Bank Application

Objective:
Develop a Java program for a bank application that performs deposit, withdrawal, transfer, loan application,
and balance check operations with proper input validation and error handling.
The program should utilize a Bank interface, BankAccount and Customer classes, and an ATM class to demonstrate the application's functionality.
Use custom exceptions, switch-case statements, try-with-resources, and multi-catch blocks for error handling.

Requirements:

Custom Exceptions:

-> class InsufficientFundsException(Checked):
  Constructor:
  Parameterized Constructor with String as parameter
  Purpose: Thrown when there are not enough funds for a withdrawal or transfer.

-> class InvalidAmountException(Unchecked):
  Constructor:
  Parameterized Constructor with String as parameter
  Purpose: Thrown when an invalid amount is entered for any transaction.

-> class AccountNotFoundException(Checked):
  Constructor:
  Parameterized Constructor with String as parameter
  Purpose: Thrown when an account is not found during a transfer.

-> class LoanNotAllowedException(Unchecked):
  Constructor:
  Parameterized Constructor with String as parameter
  Purpose:
  Thrown when the loan amount exceeds the allowed limit.

Bank Interface:

Methods:

method name: deposit
return type : void
parameters : double amount

throws InvalidAmountException
Deposits the specified amount into the account. Throws InvalidAmountException if the amount is less than or equal to zero.

Method name: withdraw
return type : void
parameters : double amount
throws InsufficientFundsException, InvalidAmountException:
Withdraws the specified amount from the account. Throws InsufficientFundsException if the amount is greater than the account balance.
Throws InvalidAmountException if the amount is less than or equal to zero.

Method name: transfer
return type : void
parameters : (BankAccount toAccount, double amount)
throws InsufficientFundsException, AccountNotFoundException, InvalidAmountException:
Transfers the specified amount to another account. Throws InsufficientFundsException if the amount is greater than the account balance.
Throws AccountNotFoundException if the target account is not found.
Throws InvalidAmountException if the amount is less than or equal to zero.

Method name: applyForLoan
return type : void
parameters : (double amount)
throws LoanNotAllowedException, InvalidAmountException:
Applies for a loan of the specified amount(50000). Throws LoanNotAllowedException if the loan amount exceeds the allowed limit(50000) or if the balance is less than specified amount(50000).
Throws InvalidAmountException if the amount is less than or equal to zero.

Method name: getBalance
return type : double
Returns the current balance of the account.

-> BankAccount Class(BLC):
  Attributes:
  accountNumber: long: private
  balance: double: private

Constructor:
Constructor to initialize accountNumber and initial balance.

Implement all methods from the Bank interface.
deposit(double amount): Increases the account balance by the specified amount. Throws InvalidAmountException if the amount is less than or equal to zero.

withdraw(double amount): Decreases the account balance by the specified amount. Throws InsufficientFundsException if the amount is greater than the account balance. Throws InvalidAmountException if the amount is less than or equal to zero.

transfer(BankAccount toAccount, double amount): Transfers the specified amount to another account. Throws InsufficientFundsException if the amount is greater than the account balance. Throws AccountNotFoundException if the target account is not found(i,e if toAccount is null). Throws InvalidAmountException if the amount is less than or equal to zero.

applyForLoan(double amount): Allows the user to apply for a loan. Throws LoanNotAllowedException if the loan amount exceeds the allowed limit. Throws InvalidAmountException if the amount is less than or equal to zero.

getBalance(): Returns the current balance of the account.

-> Customer Class(BLC):
Attributes:
name: String: private
account: BankAccount: private //HAS-A

Constructor:
Constructor to initialize the customer's name and account.

Methods : Getter methods for name & account

-> ATM Class(ELC):

a) create Main method.

b) In main method create object like bellow.

```
BankAccount acc1 = new BankAccount(1111, 60000);
    BankAccount acc2 = new BankAccount(2222, 3000);
    Customer customer1 = new Customer("Alice", acc1);
```

Customer customer2 = new Customer("Bob", acc2);

c) Display menu using switch-case:
Select an option :
1. Deposit
2. Withdraw
3. Transfer
4. Loan Application
5. Check Balance
6. Exit

d) Read the name of the customer from user
Match name to either customer1 or customer2
Perform the operation on the matched customer's account

e) Uses try-catch blocks for error handling.

TEST CASE 1 :

Input :
Enter your option : 3
Enter Customer name : alice
Enter amount to transfer : 11000
Output :
Deposit successful. New balance: 14000.0
Transfer successful.

TEST CASE 2 :

Input :
Enter your option : 4
Enter Customer name : alice
Enter loan amount to apply: 100000
Output :
Invalid operation: Loan not allowed. Either amount exceeds limit or balance is too low.

TEST CASE 3 :

Input :

Enter your option : 1
Enter Customer name : alice
Enter amount to deposit: 15000
Output :
Deposit successful. New balance: 64000.0

TEST CASE 4 :

Input :
Enter your option : 4
Enter Customer name : alice
Enter loan amount to apply: 10000
Output :
Loan approved. New balance: 74000.0

TEST CASE 5 :

Input :
Enter your option : 6
Output :
Thank you for using the ATM. Goodbye!

=> NOTE : TEST REMAINING ALL VALID & INVALID TEST CASES BY YOUR OWN AND PRACTICE WELL....