Day 5
Transaction Management

In the previous session, the topic of discussion was transaction management and transaction management is a very important and crucial concept in JDBC. And one of the most commonly asked interview questions, It was also hinted in the previous class that transaction management is something which is used to achieve a certain property of a database, which we call the ACID properties, where

A – Atomicity

C – Consistency

I – Isolation

D – Durability

What atomicity and consistency mean is the most important discussion in today's session, because these are the two things that you can achieve through JDBC and transaction management. And isolation and durability are something that you have to take care of when you design the database itself.

Atomicity means whenever there is a set of queries, which is bundled together and called a transaction, then all the queries must execute or none must execute, i.e., it should be either all or nothing, **All or nothing property of a transaction is called atomicity.** Now, after a transaction, the database must move from one consistent state to another consistent state. In other words, it must be a consistent database

How do you achieve it in Java?

Well, very simple. The JDBC API provides you three methods to achieve it in the form of:

- **setAutoCommit()**
- **commit()**
- **rollback()**

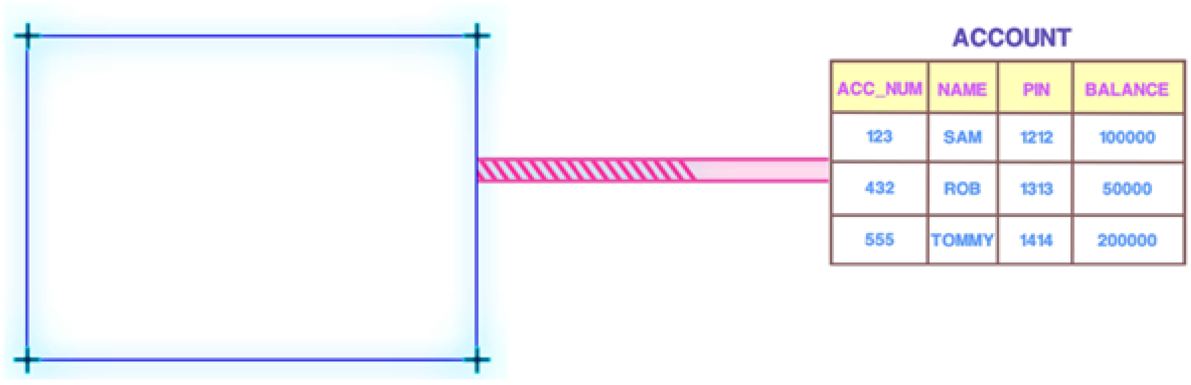
Now to understand these three methods in action, Let us take a real-time example.

Usually, whenever you hear the word transaction, you always think about sending money or receiving money. That is what we technically call a **transaction**. So let's design a simple transaction or a money transaction application, where a person can send some money to someone.

How do we do it?

Let me show you guys. Consider a console-based application and console-based applications are also known as standalone applications. Because these applications can only be accessed from your computer and it should be installed on your computer. We have a database called **tap_bank and in that, we have a table called account.**

In this account table, there are four columns: **account number, name, pin, and balance** and we have entered some dummy data. Now we will connect the application and table using JDBC. You know how this connection can be made possible.



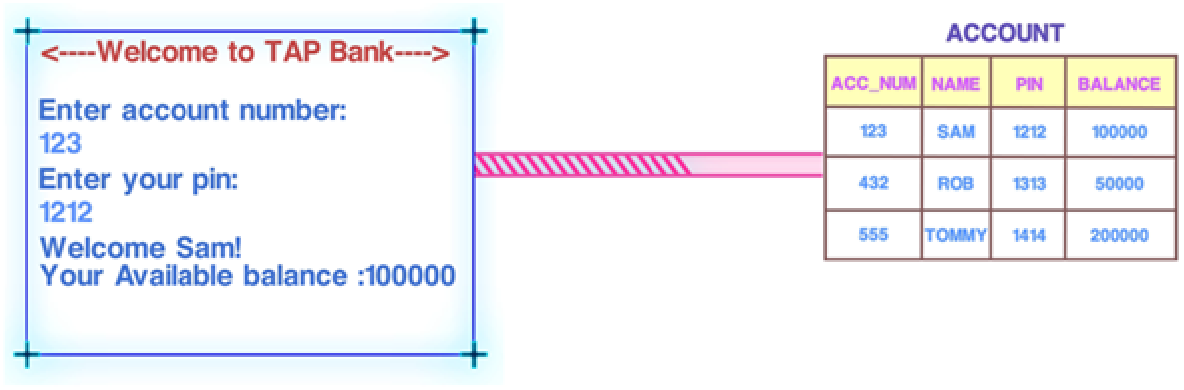
ACCOUNT

ACC_NUM	NAME	PIN	BALANCE
123	SAM	1212	100000
432	ROB	1313	50000
555	TOMMY	1414	200000

When you execute this application, a message, **"<---Welcome to Tap Bank--->",** should be displayed.

Next, it should display a message to **Enter account number**. Now a person has to enter a valid account number. In other words, it should be one of the entries present in the table called an **account.**

Let us enter the account number as **123**, which is the account number of a person called **Sam**. Next, it is going to ask for the pin. The pin, as you can see is **1212** and if in case we press enter, then we should get the following output on the window:



```
<----Welcome to TAP Bank---->

Enter account number:
123
Enter your pin:
1212
Welcome Sam!
Your Available balance :100000
```

ACCOUNT

ACC_NUM	NAME	PIN	BALANCE
123	SAM	1212	100000
432	ROB	1313	50000
555	TOMMY	1414	200000

Now to do this much one query is required, and the query will be a **SELECT** query and this is how the query looks like –

select * from account where acc_num = ? and pin = ?

and because the user gives the account number and pin, this has to be an **incomplete query**,

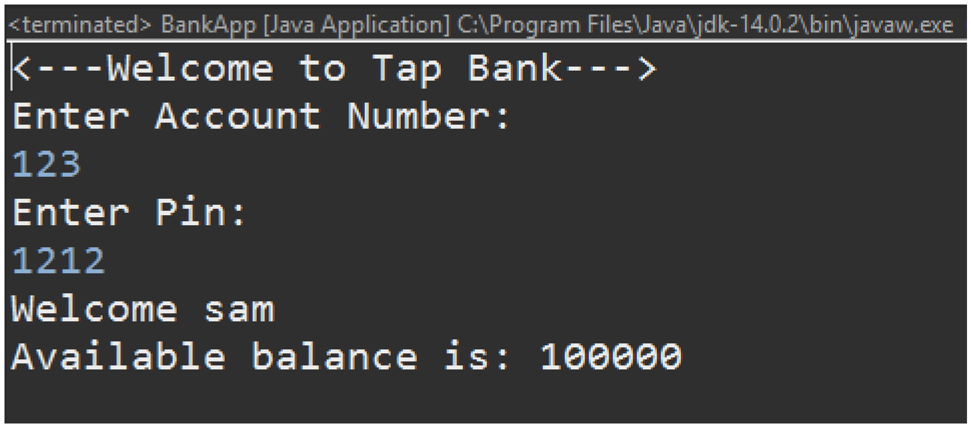
So this is the person logging in to send the money.

So let us write the code for the Login module:

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Scanner;

public class BankApp {
     public static void main(String[] args)  {
           String url = "jdbc:mysql://localhost:3306/tap_bank";
           String un = "root";
           String pwd = "root";
           Connection con = null;

           try {
                Class.forName("com.mysql.cj.jdbc.Driver");
                con=DriverManager.getConnection(url, un, pwd);
                Scanner scan =new Scanner(System.in);

//              Login Module
                System.out.println("<---Welcome to Tap Bank--->");
                System.out.println("Enter Account Number:");
                int acc_num = scan.nextInt();
                System.out.println("Enter Pin:");
                int pin = scan.nextInt();
                PreparedStatement pstmt1 = 
con.prepareStatement("select * from account where "
                                      + "acc_num = ? and pin = ? ");

                pstmt1.setInt(1, acc_num);
                pstmt1.setInt(2, pin);
                ResultSet res1 =pstmt1.executeQuery();

                res1.next();
                String name = res1.getString(2);
                int bal = res1.getInt(4);
                System.out.println("Welcome "+name);
                System.out.println("Available balance is: "+bal);

catch (Exception  e) {
                e.printStackTrace();
```

```
            }
        }
}
```

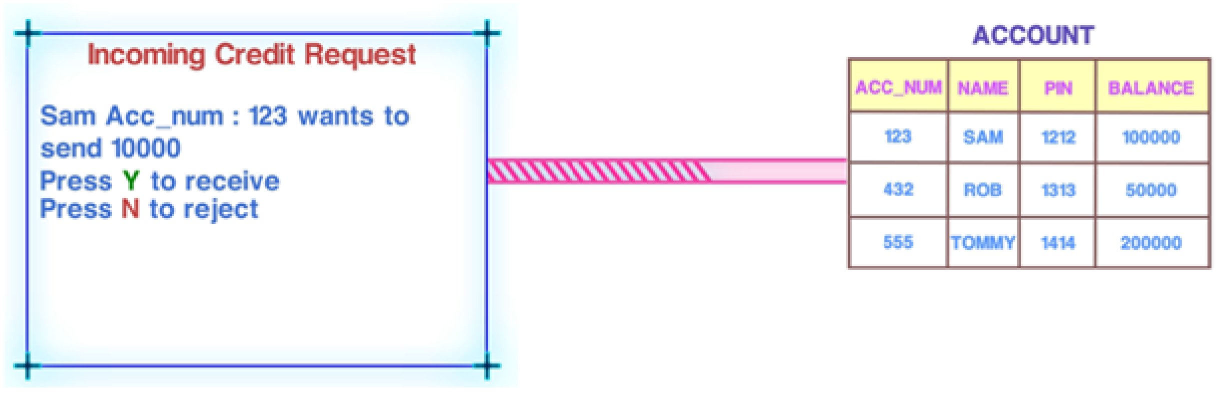
Output:



```
<terminated> BankApp [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe
<---Welcome to Tap Bank--->
Enter Account Number:
123
Enter Pin:
1212
Welcome sam
Available balance is: 100000
```

The next module is to **transfer the amount to the beneficiary account number.**

Let us assume we want to transfer the amount to **Rob** and his **account number is 432** and the amount to be transferred is **10000**. Now, if in case, we press enter, what must happen is money should not automatically get transferred, but one request will be sent to rob telling that hey, Rob, there is an incoming credit request, and the following message should be displayed.



ACC_NUM	NAME	PIN	BALANCE
123	SAM	1212	100000
432	ROB	1313	50000
555	TOMMY	1414	200000

If Rob presses **Y**, then 10,000 rupees should get deducted from Sam's account, and 10,000 rupees should be credited to Rob's, which means we have to write 2 update queries-

1. To deduct 10000 from Sam's account

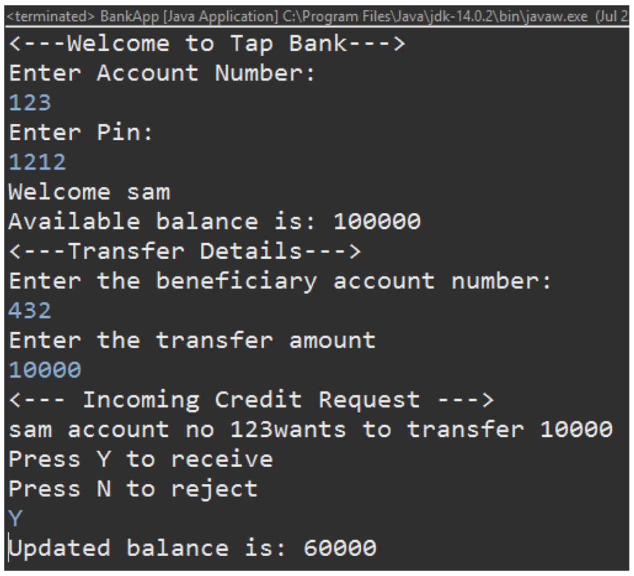
2. To credit 10000 to Rob’s account

And the queries should be incomplete query as the account number and amount should be entered by the user.

So let us write the code for the transfer module:

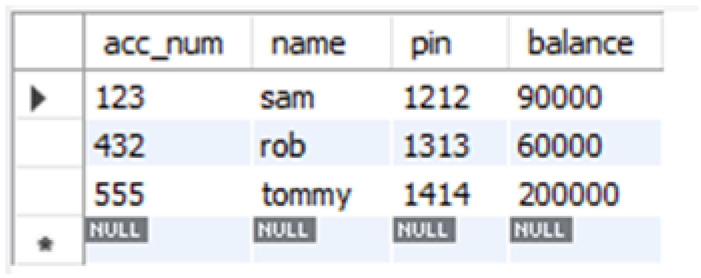
```
//               Transfer module

      System.out.println("<---Transfer Details--->");
      System.out.println("Enter the beneficiary account number:");
      int bacc_num = scan.nextInt();
      System.out.println("Enter the transfer amount");
      int t_amount = scan.nextInt();
      PreparedStatement pstmt2 = con.prepareStatement("update account
set balance = balance - ? where acc_num = ? ");

      pstmt2.setInt(1, t_amount);
      pstmt2.setInt(2, acc_num);
      pstmt2.executeUpdate();

      System.out.println("<--- Incoming Credit Request --->");
      System.out.println(name + " account no " +acc_num + "wants to
transfer "+ t_amount);
      System.out.println("Press Y to receive");
      System.out.println("Press N to reject");
      String choice = scan.next();

      if(choice.equals("Y")) {
            PreparedStatement pstmt3 =con.prepareStatement("update
account set balance = balance + ? where acc_num = ? ");
            pstmt3.setInt(1, t_amount);
            pstmt3.setInt(2, bacc_num);
            pstmt3.executeUpdate();

            PreparedStatement pstmt4 = con.prepareStatement("select *
from account where acc_num = ? ");
            pstmt4.setInt(1, bacc_num);
            ResultSet res2 =pstmt4.executeQuery();
            res2.next();
            System.out.println("Updated balance is: "
+res2.getInt(4));
            }
            else {
                  PreparedStatement pstmt5 =
con.prepareStatement("select * from account "
                                    + "where acc_num = ? ");
                  pstmt5.setInt(1, bacc_num);
                  ResultSet res2 =pstmt5.executeQuery();
                  res2.next();

                  System.out.println("Existing balance is: " +
res2.getInt(4));
                  }
```

Output:



```
<terminated> BankApp [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe (Jul 2
<---Welcome to Tap Bank--->
Enter Account Number:
123
Enter Pin:
1212
Welcome sam
Available balance is: 100000
<---Transfer Details--->
Enter the beneficiary account number:
432
Enter the transfer amount
10000
<--- Incoming Credit Request --->
sam account no 123wants to transfer 10000
Press Y to receive
Press N to reject
Y
Updated balance is: 60000
```

And the same would be updated in the database:



	acc_num	name	pin	balance
▶	123	sam	1212	90000
	432	rob	1313	60000
	555	tommy	1414	200000
*	NULL	NULL	NULL	NULL

But what if Rob had pressed **N,** i.e., he had rejected the incoming request?

Then Rob’s account should not be credited with 10000 and 10000 should not be deducted from Sam’s account. But the way we have written code, 10000 will be deducted from Sam’s account whether Rob accepts or rejects the credit request.

So how can we resolve this issue?

By setting the setAutoCommit() to false we can ensure that until the transaction is complete, the database should not be committed.

```
con.setAutoCommit(false);
```

And after the transaction is complete, it should be committed and for that we have to use-

```
con.commit()
```

But even after doing this, Sam’s account will be deducted because of the query we had written-

```
PreparedStatement pstmt2 =
con.prepareStatement("update account set "
                + "balance = balance - ? where
acc_num = ? ");
```

So we have to undo this query so that the amount will not be deducted, if Rob rejects the credit request.

How to do that?

First we have to create a save point and savepoint is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

```
Savepoint s = con.setSavepoint();
```

Then we can use the rollback() to roll back or go back to the savepoint, i.e., undo all the transactions to the savepoint.

```
con.rollback(s);
```