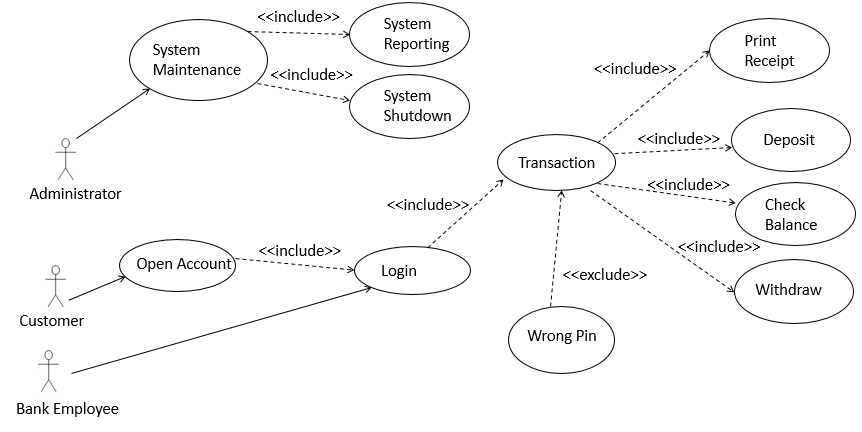
**Account Balance Check**

**Use Case Scenario**

“XYZ Bank Ltd” has computer-based Banking Information management system. They offer online customer Bank Account opening process. A customer can apply for bank account from anywhere in the world using web-based interface and submit all the required documents using online. The customer can create an account. After creating account customer can login by giving user name and password. For transaction customer must have login first,customer may give wrong PIN, then system will not authorize the customer for transaction. By transaction customer can perform certain things such as withdraw money,check money,deposit money,print receipt. A bank employee can login into a customer account. Every bank account sytem has administrator whose job is to maintain the system. System maintenance must have two things one is syatem reporting and another systesm is shutdown.



*Fig: Use Case Diagram*

**Details description:**

At first we have to identify the actor of the existing scenario. Then we have to find out the actor, here the actor we have customer, administrator and bank employee. For denoting the actor we use human sign. Generally, we use oval shape for activity, use **include** for must include activity, and **exclude** for may be activity.

For this scenario customer can open a bank account, by opening an account customer get his/her username and password for login. A customer must need to login for any type of transaction. So draw open account to login activity must include activity. Then transaction must include print receipt, withdraw, check balance, and deposit that’s why we use include.

After that, bank employee must need to login to manage the customer transaction. That’s why in scenario we use include for transaction. Now for transaction must include print receipt, deposit, check balance, and withdraw that’s why we use include.

We use administrator activity from the scenario for maintaining the system and it’s must perform system shutdown and system reporting. So, we use include for those activity.

When a customer give wrong pin for transaction its show an error so we use exclude for this activity as its sometime occur not every time.

**Class Diagram Scenario**

“ABC” is a bank popular bank in Bangladesh. It has many branch, every branch has branch code and city name. Branch can add account, remove account, collect money, add loan, remove loan. Every Customer has his/her own customer id, customer name, address, phone number and account number. Customer can deposit money, open account money, close account, borrow loan. Every account has its unique account id and customer id. For transaction customer need an account and need to have some amount of money in account. Customer can borrow loan from branch for this need account id and customer id. In savings account and current account calculate the charges and interest.

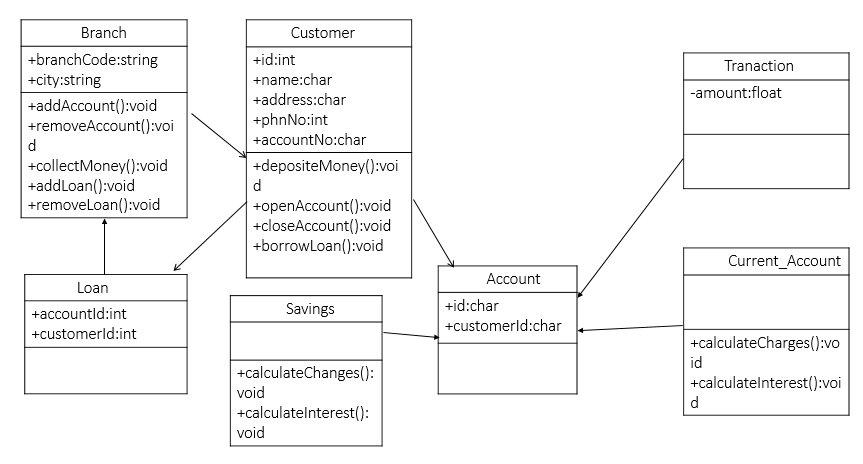


Fig: *Class Diagram*

**Details description:**

For drawing class diagram at first we have to identify class from the scenario. In this scenario we got some of classes those are Loan, Savings, Account, Current\_Account, Branch, Transaction, and Customer.

For branch class we use branchCode and city as an attributes. We use AddAccount, removeAccount, collectMoney, addLoan, removeLoan as a method because branch can performs these certain activities for branch class. In customer class we have some attributes those are id, name, address, phnNo, accountNo, also have some methods those are DepositMoney, openAccount, CloseAccount, BorrowMoney because cutomer perform these activities.

Loan also have some attributes those are accountId and customerId. In account class we have Id, and customerId attributes. Saving class we have calculateChanges, calculateInterest as method. Current\_Account has calculateChanges, calculateInterest as method. Transaction has one attribute amount. Transaction, saving, Current\_Account inherit from Account. Customer own an account that’s why we use association. Customer can take loan so we use association. Loan inherit from Branch.

**Activity Diagram Scenario**

In a bank management system, a customer has an account. Now, customer wants to know his account balance. For this, a bank employee enter his/her name into the interface of enter your name then enter his account number into the account number interface. Then verify that information that it is valid or invalid if it is valid then it will check whether it has balance zero or not. If it has, balance then will go to the account and can do the some of operation like transfer balance. I fit has zero balance customer will show that his/her account is null. I account number is invalid customer will not see his/her balance. If the account number is invalid then he will get chance to re-enter his account number.

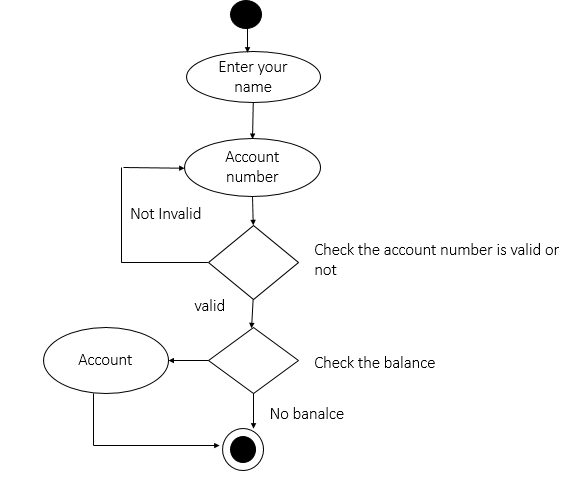


Fig: *Activity Diagram*

**Details description:**

For activity diagram we use some symbol. We use diamond shape for denote decision making, black circle for start symbol where we start, oval shape for activity what we do, and black circle with white border for end where we end our activity.

For account checking of a customer a bank employee first enter the name of the account holder then give the account number. Then the system will authenticate that the given account number is correct or wrong. In case of wrong the system will promote reenter account number then bank employee renter the account number. I f account number is valid the system will go to check is there any balance or not. If any balance the system go to the account for windrow amount, transfer amount like these operations. Then system end the activity. If no balance the system will end immediately.

**Sequence Diagram Scenario**

In an ATM system account holder gives the identity to the ATM then ATM request for authentication to the account holder and account holder provide authentication to the ATM. Then ATM request for authentication identity to bank. If the information is matched then bank confirm identity and give those Information to the ATM.

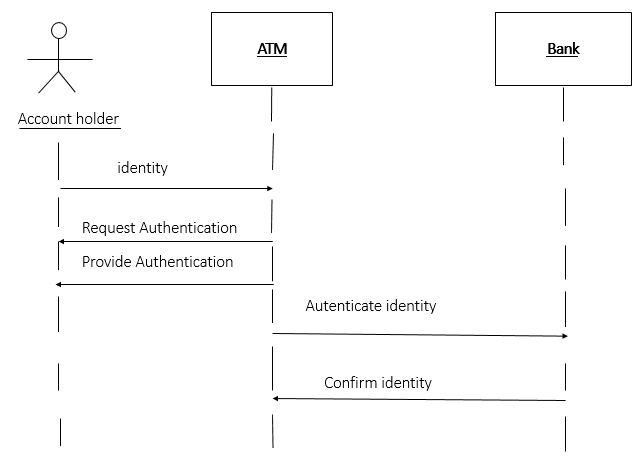


Fig: *Sequence Diagram*

**Details description:**

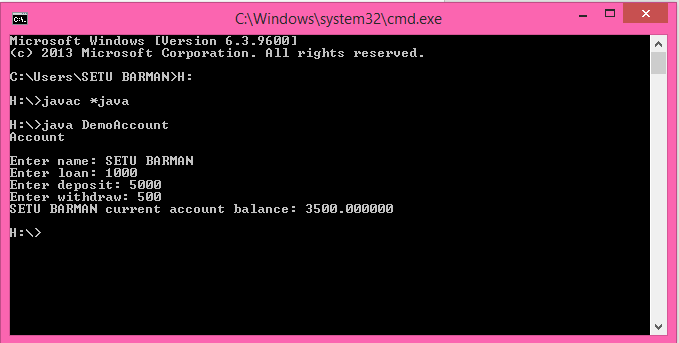
For sequence diagram, first we need to identify the object of out=r system. In above scenario, account holder, ATM and bank are our system object.

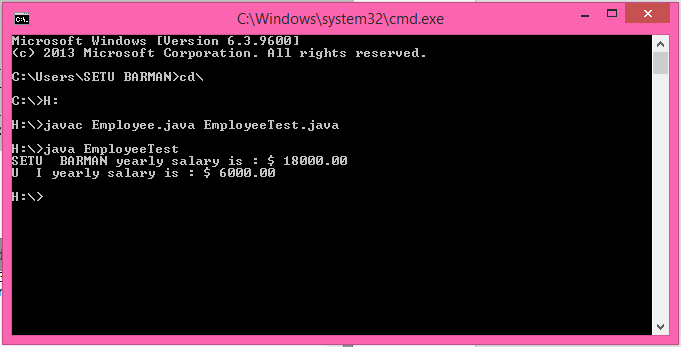
In this scenario first account holder give his/her identity to ATM so arrow shows Account holder to ATM. Then ATM authenticate the identity and provide the authentication to the account holder. So arrow are show ATM to account holder.

Then ATM authenticate the identity of account holder and bank will confirm the identity is valid or invalid. So arrow shows bank to ATM.

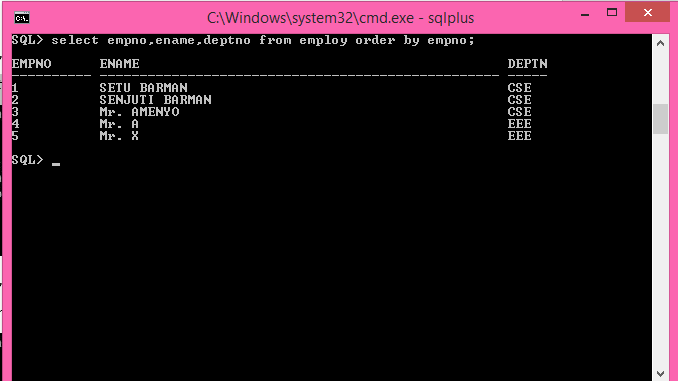
**Balance Check GUI:**

**Account details**





**Employee Table**



**Source Code**

**Create Table**

*create table employ(*

*empno varchar2(10),*

*ename varchar2(50),*

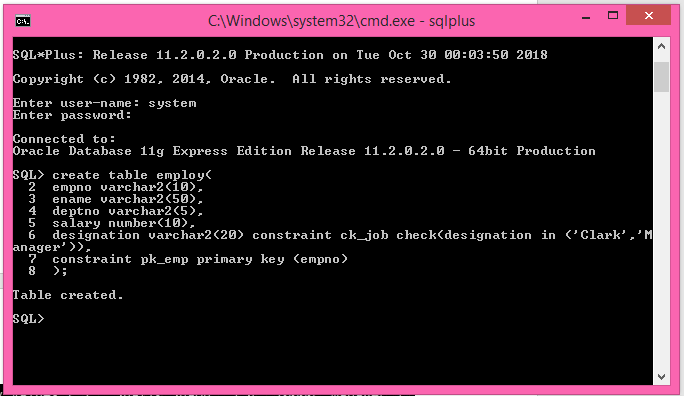
*deptno varchar2(5),*

*salary number(10),*

*designation varchar2(20) constraint ck\_job check(designation in ('Clark','Manager')),*

*constraint pk\_emp primary key (empno)*

*);*



**Data Insert:**

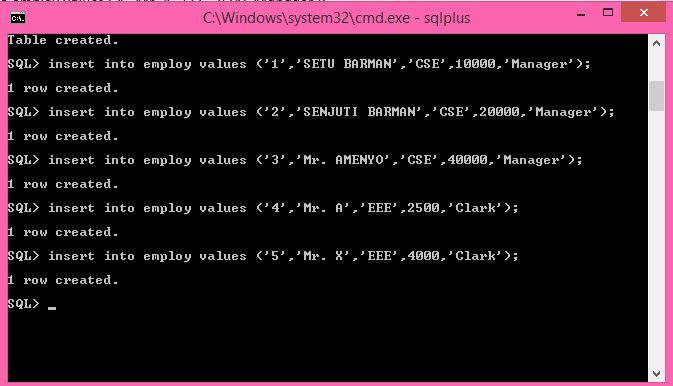
*insert into employ values ('1','SETU BARMAN','CSE',10000,'Manager');*

*insert into employ values ('2','SENJUTI BARMAN','CSE',20000,'Manager');*

*insert into employ values ('3','Mr. AMENYO','CSE',40000,'Manager');*

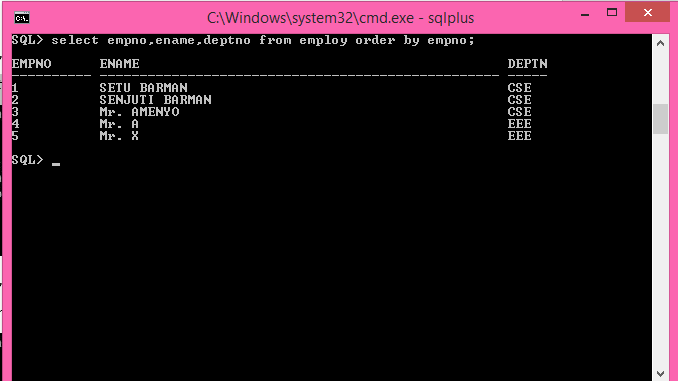
*insert into employ values ('4','Mr. A','EEE',2500,'Manager');*

*insert into employ values ('5','Mr. X','EEE',4000,'Clark');*



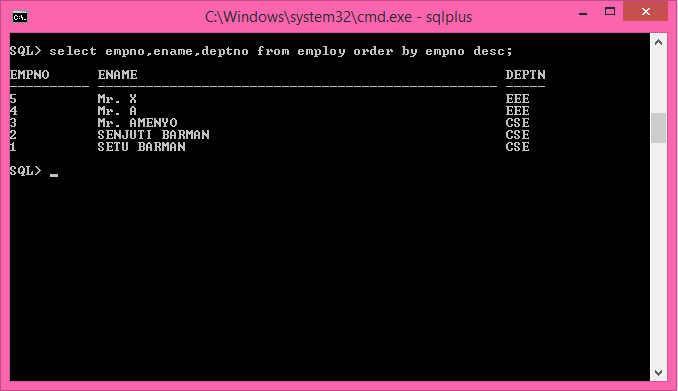
1. Display ename, e,pno, deptno from employ table sorted [ascending] on empno:

*select empno,ename,deptno from employ order by empno;*



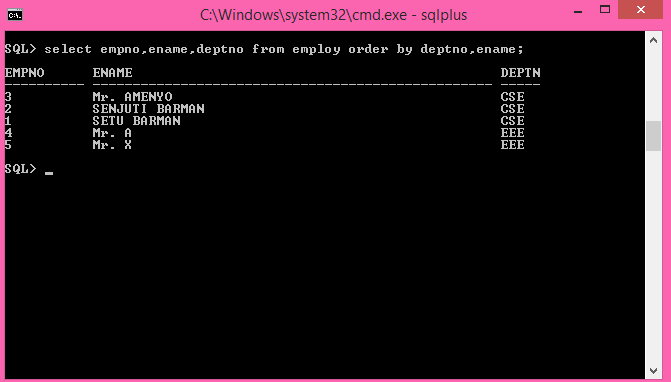
2. Display ename, e,pno, deptno from employ table sorted [decending] on empno:

*select empno,ename,deptno from employ order by empno desc;*



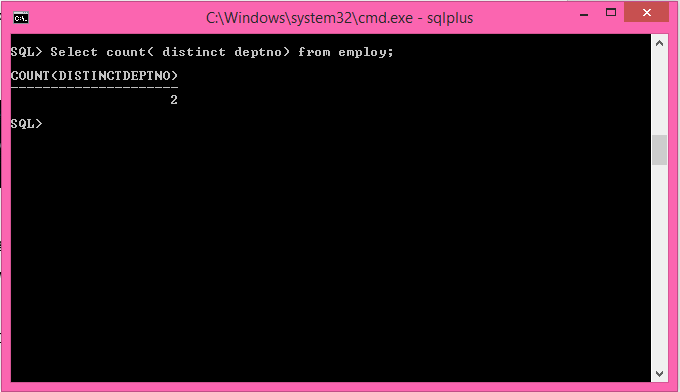
3. Sorting on multiple columns:

*select empno,ename,deptno from employ order by deptno,ename;*



4. Display the number of distinct deptno in the employ table: [distinct : to avoid duplicates]

*Select count( distinct deptno) from employ;*



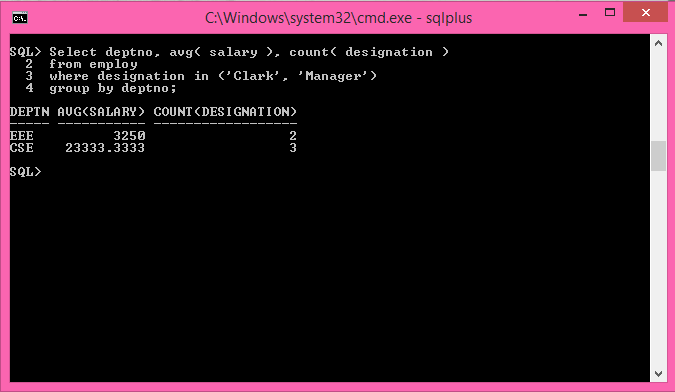
5. Determine the average salary and number of MANAGER and CLARK in each department:

*Select deptno, avg( salary ), count( designation )*

*from employ*

*where designation in ('Clark', 'Manager')*

*group by deptno;*

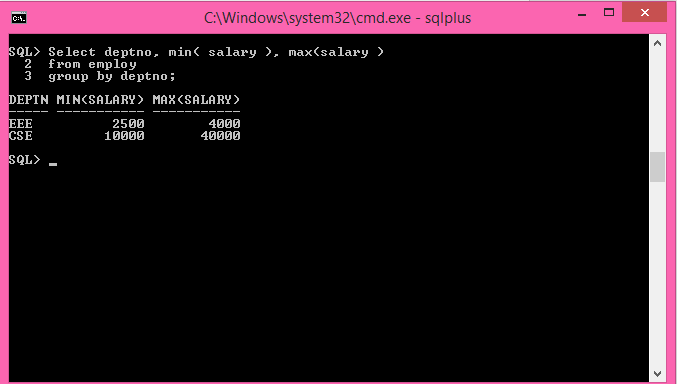


6. Write a query to display deptno, lowest and highest salary of each department:

*Select deptno, min( salary ), max(salary )*

*from employ*

*group by deptno;*



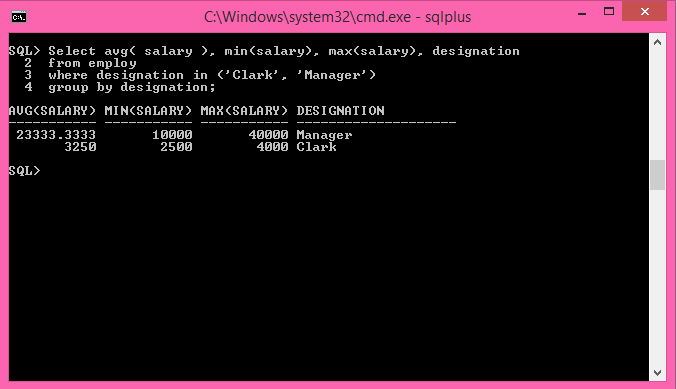
7. Compute the average, minimum, and maximum salaries of those group of employees having the designation of **clerk** or **manager**:

*Select avg( salary ), min(salary), max(salary), designation*

*from employ*

*where designation in ('Clark', 'Manager')*

*group by designation;*



**Having** clause: comes only with *group by* and after the group by operation  
**Syntax**: having < condition >

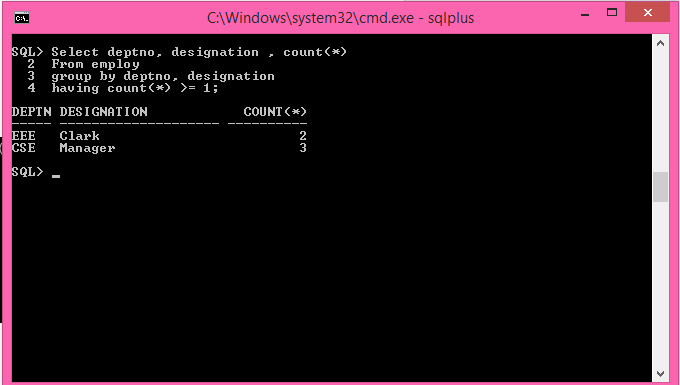
8. Write a query to display only those designation from a particular department having 1 or more employees working on it:

Select deptno, designation , count(\*)

From employ

group by deptno, designation

having count(\*) >= 1;



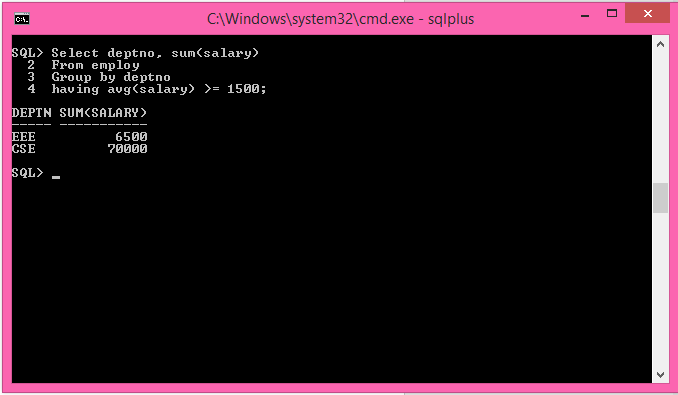
9. Write a query to display deptno, totalsalary of all employees who earn avg salary greater than or equal to 1500:

Select deptno, sum(salary)

From employ

Group by deptno

having avg(salary) >= 1500;



**JAVA**

**We can find employees name, Loan, Deposit, Withdraw, & Current Balance:**

**Account.java**

public class Account {

String name;

private double loan;

private double deposit;

private double withdraw;

private double savingsaccount;

public Account(String name, double loan, double deposit,double withdraw, double savingsaccount)

{

System.out.println("Account");

System.out.println("");

this.name=name;

this.loan=loan;

this.deposit=deposit;

this.withdraw=withdraw;

this.savingsaccount=savingsaccount;

}

public void setName(String name)

{

this.name=name;

}

public String getName()

{

return name;

}

public void setLoan(double newLoan)

{

loan=newLoan;

}

public double getLoan()

{

return loan;

}

public void loan()

{

System.out.println("Your loan ammount is "+this.loan);

}

public void setDeposit(double newDeposit)

{

deposit=newDeposit;

}

public double getDeposit()

{

return deposit;

}

public void deposit()

{

System.out.println("Your deposit ammount is "+this.deposit);

}

public void setWithdraw(double newWithdraw)

{

withdraw=newWithdraw;

}

public double getWithdraw()

{

return withdraw;

}

public void withdraw()

{

System.out.println("Your withdraw ammount is "+this.withdraw);

}

}

**SavingsAccount.java**

public class SavingsAccount extends Account {

private double savingsaccount;

public SavingsAccount(String name, double loan, double deposit, double withdraw, double savingsaccount)

{

super(name,loan,deposit,withdraw,savingsaccount);

setSavingsAccount(savingsaccount);

}

@Override

public void deposit()

{

System.out.println("Accountcheck to "+getName()+ "with balance "+savingsaccount);

}

private void setSavingsAccount(double newSavingsAccount) {

if(newSavingsAccount>=0.0)

{

savingsaccount=newSavingsAccount;

}

}

public double getSavingsAccount()

{

return savingsaccount;

}

public double computeBalance()

{

double result;

result = getDeposit()-(getLoan()+getWithdraw());

System.out.print(""+this.name+" ");

System.out.printf("current account balance: %f",result);

return savingsaccount;

}

}

**DemoAccount.java**

import java.util.Scanner;

public class DemoAccount {

public static void main(String[] args) {

String name;

double loan=0;

double deposit=0;

double withdraw=0;

double savingsaccount=0;

Scanner input = new Scanner(System.in);

SavingsAccount s= new SavingsAccount("name",loan,deposit,withdraw,savingsaccount);

System.out.print("Enter name: ");

name = input.nextLine();

s.setName(name);

System.out.print("Enter loan: ");

loan = input.nextDouble();

s.setLoan(loan);

System.out.print("Enter deposit: ");

deposit = input.nextDouble();

s.setDeposit(deposit);

System.out.print("Enter withdraw: ");

withdraw = input.nextDouble();

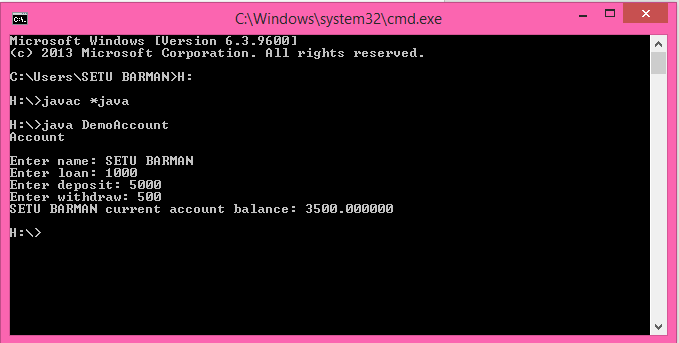
s.setWithdraw(withdraw);

s.computeBalance();

System.out.println("");

}

}



**#We can find employees yearly salary:**

**Employee.java**

**Source Code**

public class Employee{

private String firstname;

private String lastname;

private double salary;

public Employee(String firstname,String lastname,double salary){

this.firstname=firstname;

this.lastname=lastname;

this.salary=salary;

}

public void setFirstname(String firstname)

{

this.firstname=firstname;

}

public void setLastname(String lastname)

{

this.lastname=lastname;

}

public void setSalary(double salary)

{

if(salary>0.0)

this.salary=salary;

}

public String getFirstname()

{

return firstname;

}

public String getLastname()

{

return lastname;

}

public double getSalary()

{

return salary;

}

}

**EmployeeTest.java**

**Source Code**

public class EmployeeTest{

public static void main(String[] args){

Employee obj1 = new Employee("SETU","BARMAN",1500.00);

Employee obj2 = new Employee("U","I",500.00);

System.out.printf("%s %s yearly salary is : $ %.2f %n",obj1.getFirstname(),obj1.getLastname(),obj1.getSalary()\*12);

System.out.printf("%s %s yearly salary is : $ %.2f %n",obj2.getFirstname(),obj2.getLastname(),obj2.getSalary()\*12);

}

}

**Output:**

