Bank Account System (Features: create account,Deposit, withdraw, check balance., (Concepts' Encapsulation, Exception Handling)

```
package bankaccount;

import java.util.*;

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

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

//Encapsulated BankAccount class
class BankAccount {
 private final String accountNumber;
 private final String accountHolderName;
```

```
 private double balance;


 public BankAccount(String accountNumber, String
accountHolderName, double initialDeposit) throws
InvalidAmountException {
    if (initialDeposit < 0) throw new InvalidAmountException("Initial
deposit cannot be negative");
    this.accountNumber = accountNumber;
    this.accountHolderName = accountHolderName;
    this.balance = initialDeposit;
 }

 // Getters (no setters for accountNumber/accountHolderName to keep
them immutable)
 public String getAccountNumber() {
    return accountNumber;
 }

 public String getAccountHolderName() {
    return accountHolderName;
 }

 public double getBalance() {
    return balance;
 }
```

```
 // Deposit method with validation
 public void deposit(double amount) throws InvalidAmountException
{
    if (amount <= 0) throw new InvalidAmountException("Deposit
amount must be positive");
    balance += amount;
 }

 // Withdraw method with validation and custom exception for
insufficient funds
 public void withdraw(double amount) throws
InvalidAmountException, InsufficientFundsException {
    if (amount <= 0) throw new
InvalidAmountException("Withdrawal amount must be positive");
    if (amount > balance) throw new
InsufficientFundsException("Insufficient funds. Available balance: "
+ balance);
    balance -= amount;
 }

 @Override
 public String toString() {
    return String.format("Account[%s] - %s - Balance: %.2f",
accountNumber, accountHolderName, balance);
 }
```

```
}

//Manager that holds accounts and provides create/search operations
class Bank {
 private final Map<String, BankAccount> accounts = new HashMap<>();
 private final Random rng = new Random();

 // Create a unique account number (simple implementation)
 private String generateAccountNumber() {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < 10; i++) sb.append(rng.nextInt(10));
    String accNum = sb.toString();
    if (accounts.containsKey(accNum)) return generateAccountNumber();
    return accNum;
 }


 public BankAccount createAccount(String holderName, double initialDeposit) throws InvalidAmountException {
    String accNum = generateAccountNumber();
    BankAccount acc = new BankAccount(accNum, holderName, initialDeposit);
    accounts.put(accNum, acc);
```

```
    return acc;
 }


 public Optional<BankAccount> findAccount(String accountNumber)
{
    return Optional.ofNullable(accounts.get(accountNumber));
 }


 public Collection<BankAccount> listAccounts() {
    return accounts.values();
 }
}


public class BankAccountSystem {
 private static final Scanner scanner = new Scanner(System.in);
 private static final Bank bank = new Bank();

 public static void main(String[] args) {
    System.out.println("Welcome to Simple Bank Account System");
    boolean running = true;
```

```
while (running) {
    printMenu();
    System.out.print("Choose option: ");
    String choice = scanner.nextLine().trim();

    switch (choice) {
        case "1":
            createAccountFlow();
            break;
        case "2":
            depositFlow();
            break;
        case "3":
            withdrawFlow();
            break;
        case "4":
            checkBalanceFlow();
            break;
        case "5":
            listAllAccounts();
            break;
        case "0":
            running = false;
            System.out.println("Exiting... Thank you!");
```

```
                break;
            default:
                System.out.println("Invalid option. Try again.");
        }

        System.out.println();
    }

    scanner.close();
}

private static void printMenu() {
    System.out.println("-----------------------------------------");
    System.out.println("1. Create account");
    System.out.println("2. Deposit");
    System.out.println("3. Withdraw");
    System.out.println("4. Check balance");
    System.out.println("5. List all accounts (demo)");
    System.out.println("0. Exit");
    System.out.println("-----------------------------------------");
}

private static void createAccountFlow() {
    try {
```

```
        System.out.print("Enter account holder name: ");
        String name = scanner.nextLine().trim();
        System.out.print("Enter initial deposit: ");
        double initial =
Double.parseDouble(scanner.nextLine().trim());

        BankAccount acc = bank.createAccount(name, initial);
        System.out.println("Account created successfully!");
        System.out.println("Account Number: " +
acc.getAccountNumber());
        System.out.println(acc);
    } catch (InvalidAmountException e) {
        System.out.println("Error: " + e.getMessage());
    } catch (NumberFormatException e) {
        System.out.println("Invalid number format. Please enter a valid
amount.");
    }
 }

 private static void depositFlow() {
    try {
        System.out.print("Enter account number: ");
        String accNum = scanner.nextLine().trim();
        Optional<BankAccount> opt = bank.findAccount(accNum);
        if (!opt.isPresent()) {
```

```
            System.out.println("Account not found.");
            return;
        }

        System.out.print("Enter deposit amount: ");
        double amt = Double.parseDouble(scanner.nextLine().trim());
        BankAccount acc = opt.get();
        acc.deposit(amt);
        System.out.println("Deposit successful. New balance: " +
acc.getBalance());
    } catch (InvalidAmountException e) {
        System.out.println("Error: " + e.getMessage());
    } catch (NumberFormatException e) {
        System.out.println("Invalid number format. Please enter a valid
amount.");
    }
 }

 private static void withdrawFlow() {
    try {
        System.out.print("Enter account number: ");
        String accNum = scanner.nextLine().trim();
        Optional<BankAccount> opt = bank.findAccount(accNum);
        if (!opt.isPresent()) {
```

```
            System.out.println("Account not found.");
            return;
        }

        System.out.print("Enter withdrawal amount: ");
        double amt = Double.parseDouble(scanner.nextLine().trim());
        BankAccount acc = opt.get();
        acc.withdraw(amt);
        System.out.println("Withdrawal successful. New balance: " +
acc.getBalance());
    } catch (InvalidAmountException | InsufficientFundsException e)
{
        System.out.println("Error: " + e.getMessage());
    } catch (NumberFormatException e) {
        System.out.println("Invalid number format. Please enter a valid
amount.");
    }
 }

 private static void checkBalanceFlow() {
    System.out.print("Enter account number: ");
    String accNum = scanner.nextLine().trim();
    Optional<BankAccount> opt = bank.findAccount(accNum);
    if (!opt.isPresent()) {
        System.out.println("Account not found.");
```

```
            return;
        }

        BankAccount acc = opt.get();
        System.out.println(acc);
    }

    private static void listAllAccounts() {
        Collection<BankAccount> all = bank.listAccounts();
        if (all.isEmpty()) {
            System.out.println("No accounts found.");
            return;
        }
        System.out.println("Listing accounts (demo):");
        for (BankAccount acc : all) System.out.println(acc);
    }
}
```