BANK MANAGEMENT SYSTEM

GROUP NAME	ID
BONI YEHYA	15,369/21
DANIEL JEMO	15,344/21
DANIEL SAHLE	15,551/21
EDEN ERGE	15,363/21
ELSHADAY FEKADE	15,565/21
SAMUEL GETACHEW	15,345/21

CONTENT

Abstract

This project is mainly developed for the account division of a banking

Sector to provide better interface of the entire banking transaction. This system

Is aimed to give a better out look to the user interface and to implement all the banking transaction like:

New account creation: whenever a new customer comes this system facilitate

To create an account. The customer must provide information regarding the type of

Account he wants to open amount of deposit ,address ,phone number

Deposit amount: customer can deposit money into the bank. The customer needs

To provide information like his social security number and the amount.

Withdraw from account: customer can his withdraw his money from account.

He needs to provide his social security number as well as the amount which he

Wants to withdraw.

View account information: customer can view his account information. He need

To provide his social security number to the employer in order to view his account detail

LOAN : customer can loan from the to finance specific types of expenditures.

Scope

This project is developed to make the work of bank employer much easier int the process of maintaining the records of the customer.

This helps the customer to provide easy deposit/withdrawal and makes the process

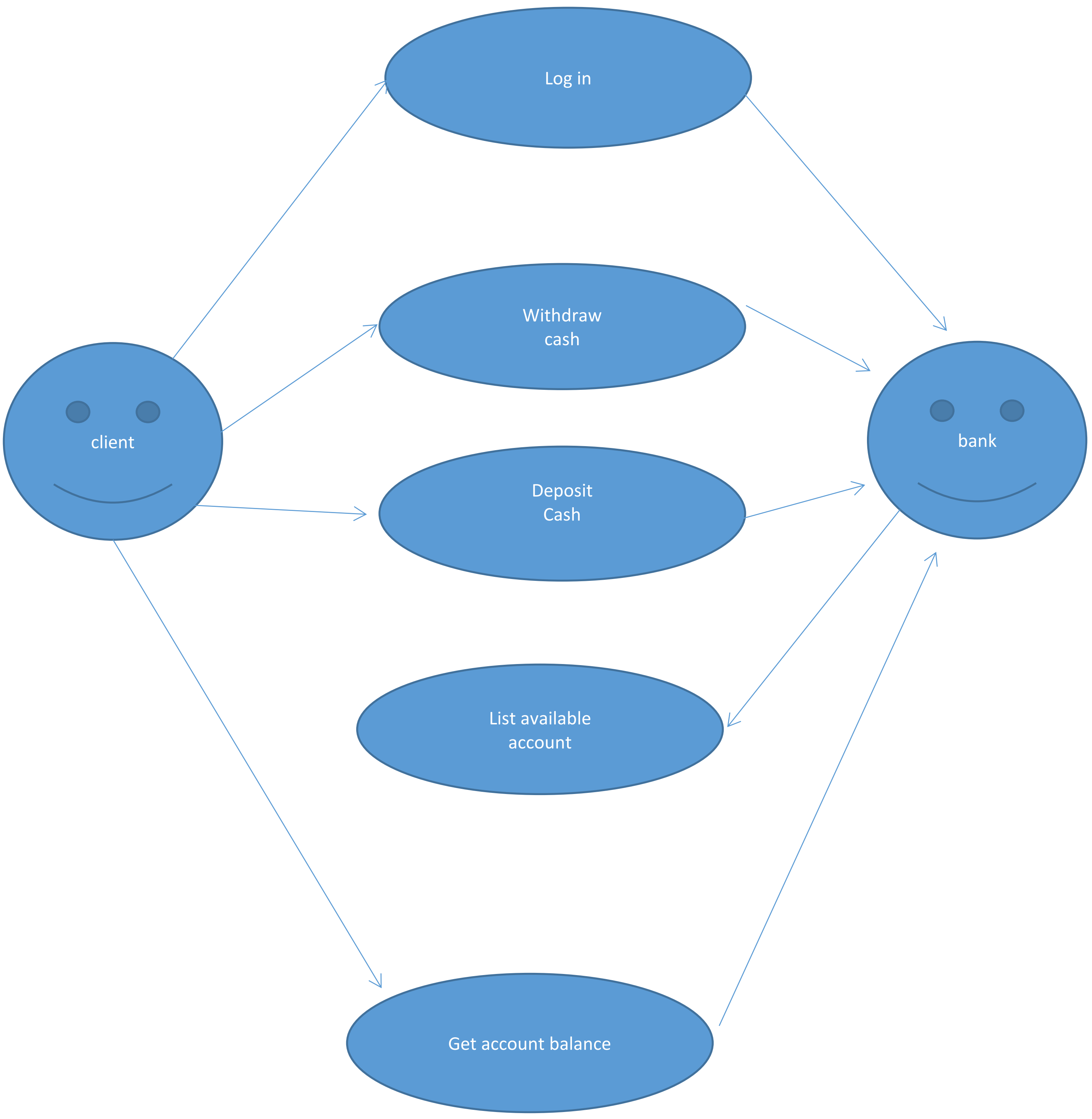
User friendly

In this way, it can provide best security ever. The employer can execute the process

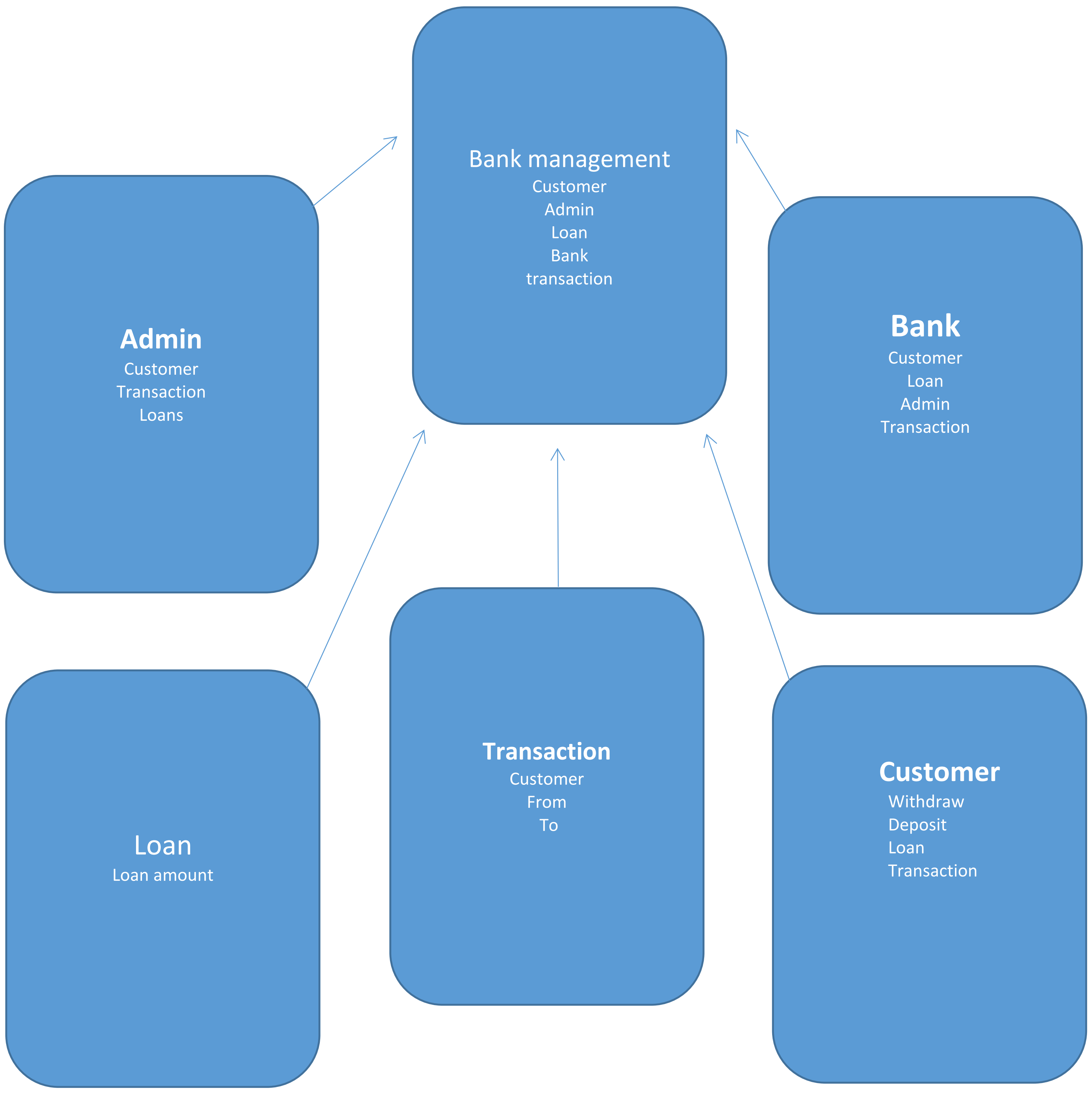
of deposit /withdraw of the amount with maximum security.

This system also provide accuracy.

This also reduces the time taken for the customer to complete his transaction.

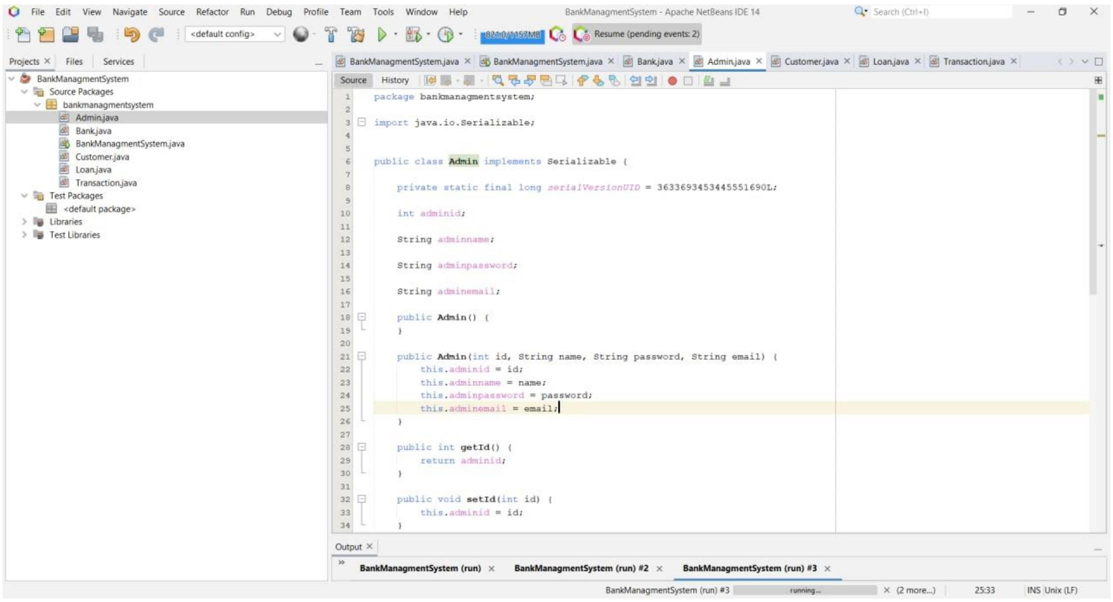


USER CASE DIAGRAM



CLASS DIAGRAM

admin



```java
package bankmanagmentsystem;

import java.io.Serializable;


public class Admin implements Serializable {

    private static final long serialVersionUID = 3633693453445551690L;

    int adminid;

    String adminname;

    String adminpassword;

    String adminemail;

    public Admin() {
    }

    public Admin(int id, String name, String password, String email) {
        this.adminid = id;
        this.adminname = name;
        this.adminpassword = password;
        this.adminemail = email;
    }

    public int getId() {
        return adminid;
    }

    public void setId(int id) {
        this.adminid = id;
    }
```

The code is a class that implements the Serializable interface.

The Admin class has three fields: id, name, and password.

It also has an email field which is used to store the administrator's email address.

The constructor of this class takes in three parameters: the ID number, a string for the name of the admin, and a string for their password.

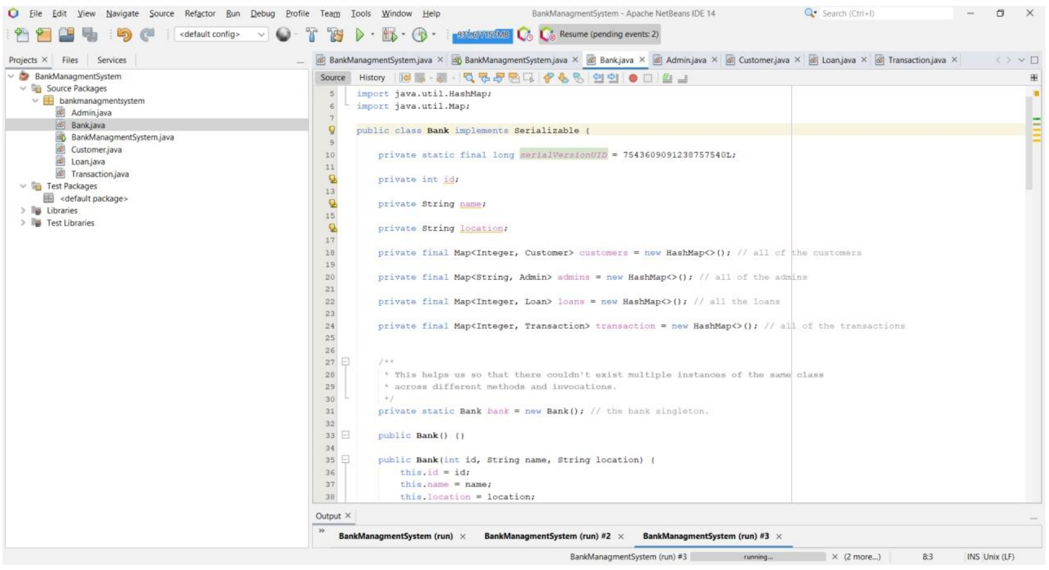
This is how they are stored in memory when instantiated by calling new Admin().

The code is a class that implements the Serializable interface.

The Admin class contains three variables, id, name and password.

The Admin class also has methods to set these variables as well as get them back.

Bank



```java
import java.util.HashMap;
import java.util.Map;

public class Bank implements Serializable {

    private static final long serialVersionUID = 7543609091238757540L;

    private int id;

    private String name;

    private String location;

    private final Map<Integer, Customer> customers = new HashMap<>(); // all of the customers

    private final Map<String, Admin> admins = new HashMap<>(); // all of the admins

    private final Map<Integer, Loan> loans = new HashMap<>(); // all the loans

    private final Map<Integer, Transaction> transaction = new HashMap<>(); // all of the transactions


    /**
     * This helps us so that there couldn't exist multiple instances of the same class
     * across different methods and invocations.
     */
    private static Bank bank = new Bank(); // the bank singleton.

    public Bank() {}

    public Bank(int id, String name, String location) {
        this.id = id;
        this.name = name;
        this.location = location;
```

The code is a class named Bank.

It implements the Serializable interface, which means that it can be saved to a file and loaded from another file.

The Bank class has three fields: id, name, and location.

The first line of code is the constructor for this class.

This is where you would put your initialization code if you were creating an instance of this object in Java programming language.

In this case, there are no initializations needed because we're just declaring what these variables will contain when they're created later on in the program's execution process (i.e., when someone creates an instance of our bank).

The next line declares that all instances of our bank must implement the Serializable interface so that they can be saved to files and loaded from other files without any problems with serialization/deserialization errors occurring during runtime (when someone tries to save or load them).

Next comes a private static final long serialVersionUID field declaration; it's used by Java programs as a unique identifier for each version of its classes so that two different versions don't accidentally overwrite one another while being edited by different programmers at different times in their development process (which could cause bugs or crashes).

Then comes some declarations

The code is meant to create a bank object that can be serialized and deserialized.

The Bank class implements the Serializable interface, which means it can be serialized and deserialized.

The code above contains a map of all the customers, admins, loans, and transactions.

The code above is an example of a class that has three maps: one for each type of data in the program.

The first map is called "customers", which stores all the customer information; the second map is called "admins", which stores all the admin information; and finally there's a third map called "loans" or "transactions".

This last one stores all loan-related information.

The code attempts to create a map of all the customers, admins, loans, and transactions.

The code creates an empty map called "customers" and then creates a new HashMap for each of the other maps.

Each HashMap is initialized with null values in order to avoid any errors during initialization.

private final Map customers = new HashMap<>(); // all of the customers private final Map admins = new HashMap<>(); // all of the admins private final Map loans = new HashMap<>(); // all the loans private final Map transaction = new HashMap<

The code is trying to create a Bank object.

The code creates the singleton instance of the class, and then it sets up some properties for that instance.

It also has a method called getbank() which returns the bank object.

The Bank class is declared as private static final, meaning it can only be instantiated once in any given program run (in this case, by calling new Bank()).

This helps us so that there couldn't exist multiple instances of the same class across different methods and invocations.

The getbank() method returns an instance of our singleton bank object with id=0 and name="Bank" and location="New York".

The code is a Java 8 code.

The code creates a Bank object and initializes it with the default values of id = 0, name = "Bank", location = null.

The getbank() method returns the same instance of the Bank class that was created in the first line of code.

The code starts by declaring a Bank object.

It then sets the previous Bank object to null and assigns the Bank object the passed value.

The code starts by declaring an Admin object with an email address as its key.

The code then adds that admin to the admins HashMap using addadmin().

The code starts by declaring a Customer object with an ID as its key.

The code then adds that customer to the customers HashMap using addcustomer().

The code starts by declaring a Loan object with an ID as its key.

The code then adds that loan to the loans HashMap using addloan().

The final line of this block declares a Transaction object with an ID as its key, which is added to transactions hashmap using addtransaction()

The code is meant to add a new object to the HashMap.

The code starts by adding the passed in object to the admins HashMap.

This is done by calling the addadmin() method and passing in an Admin object.

The same thing is done for Customer, Loan, and Transaction objects.

The code starts by declaring a variable called customers.

This is the HashMap that will store all of the customer objects in this program.

Next, it declares a variable called loans and sets its value to be an empty collection.

Then, it creates two methods: getCustomers() and getloan().

The first method returns all of the customer objects stored in the customers HashMap while the second method returns all of the loan objects stored in loans.

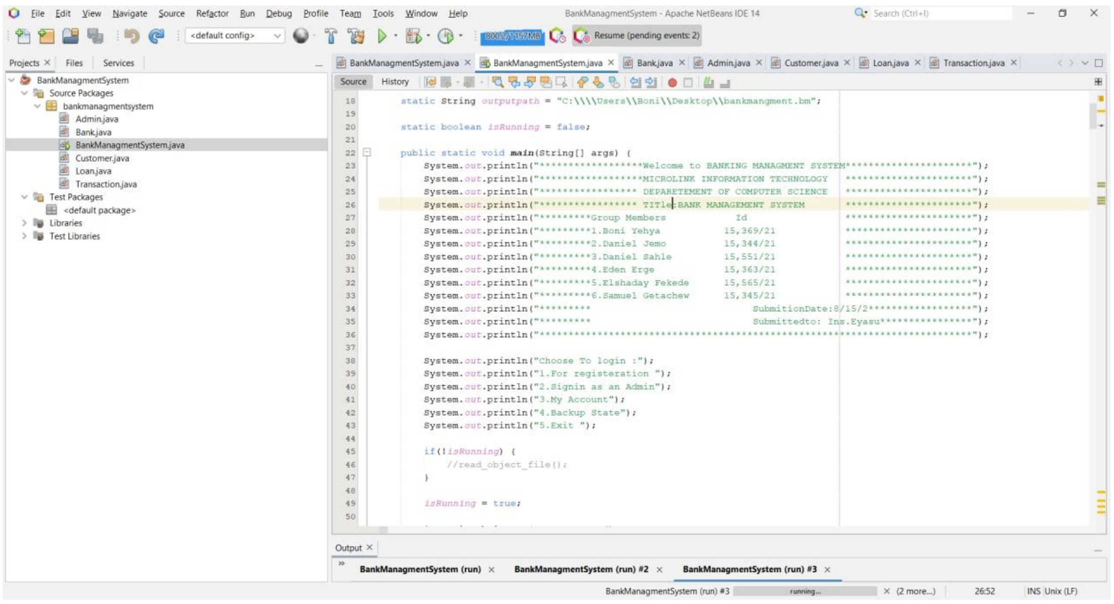
The update() method takes one parameter which is an object representing a new Customer object to be added to customers' HashMap with ID given as input parameter id.

It then updates that entry by adding it into customers' Hashmap with c as its key and id as its value.

The code is used to update a customer in the customers HashMap.

The code uses the new keyword to create a new object and assign it to the variable c. The code then uses the put method of the HashMap to add an entry for this customer with their ID.

Bank management system



```java
    static String outputpath = "C:\\\\Users\\Boni\\Desktop\\bankmangment.bm";

    static boolean isRunning = false;

    public static void main(String[] args) {
        System.out.println("*****************Welcome to BANKING MANAGMENT SYSTEM*********************");
        System.out.println("*****************MICROLINK INFORMATION TECHNOLOGY   *********************");
        System.out.println("***************** DEPARETEMENT OF COMPUTER SCIENCE  *********************");
        System.out.println("***************** TITle:BANK MANAGEMENT SYSTEM      *********************");
        System.out.println("*********Group Members            Id                *********************");
        System.out.println("*********1.Boni Yehya           15,369/21           *********************");
        System.out.println("*********2.Daniel Jemo          15,344/21           *********************");
        System.out.println("*********3.Daniel Sahle         15,551/21           *********************");
        System.out.println("*********4.Eden Erge            15,363/21           *********************");
        System.out.println("*********5.Elshaday Fekede      15,565/21           *********************");
        System.out.println("*********6.Samuel Getachew      15,345/21           *********************");
        System.out.println("*********                         SubmitionDate:8/15/2*****************");
        System.out.println("*********                         Submittedto: Ins.Eyasu***************");
        System.out.println("************************************************************************");

        System.out.println("Choose To login :");
        System.out.println("1.For registeration ");
        System.out.println("2.Signin as an Admin");
        System.out.println("3.My Account");
        System.out.println("4.Backup State");
        System.out.println("5.Exit ");

        if(!isRunning) {
            //read_object_file();
        }

        isRunning = true;
```

The code starts by declaring a class called "bankmanagmentsystem".
The next line imports the java.io package, which contains classes for input and output streams, as well as exceptions.
The next line declares an object of type FileInputStream that will be used to read data from a file on disk.
The next line declares an object of type FileOutputStream that will be used to write data to a file on disk.
Next, we declare two objects of type ObjectInputStream and ObjectOutputStream which are used for reading and writing Java objects in binary format (i.e., byte arrays).
These objects are required because they allow us to use methods like readObject() or writeObject(), which can only operate on byte arrays rather than String values or primitive types like ints or floats.
Finally, we declare two variables: one is an instance of Scanner that reads input from the user; the other is an instance of Random whose constructor takes in a seed value so it can generate random numbers with high accuracy over time without repeating itself too often (this is important when generating unique IDs).
The code is a Java program that creates a bank management system.
The code starts by creating an object called "bankmanagmentsystem".
This object has two methods, one for input and one for output.
The input method reads in data from the keyboard and stores it in an array of objects called "inputData".
The output method writes out data to the screen.
The code is a Java program that is used to generate random IDs.
The code starts by creating an instance of the Random class, which is then used to create a new random number generator.
The outputpath variable stores the path where the generated bank management system will be saved on your computer's desktop.
The main method has two purposes: first, it prints out some text and second, it begins executing the program.
When you run this program for the first time, nothing happens because there are no statements in main that execute anything yet.
After running through all of these statements and reaching line 15 (which executes System.out), you see Welcome to BANKING MANAGEMENT SYSTEM printed out on your screen followed by a bunch of asterisks (*).
This means that everything went well and now you can start using this system!
The code is used to generate random IDs.
The code is used to generate random IDs.
The code starts with a comment that says "MICROLINK INFORMATION TECHNOLOGY".
This is followed by the text "DEPARETEMENT OF COMPUTER SCIENCE" and then "TITle:BANK MANAGEMENT SYSTEM".
The code starts with a comment that says MICROLINK INFORMATION TECHNOLOGY.
The next line has the text DEPARETEMENT OF COMPUTER SCIENCE, which is followed by TITLE:BANK MANAGEMENT SYSTEM.
The code will print the following: MICROLINK INFORMATION TECHNOLOGY DEPARETEMENT OF COMPUTER SCIENCE TITle:BANK MANAGEMENT SYSTEM
The code starts by printing the text Submitted to: Ins.Eyasu" on a new line.
This is followed by the text The next line prints "Choose To login :", and then asks for input from the user, which will be either 1 or 2.
If it's 1, then it will print "For registeration", and if it's 2, then it will print "Signin as an Admin."

The next two lines are used to display information about what has been submitted to this program.
It first displays that they have submitted something called a form with some fields in them (name of company, address).
Then after that is printed out, there is another line that says "My Account."
The code is used to create a login form.
The code is used to create a login form with the following options: 1.For registeration 2.Signin as an Admin 3.My Account 4.Backup State 5. Exit

The code starts by asking the user to enter their name.
The input variable is then used to store the string entered by the user.
Next, it asks for an email address and a password.
The code starts by asking the user to enter their name.
The input variable is then used to store the string entered by the user.
Next, it asks for an email address and a password.
The code will ask the user to enter their name, email and password.
The code above will then check if the input is valid by checking if it has a length of at least six characters.
If it does not have a length of at least six characters, then the program will print an error message that says "Please enter your name."
The code starts by creating a new Customer object.
The constructor takes in the nextInt() function, which returns an int value between 0 and 9.
It also takes in the name, email, password, and amount of money that is currently on deposit with this customer.
The code then goes to Bank.getbank().addcustomer(cu); This line adds the newly created customer to the bank's list of customers.
Next it asks for input from the user using System.out.println("Enter:"); It prints out "1."
if they want to go back to main menu or "2."
if they want to exit the system; otherwise it will print out an integer value corresponding with whatever key was pressed by the user (in this case b).
If b equals 1 then main() is called; otherwise if b equals 2 then exit() is called
The code is executed when the user enters a valid email and password.
If the customer exists, then it prints "Welcome " + currentCustomer.getName() and to the console.

The code starts by asking the user to enter their email and password.
If they enter an email that is epicdream@gmail.com and a password of 0000, then it prints "Welcome Admin Enter:".
The code starts by declaring two variables called input and output.
The input variable will be used for the user's input, while the output variable will print what the program does next.
Next, there are three if statements in this code: one at line 5, one at line 7, and one at line 9.
These if statements check whether or not the entered email is epicdream@gmail.com with a password of 0000; if so then it prints "Welcome Admin Enter."
Otherwise it prints "Invalid Email/Password" on screen instead of printing anything else out on screen because those lines would have been skipped over due to these conditions being met (the condition was false).
The code attempts to sign in the user "admin" with their email address and password.
The code will only work if the user enters the correct email address and password.

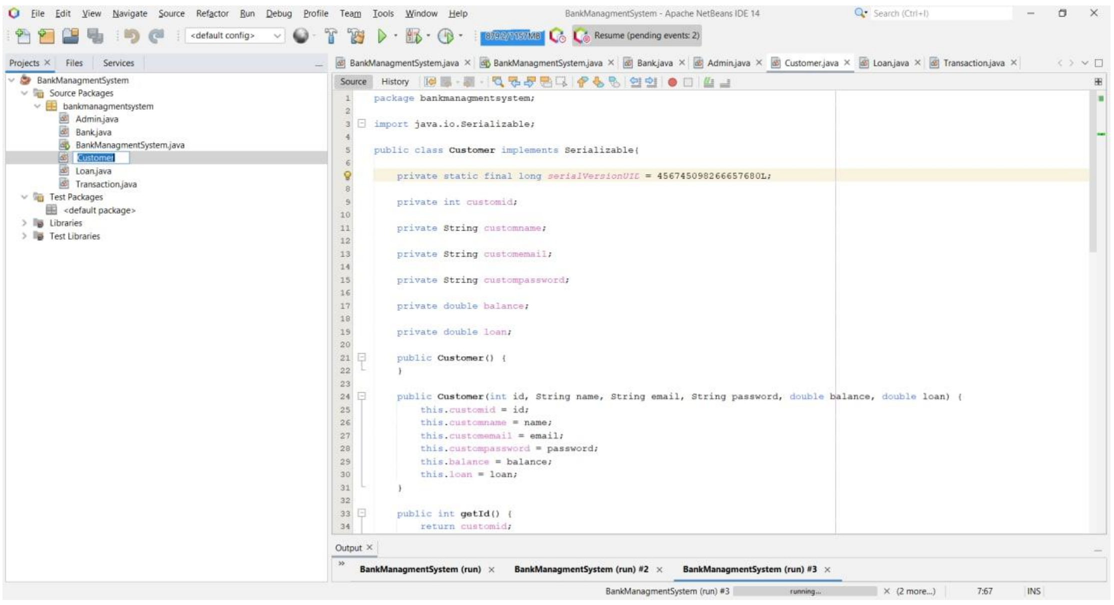
If they enter an incorrect email or password, then it prints out a message saying that they are not logged in.
The code starts by asking the user to input a number.
If they enter anything other than 1, 2, 3, 4 or 5 then it will print "Unknown Input" and continue on with the rest of the code.
If they enter 1 then it will call view_customers().
It also prints out "To View Customers".
If they enter 2 then it will call view_transaction().
It also prints out "To View Transaction".
If they enter 3 then it will call view_loans().
It also prints out "To View Loans".
The code is a switch statement that has four cases.
The first case is for when the user enters 1, which will then call the view_customers() function.
The second case is for when the user enters 2, which will then call the view_transaction() function.
The third case is for when the user enters 3, which will then call the view_loans() function.
And finally, if they enter 4 or 5, it will go back to main().
The code starts by creating a new class called MainMenu.
This is the main menu for the program.
It has two methods: one to go back to the main menu and another to sign in as an administrator.
The next line creates a new instance of this class, which will be used later on when we want to enter our username or password into it.
The input variable is then set up with three possible options: "1" (to go back), "2" (to sign in as an administrator), and "3" (exit).
Next, the switch statement checks if b equals 1 or 2; if that's not true, then it checks if b equals 3; otherwise, it prints out Not implemented!
The code will print the following message: Password or username is not correct!
Enter: 1.To go back to main menu 2.Retry 3.Exit the system
The code starts by printing out a message that says Customers
It then prints out the ID, name, email, and balance for each customer.
Then it prints out an input prompt asking the user to enter one of four options: Delete Customer, Edit Customer, Back to Main menu or Exit System.
The code starts by checking if there is an input from the user.
If there is no input from the user then it will print out "Unknown Input" and end its execution.
The next line checks what option was selected by the user with a switch statement which has been explained in detail above.
The code is used to print out the ID, name, email and balance of all customers in the bank.
The code in the above example is a loop that prints out all of the loans in the bank.
The code starts by printing "****************** Loans ***********************" and then it prints an ID, By, Userid, Amount, Interest rate for each loan.
The first line of code creates a variable called l which will hold all of the loans in Bank.getbank().getloan() .
Then on Line 2 we create another variable called SystemOut which will print to standard output (the screen).
On Line 3 we start our loop with a for statement that iterates through every loan in l .
We use getBank() to access the bank object and getLoan() to access each individual loan inside of it.

Finally on Line 4 we print out what was returned from getLoan() , so you can see how many loans were found inside of Bank.getbank().getloan().
The code will print out the following: Id\t\tBy\t\tUserId\t\tAmount\t\tInterestRate 1.
2.
3.4.5.6.7 1.
Bank Account \ tBank Account \ tUserid 1 \ t$100,000 \ t0%
The code asks the user to enter an amount, and then calculates the new balance.
If the input is less than 0, it prints "Invalid Amount" and calls customer_deposit again with c as a parameter.
The code asks for input from the user in order to calculate how much they want to deposit into their account.
It then calculates what their new balance will be after adding that amount of money.
The code will prompt the user to input a deposit amount.
If the user inputs an invalid amount, then it will print out an error message and terminate.
Otherwise, it will calculate the new balance of the customer account and update that balance in their account.
The code above attempts to calculate how much money is left in a customer's account after they have made a deposit.
The code starts with a main() function that is called when the user presses 1.
The first thing this function does is to create an instance of the Customer class and assign it to c. The next line in main() creates a new System object, which will be used for input purposes later on.
Next, the code asks for input from the user by printing "Please Enter: " followed by two blanks.
This prompts the user to enter their desired amount of money they want to withdraw from their account.
After getting input from the user, it prints out "Invalid Amount" if they entered an invalid number or if they entered more than what was available in their account balance (in case there are no funds left).
It then goes back up one level and calls customer_withdraw().
If you press 2 instead of 1, it exits out of this program altogether!
If you enter anything other than 0 or -1 as your answer (i.e., any number), then customer_withdraw() gets called again with whatever value you entered as its argument (c).
After doing so, customer_withdraw() subtracts that amount from c's current balance and sets c's new balance equal to that minus wb
The code asks the user to enter a number and then deducts that amount from the customer's balance.
The code also prints out a message saying "Deposited Successfully!
your new balance is " + new_balance" after the amount has been deducted.
The code starts by declaring a variable called crieteria.
This is the amount of money that will be borrowed from the bank.
The code then calculates how much money will be loaned to the customer based on their current balance and how many loans they have already taken out, which is stored in l. The if statement checks whether or not there is enough money in the account to pay back this loan plus interest, which would require an amount equal to crieteria multiplied by 0.3 (the rate of interest).
If so, then a new Loan object with these values is created and added to Bank's list of loans using getbank().
The borrower's account number (c) is set as well as their name and ID number for reference later on when it comes time to repay this loan.

Finally, ln gets its own value assigned before being added into the borrower's account balance with += operator.

The code attempts to check if the balance of the account is greater than the loan amount and if it is, then a new Loan object will be created with that amount.
If it isn't, then nothing will happen.

CUSTOMER

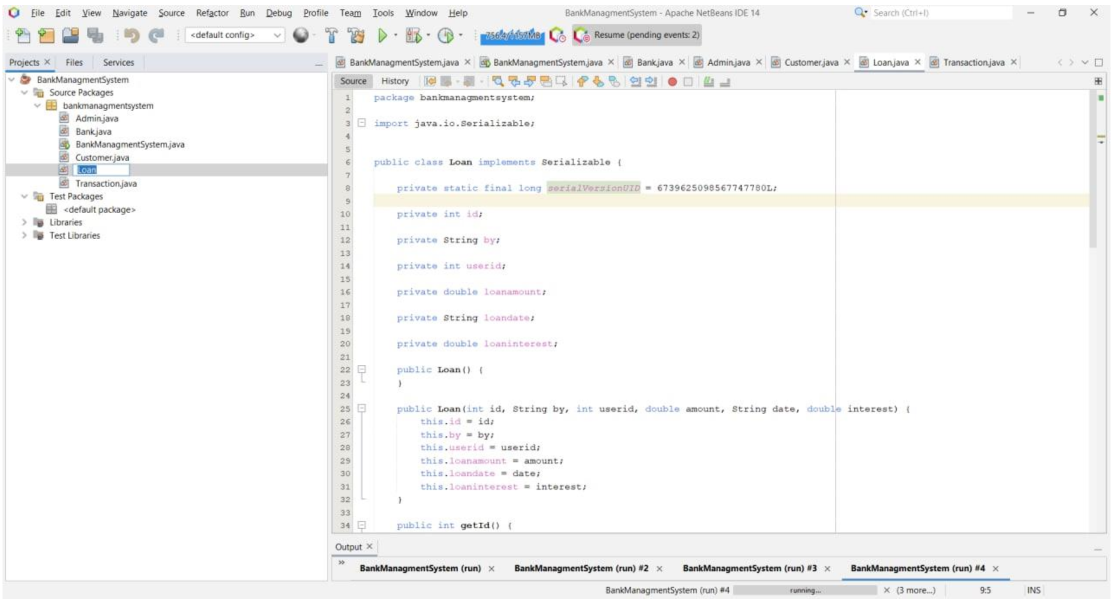


```java
package bankmanagmentsystem;

import java.io.Serializable;

public class Customer implements Serializable{

    private static final long serialVersionUID = 456745098266657680L;

    private int customid;

    private String customname;

    private String customemail;

    private String custompassword;

    private double balance;

    private double loan;

    public Customer() {
    }

    public Customer(int id, String name, String email, String password, double balance, double loan) {
        this.customid = id;
        this.customname = name;
        this.customemail = email;
        this.custompassword = password;
        this.balance = balance;
        this.loan = loan;
    }

    public int getId() {
        return customid;
```

The code starts by asking the user to enter an id.
The input variable is a text field that will be used for this purpose.
Next, it prints "Enter the id of the customer: " and then asks for an integer from the user.
It then stores this value in a variable called id.
The next line of code gets a reference to Bank's Customer class using getbank().getCustomer(id).
If there is no error, it calls remove() on Customers with c as its argument and prints "Delete Successfull!"
followed by Enter:
The code is used to delete a customer from the bank database.
The code will prompt for the id of the customer, and then find that customer in the database.
If found, it will remove that customer from all other customers and print out "Delete Successful!"
The code is a method that asks the user to enter an id and then gets the customer with that id.
If there is no customer with that id, it prints "Unknown Input".
The code starts by asking for input from the user.
The nextInt() function returns an int value which will be used in the switch statement later on.
Next, it checks if there is a Customer object stored in Bank class using getbank().getCustomer(id).
If there is one, it prints out "Editing" followed by their name and then enters new values into fields of this object.
It also prints out "Enter New Name: ".

After entering new values into fields of this object, it asks for input again to enter email and password.
The code is meant to edit a customer.
The code first asks the user for the id of the customer they want to edit.
After that, it will ask them for their name and email address.
Finally, it will ask them for their password.
The code is meant to exit the system by returning back to main menu.
It does this by asking the user if they would like to go back or exit from this menu with an input box asking them what they would like to do next in a loop until they press "Exit" button on screen.
The code starts by creating a new Customer object.
The code then asks the user to enter their ID, name, email, password and balance.
The code then uses this information to update the bank account of the customer with that ID.
The send_money function is called when you want to transfer money from one account to another.
It takes in a Customer object as an argument and checks if there is already a customer with that ID in the Bank class's list of customers (Bank.getbank().getCustomer(id)).
If so, it sends them again using send_money(c).
Otherwise it creates a new customer record for them and updates their bank account with that id using Bank.getbank().update(newC, id).
The code is a simple Java program that sends money from one bank account to another.
The first thing the code does is create a new Customer object with the given id, name, email, password and balance.
Next, it uses the Bank class to update this customer's balance by calling its update method with the given id of the receiver.

LONE

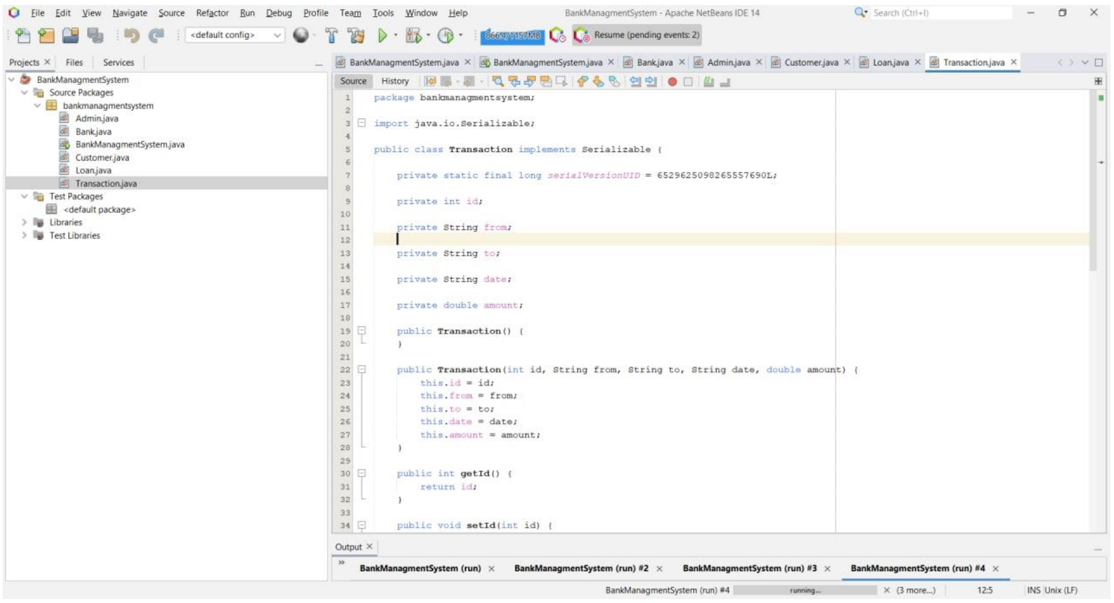


```java
package bankmanagmentsystem;

import java.io.Serializable;


public class Loan implements Serializable {

    private static final long serialVersionUID = 6739625098567747780L;

    private int id;

    private String by;

    private int userid;

    private double loanamount;

    private String loandate;

    private double loaninterest;

    public Loan() {
    }

    public Loan(int id, String by, int userid, double amount, String date, double interest) {
        this.id = id;
        this.by = by;
        this.userid = userid;
        this.loanamount = amount;
        this.loandate = date;
        this.loaninterest = interest;
    }

    public int getId() {
```

The code is a class that implements the Serializable interface.
The Loan class has two private fields: id and by.
It also has three public methods, getId(), setId(), and getBy().
The first method returns the value of the id field, which is an int type variable.
The second method sets the value of the id field to whatever argument passed in as a parameter (in this case, it would be set to 6739625098677477780L).
Finally, getBy() returns what is stored in by field.

The code starts with some boilerplate code for creating objects of type Loan using new operator on line 1-2.
Then there are four variables declared on lines 3-5: loanamount (double), loandate (String), loaninterest (double), and userid (int).
These variables are initialized on lines 6-7 with values from 0 to 1000000 respectively; these values represent how much money you want to borrow or lend out at any given time.
On line 8 we have our constructor function that takes no arguments but creates an object called "Loan" with default values for all its fields except for those specified above when they were initialized via their respective constructors.
The code is an example of a class that implements the Serializable interface.
The Loan class contains private fields for id, by, userid, loanamount, loandate, and loaninterest.
The Loan constructor takes in the id of the object being created as well as the by and userid of the object being created.
The Loan constructor also takes in a double value for amount and date as well as a double value for interest.

TANSACTIOMN


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
BankManagmentSystem - Apache NetBeans IDE 14
Resume (pending events: 2)
Projects Files Services
BankManagmentSystem
Source Packages
bankmanagmentsystem
Admin.java
Bank.java
BankManagmentSystem.java
Customer.java
Loan.java
Transaction.java
Test Packages
<default package>
Libraries
Test Libraries
BankManagmentSystem.java BankManagmentSystem.java Bank.java Admin.java Customer.java Loan.java Transaction.java
Source History
Output
BankManagmentSystem (run) BankManagmentSystem (run) #2 BankManagmentSystem (run) #3 BankManagmentSystem (run) #4
running...

```java
package bankmanagmentsystem;

import java.io.Serializable;

public class Transaction implements Serializable {

    private static final long serialVersionUID = 6529625098265557690L;

    private int id;

    private String from;

    private String to;

    private String date;

    private double amount;

    public Transaction() {
    }

    public Transaction(int id, String from, String to, String date, double amount) {
        this.id = id;
        this.from = from;
        this.to = to;
        this.date = date;
        this.amount = amount;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
```

The code is a class that implements the Serializable interface.
This means that it can be saved to a file and read back in later.
The Transaction class has three private variables: id, from, and to.
These are used for storing information about the transaction such as what date it was created on or who it is going to.
It also has an amount variable which stores how much money is being sent or received in this transaction.
The constructor of the Transaction class takes four arguments: id, from, to, and date.
Each argument corresponds with one of the private variables mentioned above so they can be set when creating a new instance of this object using these values as parameters
The code is used to create a class called Transaction.
The class has three fields: id, from, and to.
The Transaction class also implements the Serializable interface which means that it can be saved in a file and read back in at any time.

CONCLUSION

This project is to nurture the needs of the user In banking sector by embedding all the task of transaction taking place in bank.

Thus the bank management system is developed and executed successfully.