** become **private** members of the **class circle**. So any object of type **circle** cannot directly access these members. The previous definition of **circle** is equivalent to:

```
class circle: shape
        {
        .
        .
        .
        };
```

That is if we do not use either the **member access specifier public** or **private**, the **public** members of a base class are inherited as **private** members by default.

**Exercise 1**: **What should be the output of this program?**

```
#include <iostream>
using namespace std;

// Base class
class Shape {
protected:
    double area;
public:

    Shape(){ cout<<”Default constructor of shape called”<<endl;}
    ~Shape(){ cout<<”Destructor of shape called”<<endl;}


     double getArea() {
        return area;
    }

 };

// Derived class
class Rectangle : public Shape {
private: double length, width;
public:
    void CalculateArea() {
        area= width * height;
    }
    Rectangle(double w, double l)
    { cout<<”Parameterized constructor of Rectangle called”<<endl;
       Width=w;
      length=l;
    }
     ~Rectangle(){ cout<<”Destructor of Rectangle called”<<endl;}


};

int main() {
    Rectangle rect(2,3);

    // calculate area of rectangle
    rect.CalculateArea();

    // Accessing base class method
    cout << "Area of the rectangle: " << rect.getArea() << endl;

    return 0;
}
```

**For all of the exercises below save your code on the learning management system (LMS) and give the screenshot of the output you get on the console in the space provided after every exercise.**

## Exercise 2:

Define **a class called bankAccount** to store a bank customer's account number and balance. Suppose that account number is of type **int**, and balance is of type **double**. Your class should, at least, provide the following operations: set the account number, retrieve the account number, retrieve the balance, deposit and withdraw money, and print account information. Add appropriate constructors.

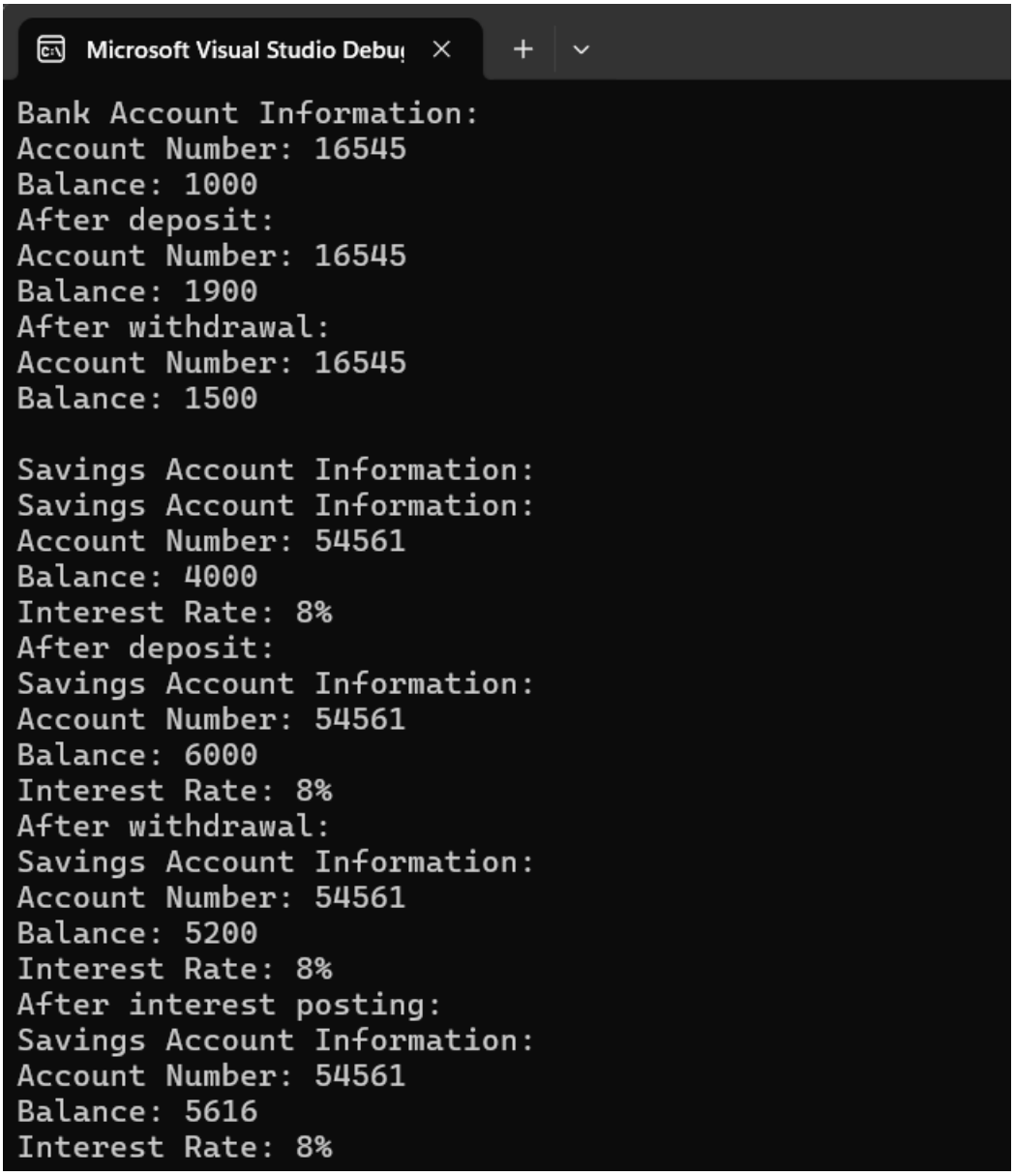
Every bank offers a savings account. Derive the **class savingsAccount** from the **class bankAccount**. This class inherits members to store the account number and the balance from the base class. A customer with a savings account typically receives interest, makes deposits, and withdraws money. In addition to the operations inherited from the base class, this class should provide the following operations: set interest rate, retrieve interest rate, post interest, and print account information. Add appropriate constructors.

Write a program to test your classes designed.

```
#include <iostream>
using namespace std;

class bankAccount {
protected:
   int accountNumber;
   double balance;

public:
   // Constructor
   bankAccount(int accNum, double bal) : accountNumber(accNum), balance(bal) {}

   // Member functions
   void setAccountNumber(int accNum) {
       accountNumber = accNum;
   }

   int getAccountNumber() const {
       return accountNumber;
   }

   double getBalance() const {
       return balance;
```

```
    }

    void deposit(double amount) {
        balance += amount;
    }

    void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        }
        else {
            cout << "Insufficient funds!" << endl;
        }
    }

    void printAccountInfo() const {
        cout << "Account Number: " << accountNumber << endl;
        cout << "Balance: " << balance << endl;
    }
};

class savingsAccount : public bankAccount {
private:
    double interestRate;

public:
    // Constructor
    savingsAccount(int accNum, double bal, double intRate) : bankAccount(accNum,
bal), interestRate(intRate) {}

    // Member functions
    void setInterestRate(double intRate) {
        interestRate = intRate;
    }

    double getInterestRate() const {
        return interestRate;
    }

    void postInterest() {
        balance += balance * (interestRate / 100);
    }

    void printAccountInfo() const {
        cout << "Savings Account Information:" << endl;
```

```
        bankAccount::printAccountInfo();
        cout << "Interest Rate: " << interestRate << "%" << endl;
    }
};

int main() {
    //  bankAccount class
    bankAccount account1(16545, 1000);
    cout << "Bank Account Information:" << endl;
    account1.printAccountInfo();
    account1.deposit(900);
    cout << "After deposit:" << endl;
    account1.printAccountInfo();
    account1.withdraw(400);
    cout << "After withdrawal:" << endl;
    account1.printAccountInfo();

    //  savingsAccount class
    savingsAccount savings1(54561, 4000, 8);
    cout << endl << "Savings Account Information:" << endl;
    savings1.printAccountInfo();
    savings1.deposit(2000);
    cout << "After deposit:" << endl;
    savings1.printAccountInfo();
    savings1.withdraw(800);
    cout << "After withdrawal:" << endl;
    savings1.printAccountInfo();
    savings1.postInterest();
    cout << "After interest posting:" << endl;
    savings1.printAccountInfo();

    return 0;
}
```




## Exercise 3:

Design a system to manage employees within a company which has a name and sales tax number. Every **Employee** has a **name, ID**, **designation,** and **salary**. Managers have a department, engineer has a specialization and salesperson has a sales target.

Please decide the classes needed and their relationships. Write the prototype of these classes in a header file called company.h.

Implement these classes in a file called company.cpp

In a driver.cpp file write a main function that creates an object of company. Then creates a dynamic array of employees within the company. A sample is given below

```
int main()
{
```

```
// creates a company with name, sales tax number and number of employess
     Company *syst= new Company("System Private Ltd", "111-121-131", 10);

   for(int i=0; i< syst->getNumberEmployees(); i++)
   {
        //ask user the type of employee and add objects of manager, engineer or salesperson
    }

     Syst->DisplayEmployees();

    return 0;
}
```

**Header File**

```
#ifndef COMPANY_H
#define COMPANY_H

#include <string>
using namespace std;

class Employee {
protected:
    string name;
    int ID;
    string designation;
    double salary;

public:

    Employee(string empName, int empID, string empDesignation, double empSalary);

    ~Employee();

   virtual void display();
};

class Manager : public Employee {
private:
    string department;

public:
    Manager(string empName, int empID, string empDesignation, double empSalary,
string empDepartment);
```

```
    void display();
};

class Engineer : public Employee {
private:
    string specialization;

public:
    Engineer(string empName, int empID, string empDesignation, double empSalary,
string empSpecialization);

    void display();
};

class Salesperson : public Employee {
private:
    double salesTarget;

public:
    Salesperson(string empName, int empID, string empDesignation, double
empSalary, double empSalesTarget);

    void display();
};

class Company {
private:
    string name;
    string salesTaxNumber;
    int numberOfEmployees;
    Employee** employees;

public:
    Company(string companyName, string companySalesTaxNumber, int numEmployees);

    ~Company();

    void addEmployee(Employee *emp,int index);

    int getNumberEmployees();

    void displayEmployees();
};

#endif
```

```cpp
// CPP
#include "company.h"
#include <iostream>
using namespace std;

// Employee class

Employee::Employee(string empName, int empID, string empDesignation, double
empSalary)
    : name(empName), ID(empID), designation(empDesignation), salary(empSalary) {}

Employee::~Employee() {}

void Employee::display() {

    cout << "Name: " << name << ", ID: " << ID << ", Designation: " <<
designation << ", Salary: $" << salary << endl;


}
// Manager class
Manager::Manager(string empName, int empID, string empDesignation, double
empSalary, string empDepartment)
    : Employee(empName, empID, empDesignation, empSalary),
department(empDepartment) {}

void Manager::display() {
    cout << "Manager Information:" << endl;
    Employee::display();
    cout << "Department: " << department << endl;

}

// Engineer class
Engineer::Engineer(string empName, int empID, string empDesignation, double
empSalary, string empSpecialization)
    : Employee(empName, empID, empDesignation, empSalary),
specialization(empSpecialization) {}

void Engineer::display() {
    cout << "Engineer Information:" << endl;
    Employee::display();
    cout << "Specialization: " << specialization << endl;
```

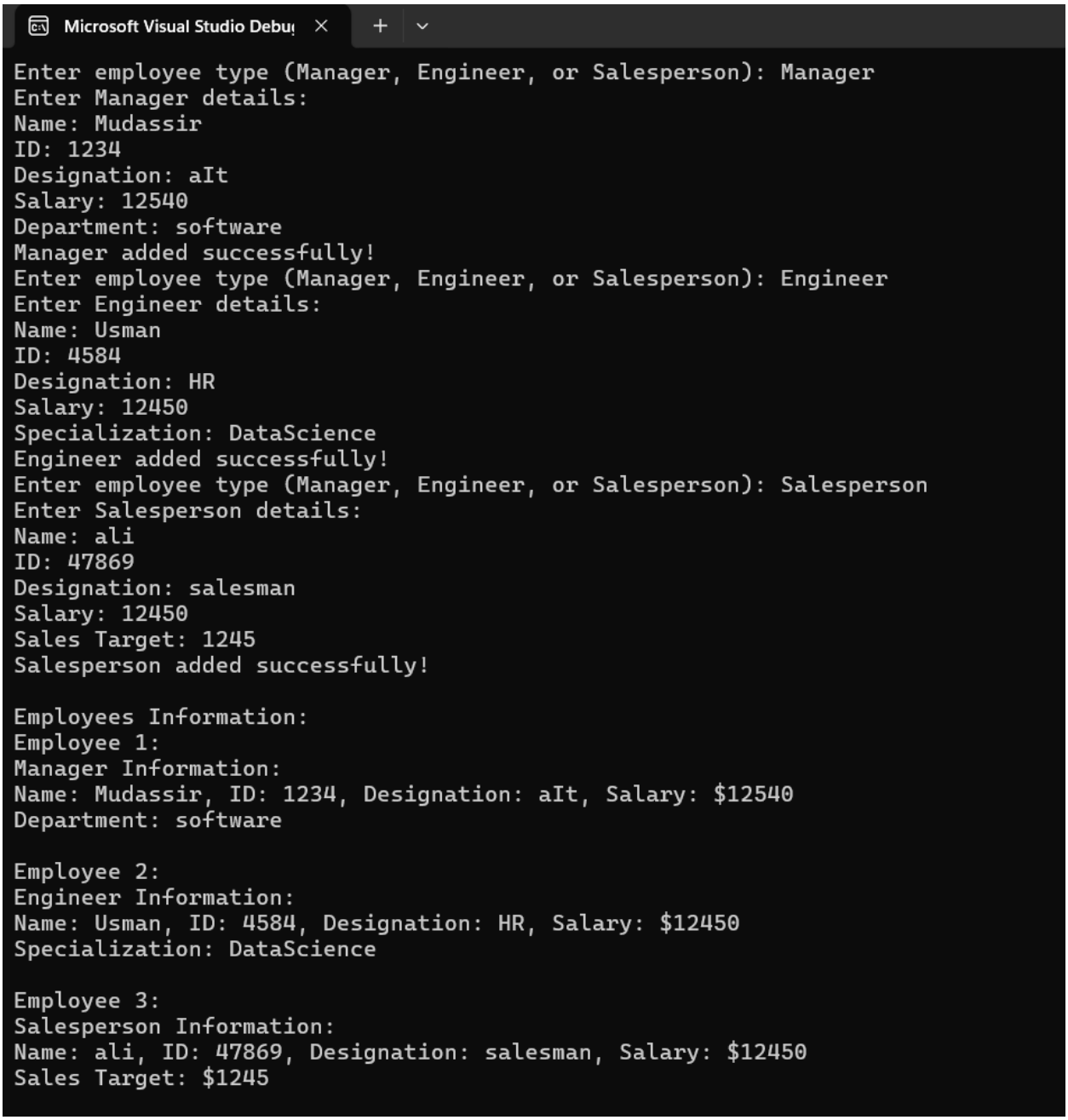
```
}

// Salesperson class
Salesperson::Salesperson(string empName, int empID, string empDesignation, double
empSalary, double empSalesTarget)
    : Employee(empName, empID, empDesignation, empSalary),
salesTarget(empSalesTarget) {}

void Salesperson::display() {
    cout << "Salesperson Information:" << endl;
    Employee::display();
    cout << "Sales Target: $" << salesTarget << endl;

}

// Company class
Company::Company(string companyName, string companySalesTaxNumber, int
numEmployees)
    : name(companyName), salesTaxNumber(companySalesTaxNumber){

    numberOfEmployees = numEmployees;

  employees = new Employee*[numEmployees];
}

Company::~Company() {
    for (int i = 0; i < numberOfEmployees; i++) {
        delete employees[i];
    }
    delete[] employees;
}

void Company::addEmployee(Employee * emp,int index) {
    employees[index] = emp;
}

int Company::getNumberEmployees() {
    return numberOfEmployees;
}

void Company::displayEmployees() {
    for (int i = 0; i < numberOfEmployees; i++) {

        cout << "Employee " << (i + 1) << ":" << endl;
        employees[i]->display();
```

```
        cout << endl;
    }
}
```

**Driver**

```
// driver.cpp

#include "company.h"
#include <iostream>

int main() {
    // Create a company with name, sales tax number, and number of employees
    Company* syst = new Company("System Private Ltd", "111-121-131", 3);

    for (int i = 0; i < syst->getNumberEmployees(); i++) {
        // Ask user the type of employee and add objects of manager, engineer or
salesperson
       string empType;
       cout << "Enter employee type (Manager, Engineer, or Salesperson): ";
       cin >> empType;

        string name;
        int ID;
        string designation;
        double salary;

        cout << "Enter " << empType << " details:" << std::endl;
        cout << "Name: ";
        cin >> name;
        cout << "ID: ";
        cin >> ID;
        cout << "Designation: ";
        cin >> designation;
        cout << "Salary: ";
        cin >> salary;

        if (empType == "Manager") {
            string department;
            cout << "Department: ";
            cin >> department;
            syst->addEmployee (new Manager(name, ID, designation, salary,
department),i);
```

```
            std::cout << "Manager added successfully!" << endl;
        }
        else if (empType == "Engineer") {
            string specialization;
            cout << "Specialization: ";
            cin >> specialization;
            syst->addEmployee( new Engineer(name, ID, designation, salary,
specialization),i);
            cout << "Engineer added successfully!" << endl;
        }
        else if (empType == "Salesperson") {
            double salesTarget;
            cout << "Sales Target: ";
            cin >> salesTarget;
            syst->addEmployee(new Salesperson(name, ID, designation, salary,
salesTarget),i);
            cout << "Salesperson added successfully!" << endl;
        }
    }

    cout << "\nEmployees Information:\n";
    syst->displayEmployees();

    delete syst;

    return 0;
}
```




## Multiple Inheritance:

C++ class can inherit members from more than one class and here is the extended syntax:

```
class derived-class: access base-a, access base-b...
{
//member list
```

```
};
```

**Example: Understand the concept of multiple inheritance by the following example and predict the output of the program**

```
#include <iostream>
#include <string>

// Base class 1
class Bird {
public:
    // Bird-specific action
    void fly() const {
        std::cout << "Flying in the sky!" << std::endl;
    }
};

// Base class 2
class Fish {
public:
    // Fish-specific action
    void swim() const {
        std::cout << "Swimming in the water!" << std::endl;
    }
};

// Derived class
class Seabird : public Bird, public Fish {
public:
    // Display actions of seabird
    void displayActions() const {
        fly();  // Accessing fly() from Bird
        swim(); // Accessing swim() from Fish
    }
};

int main() {
    // Create a seabird object
    Seabird seabird;

    // Display actions of seabird
    seabird.displayActions();

    return 0;
}
```