



Faculty of Engineering - Ain Shams University

Computer Engineering and Software Systems Program

CSE346: Advanced Database Systems

Project Proposal

Team members

Kareem Wael Hasan Ahmed	2001151
Ahmed Tarek Aboelmakarem Ahmed	20P6897
Ziad Ahmed Wahidallah Amer	20P5016
Karim Bassel Samir	20P6794
Fady Fady Fouad	20P7341

Github Repo: <https://github.com/KareemWael1/BankSys>

Table of Contents

Relevance.....	4
Abstract.....	4
Approach	5
1- ER Diagram	5
2- ObjectDB	6
3- Classes and JPA.....	8
4- Class Diagram	9
5- Database Population	11
6- GUI.....	12
Results and Analysis	14
1- Forms	14
2- Queries	20
3- Discussion.....	25
References:.....	25
Appendix.....	26

List of figures

Figure 1: ER Diagram.....	5
Figure 2: ObjectDB Schema GUI	6
Figure 3: ObjectDB Query GUI.....	7
Figure 4: Sample Entity to be a table in the database	8
Figure 5: Class Diagram	10
Figure 6: List of names to generate combinations	11
Figure 7: Generation of transactions for some account	11
Figure 8: Database Information.....	12
Figure 9: Main Menu	12
Figure 10: Sample Form	13
Figure 11: Sample Query (Query 1 button job)	13
Figure 12: Sample Query Result	13
Figure 13: Add Customer.....	14
Figure 14: Added Customer	14
Figure 15: Add Account.....	15
Figure 16: Added Account	15
Figure 17: Add Transaction	16
Figure 18: Added Transaction	16
Figure 19: Add Employee.....	17
Figure 20: Added Employee	17
Figure 21: Add Service	18
Figure 22: Added Service.....	18
Figure 23: Add Branch.....	19
Figure 24: Added Branch	19
Figure 25: Query 1 result.....	20
Figure 26: Query 2 result.....	21
Figure 27: Query 3 result.....	22
Figure 28: Query 4 result.....	23
Figure 29: Query 4 result (reduced execution time due to caching)	23
Figure 30: Query 5 result.....	24

Relevance

The project is a Database Management System (DBMS) for a Bank utilizing Object-Oriented Database (OODB) called objectDB, directly aligns with the focus of the class on Object-Oriented Database Systems (OODB).

Abstract

This project aims to develop a robust Database Management System (DBMS) tailored for a banking environment using an Object-Oriented Database (OODB) approach with objectDB. By leveraging the principles of object-oriented design, it seeks to provide efficient data storage, retrieval, and management for complex banking operations while ensuring scalability and flexibility.

Approach

1- ER Diagram

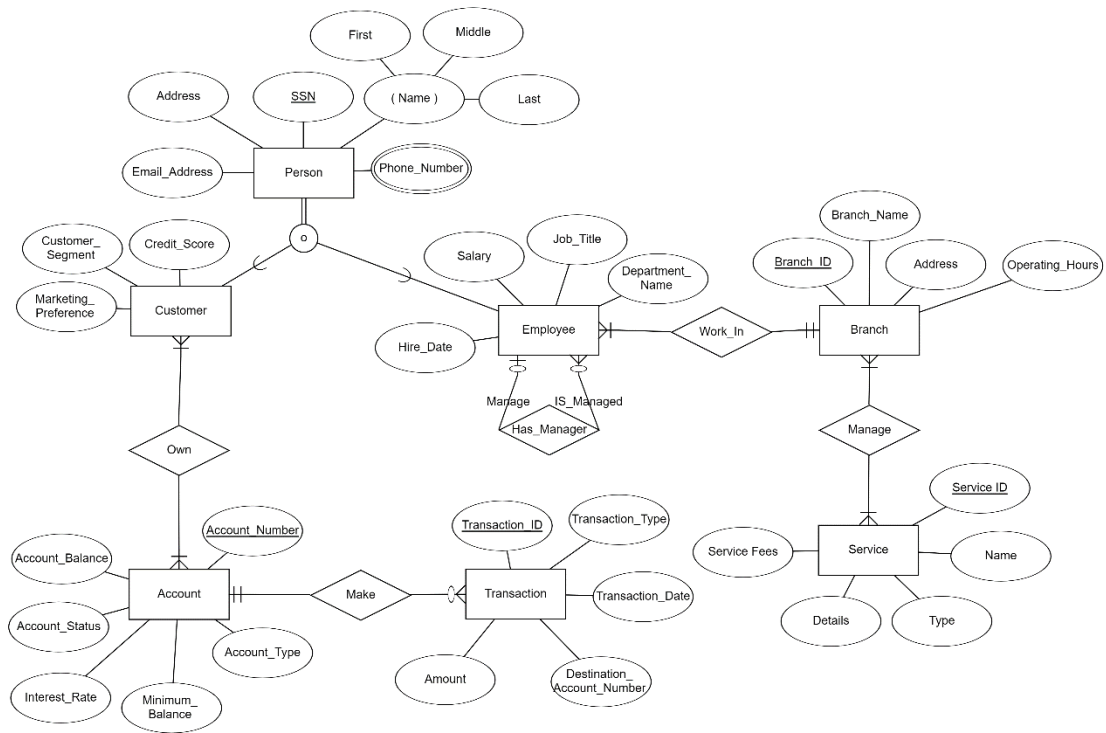


Figure 1: ER Diagram

2- ObjectDB

ObjectDB is the most productive software for developing Java database applications using the Java Persistence API (JPA). It is the first persistence solution that combined a powerful database with JPA support in one product, sparing the need to integrate an external JPA ORM with a database.

It comes with a GUI (integrated into our application) that provides useful features like showing the classes in the schema and their information:

- Number of objects in the table
- Table Size in bytes
- Fields and their datatype
- Whether it inherits from other classes (for example: figure 2 here shows that Customer extends Person, the keyword extends is equivalent to the keyword UNDER we learnt in the lectures)

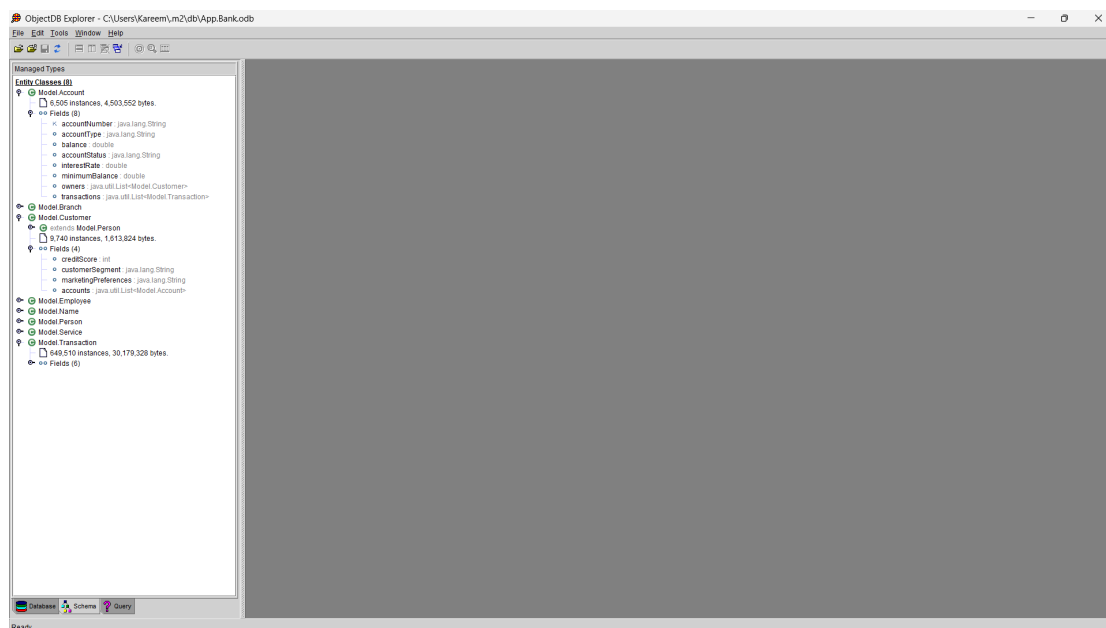


Figure 2: ObjectDB Schema GUI

ObjectDB Query GUI provides convenient space to enter and see the results of the query as a list of objects, we can expand each object in the list and see its attributes, if any of the attributes is an object (or list of objects) we can expand it as well.

It provides many useful information such as limiting the results if there are many tables, showing query execution plan, and evaluating performance in terms of execution time.

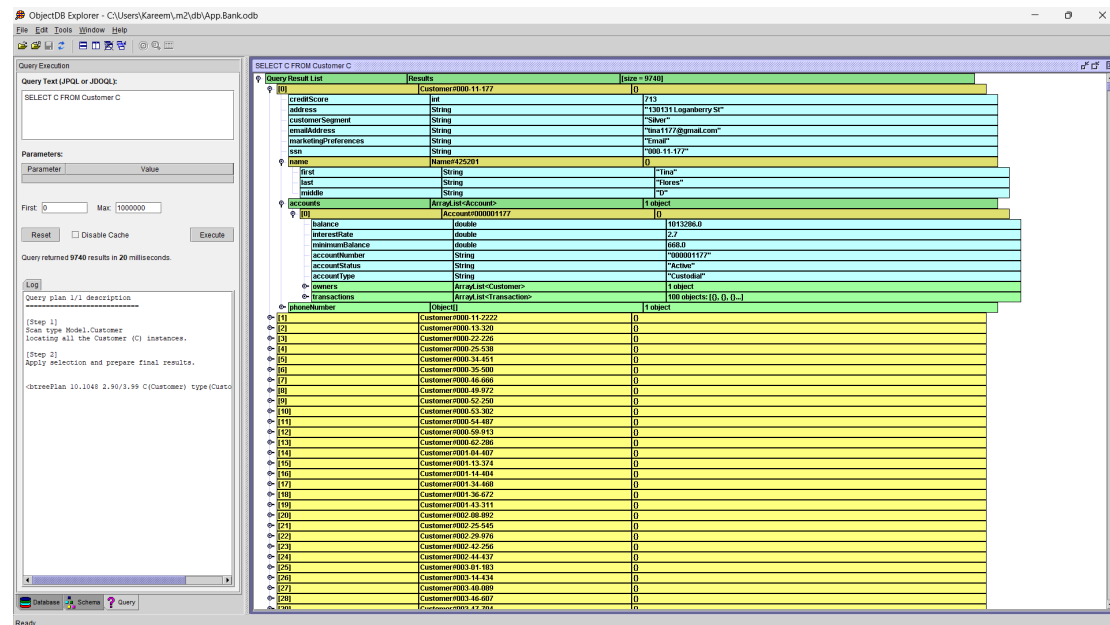



Figure 3: ObjectDB Query GUI

3- Classes and JPA

The system is built using Java programming language with the aid of objectDB dependency.. the linking with them is done with JPA library which facilitates creating tables/types in the schema using java classes (or more precisely: java entities)

JPA stands for Java Persistence API. It's a specification that defines how databases are accessed from within a Java application. JPA essentially acts as a middle layer that translates between Java objects and database tables. This allows us to work with data using plain old Java objects (POJOs) instead of having to write complex SQL queries.

As shown in the following figure, Java entities is annotated with the keyword `@Entity`, this is a JPA annotation that tells the database that this class should be translated into a table in the database upon compilation. The primary key is also annotated with the keyword `@Id`, it could be as well annotated with the keyword `@GeneratedValue` to tell the database that upon each insertion it should generate a new number (default is sequential generation) to work as an ID for the newly inserted object, again, this is optional as we can insert the object with the primary key manually as we do in Customer/Employee SSN and Account Number.



```
@Entity
public abstract class Person {

    @Id
    protected String ssn;

    protected Name name;
    protected String address;
    protected String emailAddress;
    protected String[] phoneNumber;
```

Figure 4: Sample Entity to be a table in the database

Note that the full code is in the Appendix.

4- Class Diagram

The following diagram shows the classes and their dependencies mapped from the ER diagram shown earlier, note that composite attributes in OODB needs to have its own type, that's why there is an extra class for attribute name (which is composed of First, Middle, Last).

Note that relations are represented as attributes in each object and inheritance is

One-to-Many

Account-Transaction (An account may have many transactions but a transaction must belong to one Account).

Branch-Employee (A branch must have many employees, but an employee must work in one branch)

Many-to-Many

Customer-Account (A customer must have one or more accounts while an account must belong to one or more customers), This is known as joint account (shared by 2 customers) and is common for married couples in Europe/USA.

Branch-Service (A branch must provide many services, while a service must be provided by many branches)

Inheritance

Customer and Employee inherit from **Person** as indicated by the blue arrows in the diagram.

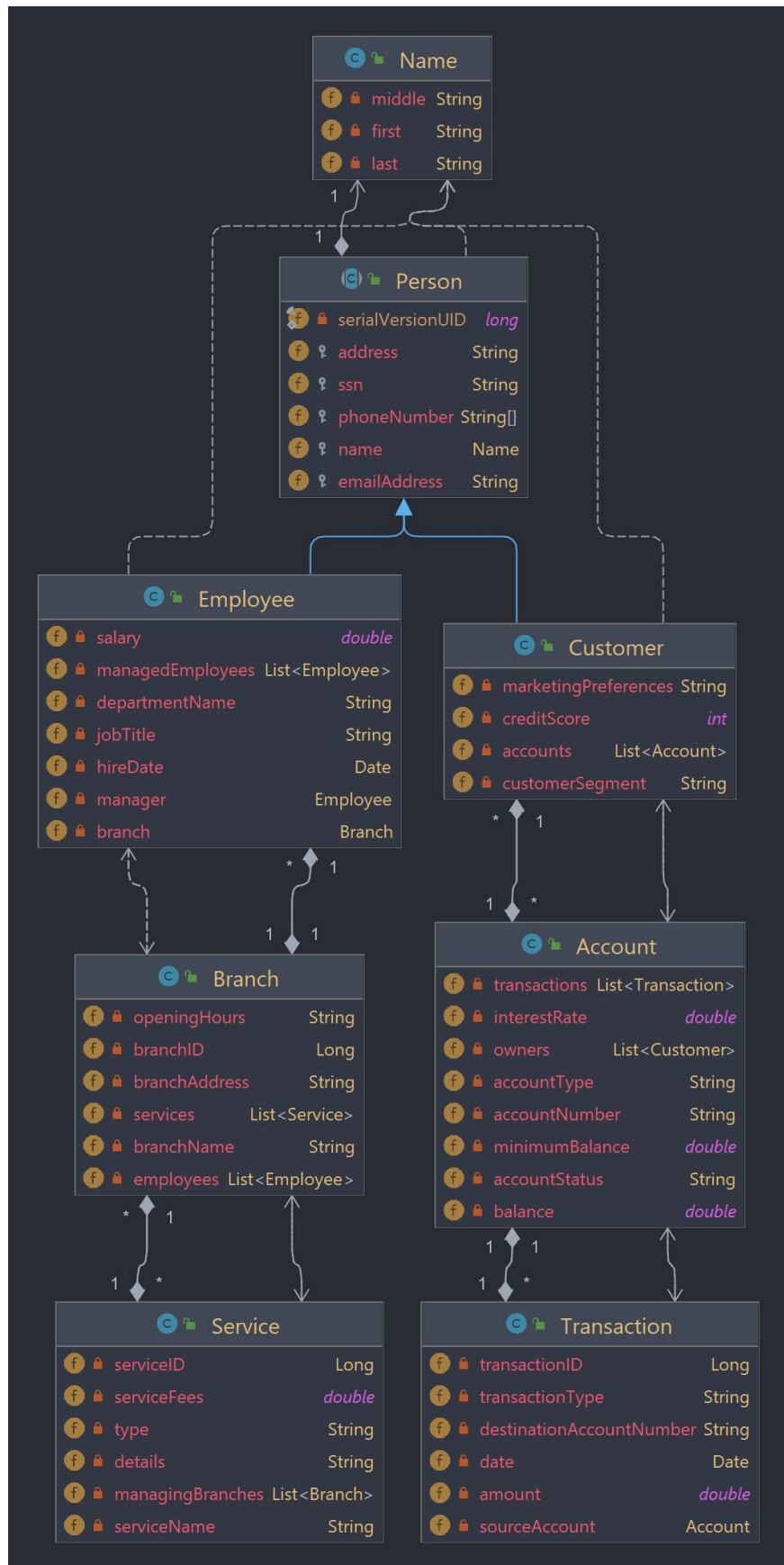


Figure 5: Class Diagram

5- Database Population

In order to test the queries and their performance we need many tuples in the database, but our domain is a Banking System and banking data is extremely confidential, so we seeked a smart randomization to generate relevant data.

This is provided in class **DatabaseUtils.java**

```
// List of first names
private static final String[] FIRST_NAMES = {"Alice", "Bob", "Charlie", "David", "Eve", "Frank", "G",
    "Kevin", "Laura", "Michael", "Nancy", "Oscar", "Peggy", "Quincy", "Rita", "Steve", "Tina",
    "Yvonne", "Zach", "Olivia", "Peter", "Quinn", "Rose", "Sam", "Tara", "Uma", "Violet", "Will"};

// List of last names
private static final String[] LAST_NAMES = {"Smith", "Johnson", "Williams", "Jones", "Brown", "David",
    "Anderson", "Thomas", "Jackson", "White", "Harris", "Martin", "Thompson", "Garcia", "Martinez",
    "Lewis", "Lee", "Walker", "Hall", "Allen", "Young", "Hernandez", "King", "Wright", "Lopez",
    "Baker", "Gonzalez", "Nelson", "Carter", "Mitchell", "Perez", "Roberts", "Turner", "Phillips",
    "Edwards", "Collins", "Stewart", "Sanchez", "Morris", "Rogers", "Reed", "Cook", "Morgan", "Meyer",
    "Cooper", "Richardson", "Cox", "Howard", "Ward", "Torres", "Peterson", "Gray", "Ramirez", "Hicks",
    "Sanders", "Price", "Bennett", "Wood", "Barnes", "Ross", "Henderson", "Cole", "Jenkins", "Perry",
    "Hughes", "Flores", "Washington", "Butler", "Simmons", "Foster", "Gonzales", "Bryant", "Alexander",
    "Hayes" };

// List of middle initials
private static final String[] MIDDLE_INITIALS = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J",
    "S", "T", "U", "V", "W", "X", "Y", "Z"};
```

Figure 6: List of names to generate combinations

```
Account account = new Account(accountNumber, accountType, balance, status, interestRate, minimumBalance, customers);
em.persist(account);

// Generate 100 transactions for the account
for (int j = 1; j <= 10; j++) {
    // Generate random transaction type
    String transactionType = TRANSACTION_TYPES[random.nextInt(TRANSACTION_TYPES.length)];
    // Generate random transaction amount
    double transactionAmount = 1 + random.nextInt( bound: 10000);
    // Generate random transaction date
    Date transactionDate = new Date( year: 2000 + random.nextInt( bound: 23), random.nextInt( bound: 12), random.nextInt( bound: 28));
    String destinationAccountNumber = null;
    if (transactionType.equals("Transfer")) {
        if(i < 10){
            transactionType = "Deposit";
        }
        else {
            destinationAccountNumber = String.format("%09d", random.nextInt( bound: i - 1) + 1);
        }
    }
    Transaction transaction = new Transaction(transactionType, transactionAmount, transactionDate, destinationAccountNumber);
    em.persist(transaction);
}
em.getTransaction().commit();
}
```

Figure 7: Generation of transactions for some account

We initially had 1,021,060 objects in the Database, but after removing irrelevant objects and illogical ones we settled on a sample population of 677,548 objects as shown in the next figure.

Database Details	
File Size:	36,962,304
Total Objects:	677,548
Next Object Id:	966,066
Last Modified:	
2024-05-16 04:53:37	

Figure 8: Database Information

6- GUI

Using Swing library, a simple GUI was created for supporting forms to insert data into the database.

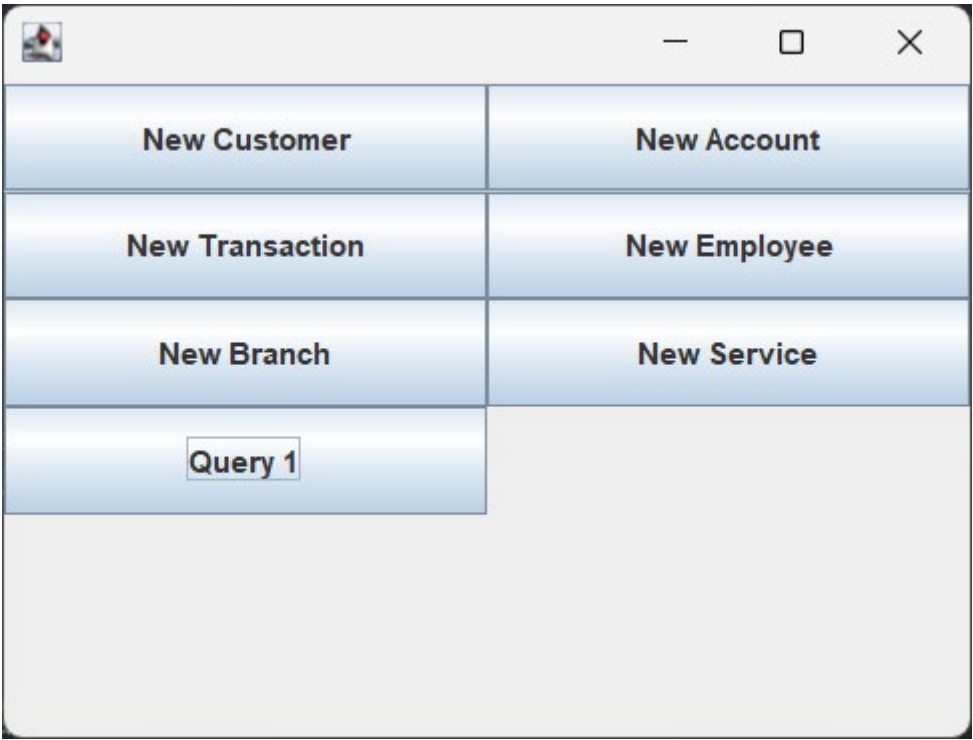
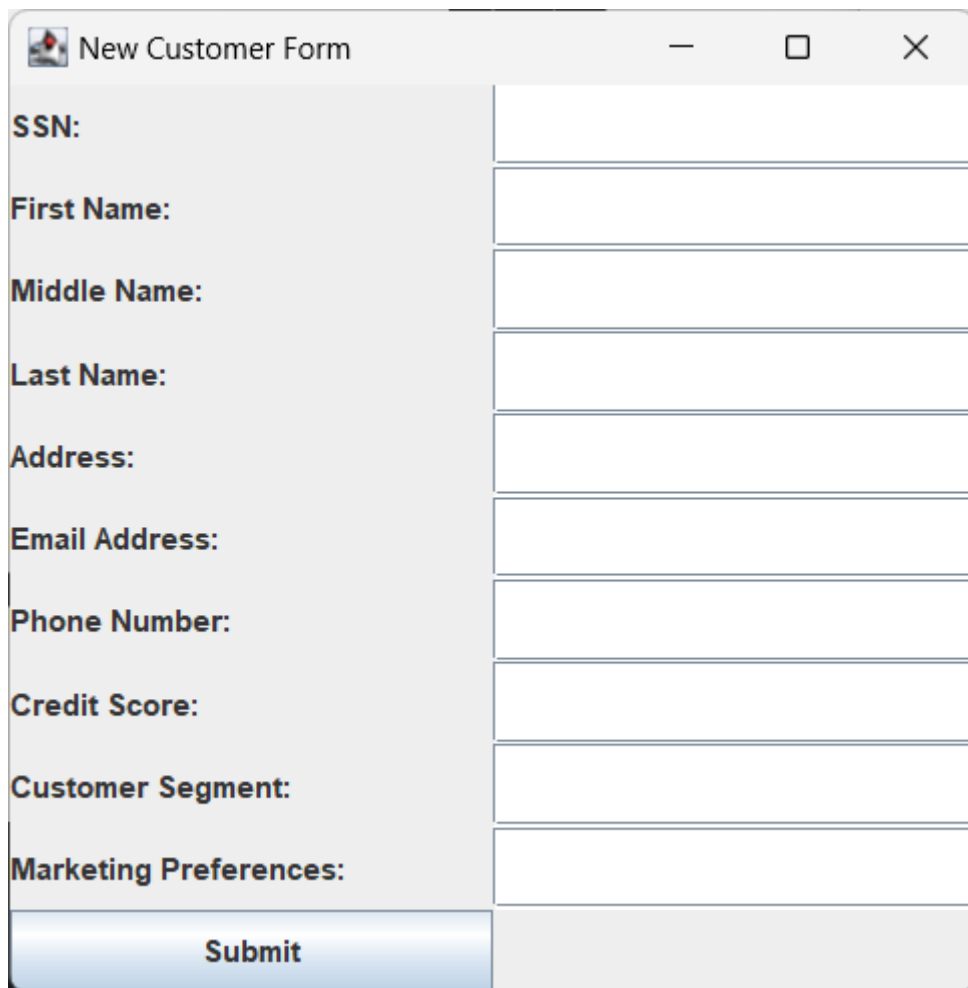


Figure 9: Main Menu



A screenshot of a Java Swing window titled "New Customer Form". The window has a standard title bar with minimize, maximize, and close buttons. The form contains several text input fields for the following labels: SSN, First Name, Middle Name, Last Name, Address, Email Address, Phone Number, Credit Score, Customer Segment, and Marketing Preferences. At the bottom left of the form is a "Submit" button.

Figure 10: Sample Form

```
// Find the number of customer objects in the database.  
Query q1 = em.createQuery("SELECT COUNT(c) FROM Customer c");  
System.out.println("Total Customers: " + q1.getSingleResult());
```

Figure 11: Sample Query (Query 1 button job)

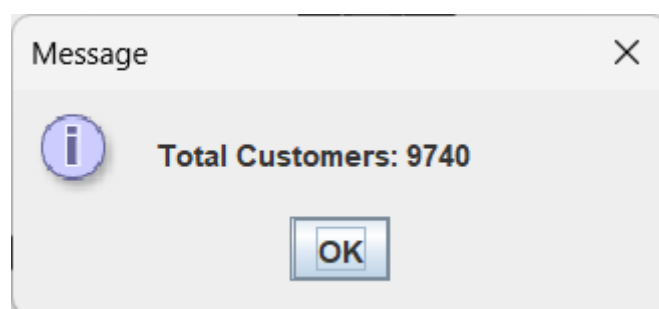


Figure 12: Sample Query Result

Results and Analysis

1- Forms

Add Customer

New Customer

New Account

New Transaction

New Employee

New Customer Form

SSN:

123456

First Name:

Ahmed

Middle Name:

Tarek

Last Name:

Ahmed

Address:

Tanta

Email Address:

ahmed@gmail.com

Phone Number:

0123456789

Credit Score:

100

Customer Segment:

VIP

Marketing Preferences:

Mail

Submit

Figure 13: Add Customer

Result in query showing that customer in the database

Query Execution

Query Text (JSQL or JDOQL):

SELECT FROM Customer c
WHERE c.ssn = '123456'

Parameters:

Parameter

Value

First: 0

Max: 1000000

Reset

Disable Cache

Execute

Query returned one result in 9 milliseconds.

Log

Locating Customer (c) instances that satisfy:
(c.ssn='123456').
[Step 2]
Apply selection and prepare final results.
<treePlan 1.0087 0.96/2.08 c(Customer) type(Cus

Query Result List

Results

[size = 1]

creditScore

Int

100

address

String

"Tanta"

customerSegment

String

"VIP"

emailAddress

String

"ahmed@gmail.com"

marketingPreferences

String

"Mail"

ssn

String

"123456"

name

Name#109419

0

first

String

"Ahmed"

last

String

"Ahmed"

middle

String

"Tarek"

accounts

ArrayList<Account>

1 object

balance

double

999.0

interestRate

double

3.0

minimumBalance

double

99.0

accountNumber

String

"98765"

accountStatus

String

"Active"

accountType

String

"Savings"

owners

ArrayList<Customer>

1 object

customer

Customer#123456 (R)

0

transactions

ArrayList<Transaction>

1 object

amount

double

150.0

date

Date

May 16, 2024, 4:16:50 AM

destinationAccountNu...

String

"100007806"

transactionID

Long

109422

transactionType

String

"money transfer"

sourceAccount

Account#98765 (R)

0

phoneNumber

Object[]

1 object

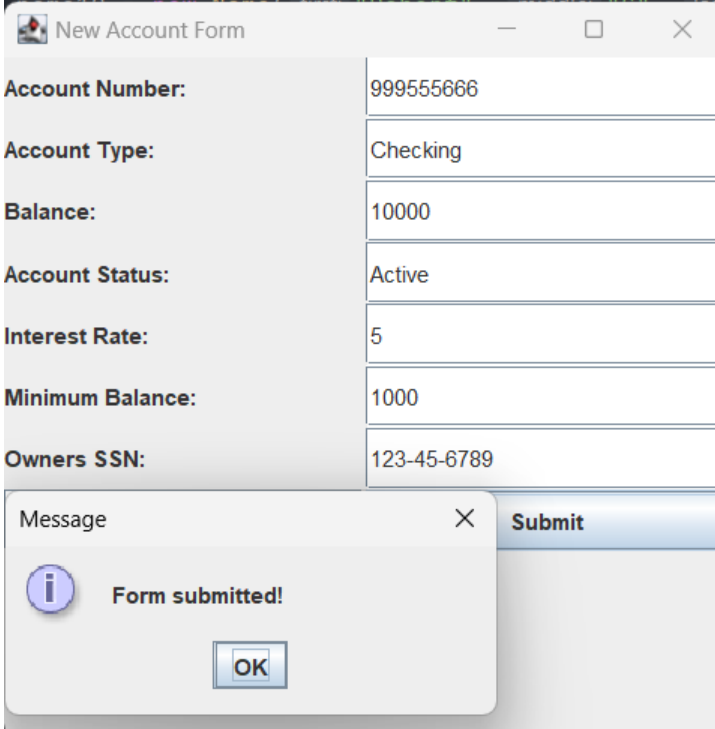
phone

String

"0123456789"

Figure 14: Added Customer

Add Account



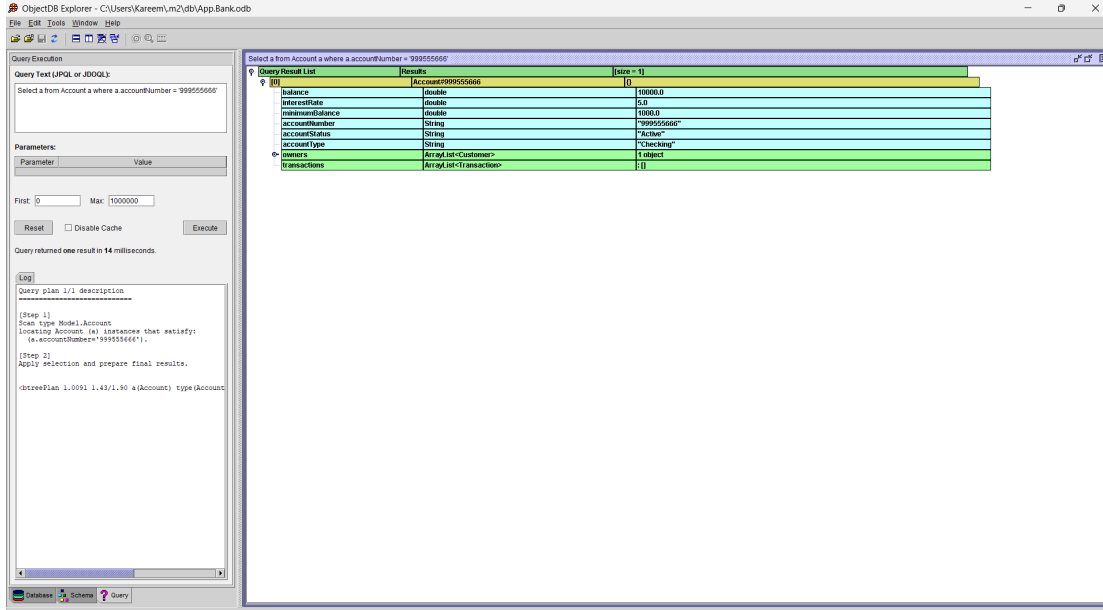
A screenshot of a web application window titled "New Account Form". The form contains several input fields with the following values: Account Number: 999555666, Account Type: Checking, Balance: 10000, Account Status: Active, Interest Rate: 5, Minimum Balance: 1000, and Owners SSN: 123-45-6789. A "Submit" button is located at the bottom right. A modal message box is overlaid on the form, displaying an information icon, the text "Form submitted!", and an "OK" button.

Account Number:	999555666
Account Type:	Checking
Balance:	10000
Account Status:	Active
Interest Rate:	5
Minimum Balance:	1000
Owners SSN:	123-45-6789

Message: Form submitted! OK

Figure 15: Add Account

The result query showing that account in the database



A screenshot of the ObjectDB Explorer application showing a query result. The query is "Select a from Account a where a.accountNumber = 999555666". The results are displayed in a table with columns: balance, interestRate, minimumBalance, accountNumber, accountStatus, accountType, owners, and transactions. The table contains one row of data for the account with number 999555666.

balance	interestRate	minimumBalance	accountNumber	accountStatus	accountType	owners	transactions
10000.0	5.0	1000.0	999555666	Active	Checking	Account-Owner	Account-Transactions

Figure 16: Added Account

Add Transaction

New Transaction Form

Transaction Type:
Deposit

Amount:
200

Destination Account Number:

Source Account Number:
999555666

Submit

Message
Form submitted!
OK

Figure 17: Add Transaction

The result query showing that transaction in the database, notice that the balance in its source account now is 10,200. Which means the 200 deposited amount where added to that account original balance (10,000) and got updated and committed automatically.

ObjectDB Explorer - C:\Users\Kareem\m2\db\App.Bank.odb

Query Execution
Query Text (JPQL or JDOQL):
Select t from Transaction t where t.sourceAccount.accountNumber = 999555666

Parameters:
Parameter Value

First: 0 Max: 1000000

Execute

Query returned one result in 213 milliseconds.

Log

```

Step 1: Process Transaction (t) instances
[Step 1a]
Scan type Model.Transaction
locating all the Transaction (t) instances.
[Step 1b]
Evaluate fields in Transaction (t) instances.
Step 2: Process Account (v1) instances
(for every result of step 1)
[Step 2a]
Iterate over all the instances (v1) in t.sourceAccount.
[Step 2b]
Filter the results of step 2a
retaining only results that satisfy:
(v1.accountNumber=999555666).
Step 3: Apply selection
Apply selection and prepare final results.
<multisetPlan 13.9737 5.84/5.81 t,v1>
<extractPlan 12.9314 4.17/5.61 t(Transaction)>
<filterPlan 11.9314 4.17/5.61 t(Transaction) type(Transaction[all]) />
<extractPlan>
<filterPlan 9.0212 0.02/0.00 v1(AccountNumber=999555666)>
<boundPlan 0.0 0.00/0.00 v1(Account) bound(t.sourceAccount) />
</filterPlan>
</multisetPlan>

```

Query Result List (size = 1)

Amount	Location	Results
200.0	Transaction900006	0
Date	Date	May 16, 2024 10:45:27 PM
destinationAccountNumber	String	null
transactionId	Long	966066
transactionType	String	"Deposit"
sourceAccount	Account999555666	0
Balance	Double	10200.9
interestRate	Double	5.0
interestBalance	Double	19800.0
accountNumber	String	"999555666"
accountStatus	String	"Active"
accountType	String	"Checking"
owner	ArrayList<Customer>	1 object
transactions	ArrayList<Transaction>	1 object

Figure 18: Added Transaction

Add Employee

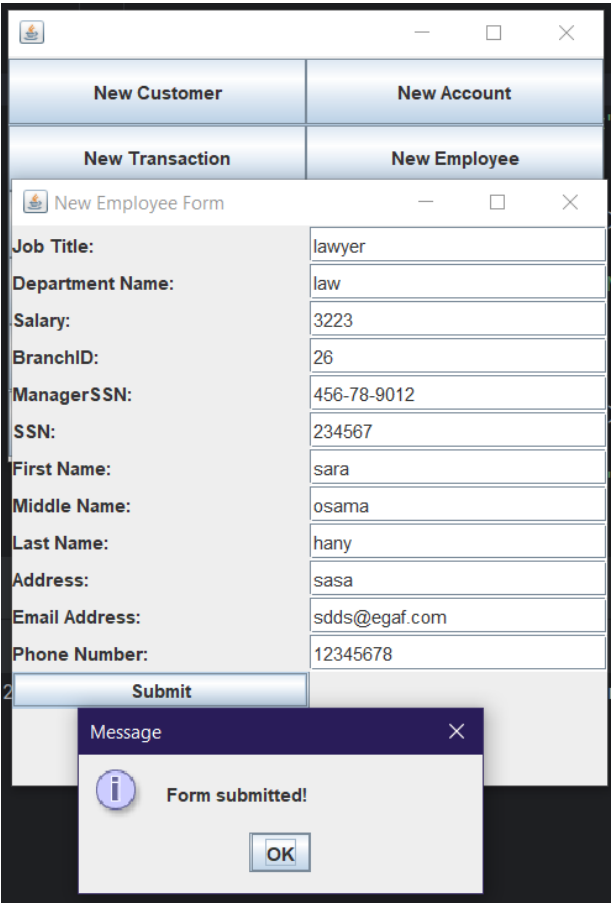


Figure 19: Add Employee

Result in query showing that employee in the database

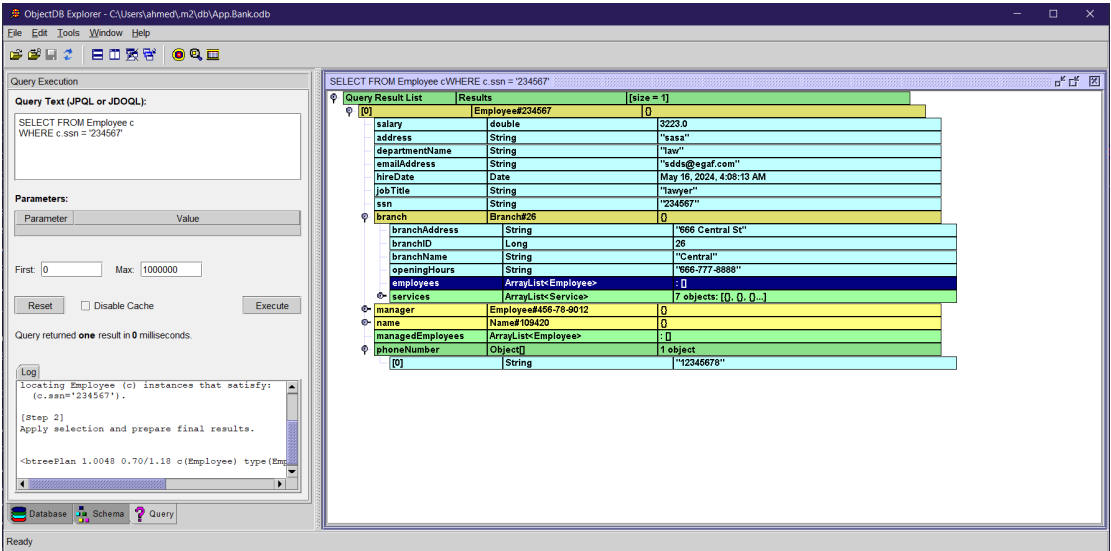


Figure 20: Added Employee

Add Service

Message

Form submitted!

OK

New Service Form

Service Name:

visa

Service Fees:

33

Type:

VIP

Details:

get a visa

Managing Branches(enter branch IDs separated by space):

26 27

Submit

Figure 21: Add Service

Result in query showing that service in the database

ObjectDB Explorer - C:\Users\ahmedf_m2\obj\App.Bank.odt

FileEditToolsWindowHelp

Query Execution

Query Text (JSQL or JDOQL):

SELECT FROM Employee c
WHERE c.ssn = '234567'

Parameters:

Parameter	Value
First	0
Max	1000000

☐ Disable Cache

Query returned one result in 0 milliseconds.

Log

locating Employee (c) instances that satisfy:
(c.ssn='234567').

[Step 2]
Apply selection and prepare final results.

<btreesPlan 1.0048 0.70/1.18 c(Employee) type (Em

Database

Schema

Query

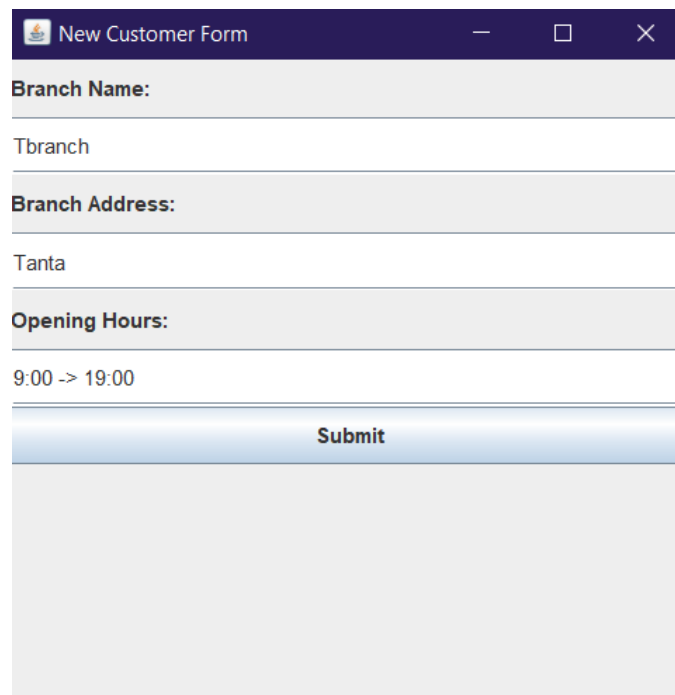
Ready

SELECT FROM Employee c WHERE c.ssn = '234567'

Y	W	U
salary	double	3223.0
address	String	"sasa"
departmentName	String	"law"
emailAddress	String	"sadda@egaf.com"
hireDate	Date	May 16, 2024, 4:08:13 AM
jobTitle	String	"lawyer"
ssn	String	"234567"
branch	Branch#26	0
branchAddress	String	"666 Central St"
branchID	Long	26
branchName	String	"Central"
openingHours	String	"666-777-8888"
employees	Array<List<Employee>	1
services	Array<List<Service>	7 objects: [0, 0, 0, ...]
0	Service#42	0
1	Service#34	0
2	Service#36	0
3	Service#38	0
4	Service#40	0
5	Service#109417	0
6	Service#109421	0
serviceFees	double	33.0
details	String	"get a visa"
serviceID	Long	109421
serviceName	String	"visa"
type	String	"VIP"
managingBranches	Array<List<Branch>	2 objects: [0, 0]
manager	Employee#456-78-9012	0
name	Name#109420	0
managedEmployees	Array<List<Employee>	1
phoneNumber	Object[]	1 object
0	String	"12345678"

Figure 22: Added Service

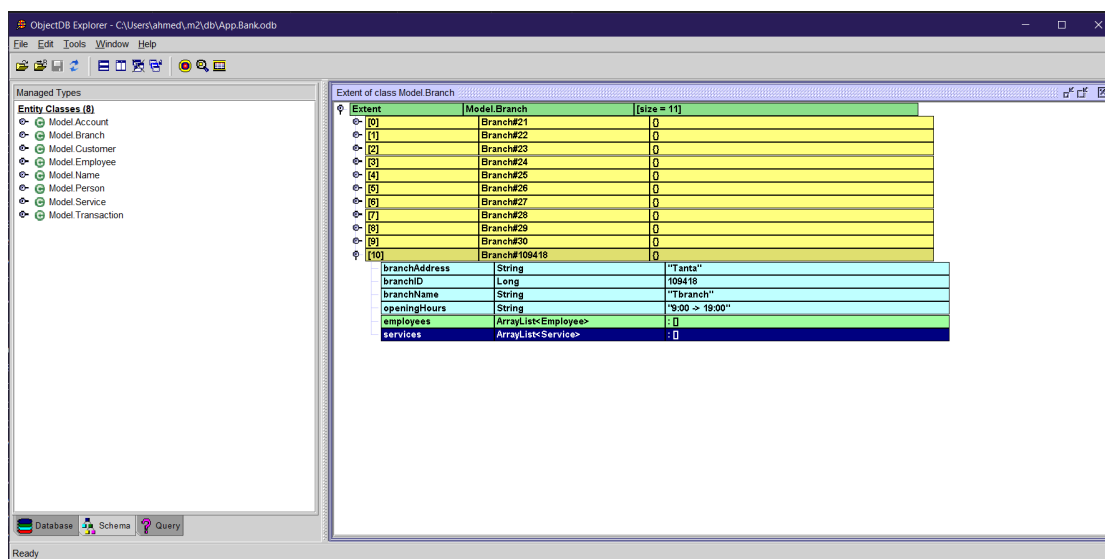
Add Branch



A screenshot of a web form titled "New Customer Form". The form has three input fields: "Branch Name:" with the value "Tbranch", "Branch Address:" with the value "Tanta", and "Opening Hours:" with the value "9:00 -> 19:00". Below these fields is a blue "Submit" button.

Figure 23: Add Branch

Result in query showing that branch in the database



A screenshot of the ObjectDB Explorer application. The left pane shows "Managed Types" with a tree of entity classes. The right pane shows the "Extent of class Model.Branch" with a table of 11 rows. The table columns are "branchID", "branchName", "branchAddress", "openingHours", "employees", and "services". The last row (branchID 10) shows the data for the newly added branch.

branchID	branchName	branchAddress	openingHours	employees	services
0	Branch#21				
1	Branch#22				
2	Branch#23				
3	Branch#24				
4	Branch#25				
5	Branch#26				
6	Branch#27				
7	Branch#28				
8	Branch#29				
9	Branch#30				
10	Branch#109416	"Tanta"	"9:00 -> 19:00"	[:]	[:]

Figure 24: Added Branch

2- Queries

- 1- Retrieve all customers names and accounts balances that have a balance more than a million \$, only if they have saving accounts.

```
SELECT C.name, C.accounts.balance
```

FROM Customer C

WHERE C.accounts.accountType = 'Savings' and C.accounts.balance > 1000000

Query Execution

Query Text (SQL or JDOQL):

```
SELECT C.name, C.accounts.balance FROM Customer C WHERE C.accounts.accountType = 'Savings' and C.accounts.balance > 1000000
```

Parameters:

Parameter	Value
First	0
Max	1000000

☐ Disable Cache

Query returned 495 results in 8 milliseconds

(for every result of step 1)

[Step 2a]
Iterate over all the instances (vcl) in C.accounts.

[Step 2b]
Retrieve fields in Account (vcl) instances.

[Step 2c]
Filter the results of step 2a retaining only results that satisfy:
(vcl.balance>1000000).

[Step 2d]
Filter the results of step 2c retaining only results that satisfy:
(vcl.accountType='Savings').

Step 3: Apply selection

Apply selection and prepare final results.

```
<multiVarPlan 14.18962 4.592/3.99 C,vcl>
<extractPlan 10.1040 2.803/3.99 C (Customer)>
<extractPlan 10.1040 2.803/3.99 C (Customer)Type (Customer)[all] />
</extractPlan>
<filterPlan 1.0424 0.52/0.00 (vcl.accountType='Savings')>
</filterPlan>
<filterPlan 2.2212 0.50/0.00 (vcl.balance>1000000)>
<extractPlan 2.0 0.48/0.00 vcl (Account)>
<boundPlan 0.0 0.0/0.00 vcl (Account) bound (C.accounts) />
</extractPlan>
</filterPlan>
</filterPlan>
</multiVarPlan>
```

SELECT C.name, C.accounts.balance FROM Customer C WHERE C.accounts.accountType = 'Savings' and C.accounts.balance > 1000000

Query Result List	Results	[size = 495]																				
[0]	<table border="1"> <thead> <tr> <th>C.name</th> <th>String</th> <th>Name\$151506</th> <th>()</th> </tr> </thead> <tbody> <tr> <td>first</td> <td>String</td> <td>"Tami"</td> <td></td> </tr> <tr> <td>last</td> <td>String</td> <td>"Anderson"</td> <td></td> </tr> <tr> <td>middle</td> <td>String</td> <td>"K"</td> <td></td> </tr> <tr> <td>C.accounts.balance</td> <td>double</td> <td>1002019.0</td> <td></td> </tr> </tbody> </table>	C.name	String	Name\$151506	()	first	String	"Tami"		last	String	"Anderson"		middle	String	"K"		C.accounts.balance	double	1002019.0		(Model Name\$3156, 1002019.0)
C.name	String	Name\$151506	()																			
first	String	"Tami"																				
last	String	"Anderson"																				
middle	String	"K"																				
C.accounts.balance	double	1002019.0																				
[1]	<table border="1"> <thead> <tr> <th>C.name</th> <th>String</th> <th>Name\$547484</th> <th>()</th> </tr> </thead> <tbody> <tr> <td>first</td> <td>String</td> <td>"Tami"</td> <td></td> </tr> <tr> <td>last</td> <td>String</td> <td>"Powell"</td> <td></td> </tr> <tr> <td>middle</td> <td>String</td> <td>"K"</td> <td></td> </tr> <tr> <td>C.accounts.balance</td> <td>double</td> <td>1075949.0</td> <td></td> </tr> </tbody> </table>	C.name	String	Name\$547484	()	first	String	"Tami"		last	String	"Powell"		middle	String	"K"		C.accounts.balance	double	1075949.0		(Model Name\$3156, 1075949.0)
C.name	String	Name\$547484	()																			
first	String	"Tami"																				
last	String	"Powell"																				
middle	String	"K"																				
C.accounts.balance	double	1075949.0																				
[2]	<table border="1"> <thead> <tr> <th>C.name</th> <th>String</th> <th>Name\$231340</th> <th>()</th> </tr> </thead> <tbody> <tr> <td>first</td> <td>String</td> <td>"Tami"</td> <td></td> </tr> <tr> <td>last</td> <td>String</td> <td>"Chase"</td> <td></td> </tr> <tr> <td>middle</td> <td>String</td> <td>"K"</td> <td></td> </tr> <tr> <td>C.accounts.balance</td> <td>double</td> <td>1193001.0</td> <td></td> </tr> </tbody> </table>	C.name	String	Name\$231340	()	first	String	"Tami"		last	String	"Chase"		middle	String	"K"		C.accounts.balance	double	1193001.0		(Model Name\$3156, 1193001.0)
C.name	String	Name\$231340	()																			
first	String	"Tami"																				
last	String	"Chase"																				
middle	String	"K"																				
C.accounts.balance	double	1193001.0																				
[3]	<table border="1"> <thead> <tr> <th>C.name</th> <th>String</th> <th>Name\$122717</th> <th>()</th> </tr> </thead> <tbody> <tr> <td>first</td> <td>String</td> <td>"Kina"</td> <td></td> </tr> <tr> <td>last</td> <td>String</td> <td>"Trent"</td> <td></td> </tr> <tr> <td>middle</td> <td>String</td> <td>"K"</td> <td></td> </tr> <tr> <td>C.accounts.balance</td> <td>double</td> <td>1104430.0</td> <td></td> </tr> </tbody> </table>	C.name	String	Name\$122717	()	first	String	"Kina"		last	String	"Trent"		middle	String	"K"		C.accounts.balance	double	1104430.0		(Model Name\$3156, 1104430.0)
C.name	String	Name\$122717	()																			
first	String	"Kina"																				
last	String	"Trent"																				
middle	String	"K"																				
C.accounts.balance	double	1104430.0																				
[4]	<table border="1"> <thead> <tr> <th>C.name</th> <th>String</th> <th>Name\$179299</th> <th>()</th> </tr> </thead> <tbody> <tr> <td>first</td> <td>String</td> <td>"Peter"</td> <td></td> </tr> <tr> <td>last</td> <td>String</td> <td>"Edie"</td> <td></td> </tr> <tr> <td>middle</td> <td>String</td> <td>"K"</td> <td></td> </tr> <tr> <td>C.accounts.balance</td> <td>double</td> <td>1031426.0</td> <td></td> </tr> </tbody> </table>	C.name	String	Name\$179299	()	first	String	"Peter"		last	String	"Edie"		middle	String	"K"		C.accounts.balance	double	1031426.0		(Model Name\$3156, 1031426.0)
C.name	String	Name\$179299	()																			
first	String	"Peter"																				
last	String	"Edie"																				
middle	String	"K"																				
C.accounts.balance	double	1031426.0																				
[5]	Result	(Model Name\$3156, 1163809.0)																				
[6]	Result	(Model Name\$3156, 1163809.0)																				
[7]	Result	(Model Name\$3156, 1163809.0)																				
[8]	Result	(Model Name\$3156, 1163809.0)																				
[9]	Result	(Model Name\$3156, 1163809.0)																				
[10]	Result	(Model Name\$3156, 1163809.0)																				
[11]	Result	(Model Name\$3156, 1163809.0)																				
[12]	Result	(Model Name\$3156, 1163809.0)																				
[13]	Result	(Model Name\$3156, 1163809.0)																				
[14]	Result	(Model Name\$3156, 1163809.0)																				
[15]	Result	(Model Name\$3156, 1163809.0)																				
[16]	Result	(Model Name\$3156, 1163809.0)																				
[17]	Result	(Model Name\$3156, 1163809.0)																				
[18]	Result	(Model Name\$3156, 1163809.0)																				
[19]	Result	(Model Name\$3156, 1163809.0)																				
[20]	Result	(Model Name\$3156, 1163809.0)																				
[21]	Result	(Model Name\$3156, 1163809.0)																				
[22]	Result	(Model Name\$3156, 1163809.0)																				
[23]	Result	(Model Name\$3156, 1163809.0)																				

Figure 25: Query 1 result

2- Retrieve all employees who works in big branches that has more than 100 employees

SELECT e1

From Employee e1

WHERE Size(e1.branch.employees) > 100

The screenshot shows a query execution interface. On the left, the 'Query Text (JPQL or JOOQL):' section contains the query: `SELECT e1
From Employee e1
WHERE Size(e1.branch.employees) > 100`. Below this, the 'Parameters:' section is empty. The 'Query returned 629 results in 6 milliseconds.' message is visible. The 'Log' section shows the execution steps. On the right, the 'Query Result List' shows a table with 629 results. The table has columns: `id`, `salary`, `address`, `departmentName`, `emailAddress`, `hireDate`, `jobTitle`, `ssn`, `branch`, `manager`, `name`, `managerEmployees`, and `phoneNumber`. The first row shows an employee with `id` 1000, `salary` 50513, `address` '178179 Blackberry St', `departmentName` 'Accounting', `emailAddress` 'tyler500@gmail.com', `hireDate` '2017-1-20 02:33:02 PM', `jobTitle` 'Manager', `ssn` '7808 58 513', `branch` 'Branch0000000000', `manager` null, `name` 'Name0000000000', `managerEmployees` '0', and `phoneNumber` '0 object'.

id	salary	address	departmentName	emailAddress	hireDate	jobTitle	ssn	branch	manager	name	managerEmployees	phoneNumber
Employee1000	50513	178179 Blackberry St	Accounting	tyler500@gmail.com	2017-1-20 02:33:02 PM	Manager	7808 58 513	Branch0000000000	null	Name0000000000	0	0 object
Employee1001	14 041											
Employee1002	56 866											
Employee1003	70 153											
Employee1004	11 042											
Employee1005	44 542											
Employee1006	78 570											
Employee1007	23 459											
Employee1008	48 789											
Employee1009	31 876											
Employee1010	78 754											
Employee1011	48 293											
Employee1012	89 432											
Employee1013	28 230											
Employee1014	100 62 730											
Employee1015	136 63 739											
Employee1016	114 62 017											
Employee1017	152 66 545											
Employee1018	156 66 934											
Employee1019	156 65 181											
Employee1020	168 44 101											
Employee1021	169 52 548											
Employee1022	200 75 963											
Employee1023	204 19 852											
Employee1024	221 67 547											
Employee1025	236 75 964											
Employee1026	248 31 881											
Employee1027	249 44 381											
Employee1028	260 67 548											
Employee1029	272 81 215											
Employee1030	284 69 732											
Employee1031	332 18 850											
Employee1032	345 67 8861											
Employee1033	352 68 745											
Employee1034	356 68 134											
Employee1035	369 65 523											
Employee1036	383 61 011											
Employee1037	376 83 579											
Employee1038	388 72 666											

Figure 26: Query 2 result

- WHERE e.manager.salary > 100000 and e.manager.branch.branchName = 'Main'

Figure 27: Query 3 result

- 4- Retrieve all transactions that their amount is greater than 1000 \$ and made by Premium Customers

SELECT t

FROM Transaction t

WHERE t.amount > 1000 and t.sourceAccount.owners.customerSegment = 'Premium'

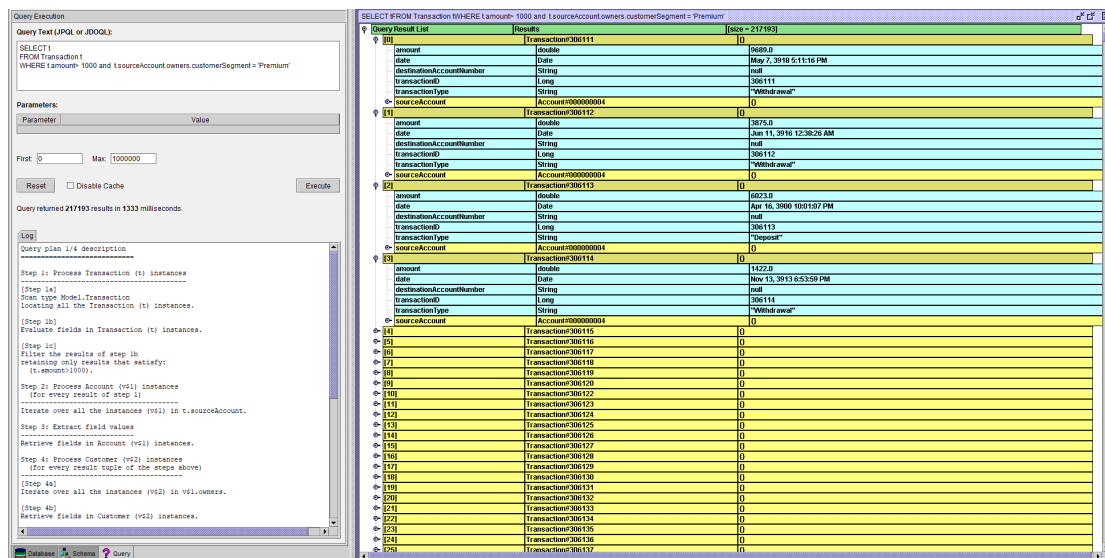


Figure 28: Query 4 result

Notice that the results count is > 200,000 and the query returned in < 2 seconds

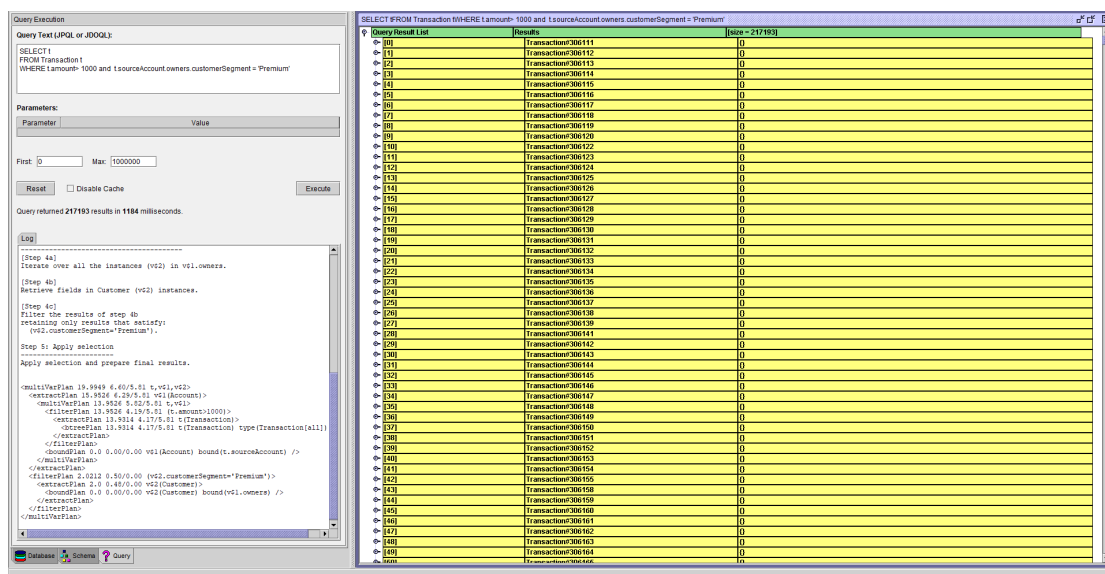


Figure 29: Query 4 result (reduced execution time due to caching)

After running the query again, the time went down from 1.3 to 1.1 seconds

- 5- Retrieve all customer's email addresses if the have Email but not SMS in the marketing preferences

SELECT C.emailAddress

FROM Customer C

WHERE C.marketingPreferences LIKE '%Email%'

AND NOT C.marketingPreferences LIKE '%SMS%'

The screenshot shows a database query execution interface. On the left, the 'Query Execution' panel displays the SQL query: `SELECT C.emailAddress FROM Customer C WHERE C.marketingPreferences LIKE '%Email%' AND NOT C.marketingPreferences LIKE '%SMS%'`. Below the query, it indicates 'Query returned 2617 results in 7 milliseconds'. The right panel shows the 'Query Result List' with columns 'id', 'Results', and 'Size'. The results are a list of email addresses, such as 'tina117@gmail.com', 'tina117@gmail.com', 'tina117@gmail.com', etc., each with a size of 2617.

id	Results	Size
101	String	2617
102	String	2617
103	String	2617
104	String	2617
105	String	2617
106	String	2617
107	String	2617
108	String	2617
109	String	2617
110	String	2617
111	String	2617
112	String	2617
113	String	2617
114	String	2617
115	String	2617
116	String	2617
117	String	2617
118	String	2617
119	String	2617
120	String	2617
121	String	2617
122	String	2617
123	String	2617
124	String	2617
125	String	2617
126	String	2617
127	String	2617
128	String	2617
129	String	2617
130	String	2617
131	String	2617
132	String	2617
133	String	2617
134	String	2617
135	String	2617
136	String	2617
137	String	2617
138	String	2617
139	String	2617
140	String	2617
141	String	2617
142	String	2617
143	String	2617
144	String	2617
145	String	2617
146	String	2617
147	String	2617
148	String	2617
149	String	2617
150	String	2617

Figure 30: Query 5 result

3- Discussion

Both the Insert and Select queries shown in this section demonstrated the ease of using the application providing robust integration between the Application itself in Java (and its GUI) and The database.

User-Friendly Interaction:

- **Straightforward Queries:** These examples demonstrate that the application allows users to interact with the database using simple and common commands. This makes it easier to learn and use for beginners, even those without extensive database experience.
- **GUI Integration:** The seamless integration between the Java application and its GUI indicates that the application likely provides a user-friendly interface for interacting with the database. This interface could include buttons, drop-down menus, or text fields that allow users to easily construct and execute queries without needing to write raw code.

Performance Advantages:

- **Scalability:** The passage emphasizes that the application's execution time remains around 1 second even as the query result size increases. This suggests the application is well-suited for handling large datasets without significant performance degradation. This scalability is crucial for applications that manage ever-growing amounts of data.
- **Efficiency Compared to Relational Databases:** The passage highlights a key advantage over traditional relational databases. Relational databases often rely on joining multiple tables to answer complex queries. This process can be time-consuming and resource-intensive, leading to slower query execution. The application, on the other hand, seems to avoid such overhead, resulting in faster performance.

References:

[1] "ObjectDB - Object Database for Java (JPA/JDO)" Available at:
<https://www.objectdb.com/>.

[2] "Investopedia. (n.d.). Joint Account. Retrieved from
<https://www.investopedia.com/terms/j/jointaccount.asp>"

Appendix

Bank.java (main class)

```
package App;

import javax.persistence.*;
import javax.swing.*;
import java.util.*;
import Model.*;

public class Bank {
    private static EntityManagerFactory emf;
    private static EntityManager em;

    public static void main(String[] args) {
        // Open a database connection
        // (create a new database if it doesn't exist yet):
        emf =
Persistence.createEntityManagerFactory("$objectdb/db/App.Bank.odb");
        em = emf.createEntityManager();

        //      DatabaseUtils.populateDatabase(); // comment again after
run
        //DatabaseUtils.populateDatabase2();
        // Find the number of Customer objects in the database:
        Query q1 = em.createQuery("SELECT COUNT(c) FROM Customer c");
        System.out.println("Total Customers: " +
q1.getSingleResult());

        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                // Create an instance of the Home form
                Home homeForm = new Home(emf, em);

                // Set the Home form to be visible
                homeForm.setVisible(true);
            }
        });
    }

    public static EntityManager getEntityManager() {
        return em;
    }
}
```

DatabaseUtils.java (for population)

```
package App;

import Model.*;

import javax.persistence.EntityManager;
import javax.persistence.Query;
import java.util.*;

public class DatabaseUtils {

    private static EntityManager em;

    // List of first names
    private static final String[] FIRST_NAMES = {"Alice", "Bob",
"Charlie", "David", "Eve", "Frank", "Grace", "Heidi", "Ivan", "Judy",
"Kevin", "Laura", "Michael", "Nancy", "Oscar", "Peggy",
"Quincy", "Rita", "Steve", "Tina", "Ursula", "Victor", "Wendy",
"Xavier",
"Yvonne", "Zach", "Olivia", "Peter", "Quinn", "Rose",
"Sam", "Tara", "Uma", "Violet", "Will", "Xena", "Yara", "Zara" };
    // List of last names
    private static final String[] LAST_NAMES = {"Smith", "Johnson",
"Williams", "Jones", "Brown", "Davis", "Miller", "Wilson", "Moore",
"Taylor",
"Anderson", "Thomas", "Jackson", "White", "Harris",
"Martin", "Thompson", "Garcia", "Martinez", "Robinson", "Clark",
"Rodriguez",
"Lewis", "Lee", "Walker", "Hall", "Allen", "Young",
"Hernandez", "King", "Wright", "Lopez", "Hill", "Scott", "Green",
"Adams",
"Baker", "Gonzalez", "Nelson", "Carter", "Mitchell",
"Perez", "Roberts", "Turner", "Phillips", "Campbell", "Parker",
"Evans",
"Edwards", "Collins", "Stewart", "Sanchez", "Morris",
"Rogers", "Reed", "Cook", "Morgan", "Bell", "Murphy", "Bailey",
"Rivera",
"Cooper", "Richardson", "Cox", "Howard", "Ward",
"Torres", "Peterson", "Gray", "Ramirez", "James", "Watson", "Brooks",
"Kelly",
"Sanders", "Price", "Bennett", "Wood", "Barnes", "Ross",
"Henderson", "Cole", "Jenkins", "Perry", "Powell", "Long",
"Patterson",
"Hughes", "Flores", "Washington", "Butler", "Simmons",
"Foster", "Gonzales", "Bryant", "Alexander", "Russell", "Griffin",
"Diaz",
"Hayes" };
    // List of middle initials
    private static final String[] MIDDLE_INITIALS = {"A", "B", "C",
"D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q",
"R",
"S", "T", "U", "V", "W", "X", "Y", "Z"};

    // List of addresses
    private static final String[] ADDRESSES = {"123 Main St", "456
Elm St", "789 Oak St", "1011 Pine St", "1213 Maple St", "1415 Cedar
St",
"1617 Birch St", "1819 Spruce St", "2021 Ash St", "2223
Walnut St", "2425 Chestnut St", "2627 Poplar St", "2829 Pineapple
```

```

St",
    "3031 Orange St", "3233 Banana St", "3435 Grape St",
    "3637 Cherry St", "3839 Lemon St", "4041 Lime St", "4243 Blueberry
St",
    "4445 Raspberry St", "4647 Strawberry St", "4849
Blackberry St", "5051 Raspberry St", "5253 Cranberry St", "5455
Boysenberry St",
    "5657 Gooseberry St", "5859 Elderberry St", "6061
Mulberry St", "6263 Loganberry St", "6465 Dewberry St", "6667
Huckleberry St",
    "6869 Marionberry St", "7071 Tayberry St", "7273
Salmonberry St", "7475 Cloudberry St", "7677 Blackberry St", "7879
Raspberry St",
    "8081 Blueberry St", "8283 Strawberry St", "8485
Raspberry St", "8687 Cranberry St", "8889 Boysenberry St", "9091
Gooseberry St",
    "9293 Elderberry St", "9495 Mulberry St", "9697
Loganberry St", "9899 Dewberry St", "100101 Huckleberry St", "102103
Marionberry St",
    "104105 Tayberry St", "106107 Salmonberry St", "108109
Cloudberry St", "110111 Blackberry St", "112113 Raspberry St",
    "114115 Blueberry St",
    "116117 Strawberry St", "118119 Raspberry St", "120121
Cranberry St", "122123 Boysenberry St", "124125 Gooseberry St",
    "126127 Elderberry St",
    "128129 Mulberry St", "130131 Loganberry St", "132133
Dewberry St", "134135 Huckleberry St", "136137 Marionberry St",
    "138139 Tayberry St",
    "140141 Salmonberry St", "142143 Cloudberry St",
    "144145 Blackberry St", "146147 Raspberry St", "148149
Blueberry St", "150151 Strawberry St", "152153 Raspberry St", "154155
Cranberry St",
    "156157 Boysenberry St", "158159 Gooseberry St", "160161
Elderberry St", "162163 Mulberry St", "164165 Loganberry St", "166167
Dewberry St",
    "168169 Huckleberry St", "170171 Marionberry St", "172173
Tayberry St", "174175 Salmonberry St", "176177 Cloudberry St",
    "178179 Blackberry St",
    "180181 Raspberry St", "182183 Blueberry St", "184185
Strawberry St", "186187 Raspberry St", "188189 Cranberry St", "190191
Boysenberry St",
    "192193 Gooseberry St", "194195 Elderberry St", "196197
Mulberry St", "198199 Loganberry St", "200201 Dewberry St", "202203
Huckleberry St",
    "204205 Marionberry St", "206207 Tayberry St", "208209
Salmonberry St", "210211 Cloudberry St", "212213 Blackberry St",
    "214215 Raspberry St"};

    // List of customer segments
    private static final String[] CUSTOMER_SEGMENTS = {"Silver",
    "Gold", "Platinum", "Premium"};

    // List of marketing preferences
    private static final String[] MARKETING_PREFERENCES = {"Email",
    "Phone", "Mail", "SMS", "Email, Phone", "Email, Mail", "Email, SMS",
    "Phone, Mail", "Phone, SMS", "Mail, SMS", "Email, Phone, Mail",
    "Email, Phone, SMS", "Email, Mail, SMS", "Phone, Mail, SMS", "Email,
    Phone, Mail, SMS"};

    // List of account types

```

```

    private static final String[] ACCOUNT_TYPES = {"Checking",
    "Savings", "Custodial", "Trust"};

    // List of account statuses
    private static final String[] ACCOUNT_STATUSES = {"Active",
    "Inactive", "Closed"};

    // List of bank employee job titles
    private static final String[] JOB_TITLES = {"Teller", "Manager",
    "Engineer", "Analyst", "Technician", "Designer", "Developer",
    "Accountant", "Administrator", "Consultant"};

    // List of bank employee departments
    private static final String[] DEPARTMENTS = {"Accounting",
    "Administration", "Customer Service", "Engineering", "Finance",
    "Human Resources", "IT", "Marketing", "Consulting"};

    // List of transaction types
    private static final String[] TRANSACTION_TYPES = {"Deposit",
    "Withdrawal", "Transfer"};

    private static final Random random = new Random();

    public static void populateDatabase() {
        em = Bank.getEntityManager();

        // Store 10 Customer objects in the database:
        em.getTransaction().begin();
        Name name1 = new Name("John", "Q", "Doe");
        Name name2 = new Name("Jane", "", "Smith");
        Name name3 = new Name("Alice", "R", "Johnson");
        Name name4 = new Name("Michael", "A", "Johnson");
        Name name5 = new Name("Emily", "B", "Williams");
        Name name6 = new Name("David", "C", "Brown");
        Name name7 = new Name("Sarah", "D", "Miller");
        Name name8 = new Name("James", "E", "Davis");
        Name name9 = new Name("Mary", "F", "Wilson");
        Name name10 = new Name("Robert", "G", "Martinez");
        em.persist(name1);
        em.persist(name2);
        em.persist(name3);
        em.persist(name4);
        em.persist(name5);
        em.persist(name6);
        em.persist(name7);
        em.persist(name8);
        em.persist(name9);
        em.persist(name10);
        Customer customer1 = new Customer("123-45-6789", name1, "123
Main St", "john@example.com", new String[]{"1234567890"}, 750,
"Premium", "Email, SMS");
        Customer customer2 = new Customer("987-65-4321", name2, "456
Elm St", "jane@example.com", new String[]{"0987654321"}, 700, "Gold",
"Email");
        Customer customer3 = new Customer("543-21-9876", name3, "789
Oak St", "alice@example.com", new String[]{"5551234567",
"5559876543"}, 800, "Platinum", "SMS");
        Customer customer4 = new Customer("111-22-3333", name4, "555
Pine St", "michael@example.com", new String[]{"1112223333"}, 720,

```

```

"Gold", "Email");
    Customer customer5 = new Customer("444-55-6666", name5, "777
Cedar St", "emily@example.com", new String[]{"4445556666"}, 690,
"Silver", "SMS");
    Customer customer6 = new Customer("777-88-9999", name6, "999
Maple St", "david@example.com", new String[]{"7778889999"}, 780,
"Platinum", "Email, SMS");
    Customer customer7 = new Customer("000-11-2222", name7, "111
Walnut St", "jessica@example.com", new String[]{"0001112222"}, 700,
"Gold", "SMS");
    Customer customer8 = new Customer("333-44-5555", name8, "222
Birch St", "ryan@example.com", new String[]{"3334445555"}, 760,
"Premium", "Email");
    Customer customer9 = new Customer("666-77-8888", name9, "333
Oak St", "sophia@example.com", new String[]{"6667778888"}, 730,
"Gold", "SMS");
    Customer customer10 = new Customer("999-00-1111", name10,
"444 Elm St", "matthew@example.com", new String[]{"9990011111"}, 710,
"Silver", "Email");
    em.persist(customer1);
    em.persist(customer2);
    em.persist(customer3);
    em.persist(customer4);
    em.persist(customer5);
    em.persist(customer6);
    em.persist(customer7);
    em.persist(customer8);
    em.persist(customer9);
    em.persist(customer10);
    em.getTransaction().commit();

    // Store 10 Account objects in the database:
    em.getTransaction().begin();
    Account account1 = new Account("1234567890", "Checking",
1000.00, "Active", 3.1, 100, List.of(customer1, customer3));
    Account account2 = new Account("0987654321", "Savings",
2000.00, "Active", 3.2, 200, List.of(customer2));
    Account account3 = new Account("5551234567", "Checking",
3000.00, "Active", 3.3, 300, List.of(customer3));
    Account account4 = new Account("5559876543", "Savings",
4000.00, "Active", 3.4, 400, List.of(customer3));
    Account account5 = new Account("1112223333", "Checking",
5000.00, "Active", 3.5, 500, List.of(customer4));
    Account account6 = new Account("4445556666", "Savings",
6000.00, "Active", 3.6, 600, List.of(customer5));
    Account account7 = new Account("7778889999", "Checking",
7000.00, "Active", 3.7, 700, List.of(customer6));
    Account account8 = new Account("0001112222", "Savings",
8000.00, "Active", 3.8, 800, List.of(customer7));
    Account account9 = new Account("3334445555", "Checking",
9000.00, "Active", 3.9, 900, List.of(customer8));
    Account account10 = new Account("6667778888", "Savings",
10000.00, "Active", 4.0, 1000, List.of(customer9));
    em.persist(account1);
    em.persist(account2);
    em.persist(account3);
    em.persist(account4);
    em.persist(account5);
    em.persist(account6);
    em.persist(account7);
    em.persist(account8);

```

```

        em.persist(account9);
        em.persist(account10);
        em.getTransaction().commit();

        // Store 10 Transaction objects in the database:
        em.getTransaction().begin();
        Transaction transaction1 = new Transaction("Deposit", 100.00,
new Date(2001, 1, 1, 13, 4, 43), null, account1);
        Transaction transaction2 = new Transaction("Withdrawal",
200.00, new Date(2002, 2, 2, 14, 5, 44), null, account2);
        Transaction transaction3 = new Transaction("Deposit", 300.00,
new Date(2003, 3, 3, 15, 6, 45), null, account3);
        Transaction transaction4 = new Transaction("Withdrawal",
400.00, new Date(2004, 4, 4, 16, 7, 46), null, account4);
        Transaction transaction5 = new Transaction("Deposit", 500.00,
new Date(2005, 5, 5, 17, 8, 47), null, account5);
        Transaction transaction6 = new Transaction("Withdrawal",
600.00, new Date(2006, 6, 6, 18, 9, 48), null, account6);
        Transaction transaction7 = new Transaction("Transfer",
700.00, new Date(2007, 7, 7, 19, 10, 49),
account1.getAccountNumber(), account7);
        Transaction transaction8 = new Transaction("Transfer",
800.00, new Date(2008, 8, 8, 20, 11, 50),
account2.getAccountNumber(), account8);
        Transaction transaction9 = new Transaction("Transfer",
900.00, new Date(2009, 9, 9, 21, 12, 51),
account3.getAccountNumber(), account9);
        Transaction transaction10 = new Transaction("Transfer",
1000.00, new Date(2010, 10, 10, 22, 13, 52),
account4.getAccountNumber(), account10);
        em.persist(transaction1);
        em.persist(transaction2);
        em.persist(transaction3);
        em.persist(transaction4);
        em.persist(transaction5);
        em.persist(transaction6);
        em.persist(transaction7);
        em.persist(transaction8);
        em.persist(transaction9);
        em.persist(transaction10);
        em.getTransaction().commit();

        // Store 10 Branch objects in the database:
        em.getTransaction().begin();
        Branch branch1 = new Branch("Main", "111 Main St", "111-222-
3333");
        Branch branch2 = new Branch("North", "222 North St", "222-
333-4444");
        Branch branch3 = new Branch("South", "333 South St", "333-
444-5555");
        Branch branch4 = new Branch("East", "444 East St", "444-555-
6666");
        Branch branch5 = new Branch("West", "555 West St", "555-666-
7777");
        Branch branch6 = new Branch("Central", "666 Central St",
"666-777-8888");
        Branch branch7 = new Branch("Downtown", "777 Downtown St",
"777-888-9999");
        Branch branch8 = new Branch("Uptown", "888 Uptown St", "888-
999-0000");
        Branch branch9 = new Branch("Midtown", "999 Midtown St",

```

```

"999-000-1111");
    Branch branch10 = new Branch("Suburb", "000 Suburb St", "000-111-2222");
    em.persist(branch1);
    em.persist(branch2);
    em.persist(branch3);
    em.persist(branch4);
    em.persist(branch5);
    em.persist(branch6);
    em.persist(branch7);
    em.persist(branch8);
    em.persist(branch9);
    em.persist(branch10);

    // Store 10 Service objects in the database:
    Service savingsAccountService = new Service("Savings Account", 5.0, "Account", "Basic savings account service", List.of(branch1, branch2, branch3, branch4, branch5));
    Service checkingAccountService = new Service("Checking Account", 7.5, "Account", "Basic checking account service", List.of(branch6, branch7, branch8, branch9, branch10));
    Service loanService = new Service("Loan", 15.0, "Financial", "Loan service for various purposes", List.of(branch1, branch2, branch3, branch4, branch5));
    Service creditCardService = new Service("Credit Card", 10.0, "Financial", "Credit card service with rewards program", List.of(branch6, branch7, branch8, branch9, branch10));
    Service investmentService = new Service("Investment", 20.0, "Financial", "Investment service for stocks, bonds, etc.", List.of(branch1, branch2, branch3, branch4, branch5));
    Service mortgageService = new Service("Mortgage", 25.0, "Financial", "Mortgage service for buying properties", List.of(branch6, branch7, branch8, branch9, branch10));
    Service onlineBankingService = new Service("Online Banking", 3.0, "Digital", "Online banking service for account management", List.of(branch1, branch2, branch3, branch4, branch5));
    Service mobileBankingService = new Service("Mobile Banking", 3.0, "Digital", "Mobile banking service for on-the-go access", List.of(branch6, branch7, branch8, branch9, branch10));
    Service atmService = new Service("ATM", 1.0, "Physical", "ATM service for cash withdrawals and deposits", List.of(branch1, branch2, branch3, branch4, branch5));
    Service wireTransferService = new Service("Wire Transfer", 12.0, "Financial", "Wire transfer service for sending money domestically and internationally", List.of(branch6, branch7, branch8, branch9, branch10));
    em.persist(savingsAccountService);
    em.persist(checkingAccountService);
    em.persist(loanService);
    em.persist(creditCardService);
    em.persist(investmentService);
    em.persist(mortgageService);
    em.persist(onlineBankingService);
    em.persist(mobileBankingService);
    em.persist(atmService);
    em.persist(wireTransferService);
    em.getTransaction().commit();

    // Store 10 Employee objects in the database:
    em.getTransaction().begin();
    // Create 10 names

```



```

        Name name11 = new Name("John", "M", "Doe");
        Name name12 = new Name("Jane", "F", "Smith");
        Name name13 = new Name("James", "M", "Johnson");
        Name name14 = new Name("Jill", "F", "Williams");
        Name name15 = new Name("Jack", "M", "Brown");
        Name name16 = new Name("Jenny", "F", "Davis");
        Name name17 = new Name("Joe", "M", "Miller");
        Name name18 = new Name("Jessica", "F", "Wilson");
        Name name19 = new Name("Jerry", "M", "Moore");
        Name name20 = new Name("Julie", "F", "Taylor");
        em.persist(name11);
        em.persist(name12);
        em.persist(name13);
        em.persist(name14);
        em.persist(name15);
        em.persist(name16);
        em.persist(name17);
        em.persist(name18);
        em.persist(name19);
        em.persist(name20);
        Employee employee1 = new Employee("123-45-6789", name11, "123
Main St", "john@example.com",
        new String[]{"123-456-7890"}, "Manager",
        "Engineering", 75000.0, new Date(), branch1, null);

        Employee employee2 = new Employee("234-56-7890", name12, "456
Elm St", "jane@example.com",
        new String[]{"234-567-8901"}, "Engineer",
        "Engineering", 60000.0, new Date(), branch2, employee1);
        Employee employee3 = new Employee("345-67-8901", name13, "789
Oak St", "bob@example.com",
        new String[]{"345-678-9012"}, "Analyst", "Finance",
        55000.0, new Date(), branch3, employee1);

        Employee employee4 = new Employee("456-78-9012", name14, "101
Pine St", "mary@example.com",
        new String[]{"456-789-0123"}, "Technician", "IT",
        50000.0, new Date(), branch4, employee1);

        Employee employee5 = new Employee("567-89-0123", name15, "202
Cedar St", "david@example.com",
        new String[]{"567-890-1234"}, "Designer",
        "Marketing", 60000.0, new Date(), branch5, employee1);

        Employee employee6 = new Employee("678-90-1234", name16, "303
Maple St", "sarah@example.com",
        new String[]{"678-901-2345"}, "Developer", "IT",
        65000.0, new Date(), branch6, employee1);

        Employee employee7 = new Employee("789-01-2345", name17, "404
Walnut St", "chris@example.com",
        new String[]{"789-012-3456"}, "Accountant",
        "Finance", 58000.0, new Date(), branch7, employee1);

        Employee employee8 = new Employee("890-12-3456", name18, "505
Cherry St", "lisa@example.com",
        new String[]{"890-123-4567"}, "Administrator",
        "Administration", 70000.0, new Date(), branch8, employee1);

        Employee employee9 = new Employee("901-23-4567", name19, "606
Spruce St", "jason@example.com",

```

```

        new String[]{"901-234-5678"}, "Manager", "Marketing",
75000.0, new Date(), branch9, employee1);

    Employee employee10 = new Employee("012-34-5678", name20,
"707 Birch St", "emily@example.com",
        new String[]{"012-345-6789"}, "Consultant",
"Consulting", 80000.0, new Date(), branch10, employee1);
    em.persist(employee1);
    em.persist(employee2);
    em.persist(employee3);
    em.persist(employee4);
    em.persist(employee5);
    em.persist(employee6);
    em.persist(employee7);
    em.persist(employee8);
    em.persist(employee9);
    em.persist(employee10);
    em.getTransaction().commit();

    // 10000 Customers
    for (int i = 1; i <= 10000; i++) {
        em.getTransaction().begin();
        // Select a random first name and last name and middle
initial
        Name name = new
Name(FIRST_NAMES[random.nextInt(FIRST_NAMES.length)],
MIDDLE_INITIALS[random.nextInt(MIDDLE_INITIALS.length)],
LAST_NAMES[random.nextInt(LAST_NAMES.length)]);
        em.persist(name);
        // Generate ssn
        String ssn = String.format("%03d-%02d-%03d",
random.nextInt(1000), i/100, i%1000);
        // Generate random address
        String address =
ADDRESSES[random.nextInt(ADDRESSES.length)];
        // Generate random email
        String email = String.format("%s%d@gmail.com",
name.getFirst().toLowerCase(), i);
        // Generate random phone numbers
        String[] phoneNumbers = new
String[]{String.format("%03d%03d%04d", i%1000, i%1000, i%10000)};
        // Generate random credit score
        int creditScore = 300 + random.nextInt(700);
        // Generate random customer segment
        String segment =
CUSTOMER_SEGMENTS[random.nextInt(CUSTOMER_SEGMENTS.length)];
        // Generate random marketing preferences
        String marketingPreferences =
MARKETING_PREFERENCES[random.nextInt(MARKETING_PREFERENCES.length)];

        Customer customer = new Customer(ssn, name, address,
email, phoneNumbers, creditScore, segment, marketingPreferences);
        em.persist(customer);

        // Generate random account number
        String accountNumber = String.format("%09d", i);
        // Generate random account type
        String accountType =
ACCOUNT_TYPES[random.nextInt(ACCOUNT_TYPES.length)];
        // Generate random balance
        double balance = 1000 + random.nextInt(1000000);

```

```

        // Generate random status
        String status =
ACCOUNT_STATUSES[random.nextInt(ACCOUNT_STATUSES.length)];
        // Generate random interest rate
        double interestRate = ((int)((random.nextDouble() * 5 +
2) * 100)) / 100.0;
        // Generate random minimum balance
        double minimumBalance = 100 + random.nextInt(900);
        List<Customer> customers = new
java.util.ArrayList<>(List.of(customer));
        if(random.nextBoolean()) {
            // Select a random first name and last name and
middle initial
            Name nameS = new
Name(FIRST_NAMES[random.nextInt(FIRST_NAMES.length)],
MIDDLE_INITIALS[random.nextInt(MIDDLE_INITIALS.length)],
LAST_NAMES[random.nextInt(LAST_NAMES.length)]);
            em.persist(nameS);
            // Generate ssn
            String ssn2 = String.format("%03d-%02d-%03d",
random.nextInt(1000), i/100, i%1000);
            // Generate random address
            String address2 =
ADDRESSES[random.nextInt(ADDRESSES.length)];
            // Generate random email
            String email2 = String.format("%s%d@gmail.com",
name.getFirst().toLowerCase(), i);
            // Generate random phone numbers
            String[] phoneNumbers2 = new
String[]{String.format("%03d%03d%04d", i%1000, i%1000, i%10000)};
            // Generate random credit score
            int creditScore2 = 300 + random.nextInt(700);
            // Generate random customer segment
            String segment2 =
CUSTOMER_SEGMENTS[random.nextInt(CUSTOMER_SEGMENTS.length)];
            // Generate random marketing preferences
            String marketingPreferences2 =
MARKETING_PREFERENCES[random.nextInt(MARKETING_PREFERENCES.length)];

            Customer customerS = new Customer(ssn2, nameS,
address2, email2, phoneNumbers2, creditScore2, segment2,
marketingPreferences2);
            em.persist(customerS);
            customers.add(customerS);
        }
        Account account = new Account(accountNumber, accountType,
balance, status, interestRate, minimumBalance, customers);
        em.persist(account);

        // Generate 100 transactions for the account
        for (int j = 1; j <= 10; j++) {
            // Generate random transaction type
            String transactionType =
TRANSACTION_TYPES[random.nextInt(TRANSACTION_TYPES.length)];
            // Generate random transaction amount
            double transactionAmount = 1 + random.nextInt(10000);
            // Generate random transaction date
            Date transactionDate = new Date(2000 +
random.nextInt(23), random.nextInt(12), random.nextInt(28),
random.nextInt(24), random.nextInt(60), random.nextInt(60));
            String destinationAccountNumber = null;

```

```

        if (transactionType.equals("Transfer")) {
            if(i < 10){
                transactionType = "Deposit";
            }
            else {
                destinationAccountNumber =
String.format("%09d", random.nextInt(i - 1) + 1);
            }
        }
        Transaction transaction = new
Transaction(transactionType, transactionAmount, transactionDate,
destinationAccountNumber, account);
        em.persist(transaction);
    }
    em.getTransaction().commit();
}

ArrayList<Branch> branches = new ArrayList<>();
branches.add(branch1);
branches.add(branch2);
branches.add(branch3);
branches.add(branch4);
branches.add(branch5);
branches.add(branch6);
branches.add(branch7);
branches.add(branch8);
branches.add(branch9);
branches.add(branch10);

ArrayList<Employee> managers = new ArrayList<>();
managers.add(employee1);
managers.add(employee9);

// Generate 1000 employees
for (int i = 1; i <= 1000; i++) {
    em.getTransaction().begin();
    // Select a random first name and last name and middle
initial
    Name name = new
Name(FIRST_NAMES[random.nextInt(FIRST_NAMES.length)],
MIDDLE_INITIALS[random.nextInt(MIDDLE_INITIALS.length)],
LAST_NAMES[random.nextInt(LAST_NAMES.length)]);
    em.persist(name);
    // Generate ssn
    String ssn = String.format("%03d-%02d-%03d", (i * 36) %
1000, (i / 2) % 100, (i + 13) % 1000);
    // Generate random address
    String address =
ADDRESSES[random.nextInt(ADDRESSES.length)];
    // Generate random email
    String email = String.format("%s%d@gmail.com",
name.getFirst().toLowerCase(), i);
    // Generate random phone numbers
    String[] phoneNumbers = new
String[]{String.format("%03d%03d%04d", i % 1000, i % 1000, i %
10000)};
    // Generate random job title
    String jobTitle =
JOB_TITLES[random.nextInt(JOB_TITLES.length)];
    // Generate random department
    String department =

```

```

DEPARTMENTS[random.nextInt(DEPARTMENTS.length)];
    // Generate random salary
    double salary = 30000 + random.nextInt(170000);
    // Generate random hire date
    Date hireDate = new Date(2000 + random.nextInt(23),
random.nextInt(12), random.nextInt(28), random.nextInt(24),
random.nextInt(60), random.nextInt(60));
    // Generate random branch
    Branch branch =
branches.get(random.nextInt(branches.size()));
    Employee employee = new Employee(ssn, name, address,
email, phoneNumbers, jobTitle, department, salary, hireDate, branch,
null);

    if (jobTitle.equals("Manager")) {
        managers.add(employee);
    }
    else {
        Employee manager =
managers.get(random.nextInt(managers.size()));
        employee.setManager(manager);
    }
    em.persist(employee);
    em.getTransaction().commit();
}

}

public static void populateDatabase2(){
    EntityManager em = Bank.getEntityManager();
    List<Employee> employees = em.createQuery("SELECT e FROM
Employee e", Employee.class).getResultList();
    for(Employee employee : employees){
        em.getTransaction().begin();
        Long branchId = employee.getBranch().getBranchID();
        Branch branch = em.find(Branch.class, branchId);
        branch.addEmployee(employee);
        em.getTransaction().commit();
    }
}
}

```

Person Class

```
package Model;

import javax.persistence.*;

@Entity
public abstract class Person {

    @Id
    protected String ssn;

    protected Name name;
    protected String address;
    protected String emailAddress;
    protected String[] phoneNumber;

    Person(String ssn, Name name, String address, String
emailAddress, String[] phoneNumber) {
        this.ssn = ssn;
        this.name = name;
        this.address = address;
        this.emailAddress = emailAddress;
        this.phoneNumber = phoneNumber;
    }

    public String getSsn() {
        return ssn;
    }

    public void setSsn(String ssn) {
        this.ssn = ssn;
    }

    public Name getName() {
        return name;
    }

    public void setName(Name name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getEmailAddress() {
        return emailAddress;
    }

    public void setEmailAddress(String emailAddress) {
        this.emailAddress = emailAddress;
    }

    public String[] getPhoneNumber() {
        return phoneNumber;
    }
}
```

```

    public void setPhoneNumber(String[] phoneNumber) {
        this.phoneNumber = phoneNumber;
    }
}

```

Name Class

```

package Model;

import javax.persistence.Entity;

@Entity
public class Name {
    private String first;
    private String middle;
    private String last;

    public Name(String first, String middle, String last) {
        this.first = first;
        this.middle = middle;
        this.last = last;
    }

    public String getFirst() {
        return first;
    }

    public void setFirst(String first) {
        this.first = first;
    }

    public String getMiddle() {
        return middle;
    }

    public void setMiddle(String middle) {
        this.middle = middle;
    }

    public String getLast() {
        return last;
    }

    public void setLast(String last) {
        this.last = last;
    }

    @Override
    public String toString() {
        return "'" + first + " " + middle + " " + last + "'";
    }
}

```

Customer Class

```
package Model;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;
import javax.persistence.Entity;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

@Entity
public class Customer extends Person implements Serializable {
    private int creditScore;
    private String customerSegment;
    private String marketingPreferences;
    private List<Account> accounts = new ArrayList<>();

    public Customer(String ssn, Name name, String address, String
emailAddress, String[] phoneNumber, int creditScore, String
customerSegment, String marketingPreferences) {
        super(ssn, name, address, emailAddress, phoneNumber);
        this.creditScore = creditScore;
        this.customerSegment = customerSegment;
        this.marketingPreferences = marketingPreferences;
        for (Account account: accounts){
            account.addOwner(this);
        }
    }

    public int getCreditScore() {
        return creditScore;
    }

    public void setCreditScore(int creditScore) {
        this.creditScore = creditScore;
    }

    public String getCustomerSegment() {
        return customerSegment;
    }

    public void setCustomerSegment(String customerSegment) {
        this.customerSegment = customerSegment;
    }

    public String getMarketingPreferences() {
        return marketingPreferences;
    }

    public void setMarketingPreferences(String marketingPreferences)
{
        this.marketingPreferences = marketingPreferences;
    }

    public List<Account> getAccounts() {
        return accounts;
    }
}
```



```

    public void setAccounts(List<Account> accounts) {
        this.accounts = accounts;
    }

    public void addAccount(Account account) {
        accounts.add(account);
    }

    public void removeAccount(Account account) {
        accounts.remove(account);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Customer customer = (Customer) o;

        if (creditScore != customer.creditScore) return false;
        if (!customerSegment.equals(customer.customerSegment)) return
false;
        if
(!marketingPreferences.equals(customer.marketingPreferences)) return
false;
        return Objects.equals(accounts, customer.accounts);
    }

    @Override
    public int hashCode() {
        int result = creditScore;
        result = 31 * result + customerSegment.hashCode();
        result = 31 * result + marketingPreferences.hashCode();
        result = 31 * result + (accounts != null ?
accounts.hashCode() : 0);
        return result;
    }

    @Override
    public String toString() {
        return "Customer{" +
            "ssn='" + ssn + '\'' +
            ", name='" + name +
            ", address='" + address + '\'' +
            ", emailAddress='" + emailAddress + '\'' +
            ", phoneNumber='" + Arrays.toString(phoneNumber) +
            ", creditScore='" + creditScore +
            ", customerSegment='" + customerSegment + '\'' +
            ", marketingPreferences='" + marketingPreferences +
'\'' +
            ", accounts='" + accounts +
            "'}";
    }
}

```

Employee Class

```
package Model;

import javax.persistence.Entity;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import java.util.*;

@Entity
public class Employee extends Person{
    private String jobTitle;
    private String departmentName;
    private double salary;
    private Date hireDate;
    private Branch branch;
    private Employee manager;
    private List<Employee> managedEmployees;

    public Employee(String ssn, Name name, String address, String email, String[] phoneNumbers, String jobTitle,
                    String departmentName, double salary, Date hireDate, Branch branch, Employee manager,
                    List<Employee> managedEmployees) {
        super(ssn, name, address, email, phoneNumbers);
        this.jobTitle = jobTitle;
        this.departmentName = departmentName;
        this.salary = salary;
        this.hireDate = hireDate;
        this.branch = branch;
        this.manager = manager;
        this.managedEmployees = managedEmployees;
        for (Employee employee: managedEmployees){
            employee.setManager(this);
        }
        manager.addManagedEmployee(this);
    }

    public Employee(String ssn, Name name, String address, String email, String[] phoneNumbers, String jobTitle,
                    String departmentName, double salary, Date hireDate, Branch branch, Employee manager) {
        this(ssn, name, address, email, phoneNumbers, jobTitle,
            departmentName, salary, hireDate, branch,
            manager, new ArrayList<Employee>());
    }

    public String getJobTitle() {
        return jobTitle;
    }

    public void setJobTitle(String jobTitle) {
        this.jobTitle = jobTitle;
    }

    public String getDepartmentName() {
        return departmentName;
    }
}
```

```

    public void setDepartmentName(String departmentName) {
        this.departmentName = departmentName;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public Date getHireDate() {
        return hireDate;
    }

    public void setHireDate(Date hireDate) {
        this.hireDate = hireDate;
    }

    public Branch getBranch() {
        return branch;
    }

    public void setBranch(Branch branch) {
        this.branch = branch;
    }

    public Employee getManager() {
        return manager;
    }

    public void setManager(Employee manager) {
        this.manager = manager;
    }

    public List<Employee> getManagedEmployees() {
        return managedEmployees;
    }

    public void setManagedEmployees(List<Employee> managedEmployees)
{
        this.managedEmployees = managedEmployees;
    }

    public void addManagedEmployee(Employee employee) {
        managedEmployees.add(employee);
    }

    public void removeManagedEmployee(Employee employee) {
        managedEmployees.remove(employee);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Employee employee = (Employee) o;

        if (Double.compare(employee.salary, salary) != 0) return

```

```

false;
    if (!jobTitle.equals(employee.jobTitle)) return false;
    if (!departmentName.equals(employee.departmentName)) return
false;
    if (!hireDate.equals(employee.hireDate)) return false;
    if (!branch.equals(employee.branch)) return false;
    if (!Objects.equals(manager, employee.manager)) return false;
    return managedEmployees.equals(employee.managedEmployees);
}

@Override
public int hashCode() {
    int result;
    long temp;
    result = jobTitle.hashCode();
    result = 31 * result + departmentName.hashCode();
    temp = Double.doubleToLongBits(salary);
    result = 31 * result + (int) (temp ^ (temp >>> 32));
    result = 31 * result + hireDate.hashCode();
    result = 31 * result + branch.hashCode();
    result = 31 * result + (manager != null ? manager.hashCode()
: 0);
    result = 31 * result + managedEmployees.hashCode();
    return result;
}

@Override
public String toString() {
    return "Employee{" +
        "ssn='" + ssn + '\'' +
        ", name=" + name +
        ", address='" + address + '\'' +
        ", emailAddress='" + emailAddress + '\'' +
        ", phoneNumber=" + Arrays.toString(phoneNumber) +
        ", jobTitle='" + jobTitle + '\'' +
        ", departmentName='" + departmentName + '\'' +
        ", salary=" + salary +
        ", hireDate=" + hireDate +
        ", branch=" + branch +
        ", manager=" + manager +
        ", managedEmployees=" + managedEmployees +
        '}';
}
}

```

Account Class

```
package Model;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import javax.persistence.*;

@Entity
public class Account {
    @Id
    private String accountNumber;
    private String accountType;
    private double balance;
    private String accountStatus;
    private double interestRate;
    private double minimumBalance;
    private List<Customer> owners;
    private List<Transaction> transactions = new ArrayList<>();

    public Account(String accountNumber, String accountType, double
balance, String accountStatus,
                    double interestRate, double minimumBalance,
List<Customer> owners) {
        this.accountNumber = accountNumber;
        this.accountType = accountType;
        this.balance = balance;
        this.accountStatus = accountStatus;
        this.interestRate = interestRate;
        this.minimumBalance = minimumBalance;
        this.owners = owners;
        for(Customer owner : owners) {
            owner.addAccount(this);
        }
        for (Transaction transaction : transactions) {
            transaction.setSourceAccount(this);
        }
    }

    public void setAccountNumber(String accountNumber) {
        this.accountNumber = accountNumber;
    }

    public String getAccountNumber() {
        return accountNumber;
    }

    public void setAccountType(String accountType) {
        this.accountType = accountType;
    }

    public String getAccountType() {
        return accountType;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }

    public double getBalance() {
```

```

        return balance;
    }

    public void setAccountStatus(String accountStatus) {
        this.accountStatus = accountStatus;
    }

    public String getAccountStatus() {
        return accountStatus;
    }

    public void setInterestRate(double interestRate) {
        this.interestRate = interestRate;
    }

    public double getInterestRate() {
        return interestRate;
    }

    public void setMinimumBalance(double minimumBalance) {
        this.minimumBalance = minimumBalance;
    }

    public double getMinimumBalance() {
        return minimumBalance;
    }

    public void setOwners(List<Customer> owners) {
        this.owners = owners;
    }

    public List<Customer> getOwners() {
        return owners;
    }

    public void setTransactions(List<Transaction> transactions) {
        this.transactions = transactions;
    }

    public List<Transaction> getTransactions() {
        return transactions;
    }

    public void addTransaction(Transaction transaction) {
        transactions.add(transaction);
        if (Objects.equals(transaction.getTransactionType(),
"Deposit")) {
            deposit(transaction.getAmount());
        } else if (Objects.equals(transaction.getTransactionType(),
"Withdrawal")) {
            withdraw(transaction.getAmount());
        } else if (Objects.equals(transaction.getTransactionType(),
"Transfer")) {
            transfer(transaction.getAmount(),
transaction.getDestinationAccount());
        }
    }

    public void removeTransaction(Transaction transaction) {
        transactions.remove(transaction);
    }

```

```

public void addOwner(Customer owner) {
    owners.add(owner);
}

public void removeOwner(Customer owner) {
    owners.remove(owner);
}

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

public void withdraw(double amount) {
    balance -= amount;
}

public void transfer(double amount, Account destinationAccount) {
    balance -= amount;
    destinationAccount.deposit(amount);
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Account account = (Account) o;

    if (Double.compare(account.balance, balance) != 0) return
false;
    if (Double.compare(account.interestRate, interestRate) != 0)
return false;
    if (Double.compare(account.minimumBalance, minimumBalance) !=
0) return false;
    if (!accountNumber.equals(account.accountNumber)) return
false;
    if (!accountType.equals(account.accountType)) return false;
    if (!accountStatus.equals(account.accountStatus)) return
false;
    if (!owners.equals(account.owners)) return false;
    return transactions.equals(account.transactions);
}

@Override
public int hashCode() {
    int result;
    long temp;
    result = accountNumber.hashCode();
    result = 31 * result + accountType.hashCode();
    temp = Double.doubleToLongBits(balance);
    result = 31 * result + (int) (temp ^ (temp >>> 32));
    result = 31 * result + accountStatus.hashCode();
    temp = Double.doubleToLongBits(interestRate);
    result = 31 * result + (int) (temp ^ (temp >>> 32));
    temp = Double.doubleToLongBits(minimumBalance);
    result = 31 * result + (int) (temp ^ (temp >>> 32));
    result = 31 * result + owners.hashCode();
    result = 31 * result + transactions.hashCode();
    return result;
}

```

```

@Override
public String toString() {
    return "Account{" +
        "accountNumber=" + accountNumber + '\'' +
        ", accountType=" + accountType + '\'' +
        ", balance=" + balance +
        ", accountStatus=" + accountStatus + '\'' +
        ", interestRate=" + interestRate +
        ", minimumBalance=" + minimumBalance +
        '}';
}
}

```

Transaction Class

```

package Model;

import javax.persistence.*;
import java.util.Date;
import java.util.Objects;

import App.Bank;

@Entity
public class Transaction {
    @Id @GeneratedValue
    private Long transactionID;
    private String transactionType;
    private double amount;
    private Date date;
    private String destinationAccountNumber;
    private Account sourceAccount;

    public Transaction(String transactionType, double amount, Date
date, String destinationAccountNumber, Account sourceAccount) {
        this.transactionType = transactionType;
        this.amount = amount;
        this.date = date;
        this.destinationAccountNumber = destinationAccountNumber;
        this.sourceAccount = sourceAccount;
        if (sourceAccount != null) {
            sourceAccount.addTransaction(this);
        }
        if (destinationAccountNumber != null) {
            getDestinationAccount().deposit(amount);
        }
    }

    public Transaction(String transactionType, double amount, Date
date, Account sourceAccount){
        this(transactionType, amount, date, null, sourceAccount);
    }

    public Long getTransactionID() {
        return transactionID;
    }

    public void setTransactionID(Long transactionID) {

```



```

        this.transactionID = transactionID;
    }

    public String getTransactionType() {
        return transactionType;
    }

    public void setTransactionType(String transactionType) {
        this.transactionType = transactionType;
    }

    public double getAmount() {
        return amount;
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String getDestinationAccountNumber() {
        return destinationAccountNumber;
    }

    public void setDestinationAccountNumber(String
destinationAccountNumber) {
        this.destinationAccountNumber = destinationAccountNumber;
    }

    public Account getSourceAccount() {
        return sourceAccount;
    }

    public void setSourceAccount(Account sourceAccount) {
        this.sourceAccount = sourceAccount;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Transaction that = (Transaction) o;

        if (Double.compare(that.amount, amount) != 0) return false;
        if (!Objects.equals(transactionID, that.transactionID))
            return false;
        if (!Objects.equals(transactionType, that.transactionType))
            return false;
        if (!Objects.equals(date, that.date)) return false;
        if (!Objects.equals(destinationAccountNumber,
that.destinationAccountNumber))
            return false;
        return Objects.equals(sourceAccount, that.sourceAccount);
    }

```

```

    }

    @Override
    public int hashCode() {
        int result;
        long temp;
        result = transactionID != null ? transactionID.hashCode() :
0;
        result = 31 * result + (transactionType != null ?
transactionType.hashCode() : 0);
        temp = Double.doubleToLongBits(amount);
        result = 31 * result + (int) (temp ^ (temp >>> 32));
        result = 31 * result + (date != null ? date.hashCode() : 0);
        result = 31 * result + (destinationAccountNumber != null ?
destinationAccountNumber.hashCode() : 0);
        result = 31 * result + (sourceAccount != null ?
sourceAccount.hashCode() : 0);
        return result;
    }

    @Override
    public String toString() {
        return "Transaction{" +
            "transactionID='" + transactionID + '\'' +
            ", transactionType='" + transactionType + '\'' +
            ", amount='" + amount +
            ", date='" + date +
            ", destinationAccountNumber='" +
destinationAccountNumber + '\'' +
            ", sourceAccount='" + sourceAccount +
            '\'';
    }

    public Account getDestinationAccount() {
        // Find the account with the destination account number
        EntityManager em = Bank.getEntityManager();
        return em.find(Account.class, destinationAccountNumber);
    }
}

```

Branch Class

```

package Model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import java.util.ArrayList;
import java.util.List;

@Entity
public class Branch {
    @Id @GeneratedValue
    private Long branchID;
    private String branchName;
    private String branchAddress;
    private String openingHours;
    private List<Employee> employees;
    private List<Service> services;
}

```

```

    public Branch(String branchName, String branchAddress, String
openingHours, List<Employee> employees, List<Service> services) {
        this.branchName = branchName;
        this.branchAddress = branchAddress;
        this.openingHours = openingHours;
        this.employees = employees;
        this.services = services;
        for(Employee employee : employees) {
            employee.setBranch(this);
        }
        for(Service service : services) {
            service.addBranch(this);
        }
    }

    public Branch(String branchName, String branchAddress, String
openingHours) {
        this(branchName, branchAddress, openingHours, new
ArrayList<Employee>(), new ArrayList<Service>());
    }

    public Branch(String branchName, String branchAddress, String
openingHours, List<Service> services) {
        this(branchName, branchAddress, openingHours, new
ArrayList<Employee>(), services);
    }

    public Long getBranchID() {
        return branchID;
    }

    public void setBranchID(Long branchID) {
        this.branchID = branchID;
    }

    public String getBranchName() {
        return branchName;
    }

    public void setBranchName(String branchName) {
        this.branchName = branchName;
    }

    public String getBranchAddress() {
        return branchAddress;
    }

    public void setBranchAddress(String branchAddress) {
        this.branchAddress = branchAddress;
    }

    public String getOpeningHours() {
        return openingHours;
    }

    public void setOpeningHours(String openingHours) {
        this.openingHours = openingHours;
    }

    public List<Employee> getEmployees() {

```

```

        return employees;
    }

    public void setEmployees(List<Employee> employees) {
        this.employees = employees;
    }

    public List<Service> getServices() {
        return services;
    }

    public void setServices(List<Service> services) {
        this.services = services;
    }

    public void addEmployee(Employee employee) {
        employees.add(employee);
    }

    public void removeEmployee(Employee employee) {
        employees.remove(employee);
    }

    public void addService(Service service) {
        services.add(service);
    }

    public void removeService(Service service) {
        services.remove(service);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Branch)) return false;
        Branch branch = (Branch) o;
        return getBranchID().equals(branch.getBranchID());
    }

    @Override
    public int hashCode() {
        return branchID.hashCode();
    }

    @Override
    public String toString() {
        return "Branch{" +
            "branchID=" + branchID +
            ", branchName='" + branchName + '\'' +
            ", branchAddress='" + branchAddress + '\'' +
            ", openingHours='" + openingHours + '\'' +
            ", services=" + services +
            '}';
    }
}

```

Service Class

```
package Model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import java.util.ArrayList;
import java.util.List;

@Entity
public class Service {
    @Id @GeneratedValue
    private Long serviceID;
    private String serviceName;
    private double serviceFees;
    private String type;
    private String details;
    private List<Branch> managingBranches;

    public Service(String serviceName, double serviceFees, String
type, String details, List<Branch> managingBranches) {
        this.serviceName = serviceName;
        this.serviceFees = serviceFees;
        this.type = type;
        this.details = details;
        this.managingBranches = managingBranches;
        for(Branch branch : managingBranches) {
            branch.addService(this);
        }
    }

    public Service(String serviceName, double serviceFees, String
type, String details) {
        this(serviceName, serviceFees, type, details, new
ArrayList<Branch>());
    }

    public Long getServiceID() {
        return serviceID;
    }

    public void setServiceID(Long serviceID) {
        this.serviceID = serviceID;
    }

    public String getServiceName() {
        return serviceName;
    }

    public void setServiceName(String serviceName) {
        this.serviceName = serviceName;
    }

    public double getServiceFees() {
        return serviceFees;
    }

    public void setServiceFees(double serviceFees) {
        this.serviceFees = serviceFees;
    }
}
```

```

public String getType() {
    return type;
}

public void setType(String type) {
    this.type = type;
}

public String getDetails() {
    return details;
}

public void setDetails(String details) {
    this.details = details;
}

public List<Branch> getManagingBranches() {
    return managingBranches;
}

public void setManagingBranches(List<Branch> managingBranches) {
    this.managingBranches = managingBranches;
}

public void addBranch(Branch branch) {
    managingBranches.add(branch);
}

public void removeBranch(Branch branch) {
    managingBranches.remove(branch);
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Service service = (Service) o;

    if (Double.compare(service.serviceFees, serviceFees) != 0)
return false;
    if (!serviceID.equals(service.serviceID)) return false;
    if (!serviceName.equals(service.serviceName)) return false;
    if (!type.equals(service.type)) return false;
    return details.equals(service.details);
}

@Override
public int hashCode() {
    int result;
    long temp;
    result = serviceID.hashCode();
    result = 31 * result + serviceName.hashCode();
    temp = Double.doubleToLongBits(serviceFees);
    result = 31 * result + (int) (temp ^ (temp >>> 32));
    result = 31 * result + type.hashCode();
    result = 31 * result + details.hashCode();
    return result;
}

```

```
@Override
public String toString() {
    return "Service{" +
        "serviceID=" + serviceID +
        ", serviceName=" + serviceName + '\'' +
        ", serviceFees=" + serviceFees +
        ", type=" + type + '\'' +
        ", details=" + details + '\'' +
        '}';
}
```

GUI Classes and dependencies can be found in the github repository, link in the first page, or the submitted zip file.