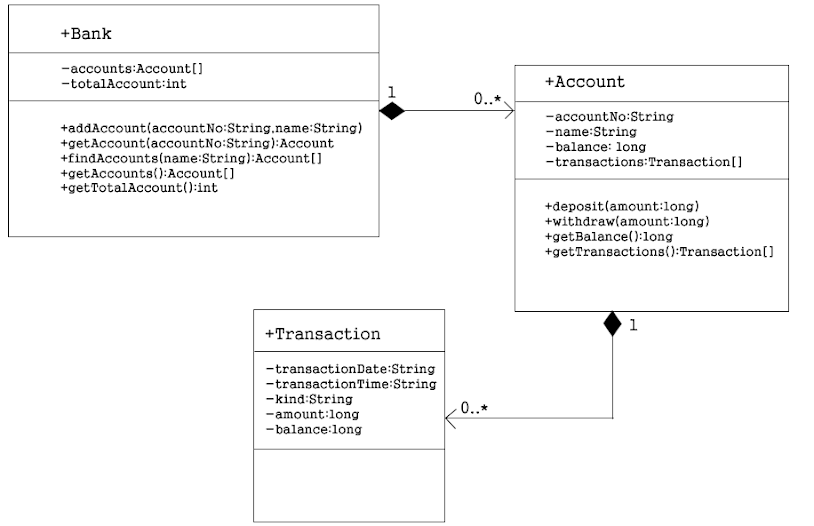
**Introduction to Distributed Systems (Lab)**

Revision of Object-oriented programming in JAVA

OOP features:

1. Encapsulation
2. Polymorphism
3. Inheritance

To see how to implement in JAVA as an example simple bank system has taken as shown below



In oop there are two main issues class and objects

* A class is blueprint or template of real-world entity which contains properties and methods as a member. Properties are characteristics of that class and methods are the task of the class. These members of a class can specify by access modifiers where do they can access like private, public or protected and -,+ and # are the symbol used to specify those identifiers respectively.
* An object is an instance of a class. So, it is possible to create many objects of a class.

From the above UML class diagram, it shows that there are three classes

1. Bank
2. Account
3. Transaction

So to implement in java

1. Create a package called JAVA
   1. In a JAVA package
      1. Main class called entry.java
      2. Bank class
      3. Transaction class
      4. Account class
2. Main class

public class entry {

    public static void main(String[] args) {

    }

}

ii. Bank class which has the following private properties

public class Bank{

private Account []  accounts = new Account[10];

private int totalAccount;

iii. transaction class which has the following private properties

public class Transaction {

  private String transactionDate;

  private String transactionTime;

  private String kind;

  private double amount;

  private double balance;

iv. Account class which has the following private properties

public class Account {

    private String accountNo;

    private String name;

    private double balance;

    private Transaction[] transactions = new Transaction[100];

    private int totalTransaction;

  static final SimpleDateFormat DATE\_FORMAT = new SimpleDateFormat("yyyy/MM/dd");

    static final SimpleDateFormat TIME\_FORMAT = new SimpleDateFormat("HH:mm:ss");

    static final String DEPOSIT = "Deposit";

    static final String WITHDRAW = "Withdrawal";

Note: all properties of Bank, Account and Transaction are private so they are hidden from out of the class. To make them accessible to outside the class without changing the access modifiers will define getter and setter property methods.

To define these methods the NetBeans IDE can generate by pressing **alt+insert** 🡺generate getter and setter

1. Bank class properties getter methods

public Account[] getAccounts() {

  return accounts;

}

public int getTotalAccount() {

  return totalAccount;

}

ii . Transaction class properties getter and setter methods

  public String getTransactionDate() {

    return transactionDate;

  }

  public void setTransactionDate(String transactionDate) {

    this.transactionDate = transactionDate;

  }

  public String getTransactionTime() {

    return transactionTime;

  }

  public void setTransactionTime(String transactionTime) {

    this.transactionTime = transactionTime;

  }

  public String getKind() {

    return kind;

  }

  public void setKind(String kind) {

    this.kind = kind;

  }

  public long getAmount() {

    return (long) amount;

  }

  public void setAmount(double amount) {

    this.amount = amount;

  }

  public long getBalance() {

    return (long) balance;

  }

  public void setBalance(double balance) {

    this.balance = balance;

  }

iii. Account class properties getter and setter methods

    public String getName() {

      return name;

    }

    public void setName(String name) {

      this.name = name;

    }

    public String getAccountNo() {

      return accountNo;

    }

    public void setAccountNo(String accountNo) {

      this.accountNo = accountNo;

    }

    public long getBalance() {

      return (long) balance;

    }

    public void setBalance(double balance) {

      this.balance = balance;

    }

    public Transaction[] getTransactions() {

      return transactions;

    }

    public int getTotalTransaction() {

      return totalTransaction;

    }

    public void setTotalTransaction(int totalTransaction) {

      this.totalTransaction = totalTransaction;

    }

    public void setTransactions(Transaction[] transactions) {

      this.transactions = transactions;

    }

Constructors : A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. (can generate by alt+insrt)

1. Account class constructors (Empty and parametrized constructors)

 public Account() {}

    public Account(String accountNo, String name) {

      this.accountNo = accountNo;

      this.name = name;

    }

    public Account(String accountNo, String name, double balance) {

      this.accountNo = accountNo;

      this.name = name;

      this.balance = balance;

    }

i.Transaction constructors (Empty and parametrized constructors)

  public Transaction() {

  }

  public Transaction(String transactionDate, String transactionTime, String kind, double amount, double balance) {

    this.transactionDate = transactionDate;

    this.transactionTime = transactionTime;

    this.kind = kind;

    this.amount = amount;

    this.balance = balance;

  }

toString method: The toString method for class Object returns a string consisting of the name of the class of which the object is an instance

i.Account toString method

@Override

    public String toString() {

      StringBuffer sb = new StringBuffer();

      sb.append(accountNo);

      sb.append("|");

      sb.append(name);

      sb.append("|");

      sb.append(balance);

      return sb.toString();

    }

ii.Transaction toString method

  @Override

  public String toString() {

    StringBuilder sb = new StringBuilder();

    sb.append(transactionDate);

    sb.append("|");

    sb.append(transactionTime);

    sb.append("|");

    sb.append(kind);

    sb.append("|");

    sb.append(amount);

    sb.append("|");

    sb.append(balance);

    return sb.toString();

  }

Bank class methods definition

When a customer opining an account there are two possibilities, he/she may deposit with initial balance or without balance. So, if a customer wants to open an account without initial balance(0) and execute the first method without balance else the second method. This is one kind of **polymorphisms**.

public void addAccount(String accountNo, String name) {

  accounts[totalAccount++] = new Account(accountNo, name);

}

public void addAccount(String accountNo, String name, double balance) {

  accounts[totalAccount++] = new Account(accountNo, name, balance);

}

Other bank methods are getAccount and findAccountByName implemented as follows.

public Account getAccount(String accountNo) {

  for (int i = 0; i < totalAccount; i++) {

    if (accountNo.equals(accounts[i].getAccountNo())) {

      return accounts[i];

    }

  }

  return null;

}

public Account[] findAccountByName(String name) {

  Account[] temp = new Account[totalAccount];

  int total = 0;

  for (int i = 0; i < totalAccount; i++) {

    if (name.equals(accounts[i].getName())) {

      temp[total++] = accounts[i];

    }

  }

  Account[] matched = new Account[total];

  for (int i = 0; i < total; i++) {

    matched[i] = temp[i];

  }

  return matched;

}

Account class method implementations as follows when a customer needs to deposit and withdraw enter the amount he/she needs to do.

  public void deposit(double amount) {

      balance = balance + amount;

      Transaction transaction = new Transaction();

      Calendar cal = Calendar.getInstance();

      Date date = cal.getTime();

      transaction.setTransactionDate(DATE\_FORMAT.format(date));

      transaction.setTransactionTime(TIME\_FORMAT.format(date));

      transaction.setAmount(amount);

      transaction.setBalance(balance);

      transaction.setKind(DEPOSIT);

      transactions[totalTransaction++] = transaction;

    }

    public void withdraw(double amount) {

      if (amount > balance) {

        return;

      }

      balance = balance - amount;

      Transaction transaction = new Transaction();

      Calendar cal = Calendar.getInstance();

      Date date = cal.getTime();

      transaction.setTransactionDate(DATE\_FORMAT.format(date));

      transaction.setTransactionTime(TIME\_FORMAT.format(date));

      transaction.setAmount(amount);

      transaction.setBalance(balance);

      transaction.setKind(WITHDRAW);

      transactions[totalTransaction++] = transaction;

    }

Open account

As denied above when a bank officer open an account to a customer the customer may open with initial value or not if he/she wants to open without initial balance addAccount() take only account number and name only as follows.

Headings

  static final String ACCOUNTS\_HEADING = "Account No|Owner|Balance";

  static final String TRANSACTIONS\_HEADING = "Date|Time|D/W|Amount|Balance";

bank object (a bank)

 Bank bank = new Bank();

    bank.addAccount("110000", "Abebe");

OR

If he/she opens with initial balance addAcount() takes three arguments account number, name and initial balance.

    bank.addAccount("78978897","Aster",4000);

see all accounts in a bank implementation

   Account[] accounts = bank.getAccounts();

    int totalAccount = bank.getTotalAccount();

    System.out.println(ACCOUNTS\_HEADING);

    for (int i = 0; i < totalAccount; i++) {

      System.out.println(accounts[i]);

    }

Deposit

 Account Abebe = bank.getAccount("110000");

    Abebe.deposit(1000);

    System.out.println(ACCOUNTS\_HEADING);

    System.out.println(Abebe);

Withdraw

Abebe.withdraw(5);

    System.out.println(ACCOUNTS\_HEADING);

    System.out.println(Abebe);

See transaction log of an account

  Transaction[] transactions = Abebe.getTransactions();

    int totalTransaction = Abebe.getTotalTransaction();

    System.out.println(TRANSACTIONS\_HEADING);

    for (int i = 0; i < totalTransaction; i++) {

      System.out.println(transactions[i]);

    }

Find by customers’ name

  Account[] matched = bank.findAccountByName("Abebe");

    System.out.println(ACCOUNTS\_HEADING);

    for (Account account : matched) {

      System.out.println(account);

    }

  }

Output

1. Print out the total account list.

Account No|Owner|Balance

110000|Abebe|0.0

78978897|Aster|4000.0

2. Deposit $ 1000 into 110000 account.

Account No|Owner|Balance

110000|Abebe|1000.0

3. Withdraw $ 500 from 110000 account.

Account No|Owner|Balance

110000|Abebe|500.0

4. Print transaction history of the 101 account.

Date|Time|D/W|Amount|Balance

2021/05/10|18:33:39|Deposit|1000.0|1000.0

2021/05/10|18:33:39|Withdrawal|500.0|500.0

5. Find Abebe's Account.

Account No|Owner|Balance

110000|Abebe|500.0

Question

Interprets the output from number 1 to 5 what it does mean and how those happened.

*Many thanks to author of http://www.java-school.net/*