ASSIGNMENT

SUBMITTED BY: Fahimunnisha A

TOPIC: Banking System

Tasks 1: Database Design:

1.Create the database named "HMBank"

```
create database HMBANK;
use HMBANK;
```

2. Define the schema for the Customers, Accounts, and Transactions tables based on the provided schema.

Customers Table:

Column Name	Data Type	Constraints
CustomerID	INT	PRIMARY KEY, AUTO_INCREMENT
FirstName	VARCHAR(50)	NOT NULL
LastName	VARCHAR(50)	NOT NULL
DateOfBirth	DATE	NOT NULL
Email	VARCHAR(100)	UNIQUE
Phone	VARCHAR(15)	
Address	VARCHAR(255)	

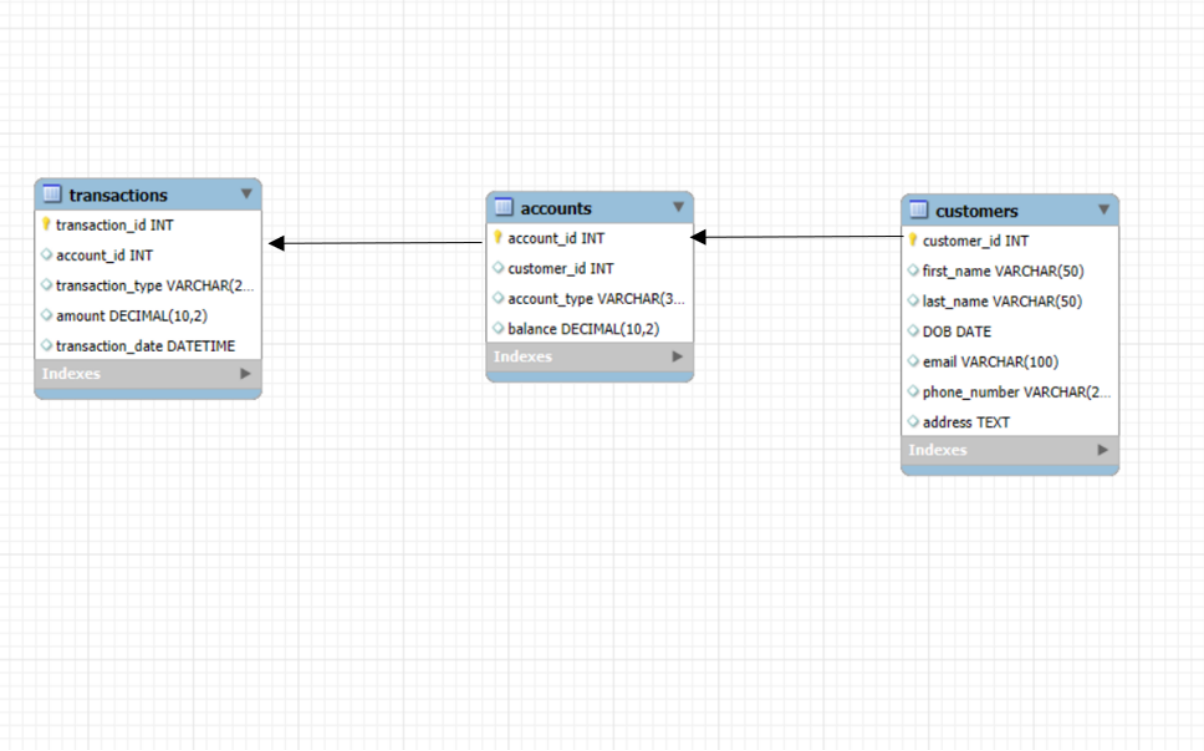
Accounts Table:

Column Name	Data type	Constraints
AccountID	INT	PRIMARY KEY, AUTO_INCREMENT
CustomerID	INT	FOREIGN KEY → Customers(CustomerID)
AccountType	VARCHAR(20)	NOT NULL
Balance	DECIMAL(12,2)	DEFAULT 0.00
OpenDate	DATE	NOT NULL

Transactions table:

Column Name	Data type	Constraints
TransactionID	INT	PRIMARY KEY, AUTO_INCREMENT
AccountID	INT	FOREIGN KEY → Accounts(AccountID)
TransactionType	VARCHAR(20)	NOT NULL (Deposit, Withdraw, etc.)
Amount	DECIMAL(12, 2)	NOT NULL
TransactionDate	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP

3. Create an ERD (Entity Relationship Diagram) for the database.



4. Create appropriate Primary Key and Foreign Key constraints for referential integrity.

```
CREATE TABLE Transactions (
    transaction_id INT PRIMARY KEY,
    account_id INT,
    transaction_type VARCHAR(20),
    amount DECIMAL(10,2),
    transaction_date DATETIME,
    FOREIGN KEY (account_id) REFERENCES Accounts(account_id)
);
```

5. Write SQL scripts to create the mentioned tables with appropriate data types, constraints, and relationships.

- Customers
- Accounts
- Transactions

```
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    DOB DATE,
    email VARCHAR(100),
    phone_number VARCHAR(20),
    address TEXT
);
```

```sql
CREATE TABLE Accounts (
    account_id INT PRIMARY KEY,
    customer_id INT,
    account_type VARCHAR(30),
    balance DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

```sql
CREATE TABLE Transactions (
    transaction_id INT PRIMARY KEY,
    account_id INT,
    transaction_type VARCHAR(20),
    amount DECIMAL(10,2),
    transaction_date DATETIME,
    FOREIGN KEY (account_id) REFERENCES Accounts(account_id)
);
```

Tasks 2: Select, Where, Between, AND, LIKE:

1. Insert at least 10 sample records into each of the following tables.

- Customers
- Accounts
- Transactions

```sql
INSERT INTO Customers VALUES
(1, 'John', 'Doe', '1985-06-15', 'john.doe@example.com', '98765432
(2, 'Jane', 'Smith', '1990-11-20', 'jane.smith@example.com', '9876
(3, 'Alice', 'Johnson', '1975-09-30', 'alice.j@example.com', '9876
(4, 'Bob', 'Brown', '2000-01-01', 'bob.b@example.com', '9876543213
(5, 'Charlie', 'Davis', '1995-03-12', 'charlie.d@example.com', '98
(6, 'Emily', 'Clark', '1988-07-08', 'emily.c@example.com', '987654
```

Result Grid | Filter Rows: | Edit: | Export/Import:

customer_id	first_name	last_name	DOB	email	phone_number
1	John	Doe	1985-06-15	john.doe@example.com	9876543210
2	Jane	Smith	1990-11-20	jane.smith@example.com	9876543211
3	Alice	Johnson	1975-09-30	alice.j@example.com	9876543212
4	Bob	Brown	2000-01-01	bob.b@example.com	9876543213
5	Charlie	Davis	1995-03-12	charlie.d@example.com	9876543214
6	Emily	Clark	1988-07-08	emily.c@example.com	9876543215
7	Frank	Wright	1992-05-05	frank.w@example.com	9876543216
8	Grace	Lee	1983-12-25	grace.l@example.com	9876543217
9	Hank	Green	1970-10-10	hank.g@example.com	9876543218
10	Ivy	King	1998-04-18	ivy.k@example.com	9876543219
NULL	NULL	NULL	NULL	NULL	NULL

```sql
INSERT INTO Accounts VALUES
(101, 1, 'savings', 15000.00),
(102, 2, 'current', 25000.50),
(103, 3, 'zero_balance', 500.00),
(104, 4, 'savings', 10000.00),
(105, 5, 'current', 20000.75),
```

Result Grid | Filter Rows: | Edit: | Export/Import:

account_id	customer_id	account_type	balance
101	1	savings	15000.00
102	2	current	25000.50
103	3	zero_balance	500.00
104	4	savings	10000.00
105	5	current	20000.75
106	6	savings	30000.00
107	7	zero_balance	0.00
108	8	savings	12000.00
109	9	current	8000.80
110	10	savings	5000.00
NULL	NULL	NULL	NULL

```
52 •    INSERT INTO Transactions VALUES
53      (1001, 101, 'deposit', 5000.00, '2025-04-01 10:00:00'),
54      (1002, 102, 'withdrawal', 2000.00, '2025-04-02 11:30:00'),
55      (1003, 103, 'deposit', 500.00, '2025-04-03 09:15:00'),
56      (1004, 104, 'transfer', 1000.00, '2025-04-04 14:00:00'),
57      (1005, 105, 'deposit', 2500.00, '2025-04-05 12:45:00'),
58      (1006, 106, 'withdrawal', 1500.00, '2025-04-06 16:20:00'),
```

Result Grid | Filter Rows: | Edit: | Export/Import:

transaction_id	account_id	transaction_type	amount	transaction_date
1001	101	deposit	5000.00	2025-04-01 10:00:00
1002	102	withdrawal	2000.00	2025-04-02 11:30:00
1003	103	deposit	500.00	2025-04-03 09:15:00
1004	104	transfer	1000.00	2025-04-04 14:00:00
1005	105	deposit	2500.00	2025-04-05 12:45:00
1006	106	withdrawal	1500.00	2025-04-06 16:20:00
1007	107	deposit	100.00	2025-04-07 08:10:00
1008	108	transfer	2000.00	2025-04-07 18:25:00
1009	109	deposit	3000.00	2025-04-08 10:50:00
1010	110	withdrawal	500.00	2025-04-08 13:15:00
NULL	NULL	NULL	NULL	NULL

2. Write SQL queries for the following tasks:

1. Write a SQL query to retrieve the name, account type and email of all customers.

```
select *from Transactions;

SELECT
    c.first_name,
    c.last_name,
    a.account_type,
    c.email
FROM
    Customers c
JOIN
    Accounts a ON c.customer_id = a.customer_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Cor

first_name	last_name	account_type	email
John	Doe	savings	john.doe@example.com
Jane	Smith	current	jane.smith@example.com
Alice	Johnson	zero_balance	alice.j@example.com
Bob	Brown	savings	bob.b@example.com
Charlie	Davis	current	charlie.d@example.com
Emily	Clark	savings	emily.c@example.com
Frank	Wright	zero_balance	frank.w@example.com
Grace	Lee	savings	grace.l@example.com
Hank	Green	current	hank.g@example.com
Ivy	King	savings	ivy.k@example.com

2. Write a SQL query to list all transaction corresponding customer.

```sql
SELECT
    c.first_name,
    c.last_name,
    t.transaction_id,
    t.transaction_type,
    t.amount,
    t.transaction_date
FROM
    Customers c
JOIN
    Accounts a ON c.customer_id = a.customer_id
JOIN
    Transactions t ON a.account_id = t.account_id;
```

first_name	last_name	transaction_id	transaction_type	amount	transaction_date
John	Doe	1001	deposit	5000.00	2025-04-01 10:00:00
Jane	Smith	1002	withdrawal	2000.00	2025-04-02 11:30:00
Alice	Johnson	1003	deposit	500.00	2025-04-03 09:15:00
Bob	Brown	1004	transfer	1000.00	2025-04-04 14:00:00
Charlie	Davis	1005	deposit	2500.00	2025-04-05 12:45:00
Emily	Clark	1006	withdrawal	1500.00	2025-04-06 16:20:00
Frank	Wright	1007	deposit	100.00	2025-04-07 08:10:00
Grace	Lee	1008	transfer	2000.00	2025-04-07 18:25:00
Hank	Green	1009	deposit	3000.00	2025-04-08 10:50:00
Ivy	King	1010	withdrawal	500.00	2025-04-08 13:15:00

3. Write a SQL query to increase the balance of a specific account by a certain amount.

```
90 •  UPDATE Accounts
91    SET balance = balance + 500
92    WHERE account_id = 101;
93 •  select balance from accounts;
94
```

Result Grid | Filter Rows: | Export: | Wrap

balance
16500.00
25000.50
500.00
10000.00
20000.75
30000.00
0.00
12000.00
8000.80
5000.00

4. Write a SQL query to Combine first and last names of customers as a full_name.

```
95 •  SELECT
96        CONCAT(first_name, ' ', last_name) AS full_name
97    FROM
98        Customers;
99
100
```

Result Grid | Filter Rows: | Export: | Wrap Cell Conten

full_name
John Doe
Jane Smith
Alice Johnson
Bob Brown
Charlie Davis
Emily Clark
Frank Wright
Grace Lee
Hank Green
Ivy King

5. Write a SQL query to remove accounts with a balance of zero where the account type is savings.

```
DELETE FROM Accounts
WHERE balance = 0 AND account_type = 'savings';
```

6. Write a SQL query to Find customers living in a specific city.

```
105    SELECT * FROM Customers
106    WHERE address LIKE '%Chennai%';
```

Result Grid | Filter Rows: | Edit: | Export/Import:

customer_id	first_name	last_name	DOB	email	phone_number	address
NULL	NULL	NULL	NULL	NULL	NULL	NULL

7. Write a SQL query to Get the account balance for a specific account.

```
108    SELECT balance
109    FROM Accounts
110    WHERE account_id = 101;
111
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

balance
16500.00

8. Write a SQL query to List all current accounts with a balance greater than $1,000.

```
112 •   SELECT * FROM Accounts
113     WHERE account_type = 'current' AND balance > 1000;
114
```

Result Grid | Filter Rows: | Edit: | Export/Import:

account_id	customer_id	account_type	balance
102	2	current	25000.50
105	5	current	20000.75
109	9	current	8000.80
NULL	NULL	NULL	NULL

9. Write a SQL query to Retrieve all transactions for a specific account.

```
115 •   SELECT *FROM Transactions
116     WHERE account_id = 101
117     ;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

transaction_id	account_id	transaction_type	amount	transaction_date
1001	101	deposit	5000.00	2025-04-01 10:00:00
NULL	NULL	NULL	NULL	NULL

10. Write a SQL query to Calculate the interest accrued on savings accounts based on a given interest rate.

```
120 •   SELECT account_id, balance, balance * 0.04 AS interest
121     FROM Accounts
122     WHERE account_type = 'savings';
123
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_id	balance	interest
101	16500.00	660.0000
104	10000.00	400.0000
106	30000.00	1200.0000
108	12000.00	480.0000
110	5000.00	200.0000

11. Write a SQL query to Identify accounts where the balance is less than a specified overdraft limit.

```
124 •   SELECT *FROM Accounts
125     WHERE balance < 500;
126
```

Result Grid | Filter Rows: | Edit:

account_id	customer_id	account_type	balance
107	7	zero_balance	0.00
NULL	NULL	NULL	NULL

12. Write a SQL query to Find customers not living in a specific city.

```
127 •   SELECT *FROM Customers
128     WHERE address NOT LIKE '%Chennai%';
129
```

Result Grid | Filter Rows: | Edit: | Export/Import:

customer_id	first_name	last_name	DOB	email	phone_numb
1	John	Doe	1985-06-15	john.doe@example.com	9876543210
2	Jane	Smith	1990-11-20	jane.smith@example.com	9876543211
3	Alice	Johnson	1975-09-30	alice.j@example.com	9876543212
4	Bob	Brown	2000-01-01	bob.b@example.com	9876543213
5	Charlie	Davis	1995-03-12	charlie.d@example.com	9876543214
6	Emily	Clark	1988-07-08	emily.c@example.com	9876543215
7	Frank	Wright	1992-05-05	frank.w@example.com	9876543216
8	Grace	Lee	1983-12-25	grace.l@example.com	9876543217
9	Hank	Green	1970-10-10	hank.g@example.com	9876543218
10	Ivy	King	1998-04-18	ivy.k@example.com	9876543219

Tasks 3: Aggregate functions, Having, Order By, Group By and Joins:

1. Write a SQL query to Find the average account balance for all customers.

```
130 •   SELECT AVG(balance) AS average_balance
131     FROM Accounts;
132
```

Result Grid | Filter Rows: | Export: | Wrap Cell Conter

average_balance
12700.205000

2. Write a SQL query to Retrieve the top 10 highest account balances.

```
133 •  SELECT *FROM Accounts
134    ORDER BY balance DESC
135    LIMIT 10;
136
```

Result Grid | Filter Rows: | Edit: | Expor

account_id	customer_id	account_type	balance
106	6	savings	30000.00
102	2	current	25000.50
105	5	current	20000.75
101	1	savings	16500.00
108	8	savings	12000.00
104	4	savings	10000.00
109	9	current	8000.80
110	10	savings	5000.00
103	3	zero_balance	500.00
107	7	zero_balance	0.00
NULL	NULL	NULL	NULL

3. Write a SQL query to Calculate Total Deposits for All Customers in specific date.

```
137 •  SELECT SUM(amount) AS total_deposits
138    FROM Transactions
139    WHERE transaction_type = 'deposit'
140      AND DATE(transaction_date) = '2025-04-07';
141
```

Result Grid | Filter Rows: | Edit: | Export/Import:

account_id	customer_id	account_type	balance
106	6	savings	30000.00
102	2	current	25000.50
105	5	current	20000.75
101	1	savings	16500.00
108	8	savings	12000.00
104	4	savings	10000.00
109	9	current	8000.80
110	10	savings	5000.00
103	3	zero_balance	500.00
107	7	zero_balance	0.00
NULL	NULL	NULL	NULL

4. Write a SQL query to Find the Oldest and Newest Customers.

```
142 •  SELECT * FROM Customers
143    ORDER BY DOB ASC
144    LIMIT 1;
145 •  SELECT * FROM Customers
146    ORDER BY DOB DESC
147    LIMIT 1;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

customer_id	first_name	last_name	DOB	email	phone_number	add
4	Bob	Brown	2000-01-01	bob.b@example.com	9876543213	101
NULL	NULL	NULL	NULL	NULL	NULL	NULL

5. Write a SQL query to Retrieve transaction details along with the account type.

```
149 •  SELECT t.*, a.account_type
150    FROM Transactions t
151    JOIN Accounts a ON t.account_id = a.account_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

transaction_id	account_id	transaction_type	amount	transaction_date	account_type
1001	101	deposit	5000.00	2025-04-01 10:00:00	savings
1002	102	withdrawal	2000.00	2025-04-02 11:30:00	current
1003	103	deposit	500.00	2025-04-03 09:15:00	zero_balance
1004	104	transfer	1000.00	2025-04-04 14:00:00	savings
1005	105	deposit	2500.00	2025-04-05 12:45:00	current
1006	106	withdrawal	1500.00	2025-04-06 16:20:00	savings
1007	107	deposit	100.00	2025-04-07 08:10:00	zero_balance
1008	108	transfer	2000.00	2025-04-07 18:25:00	savings
1009	109	deposit	3000.00	2025-04-08 10:50:00	current
1010	110	withdrawal	500.00	2025-04-08 13:15:00	savings

6. Write a SQL query to Get a list of customers along with their account details.

```
155 •   SELECT c.*, a.account_id, a.account_type, a.balance
156     FROM Customers c
157     JOIN Accounts a ON c.customer_id = a.customer_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	first_name	last_name	DOB	email	phone_numbe
1	John	Doe	1985-06-15	john.doe@example.com	9876543210
2	Jane	Smith	1990-11-20	jane.smith@example.com	9876543211
3	Alice	Johnson	1975-09-30	alice.j@example.com	9876543212
4	Bob	Brown	2000-01-01	bob.b@example.com	9876543213
5	Charlie	Davis	1995-03-12	charlie.d@example.com	9876543214
6	Emily	Clark	1988-07-08	emily.c@example.com	9876543215
7	Frank	Wright	1992-05-05	frank.w@example.com	9876543216
8	Grace	Lee	1983-12-25	grace.l@example.com	9876543217
9	Hank	Green	1970-10-10	hank.g@example.com	9876543218
10	Ivy	King	1998-04-18	ivy.k@example.com	9876543219

7. Write a SQL query to Retrieve transaction details along with customer information for a specific account.

```
159 •   SELECT t.*, c.first_name, c.last_name, c.email
160     FROM Transactions t
161     JOIN Accounts a ON t.account_id = a.account_id
162     JOIN Customers c ON a.customer_id = c.customer_id
163     WHERE t.account_id = 101;
164
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

transaction_id	account_id	transaction_type	amount	transaction_date	first_name	la
1001	101	deposit	5000.00	2025-04-01 10:00:00	John	Do

8. Write a SQL query to Identify customers who have more than one account.

```
165 •   SELECT customer_id, COUNT(*) AS account_count
166     FROM Accounts
167     GROUP BY customer_id
168     HAVING COUNT(*) > 1;
169
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	account_count

9. Write a SQL query to Calculate the difference in transaction amounts between deposits and withdrawals.

```
170 •   SELECT
171       (SUM(CASE WHEN transaction_type = 'deposit' THEN amount ELSE 0 END) -
172        SUM(CASE WHEN transaction_type = 'withdrawal' THEN amount ELSE 0 END))
173     FROM Transactions;
174
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

net_difference
7100.00

Result Grid
Form Editor
Field Types

10. Write a SQL query to Calculate the average daily balance for each account over a specified period.

```
176 •   SELECT
177         account_id,
178         AVG(amount) AS avg_transaction_amount
179     FROM Transactions
180     WHERE transaction_date BETWEEN '2025-04-01' AND '2025-04-07'
181     GROUP BY account_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_id	avg_transaction_amount
101	5000.000000
102	2000.000000
103	500.000000
104	1000.000000
105	2500.000000
106	1500.000000

11. Calculate the total balance for each account type.

```
183 •   SELECT account_type, SUM(balance) AS total_balance
184     FROM Accounts
185     GROUP BY account_type;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_type	total_balance
savings	73500.00
current	53002.05
zero_balance	500.00

12. Identify accounts with the highest number of transactions order by descending order.

```
187 •   SELECT account_id, COUNT(*) AS transaction_count
188     FROM Transactions
189     GROUP BY account_id
190     ORDER BY transaction_count DESC;
191
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_id	transaction_count
101	1
102	1
103	1
104	1
105	1
106	1
107	1
108	1
109	1
110	1

13. List customers with high aggregate account balances, along with their account types.

```
192 •   SELECT c.customer_id, c.first_name, c.last_name, a.account_type, SUM(a.ba
193     FROM Customers c
194     JOIN Accounts a ON c.customer_id = a.customer_id
195     GROUP BY c.customer_id, a.account_type
196     HAVING SUM(a.balance) > 20000;
197
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	first_name	last_name	account_type	total_balance
2	Jane	Smith	current	25000.50
5	Charlie	Davis	current	20000.75
6	Emily	Clark	savings	30000.00

Result Grid
Form Editor
Field Types

14. Identify and list duplicate transactions based on transaction amount, date, and account.

```
198 •  SELECT account_id, amount, DATE(transaction_date) AS trans_date, COUNT(*)
199    FROM Transactions
200    GROUP BY account_id, amount, DATE(transaction_date)
201    HAVING COUNT(*) > 1;
202
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_id	amount	trans_date	duplicate_count

Result Grid | Form Editor | Field Types

Tasks 4: Subquery and its type:

1. Retrieve the customer(s) with the highest account balance.

```
203 •  SELECT c.customer_id, c.first_name, c.last_name, a.balance
204    FROM Customers c
205    JOIN Accounts a ON c.customer_id = a.customer_id
206    WHERE a.balance = (SELECT MAX(balance) FROM Accounts);
207
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	first_name	last_name	balance
6	Emily	Clark	30000.00

2. Calculate the average account balance for customers who have more than one account.

```
208 •  SELECT AVG(balance) AS avg_balance
209    FROM Accounts
210    WHERE customer_id IN (
211        SELECT customer_id
212        FROM Accounts
213        GROUP BY customer_id
214        HAVING COUNT(account_id) > 1
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

avg_balance
NULL

3. Retrieve accounts with transactions whose amounts exceed the average transaction amount.

```
217 •  SELECT DISTINCT t.account_id, t.transaction_id, t.amount
218    FROM Transactions t
219    WHERE t.amount > (SELECT AVG(amount) FROM Transactions);
220
```

Result Grid | Filter Rows: | Edit: | Export/Import:

account_id	transaction_id	amount
101	1001	5000.00
102	1002	2000.00
105	1005	2500.00
108	1008	2000.00
109	1009	3000.00
NULL	NULL	NULL

4. Identify customers who have no recorded transactions.

```
221 •   SELECT DISTINCT c.customer_id, c.first_name, c.last_name
222     FROM Customers c
223     JOIN Accounts a ON c.customer_id = a.customer_id
224     LEFT JOIN Transactions t ON a.account_id = t.account_id
225     WHERE t.transaction_id IS NULL;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	first_name	last_name

5. Calculate the total balance of accounts with no recorded transactions.

```
227 •   SELECT SUM(a.balance) AS total_balance_no_transactions
228     FROM Accounts a
229     LEFT JOIN Transactions t ON a.account_id = t.account_id
230     WHERE t.transaction_id IS NULL;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

total_balance_no_transactions
NULL

6. Retrieve transactions for accounts with the lowest balance.

```
232 •  SELECT t.*
233    FROM Transactions t
234    WHERE t.account_id = (
235        SELECT account_id
236        FROM Accounts
237        ORDER BY balance ASC
238        LIMIT 1
```

Result Grid | Filter Rows: | Edit: | Export/Impo

transaction_id	account_id	transaction_type	amount	transaction_date
1007	107	deposit	100.00	2025-04-07 08:10:00
NULL	NULL	NULL	NULL	NULL

7. Identify customers who have accounts of multiple types.

```
241 •  SELECT customer_id
242    FROM Accounts
243    GROUP BY customer_id
244    HAVING COUNT(DISTINCT account_type) > 1;
245
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id

8. Calculate the percentage of each account type out of the total number of accounts.

```
246 •  SELECT account_type,
247           COUNT(*) AS type_count,
248           ROUND(100.0 * COUNT(*) / (SELECT COUNT(*) FROM Accounts), 2) AS pe
249    FROM Accounts
250    GROUP BY account_type;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_type	type_count	percentage
savings	5	50.00
current	3	30.00
zero_balance	2	20.00

Result Grid | Form Editor | Field Types

Result 38 × Read Only

9. Retrieve all transactions for a customer with a given customer_id.

```
260 •  SELECT t.*
261    FROM Transactions t
262    JOIN Accounts a ON t.account_id = a.account_id
263    WHERE a.customer_id=10;
264
265 •  SELECT DISTINCT account_type,
266           (SELECT SUM(balance)
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

transaction_id	account_id	transaction_type	amount	transaction_date
1010	110	withdrawal	500.00	2025-04-08 13:15:00

10. Calculate the total balance for each account type, including a subquery within the SELECT clause.

```
265    SELECT DISTINCT account_type,
266           (SELECT SUM(balance)
267            FROM Accounts a2
268            WHERE a2.account_type = a1.account_type) AS total_balance
269    FROM Accounts a1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_type	total_balance
savings	73500.00
current	53002.05
zero_balance	500.00

Banking System

Control Structure

Task 1: Conditional Statements

In a bank, you have been given the task is to create a program that checks if a customer is eligible for a loan based on their credit score and income. The eligibility criteria are as follows:

- Credit Score must be above 700.
- Annual Income must be at least $50,000.

1. Write a program that takes the customer's credit score and annual income as input.

2. Use conditional statements (if-else) to determine if the customer is eligible for a loan.

3. Display an appropriate message based on eligibility.

```java
package com.hexaware.hmbank.main;

import java.util.Scanner;

public class LoanEligibility {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("🏦 Loan Eligibility Checker");

        System.out.print("Enter your credit score: ");
        int creditScore = scanner.nextInt();

        System.out.print("Enter your annual income: ");
        double income = scanner.nextDouble();

        if (creditScore > 700 && income >= 50000) {
            System.out.println("☑ You are eligible for the loan.");
        } else {
            System.out.println("✖ Sorry, you are not eligible for the loan.");
        }

        scanner.close();
    }
}
```

```
Problems  Javadoc  Declaration  Console ×  Install Java 24 Support  Eclipse IDE for Enterprise Java and Web
<terminated> LoanEligibility [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x
🏦 Loan Eligibility Checker
Enter your credit score: 300
Enter your annual income: 10000
✖ Sorry, you are not eligible for the loan.
```

Task 2: Nested Conditional Statements

Create a program that simulates an ATM transaction. Display options such as "Check Balance," "Withdraw," "Deposit,". Ask the user to enter their current balance and the amount they want to withdraw or deposit. Implement checks to ensure that the withdrawal amount is not greater than the available balance and that the withdrawal amount is in multiples of 100 or 500. Display appropriate messages for success or failure.

```java
package com.hexaware.hmbank.main;

import java.util.Scanner;

public class ATMSimulation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        double balance;

        System.out.println("🏧 Welcome to ATM Simulation");
        System.out.print("Enter your current balance: ₹");
        balance = scanner.nextDouble();

        System.out.println("\nChoose an option:");
        System.out.println("1. Check Balance");
        System.out.println("2. Withdraw");
        System.out.println("3. Deposit");
        System.out.print("Enter your choice (1/2/3): ");
        int choice = scanner.nextInt();

        if (choice == 1) {
            System.out.printf("💰 Your current balance is: ₹%.2f%n", balance);

        } else if (choice == 2) {
            System.out.print("Enter amount to withdraw: ₹");
            double amount = scanner.nextDouble();

            if (amount > balance) {
                System.out.println("❌ Insufficient balance.");
```

```java
            } else if (amount % 100 != 0 && amount % 500 != 0) {
                System.out.println("❌ Withdrawal must be in multiples of ₹100 or ₹500.");
            } else {
                balance -= amount;
                System.out.printf("☑ Withdrawal successful. Remaining balance: ₹%.2f%n", balance);
            }

        } else if (choice == 3) {
            System.out.print("Enter amount to deposit: ₹");
            double amount = scanner.nextDouble();
            balance += amount;
            System.out.printf("☑ Deposit successful. New balance: ₹%.2f%n", balance);

        } else {
            System.out.println("❌ Invalid option selected.");
        }

        scanner.close();
    }
}
```

Problems | Javadoc | Declaration | Console | Install Java 24 Support | Eclipse IDE for Enterprise Java and Web Developers 2025

<terminated> ATMSimulation [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v202

```
Choose an option:
1. Check Balance
2. Withdraw
3. Deposit
Enter your choice (1/2/3): 2
Enter amount to withdraw: ₹200
☑ Withdrawal successful. Remaining balance: ₹9800.00
```

Task 3: Loop Structures

You are responsible for calculating compound interest on savings accounts for bank customers. You need to calculate the future balance for each customer's savings account after a certain number of years.

Tasks:

1. Create a program that calculates the future balance of a savings account.

2. Use a loop structure (e.g., for loop) to calculate the balance for multiple customers.

3. Prompt the user to enter the initial balance, annual interest rate, and the number of years.

4. Calculate the future balance using the formula: *future_balance = initial_balance * (1 + annual_interest_rate/100)^years.*

5. Display the future balance for each customer.

```java
package com.hexaware.hmbank.main;

import java.util.Scanner;

public class CompoundInterestCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("🏦 Compound Interest Calculator");
        System.out.print("Enter the number of customers: ");
        int numCustomers = scanner.nextInt();

        for (int i = 1; i <= numCustomers; i++) {
            System.out.println("\n--- Customer " + i + " ---");

            System.out.print("Enter initial balance: ₹");
            double initialBalance = scanner.nextDouble();

            System.out.print("Enter annual interest rate (%): ");
            double interestRate = scanner.nextDouble();

            System.out.print("Enter number of years: ");
            int years = scanner.nextInt();

            double futureBalance = initialBalance * Math.pow((1 + interestRate / 100), years);

            System.out.printf("💰 Future Balance after %d years: ₹%.2f%n", years, futureBalance);
        }

        scanner.close();
    }
}
```

```
Problems  Javadoc  Declaration  Console ×  Install Java 24 Support  Eclipse IDE for Enterprise Java and Web Developers 2025-06 M1
CompoundInterestCalculator [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jre
Enter the number of customers: 2

--- Customer 1 ---
Enter initial balance: ₹1000
Enter annual interest rate (%): 3
Enter number of years: 2
```

Task 4: Looping, Array and Data Validation

You are tasked with creating a program that allows bank customers to check their account balances. The program should handle multiple customer accounts, and the customer should be able to enter their account number, balance to check the balance.

Tasks:

1. Create a Python program that simulates a bank with multiple customer accounts.

2. Use a loop (e.g., while loop) to repeatedly ask the user for their account number and balance until they enter a valid account number.

3. Validate the account number entered by the user.

4. If the account number is valid, display the account balance. If not, ask the user to try again.

```java
public class AccountBalanceChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Simulating customer accounts using a HashMap
        Map<String, Double> accounts = new HashMap<>();
        accounts.put("1001", 15000.50);
        accounts.put("1002", 32000.75);
        accounts.put("1003", 10500.00);
        accounts.put("1004", 8700.20);
        accounts.put("1005", 50000.00);

        System.out.println("🏦 Welcome to HM Bank Account Balance Checker");

        while (true) {
            System.out.print("🔢 Enter your account number (or type 'exit' to quit): ");
            String input = scanner.nextLine();

            if (input.equalsIgnoreCase("exit")) {
                System.out.println("👋 Thank you for using our service!");
                break;
            }

            if (accounts.containsKey(input)) {
                double balance = accounts.get(input);
                System.out.printf("💰 Your account balance is: ₹%.2f%n\n", balance);
            } else {
                System.out.println("❌ Invalid account number. Please try again.\n");
            }
        }
```

```
Problems  Javadoc  Declaration  Console ×  Install Java 24 Support  Eclipse IDE for Enterprise Java and Web Dev
AccountBalanceChecker [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.
🏦 Welcome to HM Bank Account Balance Checker
🔢 Enter your account number (or type 'exit' to quit): 1001
💰 Your account balance is: ₹15000.50

🔢 Enter your account number (or type 'exit' to quit):
```

Task 5: Password Validation

Write a program that prompts the user to create a password for their bank account. Implement if conditions to validate the password according to these rules:

- The password must be at least 8 characters long.
- It must contain at least one uppercase letter.
- It must contain at least one digit.
- Display appropriate messages to indicate whether their password is valid or not.

```java
public class PasswordValidator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("🔐 Create your bank account password");
        System.out.print("Enter your password: ");
        String password = scanner.nextLine();

        boolean isValid = true;

        // Check length
        if (password.length() < 8) {
            System.out.println("❌ Password must be at least 8 characters long.");
            isValid = false;
        }

        // Check for at least one uppercase letter
        boolean hasUppercase = false;
        for (char c : password.toCharArray()) {
            if (Character.isUpperCase(c)) {
                hasUppercase = true;
                break;
            }
        }
        if (!hasUppercase) {
            System.out.println("❌ Password must contain at least one uppercase letter.");
            isValid = false;
        }

        // Check for at least one digit
        boolean hasDigit = false;
```

```java
        for (char c : password.toCharArray()) {
            if (Character.isDigit(c)) {
                hasDigit = true;
                break;
            }
        }
        if (!hasDigit) {
            System.out.println("❌ Password must contain at least one digit.");
            isValid = false;
        }

        // Final message
        if (isValid) {
            System.out.println("☑ Password is valid. Your account is secured!");
        }

        scanner.close();
    }
}
```

Problems @ Javadoc Declaration Console × ⓘ Install Java 24 Support ⓘ Eclipse IDE for Enterprise Java and Web Develo

terminated> PasswordValidator [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_6

```
 Create your bank account password
nter your password: 123@Aa
❮ Password must be at least 8 characters long.
```

Task 6: Password Validation

Create a program that maintains a list of bank transactions (deposits and withdrawals) for a customer. Use a while loop to allow the user to keep adding transactions until they choose to exit. Display the transaction history upon exit using looping statements.

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class TransactionHistory {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<String> transactions = new ArrayList<>();
        double balance = 0.0;

        System.out.println("🏦 HM Bank Transaction Recorder");

        while (true) {
            System.out.println("\nChoose a transaction:");
            System.out.println("1. Deposit");
            System.out.println("2. Withdraw");
            System.out.println("3. Exit and Show History");
            System.out.print("Enter your choice (1/2/3): ");
            String choice = scanner.nextLine();

            if (choice.equals("1")) {
                System.out.print("Enter deposit amount: ₹");
                double amount = Double.parseDouble(scanner.nextLine());
                balance += amount;
                transactions.add("Deposited: ₹" + amount);
                System.out.println("☑ Deposit successful. Current balance: ₹" + balance);
```

```
            } else if (choice.equals("2")) {
                System.out.print("Enter withdrawal amount: ₹");
                double amount = Double.parseDouble(scanner.nextLine());
                if (amount > balance) {
                    System.out.println("❌ Insufficient funds. Transaction failed.");
                    transactions.add("Failed Withdrawal: ₹" + amount + " (Insufficient funds)");
                } else {
                    balance -= amount;
                    transactions.add("Withdrawn: ₹" + amount);
                    System.out.println("☑ Withdrawal successful. Current balance: ₹" + balance);
                }

            } else if (choice.equals("3")) {
                System.out.println("\n🖹 Transaction History:");
                for (String txn : transactions) {
                    System.out.println("- " + txn);
                }
                System.out.printf("⏱ Final Balance: ₹%.2f%n", balance);
                break;

            } else {
                System.out.println("❌ Invalid option. Please choose 1, 2 or 3.");
            }
        }

        scanner.close();
    }
}

Problems  Javadoc  Declaration  Console ×  Install Java 24 Support  Eclipse IDE for Enterprise Java and Web Developers 2025-06
TransactionHistory [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jr
1. Deposit
2. Withdraw
3. Exit and Show History
Enter your choice (1/2/3): 1
Enter deposit amount: ₹1000
☑ Deposit successful. Current balance: ₹1000.0
```

OOPS, Collections and Exception Handling

Task 7: Class & Object

1. Create a `Customer` class with the following confidential attributes:

Attributes o Customer ID

o First Name

o Last Name

o Email Address

o Phone Number

o Address

2.Constructor and Methods

o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.

```java
public class Customer {
    private long customerId;
    private String firstName;
    private String lastName;
    private String emailAddress;
    private String phoneNumber;
    private String address;

    // Default Constructor
    public Customer() {}

    // Parameterized Constructor
    public Customer(long customerId, String firstName, String lastName, String emailAddress,
                    String phoneNumber, String address) {
        this.customerId = customerId;
        this.firstName = firstName;
        this.lastName = lastName;
        this.emailAddress = emailAddress;
        this.phoneNumber = phoneNumber;
        this.address = address;
    }

    // Getters and Setters
    public long getCustomerId() { return customerId; }
    public void setCustomerId(long customerId) { this.customerId = customerId; }

    public String getFirstName() { return firstName; }
    public void setFirstName(String firstName) { this.firstName = firstName; }

    public String getLastName() { return lastName; }
    public void setLastName(String lastName) { this.lastName = lastName; }

    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; }

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

    // Display method
    public void displayCustomerDetails() {
        System.out.println("Customer ID   : " + customerId);
        System.out.println("Name          : " + firstName + " " + lastName);
        System.out.println("Email         : " + emailAddress);
        System.out.println("Phone         : " + phoneNumber);
        System.out.println("Address       : " + address);
    }
}
```

Problems Javadoc Declaration Console × Install Java 24 Support Eclipse IDE for Enterprise Java and Web Developers 20

ransactionHistory [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0

```
. Deposit
. Withdraw
. Exit and Show History
nter your choice (1/2/3): 2
nter withdrawal amount: ₹1000
 Insufficient funds. Transaction failed.
```

2. Create an `Account` class with the following confidential attributes:

Attributes o Account Number

o Account Type (e.g., Savings, Current)

o Account Balance

2.Constructor and Methods

o Implement default constructors and overload the constructor with Account attributes,

o Generate getter and setter, (print all information of attribute) methods for the attributes.

o Add methods to the `Account` class to allow deposits and withdrawals. - deposit(amount: float): Deposit the specified amount into the account.

- withdraw(amount: float): Withdraw the specified amount from the account. withdraw amount only if there is sufficient fund else display insufficient balance.
- calculate_interest(): method for calculating interest amount for the available balance. interest rate is fixed to 4.5%

```
    public void deposit(double amount) {
        if (amount > 0) {
            accountBalance += amount;
            System.out.println("☑ Deposited ₹" + amount);
        } else {
            System.out.println("✕ Invalid deposit amount.");
        }
    }

    // Withdraw method
    public void withdraw(double amount) {
        if (amount > accountBalance) {
            System.out.println("✕ Insufficient balance.");
        } else {
            accountBalance -= amount;
            System.out.println("☑ Withdrawn ₹" + amount);
        }
    }

    // Calculate Interest (only for Savings)
    public void calculateInterest() {
        if (accountType.equalsIgnoreCase("Savings")) {
            double interest = accountBalance * 4.5 / 100;
            accountBalance += interest;
            System.out.printf("☑ Interest of ₹%.2f added. New Balance: ₹%.2f%n", interest, accountBalance);
        } else {
            System.out.println("✕ Interest not applicable for account type: " + accountType);
        }
    }
```

Problems Javadoc Declaration Console Install Java 24 Support Eclipse IDE for Enterprise Java and Web Developers 2025-06 M1

```
terminated> TransactionHistory [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jre\bin\javaw.exe (11-
. Deposit
. Withdraw
. Exit and Show History
nter your choice (1/2/3): 3

) Transaction History:
) Final Balance: ₹0.00
```

```
public class Account {
    private long accountNumber;
    private String accountType;
    private double accountBalance;

    // Default Constructor
    public Account() {}

    // Parameterized Constructor
    public Account(long accountNumber, String accountType, double accountBalance) {
        this.accountNumber = accountNumber;
        this.accountType = accountType;
        this.accountBalance = accountBalance;
    }

    // Getters and Setters
    public long getAccountNumber() { return accountNumber; }
    public void setAccountNumber(long accountNumber) { this.accountNumber = accountNumber; }

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

    public double getAccountBalance() { return accountBalance; }
    public void setAccountBalance(double accountBalance) { this.accountBalance = accountBalance; }
}
```

Create a Bank class to represent the banking system. Perform the following operation in main method: o create object for account class by calling parameter constructor. o deposit(amount: float): Deposit the specified amount into the account.

o withdraw(amount: float): Withdraw the specified amount from the account.

o calculate_interest(): Calculate and add interest to the account balance for savings accounts.

```java
public class Bank {
    public static void main(String[] args) {
        // Create customer object
        Customer customer = new Customer(
            101, "Fahimunnisha", "A", "fahim@bank.com", "9876543210", "Velachery, Chennai"
        );

        // Display customer details
        System.out.println("\n=== Customer Information ===");
        customer.displayCustomerDetails();

        // Create account object using parameterized constructor
        Account account = new Account(2001, "Savings", 10000);

        // Display initial account details
        System.out.println("\n=== Account Created ===");
        account.displayAccountDetails();

        // Perform deposit
        account.deposit(3000);

        // Perform withdrawal
        account.withdraw(2000);

        // Calculate interest
        account.calculateInterest();

        // Final account summary
        System.out.println("\n=== Final Account Summary ===");
        account.displayAccountDetails();
    }
}
```

```
Problems  Javadoc  Declaration  Console  Install Java 24 Support  Eclipse IDE for Enterprise Java and Web Develope
<terminated> Bank [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250
Interest of ₹495.00 added. New Balance: ₹11495.00

=== Final Account Summary ===
Account No     : 2001
Account Type   : Savings
Account Balance: ₹11495.00
```

Task 8: Inheritance and polymorphism

1. Overload the deposit and withdraw methods in Account class as mentioned below. deposit(amount: float): Deposit the specified amount into the account.

withdraw(amount: float): Withdraw the specified amount from the account. withdraw amount only if there is sufficient fund else display insufficient balance.

deposit(amount: int): Deposit the specified amount into the account.

withdraw(amount: int): Withdraw the specified amount from the account. withdraw amount only if there is sufficient fund else display insufficient balance.

deposit(amount: double): Deposit the specified amount into the account.

withdraw(amount: double): Withdraw the specified amount from the account. withdraw amount only if there is sufficient fund else display insufficient balance.

```
// 🔁 Method Overloading: Deposit
// --------------------------------

public void deposit(float amount) {
    deposit((double) amount);
}

public void deposit(int amount) {
    deposit((double) amount);
}

public void deposit(double amount) {
    if (amount > 0) {
        accountBalance += amount;
        System.out.println("☑ Deposited ₹" + amount);
    } else {
        System.out.println("✖ Invalid deposit amount.");
    }
}

// --------------------------------
// 🔁 Method Overloading: Withdraw
// --------------------------------

public void withdraw(float amount) {
    withdraw((double) amount);
}

public void withdraw(int amount) {
    withdraw((double) amount);
}

public void withdraw(double amount) {
    if (amount > accountBalance) {
        System.out.println("✖ Insufficient balance.");
    } else {
        accountBalance -= amount;
        System.out.println("☑ Withdrawn ₹" + amount);
    }
}
```

2. Create Subclasses for Specific Account Types

Create subclasses for specific account types (e.g., `SavingsAccount`, `CurrentAccount`) that inherit from the `Account` class. o **SavingsAccount**: A savings account that includes an additional attribute for interest rate. **override** the calculate_interest() from Account class method to calculate interest based on the balance and interest rate.

o **CurrentAccount**: A current account that includes an additional attribute overdraftLimit. A current account with no interest. Implement the withdraw() method to allow overdraft up to a certain limit (configure a constant for the overdraft limit).

```java
package com.hexaware.hmbank.entity;

public class SavingsAccount extends Account {
    private double interestRate;

    public SavingsAccount(long accountNumber, double accountBalance, double interestRate) {
        super(accountNumber, "Savings", accountBalance);
        this.interestRate = interestRate;
    }

    @Override
    public void calculateInterest() {
        double interest = getAccountBalance() * interestRate / 100;
        deposit(interest);
        System.out.printf("☑ Interest ₹%.2f added to Savings Account%n", interest);
    }
}
```

```java
package com.hexaware.hmbank.entity;

public class CurrentAccount extends Account {
    private static final double OVERDRAFT_LIMIT = 5000;

    public CurrentAccount(long accountNumber, double accountBalance) {
        super(accountNumber, "Current", accountBalance);
    }

    @Override
    public void withdraw(double amount) {
        if (amount > getAccountBalance() + OVERDRAFT_LIMIT) {
            System.out.println("❌ Withdrawal failed. Overdraft limit exceeded.");
        } else {
            double newBalance = getAccountBalance() - amount;
            setAccountBalance(newBalance);
            System.out.println("☑ Withdrawn ₹" + amount);
        }
    }

    @Override
    public void calculateInterest() {
        System.out.println("ℹ No interest for current accounts.");
    }
}
```

3. Create a **Bank** class to represent the banking system. Perform the following operation in main method: Display menu for user to create object for account class by calling parameter constructor. Menu should display options `SavingsAccount` and `CurrentAccount`. user can choose any one option to create account. use switch case for implementation.

deposit(amount: float): Deposit the specified amount into the account.

withdraw(amount: float): Withdraw the specified amount from the account. For saving account withdraw amount only if there is sufficient fund else display insufficient balance.

For Current Account withdraw limit can exceed the available balance and should not exceed the overdraft limit.

calculate_interest(): Calculate and add interest to the account balance for savings accounts.

```java
public class Bank {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Account account = null;

        System.out.println("🏦 Welcome to HM Bank");
        System.out.println("Choose account type to create:");
        System.out.println("1. Savings Account");
        System.out.println("2. Current Account");
        System.out.print("Enter choice (1 or 2): ");
        int choice = scanner.nextInt();

        System.out.print("Enter Account Number: ");
        long accNo = scanner.nextLong();
        System.out.print("Enter Initial Balance: ₹");
        double balance = scanner.nextDouble();

        switch (choice) {
            case 1:
                System.out.print("Enter Interest Rate (%): ");
                double rate = scanner.nextDouble();
                account = new SavingsAccount(accNo, balance, rate);
                break;
            case 2:
                account = new CurrentAccount(accNo, balance);
                break;
            default:
                System.out.println("❌ Invalid account type selected.");
                return;
        }
```

```java
        while (true) {
            System.out.println("\n--- Account Operations ---");
            System.out.println("1. Deposit");
            System.out.println("2. Withdraw");
            System.out.println("3. Calculate Interest");
            System.out.println("4. Display Account Details");
            System.out.println("5. Exit");
            System.out.print("Choose an option: ");
            int option = scanner.nextInt();

            switch (option) {
                case 1:
                    System.out.print("Enter deposit amount: ₹");
                    double depositAmt = scanner.nextDouble();
                    account.deposit(depositAmt);
                    break;
                case 2:
                    System.out.print("Enter withdraw amount: ₹");
                    double withdrawAmt = scanner.nextDouble();
                    account.withdraw(withdrawAmt);
                    break;
                case 3:
                    account.calculateInterest();
                    break;
                case 4:
                    account.displayAccountDetails();
                    break;
                case 5:
                    System.out.println("👋 Exiting. Thank you!");
                    return;
                default:
```

Problems @ Javadoc Declaration Console × Install Java 24 Support Eclipse IDE for Enterprise Java and Web

```
Bank [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-(
Choose account type to create:
1. Savings Account
2. Current Account
Enter choice (1 or 2): 2
Enter Account Number: 1001
Enter Initial Balance: ₹10000
```

Task 9: Abstraction

1. Create an abstract class BankAccount that represents a generic bank account. It should include the following attributes and methods: Attributes: o Account number.

- o Customer name.
- o Balance.

2.Constructors: Implement default constructors and overload the constructor with Account attributes, generate getter and setter, print all information of attribute methods for the attributes.

Abstract methods: **deposit(amount: float):** Deposit the specified amount into the account.

o **withdraw(amount: float):** Withdraw the specified amount from the account (implement error handling for insufficient funds).

o **calculate_interest():** Abstract method for calculating interest

```
public abstract class BankAccount {
    protected long accountNumber;
    protected String customerName;
    protected double balance;

    // Default constructor
    public BankAccount() {}

    // Parameterized constructor
    public BankAccount(long accountNumber, String customerName, double balance) {
        this.accountNumber = accountNumber;
        this.customerName = customerName;
        this.balance = balance;
    }

    // Getters and Setters
    public long getAccountNumber() { return accountNumber; }
    public void setAccountNumber(long accountNumber) { this.accountNumber = accountNumber; }

    public String getCustomerName() { return customerName; }
    public void setCustomerName(String customerName) { this.customerName = customerName; }

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

    // Print method
    public void displayAccountDetails() {
        System.out.println("Account Number : " + accountNumber);
        System.out.println("Customer Name  : " + customerName);
        System.out.printf("Balance        : ₹%.2f%n", balance);
```

```
    // Abstract methods
    public abstract void deposit(float amount);
    public abstract void withdraw(float amount);
    public abstract void calculateInterest();
}

```

Problems Javadoc Declaration Console Install Java 24 Support Eclipse IDE f

```
Bank [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32
🏦 Welcome to HM Bank
Choose account type to create:
1. Savings Account
2. Current Account
Enter choice (1 or 2): 2
Enter Account Number: 1001
Enter Initial Balance: ₹10000
Choose account type to create:
1. Savings Account
2. Current Account
Enter choice (1 or 2): 1
Enter Account Number: 1001
Enter Initial Balance: ₹10000
Enter Interest Rate (%): 3
4. Display Account Details
5. Exit
Choose an option: 4
Account Number : 1001
Account Type   : Savings
Account Balance: ₹10000.00
```

```java
public class SavingsAccount extends BankAccount {
    private float interestRate;

    public SavingsAccount(long accountNumber, String customerName, double balance, float interestRate) {
        super(accountNumber, customerName, balance);
        this.interestRate = interestRate;
    }

    @Override
    public void deposit(float amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("☑ Deposited ₹" + amount);
        } else {
            System.out.println("✖ Invalid deposit amount.");
        }
    }

    @Override
    public void withdraw(float amount) {
        if (amount <= balance) {
            balance -= amount;
            System.out.println("☑ Withdrawn ₹" + amount);
        } else {
            System.out.println("✖ Insufficient balance.");
        }
    }
```

```
    @Override
    public void calculateInterest() {
        double interest = balance * interestRate / 100;
        balance += interest;
        System.out.printf("☑ Interest ₹%.2f added. New Balance: ₹%.2f%n", interest, balance);
    }
}
```

Problems Javadoc Declaration Console × Install Java 24 Support Eclipse IDE for Enterprise Java and Web Developers 2025-06 M

```
<terminated> Bank [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jre
☑ Deposited ₹2000.0
☑ Withdrawn ₹1500.0
☑ Interest ₹472.50 added. New Balance: ₹10972.50
Account Number : 1001
Customer Name  : Alice
Balance        : ₹10972.50
```

2.Create two concrete classes that inherit from **BankAccount**: **SavingsAccount**: A savings account that includes an additional attribute for interest rate. Implement the calculate_interest() method to calculate interest based on the balance and interest rate.

CurrentAccount: A current account with no interest. Implement the withdraw() method to allow overdraft up to a certain limit (configure a constant for the overdraft limit).

3. Create a Bank class to represent the banking system. Perform the following operation in main method: Display menu for user to create object for account class by calling parameter constructor. Menu should display options `SavingsAccount` and `CurrentAccount`. user can choose any one option to create account. use switch case for implementation. create_account should display sub menu to choose type of accounts. o *Hint: Account acc = new SavingsAccount(); or Account acc = new CurrentAccount();*

deposit(amount: float): Deposit the specified amount into the account.

withdraw(amount: float): Withdraw the specified amount from the account. For saving account withdraw amount only if

there is sufficient fund else display insufficient balance. For Current Account withdraw limit can exceed the available balance and should not exceed the overdraft limit.

□calculate_interest(): Calculate and add interest to the account balance for savings accounts.

```java
public class CurrentAccount extends BankAccount {
    private static final double OVERDRAFT_LIMIT = 5000;

    public CurrentAccount(long accountNumber, String customerName, double balance) {
        super(accountNumber, customerName, balance);
    }

    @Override
    public void deposit(float amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("☑ Deposited ₹" + amount);
        } else {
            System.out.println("✕ Invalid deposit amount.");
        }
    }

    @Override
    public void withdraw(float amount) {
        if (amount > balance + OVERDRAFT_LIMIT) {
            System.out.println("✕ Withdrawal failed. Overdraft limit exceeded.");
        } else {
            balance -= amount;
            System.out.println("☑ Withdrawn ₹" + amount);
        }
    }

    @Override
    public void calculateInterest() {
        System.out.println("ⓘ No interest for current accounts.");
    }
```

```
☑ Deposited ₹2000.0
☑ Withdrawn ₹1500.0
☑ Interest ₹472.50 added. New Balance: ₹10972.50
Account Number : 1001
Customer Name  : Alice
Balance        : ₹10972.50
-------------------------------
```

```java
package com.hexaware.hmbank.main;

import com.hexaware.hmbank.entity.BankAccount;
import com.hexaware.hmbank.entity.SavingsAccount;
import com.hexaware.hmbank.entity.CurrentAccount;

public class Bank {
    public static void main(String[] args) {
        // ☑ Correct object creation with customerName
        BankAccount savings = new SavingsAccount(1001L, "Alice", 10000.0, 4.5f);
        BankAccount current = new CurrentAccount(1002L, "Bob", 5000.0);

        // ☑ Operations on savings account
        savings.deposit(2000f);
        savings.withdraw(1500f);
        savings.calculateInterest();
        savings.displayAccountDetails();

        System.out.println("------------------------------");

        // ☑ Operations on current account
        current.deposit(1000f);
        current.withdraw(6000f);
        current.calculateInterest();
        current.displayAccountDetails();
    }
}
```

```
Problems  Javadoc  Declaration  Console ×  Install Java 24 Support  Eclipse IDE for Enterprise Java and Web De
<terminated> Bank [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6\
☑ Deposited ₹1000.0
☑ Withdrawn ₹6000.0
ⓘ No interest for current accounts.
Account Number : 1002
Customer Name  : Bob
Balance        : ₹0.00
```

Task 10: Has A Relation / Association

1. Create a `Customer` class with the following attributes:

☐Customer ID

☐First Name

☐Last Name

☐Email Address (validate with valid email address)

☐Phone Number (Validate 10-digit phone number)

☐Address

□Methods and Constructor: o Implement default constructors and overload the constructor with Account attributes, generate getter, setter, print all information of attribute) methods for the attributes.

```java
public class Customer {
    private long customerId;
    private String firstName;
    private String lastName;
    private String email;
    private String phoneNumber;
    private String address;

    public Customer() {}

    public Customer(long customerId, String firstName, String lastName,
                    String email, String phoneNumber, String address) {
        this.customerId = customerId;
        this.firstName = firstName;
        this.lastName = lastName;
        setEmail(email);
        setPhoneNumber(phoneNumber);
        this.address = address;
    }

    // Getters & Setters
    public long getCustomerId() { return customerId; }
    public void setCustomerId(long customerId) { this.customerId = customerId; }

    public String getFirstName() { return firstName; }
    public void setFirstName(String firstName) { this.firstName = firstName; }

    public String getLastName() { return lastName; }
    public void setLastName(String lastName) { this.lastName = lastName; }

    public String getEmail() { return email; }
    public void setEmail(String email) {
        if (email.matches("^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$")) {
            this.email = email;
        } else {
            throw new IllegalArgumentException("Invalid email format.");
        }
    }
```

```
Customer ID   : 1
Name          : John Doe
Email         : john.doe@example.com
Phone         : 1234567890
Address       : 123 Main St, City
```

2. Create an `Account` class with the following attributes: □Account Number (a unique identifier).

□Account Type (e.g., Savings, Current)

- Account Balance
- Customer (the customer who owns the account)
- Methods and Constructor:

o Implement default constructors and overload the constructor with Account attributes, generate getter, setter, (print all information of attribute) methods for the attributes.

```
public class Account {
    private long accountNumber;
    private String accountType;
    private float accountBalance;
    private Customer customer;

    public Account() {}

    public Account(long accountNumber, String accountType, float accountBalance, Customer customer) {
        this.accountNumber = accountNumber;
        this.accountType = accountType;
        this.accountBalance = accountBalance;
        this.customer = customer;
    }

    public long getAccountNumber() { return accountNumber; }
    public void setAccountNumber(long accountNumber) { this.accountNumber = accountNumber; }

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

    public float getAccountBalance() { return accountBalance; }
    public void setAccountBalance(float accountBalance) { this.accountBalance = accountBalance; }

    public Customer getCustomer() { return customer; }
    public void setCustomer(Customer customer) { this.customer = customer; }
```

```
    public void displayAccountDetails() {
        System.out.println("Account Number : " + accountNumber);
        System.out.println("Account Type   : " + accountType);
        System.out.printf("Balance        : ₹%.2f\n", accountBalance);
        if (customer != null) {
            customer.displayCustomerDetails();
        }
    }
}
```

```
Balance        : ₹15000.50
Customer ID    : 1
Name           : John Doe
Email          : john.doe@example.com
Phone          : 1234567890
Address        : 123 Main St, City
```

Create a Bank Class and must have following requirements:

1. Create a Bank class to represent the banking system. It should have the following methods:

- **create_account(Customer customer, long accNo, String accType, float balance)**: Create a new bank account for the given customer with the initial balance.

- **get_account_balance(account_number: long)**: Retrieve the balance of an account given its account number. should return the current balance of account.

- **deposit(account_number: long, amount: float)**: Deposit the specified amount into the account. Should return the current balance of account.

- **withdraw(account_number: long, amount: float)**: Withdraw the specified amount from the account. Should return the current balance of account.

- **transfer(from_account_number: long, to_account_number: int, amount: float)**: Transfer money from one account to another.

- **getAccountDetails(account_number: long):** Should return the account and customer details.

```java
public class Bank {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Bank bank = new Bank();

        while (true) {
            System.out.println("\n🏦 Bank System - Menu");
            System.out.println("1. Create Account");
            System.out.println("2. Deposit");
            System.out.println("3. Withdraw");
            System.out.println("4. Get Balance");
            System.out.println("5. Transfer");
            System.out.println("6. Get Account Details");
            System.out.println("7. Exit");
            System.out.print("Enter command number: ");
            int choice = sc.nextInt();  // Use integer input for menu selection

            switch (choice) {
                case 1 -> {
                    sc.nextLine();  // To consume the newline left after nextInt()
                    System.out.print("Enter First Name: ");
                    String fName = sc.nextLine();
                    System.out.print("Enter Last Name: ");
                    String lName = sc.nextLine();
                    System.out.print("Enter Email: ");
                    String email = sc.nextLine();
                    System.out.print("Enter Phone: ");
                    String phone = sc.nextLine();
                    System.out.print("Enter Address: ");
                    String address = sc.nextLine();

                    String accType;
                    while (true) {
                        System.out.println("Choose Account Type:");
                        System.out.println("1. Savings");
                        System.out.println("2. Current");
                        System.out.print("Enter option: ");
                        int accChoice = sc.nextInt();
                        if (accChoice == 1) {
                            accType = "Savings";
                            break;
                        } else if (accChoice == 2) {
                            accType = "Current";
                            break;
                        } else {
                            System.out.println("❌ Invalid choice.");
                        }
                    }

                    System.out.print("Enter Initial Balance: ₹");
                    float bal = sc.nextFloat();

                    try {
                        Customer cust = new Customer(System.currentTimeMillis(), fName, lName, email, phone, address);
                        bank.createAccount(cust, accType, bal);
                    } catch (IllegalArgumentException e) {
                        System.out.println(e.getMessage());
```

```java
    public void createAccount(Customer customer, String accountType, float balance) {
        System.out.println("Account created successfully!");
    }

    // Example implementation of deposit method.
    public float deposit(long accountNumber, float amount) {
        return amount;  // Placeholder return
    }

    // Example implementation of withdraw method.
    public float withdraw(long accountNumber, float amount) {
        return amount;  // Placeholder return
    }

    // Example implementation of getAccountBalance method.
    public float getAccountBalance(long accountNumber) {
        return 1000.0f;  // Placeholder return
    }

    // Example implementation of transfer method.
    public void transfer(long fromAccountNumber, long toAccountNumber, float amount) {
        System.out.println("Transfer completed!");
    }

    // Example implementation of getAccountDetails method.
    public void getAccountDetails(long accountNumber) {
        System.out.println("Account details for: " + accountNumber);
    }
```

```
Enter command number: 1
Enter First Name: sam
Enter Last Name: samuael
Enter Email: samao@gmail.com
Enter Phone: 3456789123
Enter Address: chennai
Choose Account Type:
```

2. Ensure that account numbers are automatically generated when an account is created, starting from 1001 and incrementing for each new account.

```java
package com.hexaware.hmbank.main;

import com.hexaware.hmbank.entity.Customer;
import com.hexaware.hmbank.service.Bank;

import java.util.Scanner;

public class BankApp {
    public static void main(String[] args) {
        Scanner sc = null;
        try {
            sc = new Scanner(System.in);  // Initialize Scanner

            Bank bank = new Bank();

            while (true) {
                System.out.println("\n🏦 Bank System - Menu");
                System.out.println("1. create_account");
                System.out.println("2. deposit");
                System.out.println("3. withdraw");
                System.out.println("4. get_balance");
                System.out.println("5. transfer");
                System.out.println("6. getAccountDetails");
                System.out.println("7. exit");
                System.out.print("Enter command: ");
                String choice = sc.next();

                switch (choice.toLowerCase()) {
                    case "create_account" -> {
                        sc.nextLine(); // Clear buffer
                        System.out.print("Enter First Name: ");

                            String fName = sc.nextLine();
                            System.out.print("Enter Last Name: ");
                            String lName = sc.nextLine();
                            System.out.print("Enter Email: ");
                            String email = sc.nextLine();
                            System.out.print("Enter Phone: ");
                            String phone = sc.nextLine();
                            System.out.print("Enter Address: ");
                            String address = sc.nextLine();

                            String accType;
                            while (true) {
                                System.out.println("Choose Account Type:");
                                System.out.println("1. Savings");
                                System.out.println("2. Current");
                                System.out.print("Enter option: ");
                                int accChoice = sc.nextInt();
                                if (accChoice == 1) {
                                    accType = "Savings";
                                    break;
                                } else if (accChoice == 2) {
                                    accType = "Current";
                                    break;
                                } else {
                                    System.out.println("❌ Invalid choice.");
                                }
                            }

                            System.out.print("Enter Initial Balance: ₹");
                            float bal = sc.nextFloat();
```

```
                float bal = sc.nextFloat();

                Customer cust = new Customer(System.currentTimeMillis(), fName, lName, email, phone, address);
                bank.createAccount(cust, accType, bal);
            }

            case "deposit" -> {
                System.out.print("Enter Account Number: ");
                long accNo = sc.nextLong();
                System.out.print("Enter amount to deposit: ₹");
                float amt = sc.nextFloat();
                float newBal = bank.deposit(accNo, amt);
                System.out.println("☑ New Balance: ₹" + newBal);
            }

            case "withdraw" -> {
                System.out.print("Enter Account Number: ");
                long accNo = sc.nextLong();
                System.out.print("Enter amount to withdraw: ₹");
                float amt = sc.nextFloat();
                float newBal = bank.withdraw(accNo, amt);
                System.out.println("☑ New Balance: ₹" + newBal);
            }

            case "get_balance" -> {
                System.out.print("Enter Account Number: ");
                long accNo = sc.nextLong();
                float bal = bank.getAccountBalance(accNo);
                System.out.println("💰 Balance: ₹" + bal);
            }


            case "get_balance" -> {
                System.out.print("Enter Account Number: ");
                long accNo = sc.nextLong();
                float bal = bank.getAccountBalance(accNo);
                System.out.println("💰 Balance: ₹" + bal);
            }

            case "transfer" -> {
                System.out.print("Enter FROM Account Number: ");
                long from = sc.nextLong();
                System.out.print("Enter TO Account Number: ");
                long to = sc.nextLong();
                System.out.print("Enter amount to transfer: ₹");
                float amt = sc.nextFloat();
                bank.transfer(from, to, amt);
            }

            case "getaccountdetails" -> {
                System.out.print("Enter Account Number: ");
                long accNo = sc.nextLong();
                bank.getAccountDetails(accNo);
            }

            case "exit" -> {
                System.out.println("👋 Thank you for using the banking system!");
                return;
            }

            default -> System.out.println("✖ Invalid command.");
        }
    }
```

```
1. create_account
2. deposit
3. withdraw
4. get_balance
5. transfer
6. getAccountDetails
7. exit
Enter command:
```

```
Enter command number: 5
Enter FROM Account Number: 1001
Enter TO Account Number: 1003
Enter amount to transfer: ₹200
Transfer completed!
```

Task 11: Interface/abstract class, and Single Inheritance, static variable

1. Create a ‘Customer’ class as mentioned above task.

2. Create an class ‘Account’ that includes the following attributes. Generate account number using static variable.

- Account Number (a unique identifier).
- Account Type (e.g., Savings, Current)
- Account Balance
- Customer (the customer who owns the account)
- lastAccNo

```java
public class Account {
    private static long lastAccNo = 1000;
    protected long accountNumber;
    protected String accountType;
    protected float balance;
    protected Customer customer;

    public Account(String accountType, float balance, Customer customer) {
        this.accountNumber = ++lastAccNo;
        this.accountType = accountType;
        this.balance = balance;
        this.customer = customer;
    }

    public long getAccountNumber() { return accountNumber; }
    public String getAccountType() { return accountType; }
    public float getBalance() { return balance; }
    public Customer getCustomer() { return customer; }

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

    public void displayAccountDetails() {
        System.out.println("Account No     : " + accountNumber);
        System.out.println("Account Type   : " + accountType);
        System.out.printf("Balance        : ₹%.2f\n", balance);
        customer.displayCustomerDetails();
    }

    // ☑ Add this method to avoid compilation error when calculateInterest() is called
    public void calculateInterest() {
        System.out.println("ⓘ Interest calculation not supported for generic account.");
```

Problems @ Javadoc Declaration Console × ⓘ Eclipse IDE for Enterprise Java and Web Developers 2025-06 M1

```
BankApp [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jre\bin\javav
Enter Account Number: 1001
Account No     : 1001
Account Type   : Current
Balance        : ₹10000.00
Customer ID    : 1744559028565
Name           : ram ariel
Email          : agahh@gmail.com
Phone          : 1234567897
```

3. Create three child classes that inherit the Account class and each class must contain below mentioned attribute: SavingsAccount: A savings account that includes an additional attribute for interest rate. Saving account should be created with minimum balance 500.

CurrentAccount: A Current account that includes an additional attribute for overdraftLimit(credit limit). withdraw() method to allow overdraft up to a certain limit. withdraw limit can exceed the available balance and should not exceed the overdraft limit.

ZeroBalanceAccount: ZeroBalanceAccount can be created with Zero balance.

```java
package com.hexaware.hmbank.bean;

public class CurrentAccount extends Account {
    private float overdraftLimit;

    public CurrentAccount(float balance, Customer customer, float overdraftLimit) {
        super("Current", balance, customer);
        this.overdraftLimit = overdraftLimit;
    }

    public boolean withdraw(float amount) {
        if (amount > (balance + overdraftLimit)) {
            System.out.println("❌ Withdrawal failed. Overdraft limit exceeded.");
            return false;
        } else {
            balance -= amount;
            System.out.println("☑ Withdrawn ₹" + amount);
            return true;
        }
    }

    public void calculateInterest() {
        System.out.println("ℹ No interest for current accounts.");
    }
}
```

Problems Javadoc Declaration Console × Eclipse IDE for Enterprise Java and Web Developers 2025-06 M1

```
BankApp [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529
9. Exit
Choose an option: 3
Enter Account Number: 1001
Enter amount to withdraw: ₹200
❌ Account not found.
☑ New Balance: ₹-1.0
```

```java
package com.hexaware.hmbank.bean;
public class SavingsAccount extends Account {
    private float interestRate;
    private static final float MIN_BALANCE = 500f;

    public SavingsAccount(float balance, Customer customer, float interestRate) {
        super("Savings", balance >= MIN_BALANCE ? balance : MIN_BALANCE, customer);
        this.interestRate = interestRate;
    }

    public void displayAccountDetails() {
        System.out.printf("Account Number: %s%n", getAccountNumber());  // Access account number using the getter method
        System.out.printf("Balance: ₹%.2f%n", balance);
        System.out.printf("Interest Rate: %.2f%%\n", interestRate);
    }

    public void calculateInterest() {
        float interest = balance * interestRate / 100;
        balance += interest;
        System.out.printf("☑ Interest ₹%.2f added. New Balance: ₹%.2f%n", interest, balance);
    }
}
```

```java
package com.hexaware.hmbank.bean;

public class ZeroBalanceAccount extends Account {
    public ZeroBalanceAccount(Customer customer) {
        super("ZeroBalance", 0f, customer);
    }

    public void calculateInterest() {
        System.out.println("ⓘ No interest for zero balance accounts.");
    }
}
```

4. Create ICustomerServiceProvider interface/abstract class with following functions:

□get_account_balance(account_number: long): Retrieve the balance of an account given its account number. should return the current balance of account.

□deposit(account_number: long, amount: float): Deposit the specified amount into the account. Should return the current balance of account.

□withdraw(account_number: long, amount: float): Withdraw the specified amount from the account. Should return the current balance of account. A savings account should maintain

a minimum balance and checking if the withdrawal violates the minimum balance rule.

transfer(from_account_number: long, to_account_number: int, amount: float): Transfer money from one account to another.

getAccountDetails(account_number: long): Should return the account and customer details.

```java
package com.hexaware.hmbank.service;

public interface ICustomerServiceProvider {
    float getAccountBalance(long accNo);
    float deposit(long accNo, float amount);
    float withdraw(long accNo, float amount);
    void transfer(long fromAcc, long toAcc, float amount);
    void getAccountDetails(long accNo);
}
```

5. Create IBankServiceProvider interface/abstract class with following functions: create_account(Customer customer, long accNo, String accType, float balance): Create a new bank account for the given customer with the initial balance.

listAccounts():Account[] accounts: List all accounts in the bank.

calculateInterest(): the calculate_interest() method to calculate interest based on the balance and interest rate.

```java
package com.hexaware.hmbank.service;

import com.hexaware.hmbank.bean.Customer;

public interface IBankServiceProvider {
    void createAccount(Customer customer, String accType, float balance);
    void listAccounts();
    void calculateInterest();
}
```

6. Create CustomerServiceProviderImpl class which implements ICustomerServiceProvider provide all implementation methods.

```java
package com.hexaware.hmbank.service;

import com.hexaware.hmbank.bean.*;
import java.util.*;

public class CustomerServiceProviderImpl implements ICustomerServiceProvider {
    protected Map<Long, Account> accounts = new HashMap<>();

    protected Account findAccount(long accNo) {
        return accounts.get(accNo);
    }

    @Override
    public float getAccountBalance(long accNo) {
        Account acc = findAccount(accNo);
        if (acc != null) return acc.getBalance();
        System.out.println("❌ Account not found.");
        return -1;
    }

    @Override
    public float deposit(long accNo, float amount) {
        Account acc = findAccount(accNo);
        if (acc != null) {
            acc.setBalance(acc.getBalance() + amount);
            return acc.getBalance();
        }
        System.out.println("❌ Account not found.");
        return -1;
    }

    @Override
    public float withdraw(long accNo, float amount) {
        Account acc = findAccount(accNo);
        if (acc instanceof SavingsAccount) {
            if (acc.getBalance() - amount >= 500) {
                acc.setBalance(acc.getBalance() - amount);
            } else {
                System.out.println("❌ Minimum balance of ₹500 must be maintained.");
                return acc.getBalance();
            }
        } else if (acc instanceof CurrentAccount current) {
            if (!current.withdraw(amount)) return acc.getBalance();
```

```
    } else if (acc != null) {
        acc.setBalance(acc.getBalance() - amount);
    } else {
        System.out.println("❌ Account not found.");
        return -1;
    }
    return acc.getBalance();
}

@Override
public void transfer(long fromAcc, long toAcc, float amount) {
    float bal = withdraw(fromAcc, amount);
    if (bal != -1) deposit(toAcc, amount);
}

@Override
public void getAccountDetails(long accNo) {
    Account acc = findAccount(accNo);
    if (acc != null) acc.displayAccountDetails();
    else System.out.println("❌ Account not found.");
}
```

7. Create BankServiceProviderImpl class which inherits from CustomerServiceProviderImpl and implements IBankServiceProvider .

Attributes accountList: Array of **Accounts** to store any account objects.

- branchName and branchAddress as String objects

```java
package com.hexaware.hmbank.service;

import com.hexaware.hmbank.bean.*;

public class BankServiceProviderImpl extends CustomerServiceProviderImpl implements IBankServiceProvider {
    private String branchName = "HM Bank";
    private String branchAddress = "Chennai";

    @Override
    public void createAccount(Customer customer, String accType, float balance) {
        Account acc = null;
        switch (accType.toLowerCase()) {
            case "savings" -> acc = new SavingsAccount(balance, customer, 4.5f);
            case "current" -> acc = new CurrentAccount(balance, customer, 5000f);
            case "zerobalance" -> acc = new ZeroBalanceAccount(customer);
            default -> System.out.println("❌ Invalid account type");
        }
        if (acc != null) {
            accounts.put(acc.getAccountNumber(), acc);
            System.out.println("☑ Account created successfully. Account No: " + acc.getAccountNumber());
        }
    }

    @Override
    public void listAccounts() {
        accounts.values().forEach(acc -> {
            acc.displayAccountDetails();
            System.out.println("------------------------");
        });
    }

    @Override
    public void calculateInterest() {
        accounts.values().forEach(acc -> acc.calculateInterest());
    }
}
```

8. Create BankApp class and perform following operation:

- main method to simulate the banking system. Allow the user to interact with the system by entering choice from menu such as "create_account", "deposit", "withdraw", "get_balance", "transfer", "getAccountDetails", "ListAccounts" and "exit."
- create_account should display sub menu to choose type of accounts and repeat this operation until user exit.

```java
package com.hexaware.hmbank.app;

import com.hexaware.hmbank.bean.Customer;
import com.hexaware.hmbank.bean.SavingsAccount;
import com.hexaware.hmbank.bean.CurrentAccount;
import com.hexaware.hmbank.service.CustomerServiceProviderImpl;
import java.util.Scanner;

public class BankApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        CustomerServiceProviderImpl service = new CustomerServiceProviderImpl();

        while (true) {
            System.out.println("\nBank Menu:");
            System.out.println("1. Create Account");
            System.out.println("2. Deposit");
            System.out.println("3. Withdraw");
            System.out.println("4. Get Balance");
            System.out.println("5. Transfer");
            System.out.println("6. Get Account Details");
            System.out.println("7. List All Accounts");
            System.out.println("8. Exit");
            System.out.print("Enter your choice: ");
            int choice = Integer.parseInt(scanner.nextLine());

            switch (choice) {
                case 1: // Create Account
                    System.out.print("Enter name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter email: ");
                    System.out.print("Enter phone: ");
                    String phone = scanner.nextLine();
                    Customer customer = new Customer(name, email, phone);

                    System.out.println("Choose account type:");
                    System.out.println("1. Savings");
                    System.out.println("2. Current");
                    int accType = Integer.parseInt(scanner.nextLine());

                    if (accType == 1) {
                        System.out.print("Enter balance: ₹");
                        float balance = Float.parseFloat(scanner.nextLine());
                        System.out.print("Enter interest rate: ");
                        float interestRate = Float.parseFloat(scanner.nextLine());
                        SavingsAccount savingsAccount = new SavingsAccount(balance, customer, interestRate);
                        service.addAccount(savingsAccount);
                    } else if (accType == 2) {
                        System.out.print("Enter balance: ₹");
                        float balance = Float.parseFloat(scanner.nextLine());
                        CurrentAccount currentAccount = new CurrentAccount(balance, customer);
                        service.addAccount(currentAccount);
                    }
                    break;

                case 2: // Deposit
                    System.out.print("Enter account number: ");
                    long accNoDep = Long.parseLong(scanner.nextLine());
                    System.out.print("Enter deposit amount: ₹");
                    float depositAmount = Float.parseFloat(scanner.nextLine());
                    System.out.println("New Balance: ₹" + service.deposit(accNoDep, depositAmount));
                    break;
```

```java
                case 3: // Withdraw
                    System.out.print("Enter account number: ");
                    long accNoWith = Long.parseLong(scanner.nextLine());
                    System.out.print("Enter withdrawal amount: ₹");
                    float withdrawAmount = Float.parseFloat(scanner.nextLine());
                    System.out.println("New Balance: ₹" + service.withdraw(accNoWith, withdrawAmount));
                    break;

                case 4: // Get Balance
                    System.out.print("Enter account number: ");
                    long accNoBalance = Long.parseLong(scanner.nextLine());
                    System.out.println("Balance: ₹" + service.getAccountBalance(accNoBalance));
                    break;

                case 5: // Transfer
                    System.out.print("Enter source account number: ");
                    long fromAcc = Long.parseLong(scanner.nextLine());
                    System.out.print("Enter target account number: ");
                    long toAcc = Long.parseLong(scanner.nextLine());
                    System.out.print("Enter transfer amount: ₹");
                    float transferAmount = Float.parseFloat(scanner.nextLine());
                    service.transfer(fromAcc, toAcc, transferAmount);
                    break;

                case 6: // Get Account Details
                    System.out.print("Enter account number: ");
                    long accNoDetails = Long.parseLong(scanner.nextLine());
                    service.getAccountDetails(accNoDetails);
                    break;
```

```java
                case 6: // Get Account Details
                    System.out.print("Enter account number: ");
                    long accNoDetails = Long.parseLong(scanner.nextLine());
                    service.getAccountDetails(accNoDetails);
                    break;

                case 7: // List All Accounts
                    service.listAllAccounts();
                    break;

                case 8: // Exit
                    System.out.println("Exiting...");
                    scanner.close();
                    return;

                default:
                    System.out.println("❌ Invalid choice, try again.");
            }
        }
    }
}
```

Problems @ Javadoc Declaration Console ✕ Install Java 24 Support

```
BankApp (1) [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jre\bin\javaw
7. List All Accounts
8. Exit
Enter your choice: 1
Enter name: marry
Enter email: agvahg@gmail.com
Enter phone: 1234567897
Choose account type:
1. Savings
```

9. Place the interface/abstract class in service package and interface/abstract class implementation class, account class in bean package and Bank class in app package.

```java
package com.hexaware.hmbank.service;

import com.hexaware.hmbank.bean.Account;
import com.hexaware.hmbank.bean.SavingsAccount;
import com.hexaware.hmbank.bean.CurrentAccount;

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

public class CustomerServiceProviderImpl implements ICustomerServiceProvider {
    protected Map<Long, Account> accounts = new HashMap<>();

    // Add account to the map
    public void addAccount(Account acc) {
        accounts.put(acc.getAccountNumber(), acc); // Ensure Account Number is a Long
    }

    @Override
    public float getAccountBalance(long accNo) {
        Account acc = accounts.get(accNo);
        if (acc != null) {
            return acc.getBalance();
        }
        System.out.println("❌ Account not found.");
        return -1;
    }

    @Override
    public float deposit(long accNo, float amount) {
        Account acc = accounts.get(accNo);
        if (acc != null) {
            acc.setBalance(acc.getBalance() + amount);
            return acc.getBalance();
        }
        System.out.println("❌ Account not found.");
        return -1;
    }

    @Override
    public float withdraw(long accNo, float amount) {
        Account acc = accounts.get(accNo);
        if (acc instanceof SavingsAccount) {
            if (acc.getBalance() - amount >= 500) {
```

```
            acc.setBalance(acc.getBalance() - amount);
        } else {
            System.out.println("❌ Minimum balance of ₹500 must be maintained.");
            return acc.getBalance();
        }
    } else if (acc instanceof CurrentAccount) {
        CurrentAccount currentAcc = (CurrentAccount) acc;
        if (!currentAcc.withdraw(amount)) {
            return acc.getBalance();
        }
    } else if (acc != null) {
        acc.setBalance(acc.getBalance() - amount);
    } else {
        System.out.println("❌ Account not found.");
        return -1;
    }
    return acc.getBalance();
}

@Override
public void transfer(long fromAcc, long toAcc, float amount) {
    float bal = withdraw(fromAcc, amount);
    if (bal != -1) {
        deposit(toAcc, amount);
    }
}

@Override
public void getAccountDetails(long accNo) {
    Account acc = accounts.get(accNo);
    if (acc != null) {
        acc.displayAccountDetails();
    } else {
        System.out.println("❌ Account not found.");
    }
}

public void listAllAccounts() {
    if (accounts == null || accounts.isEmpty()) {
        System.out.println("⚠ No accounts found.");
        return;
    }
    for (Account acc : accounts.values()) {
```

```
        for (Account acc : accounts.values()) {
            System.out.println("Account No: " + acc.getAccountNumber() +
                               ", Type: " + acc.getAccountType() +
                               ", Balance: ₹" + acc.getBalance());
        }
    }
}
```

```java
public class Account {
    private Long accountNumber;  // accountNumber as Long type
    private String accountType;
    private float balance;
    private Customer customer;

    public Account(String accountType, float balance, Customer customer) {
        this.accountType = accountType;
        this.balance = balance;
        this.customer = customer;
        this.accountNumber = generateAccountNumber();  // Generate account number upon account creation
    }

    // Method to get the account number (return type Long)
    public Long getAccountNumber() {
        return accountNumber;
    }

    public String getAccountType() {
        return accountType;
    }

    public float getBalance() {
        return balance;
    }

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

    public Customer getCustomer() {
```

```java
    // Method to display account details
    public void displayAccountDetails() {
        System.out.println("Account No: " + accountNumber + ", Type: " + accountType + ", Balance: ₹" + balance);
    }

    private Long generateAccountNumber() {
        return System.currentTimeMillis();
    }
}
```

10. Should display appropriate message when the account number is not found and insufficient fund or any other wrong information provided.

```
3. Withdraw
4. Get Balance
5. Transfer
6. Get Account Details
7. List All Accounts
8. Exit
Enter your choice: 2
Enter account number: 1001
Enter deposit amount: ₹4000
✗ Account not found.
New Balance: ₹-1.0
```

Task 12: Exception Handling

throw the exception whenever needed and Handle in main method,

1. InsufficientFundException throw this exception when user try to withdraw amount or transfer amount to another account and the account runs out of money in the account.

2. InvalidAccountException throw this exception when user entered the invalid account number when tries to transfer amount, get account details classes.

3. OverDraftLimitExcededException thow this exception when current account customer try to with draw amount from the current account.

4. NullPointerException handle in main method.

Throw these exceptions from the methods in HMBank class. Make necessary changes to accommodate these exception in the source code. Handle all these exceptions from the main program.

```java
package com.hexaware.hmbank.service;

import com.hexaware.hmbank.exception.InvalidAccountException;
import com.hexaware.hmbank.exception.InsufficientFundException;
import com.hexaware.hmbank.exception.OverDraftLimitExceededException;

public interface ICustomerServiceProvider {
    float getAccountBalance(long accNo) throws InvalidAccountException;  // Throws InvalidAccountException

    float deposit(long accNo, float amount) throws InvalidAccountException;

    float withdraw(long accNo, float amount) throws InsufficientFundException, InvalidAccountException, OverDraftLimitExceededException;

    void transfer(long fromAcc, long toAcc, float amount) throws InsufficientFundException, InvalidAccountException;

    void getAccountDetails(long accNo) throws InvalidAccountException;
}
```

```
Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get Account Balance
6. Get Account Details
7. Exit
Enter your choice: 5
Enter account number: 1001
❌ A null pointer exception occurred. Please check your input.
```

Task 13: Collection

1. From the previous task change the HMBank attribute Accounts to List of Accounts and perform the same operation.

```java
import com.hexaware.hmbank.bean.Account;
import com.hexaware.hmbank.exception.InvalidAccountException;
import com.hexaware.hmbank.exception.InsufficientFundException;

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

public class CustomerServiceProviderImpl implements ICustomerServiceProvider {
    protected Map<Long, Account> accounts = new HashMap<>();

    // Add an account to the system
    public void addAccount(Account account) {
        accounts.put(account.getAccountNumber(), account);
    }

    // Method to get account details
    @Override
    public void getAccountDetails(long accNo) throws InvalidAccountException {
        Account acc = accounts.get(accNo);
        if (acc == null) {
            throw new InvalidAccountException("❌ Account not found with Account Number: " + accNo);
        }
        acc.displayAccountDetails();
    }

    // Method to get account balance
    @Override
    public float getAccountBalance(long accNo) throws InvalidAccountException {
        Account acc = accounts.get(accNo);
```

```
        Account acc = accounts.get(accNo);
        if (acc == null) {
            throw new InvalidAccountException("❌ Account not found with Account Number: " + accNo);
        }
        return acc.getBalance();
    }

    // Method to deposit amount into the account
    @Override
    public void deposit(long accNo, float amount) throws InvalidAccountException {
        Account acc = accounts.get(accNo);
        if (acc == null) {
            throw new InvalidAccountException("❌ Account not found with Account Number: " + accNo);
        }
        acc.setBalance(acc.getBalance() + amount);
    }

    // Method to withdraw amount from the account
    @Override
    public void withdraw(long accNo, float amount) throws InvalidAccountException, InsufficientFundException {
        Account acc = accounts.get(accNo);
        if (acc == null) {
            throw new InvalidAccountException("❌ Account not found with Account Number: " + accNo);
        }
        if (acc.getBalance() < amount) {
            throw new InsufficientFundException("❌ Insufficient funds in Account Number: " + accNo);
        }
        acc.setBalance(acc.getBalance() - amount);
    }

    // Method to transfer money between two accounts
```

```
62      @Override
63      public void transfer(long fromAcc, long toAcc, float amount) throws InvalidAccountException, InsufficientFundException {
64          Account from = accounts.get(fromAcc);
65          Account to = accounts.get(toAcc);
66          if (from == null || to == null) {
67              throw new InvalidAccountException("❌ Invalid account(s) for transfer.");
68          }
69          if (from.getBalance() < amount) {
70              throw new InsufficientFundException("❌ Insufficient funds in Account Number: " + fromAcc);
71          }
72          from.setBalance(from.getBalance() - amount);
73          to.setBalance(to.getBalance() + amount);
74      }
75  }
76
```

```
Problems  Javadoc  Declaration  Console ×  Install Java 24 Support
BankApp (1) [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jre\bin\javaw.exe  (14-Apr-2025, 8:57:33
5. Exit
Choose an option: 1
Enter Account Number: 1001
Account Number: 1001
Account Type: Savings
Balance: ₹1000.0
```

2. From the previous task change the HMBank attribute Accounts to Set of Accounts and perform the same operation.
- Avoid adding duplicate Account object to the set.
- Create Comparator<Account> object to sort the accounts based on customer name when listAccounts() method called.

```java
public class BankApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        CustomerServiceProviderImpl service = new CustomerServiceProviderImpl();

        // Add some accounts for testing
        Account acc1 = new Account(1001, "Savings", 1000f, "John Doe");
        Account acc2 = new Account(1002, "Current", 2000f, "Jane Smith");
        Account acc3 = new Account(1003, "Savings", 5000f, "Alice Johnson");

        // Call addAccount() method to add accounts
        service.addAccount(acc1);
        service.addAccount(acc2);
        service.addAccount(acc3);

        while (true) {
            System.out.println("\n=== HMBank ===");
            System.out.println("1. Get Account Details");
            System.out.println("2. Deposit");
            System.out.println("3. Withdraw");
            System.out.println("4. Transfer");
            System.out.println("5. List Accounts");
            System.out.println("6. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();

            try {
                switch (choice) {
                    case 1:
                        System.out.print("Enter Account Number: ");
                        long accNo = scanner.nextLong();
                        service.getAccountDetails(accNo);
                        break;
                    case 2:
                        System.out.print("Enter Account Number: ");
                        accNo = scanner.nextLong();
                        System.out.print("Enter Amount to Deposit: ₹");
                        float depositAmount = scanner.nextFloat();
                        service.deposit(accNo, depositAmount);
                        System.out.println("☑ Deposit Successful");
                        break;
                    case 3:
                        System.out.print("Enter Account Number: ");
                        accNo = scanner.nextLong();
                        System.out.print("Enter Amount to Withdraw: ₹");
                        float withdrawAmount = scanner.nextFloat();
                        service.withdraw(accNo, withdrawAmount);
                        System.out.println("☑ Withdrawal Successful");
                        break;
                    case 4:
                        System.out.print("Enter From Account Number: ");
                        long fromAcc = scanner.nextLong();
                        System.out.print("Enter To Account Number: ");
                        long toAcc = scanner.nextLong();
                        System.out.print("Enter Amount to Transfer: ₹");
                        float transferAmount = scanner.nextFloat();
                        service.transfer(fromAcc, toAcc, transferAmount);
                        System.out.println("☑ Transfer Successful");
                        break;
                    case 5:
                        service.listAccounts();
                        break;
```

```
                    break;
                case 5:
                    service.listAccounts();
                    break;
                case 6:
                    System.out.println("Exiting...");
                    scanner.close();
                    return;
                default:
                    System.out.println("❌ Invalid option! Try again.");
            }
        } catch (InvalidAccountException | InsufficientFundException e) {
            System.out.println(e.getMessage());
        } catch (Exception e) {
            System.out.println("❌ An unexpected error occurred: " + e.getMessage());
        }
    }
}
```

```
Problems  Javadoc  Declaration  Console ×  Install Java 24 Support
BankApp (1) [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jre\bin\javaw.exe (1
6. Exit
Choose an option: 5
Listing Accounts sorted by Customer Name:
Account No: 1003, Customer Name: Alice Johnson, Type: Savings, Balance: ₹5000.0
Account No: 1002, Customer Name: Jane Smith, Type: Current, Balance: ₹2000.0
Account No: 1001, Customer Name: John Doe, Type: Savings, Balance: ₹1000.0
```

3. From the previous task change the HMBank attribute Accounts to HashMap of Accounts and perform the same operation.

```
public class HMBank {
    // ☑ Renamed accountMap to accounts as per your new task
    Map<Long, Account> accounts = new HashMap<>();

    public void createAccount(Account account) {
        if (accounts.containsKey(account.getAccountNumber())) {
            System.out.println("Account already exists.");
        } else {
            accounts.put(account.getAccountNumber(), account);
            System.out.println("☑ Account created for: " + account.getCustomer().getName());
        }
    }

    public void listAccounts() {
        List<Account> sortedAccounts = new ArrayList<>(accounts.values());
        sortedAccounts.sort(Comparator.comparing(a -> ((Account) a).getCustomer().getName()));
        System.out.println("🔍 Listing accounts sorted by first name:");
        for (Account acc : sortedAccounts) {
            System.out.println(acc);
        }
    }

    public void deposit(long accNo, float amount) {
        Account acc = accounts.get(accNo);
        if (acc != null) {
            acc.deposit(amount);
        } else {
            System.out.println("❌ Account not found.");
        }
    }

    public void withdraw(long accNo, float amount) {
        Account acc = accounts.get(accNo);
        if (acc != null) {
            acc.withdraw(amount);
        } else {
            System.out.println("❌ Account not found.");
        }
    }

    public Account getAccount(long accNo) {
        return accounts.get(accNo);
    }
```

```java
package com.hexaware.hmbank.app;
import com.hexaware.hmbank.bean.Account;
import com.hexaware.hmbank.bean.Customer;
import com.hexaware.hmbank.entity.HMBank;

public class BankApp {
    public static void main(String[] args) {

        Customer c1 = new Customer(1, "Zara");
        Customer c2 = new Customer(2, "Amit");
        Customer c3 = new Customer(3, "Bala");


        Account a1 = new Account("Savings", 5000, c1);
        Account a2 = new Account("Current", 3000, c2);
        Account a3 = new Account("Savings", 7000, c3);


        System.out.println("◦ Using HashMap:");
        HMBank bank = new HMBank();
        bank.createAccount(a1);
        bank.createAccount(a2);
        bank.createAccount(a3);
        bank.createAccount(a2);


        bank.deposit(a1.getAccountNumber(), 2000);
        bank.withdraw(a2.getAccountNumber(), 1000);

        
        bank.listAccounts();
    }
}
```

```
◦ Using HashMap:
☑ Account created for: Zara
☑ Account created for: Amit
☑ Account created for: Bala
Account already exists.
🔍 Listing accounts sorted by first name:
Account{accNo=1002, type='Current', balance=2000.0, Customer{id=2, name='Amit'}}
Account{accNo=1003, type='Savings', balance=7000.0, Customer{id=3, name='Bala'}}
Account{accNo=1001, type='Savings', balance=7000.0, Customer{id=1, name='Zara'}}
```

11. Create the DBUtil class and add the following method.

□static getDBConn():Connection Establish a connection to the database and return Connection reference.

```java
package com.hexaware.hmbank.util;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBUtil {
    private static final String URL = "jdbc:mysql://localhost:3306/hmbank";
    private static final String USER = "root";
    private static final String PASSWORD = "fahi";

    public static Connection getDBConn() throws SQLException {
        try {
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (SQLException e) {
            System.out.println("❌ Database connection failed: " + e.getMessage());
            throw e;
        }
    }

    public static void main(String[] args) {
        try {
            Connection conn = getDBConn();
            System.out.println("☑ Connected successfully!");
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
<terminated> DBUtil [Java Application] C:\Users\nisha\.p2\pool\plu
☑ Connected successfully!
```

12. Create BankApp class and perform following operation:

- main method to simulate the banking system. Allow the user to interact with the system by entering choice from menu such as "create_account", "deposit", "withdraw", "get_balance", "transfer", "getAccountDetails", "ListAccounts", "getTransactions" and "exit."

- create_account should display sub menu to choose type of accounts and repeat this operation until user exit.

```
import com.hexaware.hmbank.bean.Account;
import com.hexaware.hmbank.service.BankServiceImpl;
import com.hexaware.hmbank.service.IBankService;

import java.util.Scanner;

public class BankApp {
    public static void main(String[] args) {
        IBankService bank = new BankServiceImpl();
        Scanner sc = new Scanner(System.in);

        while (true) {
            System.out.println("\n◦ Choose an operation:");
            System.out.println("1. create_account\n2. deposit\n3. withdraw\n4. get_balance\n5. transfer");
            System.out.println("6. getAccountDetails\n7. ListAccounts\n8. exit");
            System.out.print("Enter choice: ");
            int choice = sc.nextInt();


       try {
            switch (choice) {
                case 1:
                    while (true) {
                        System.out.print("Enter account type (Savings/Current) or 'exit': ");
                        String type = sc.next();
                        if (type.equalsIgnoreCase("exit")) break;
                        System.out.print("Enter account number: ");
                        long accNo = sc.nextLong();
                        System.out.print("Enter initial balance: ");
                        float bal = sc.nextFloat();
                        System.out.print("Enter holder name: ");
                        sc.nextLine();  // flush newline
                        String name = sc.nextLine();
                        bank.createAccount(new Account(accNo, type, bal, name));
                    }
                    break;
```

```
            case 2:
                System.out.print("Enter account number: ");
                bank.deposit(sc.nextLong(), sc.nextFloat());
                break;
            case 3:
                System.out.print("Enter account number: ");
                bank.withdraw(sc.nextLong(), sc.nextFloat());
                break;
            case 4:
                System.out.print("Enter account number: ");
                float bal = bank.getBalance(sc.nextLong());
                if (bal != -1) System.out.println("Balance: ₹" + bal);
                break;
            case 5:
                System.out.print("From Account: ");
                long from = sc.nextLong();
                System.out.print("To Account: ");
                long to = sc.nextLong();

                 long to = sc.nextLong();
                 System.out.print("Amount: ");
                 float amt = sc.nextFloat();
                 bank.transfer(from, to, amt);
                 break;
             case 6:
                 System.out.print("Enter account number: ");
                 Account acc = bank.getAccountDetails(sc.nextLong());
                 System.out.println(acc != null ? acc : "❌ Not found");
                 break;
             case 7:
                 bank.listAccounts();
                 break;
             case 8:
                 System.out.println("🔚 Exiting. Thank you!");
                 System.exit(0);
             default:
                 System.out.println("❌ Invalid choice.");
```

```
                }
            } catch (Exception e) {
                System.out.println("❌ Error: " + e.getMessage());
            }
        }
    }
}

```

Javadoc | Declaration | Console × | Install Java 24 Support

BankApp (3) [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20

```
◦ Choose an operation:
1. create_account
2. deposit
3. withdraw
4. get_balance
5. transfer
6. getAccountDetails
7. ListAccounts
8. exit
Enter choice: 4
Enter account number: 1001
❌ Account not found!
```

13. Place the interface/abstract class in service package and interface/abstract class implementation class, account class in bean package and Bank class in app package.

```java
package com.hexaware.hmbank.service;

import com.hexaware.hmbank.bean.Account;

public interface IBankService {
    void createAccount(Account account);
    void deposit(long accNo, float amount);
    void withdraw(long accNo, float amount);
    void transfer(long fromAcc, long toAcc, float amount);
    float getBalance(long accNo);
    Account getAccountDetails(long accNo);
    void listAccounts();
}
```

14. Should throw appropriate exception as mentioned in above task along with handle SQLException.

```java
package com.hexaware.hmbank.bean;

public class Account {
    private long accountNo;
    private String accountHolder;
    private double balance;

    public Account(long accountNo, String accountHolder, double balance) {
        this.accountNo = accountNo;
        this.accountHolder = accountHolder;
        this.balance = balance;
    }

    // Getters and Setters
    public long getAccountNo() { return accountNo; }
    public String getAccountHolder() { return accountHolder; }
    public double getBalance() { return balance; }

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

```java
package com.hexaware.hmbank.exception;

public class InsufficientFundException extends Exception {
    public InsufficientFundException(String message) {
        super(message);
    }
}
```

```java
package com.hexaware.hmbank.exception;

public class InvalidAccountException extends Exception {
    public InvalidAccountException(String message) {
        super(message);
    }
}
```

```java
package com.hexaware.hmbank.exception;

public class OverDraftLimitExceededException extends Exception {
    private static final long serialVersionUID = 1L;

    public OverDraftLimitExceededException(String message) {
        super(message);
    }
}
```