

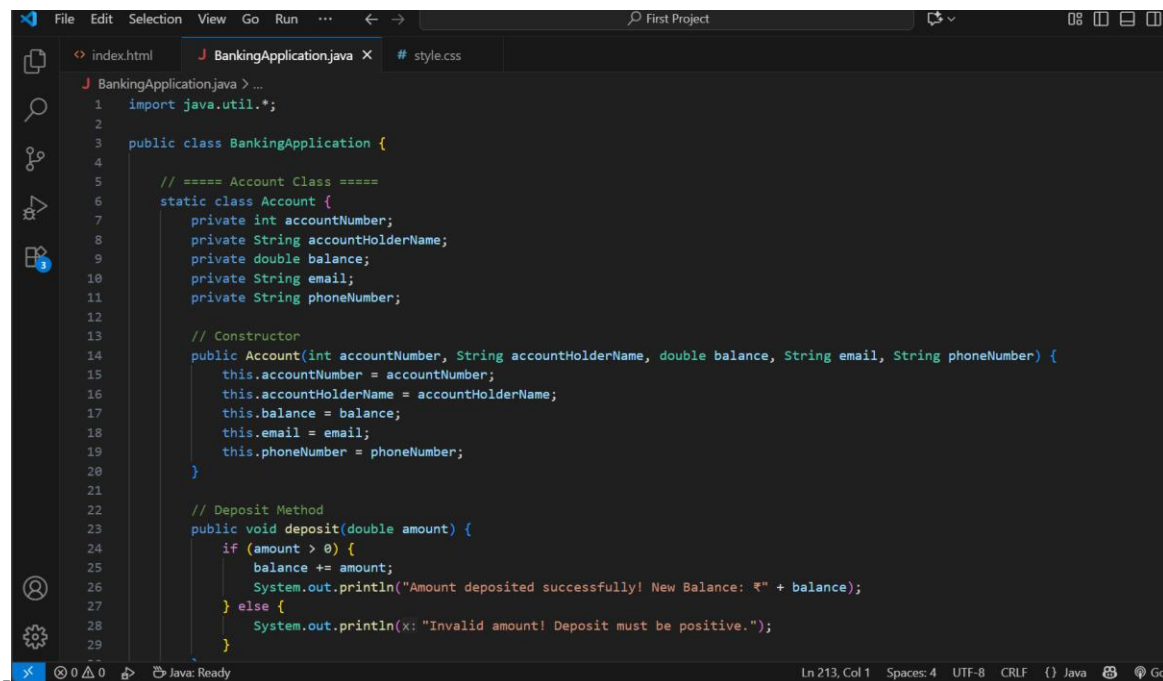
# JAVA ASSIGNMENT NUMBER: 01

Name: **Aman**

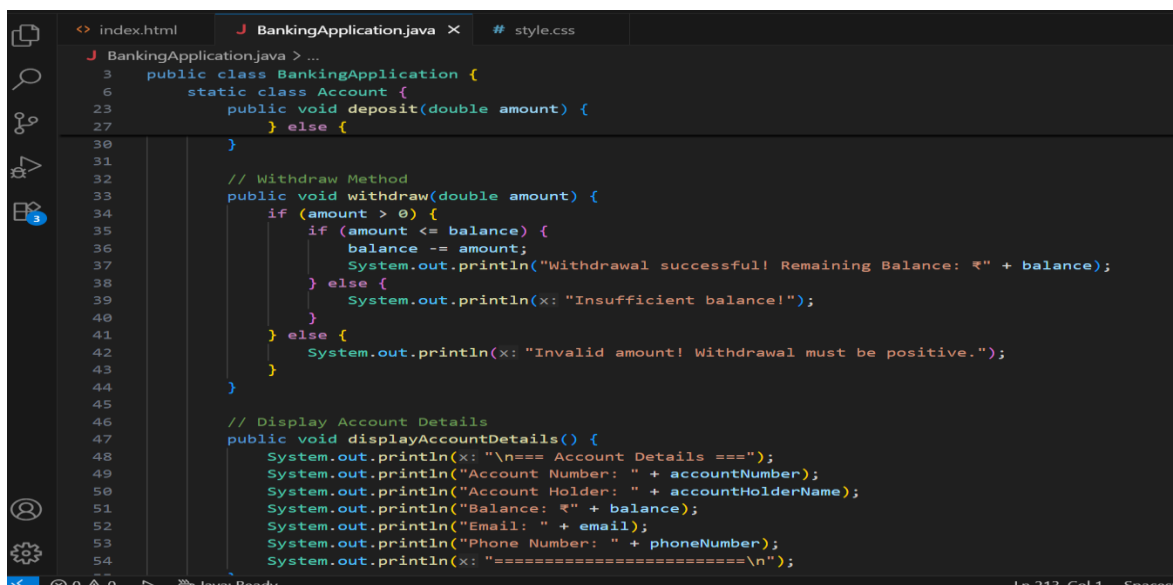
Roll no: **2401201115**

Course: **BCA (AI&DS)**

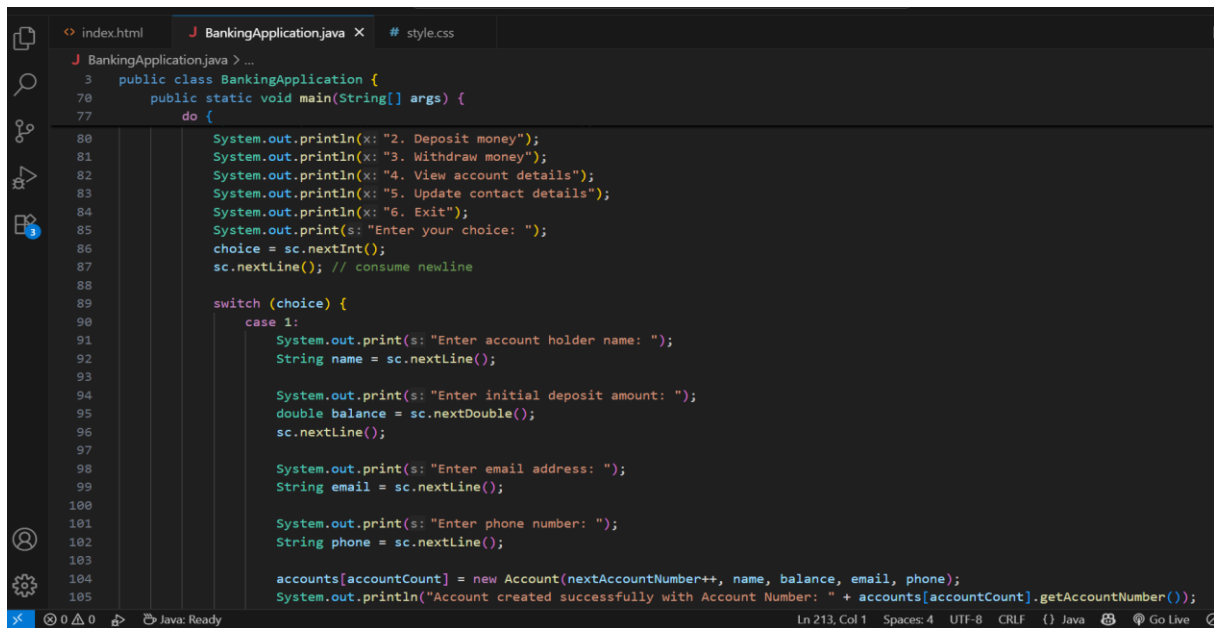
## INPUT



```
File Edit Selection View Go Run ... First Project
index.html J BankingApplication.java X # style.css
J BankingApplication.java > ...
1 import java.util.*;
2
3 public class BankingApplication {
4
5     // ===== Account Class =====
6     static class Account {
7         private int accountNumber;
8         private String accountHolderName;
9         private double balance;
10        private String email;
11        private String phoneNumber;
12
13        // Constructor
14        public Account(int accountNumber, String accountHolderName, double balance, String email, String phoneNumber) {
15            this.accountNumber = accountNumber;
16            this.accountHolderName = accountHolderName;
17            this.balance = balance;
18            this.email = email;
19            this.phoneNumber = phoneNumber;
20        }
21
22        // Deposit Method
23        public void deposit(double amount) {
24            if (amount > 0) {
25                balance += amount;
26                System.out.println("Amount deposited successfully! New Balance: ₹" + balance);
27            } else {
28                System.out.println(x: "Invalid amount! Deposit must be positive.");
29            }
30        }
31    }
32}
```

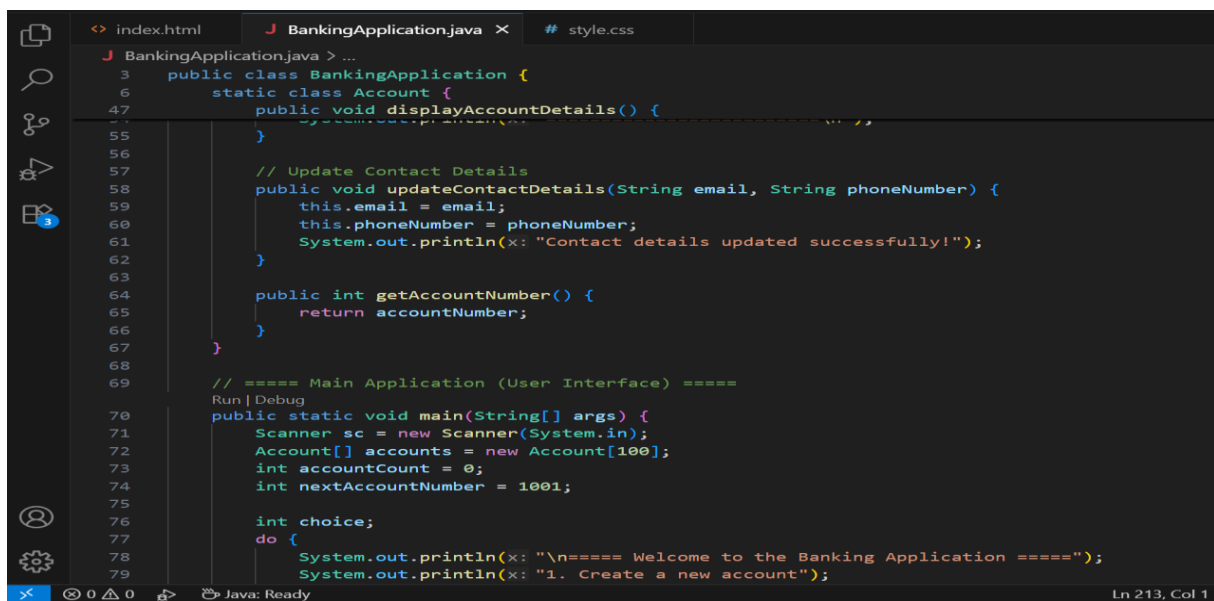


```
index.html J BankingApplication.java X # style.css
J BankingApplication.java > ...
3 public class BankingApplication {
6     static class Account {
23         public void deposit(double amount) {
27             } else {
30         }
31     }
32
33     // Withdraw Method
34     public void withdraw(double amount) {
35         if (amount > 0) {
36             if (amount <= balance) {
37                 balance -= amount;
38                 System.out.println("Withdrawal successful! Remaining Balance: ₹" + balance);
39             } else {
40                 System.out.println(x: "Insufficient balance!");
41             }
42         } else {
43             System.out.println(x: "Invalid amount! Withdrawal must be positive.");
44         }
45     }
46
47     // Display Account Details
48     public void displayAccountDetails() {
49         System.out.println(x: "\n=== Account Details ===");
50         System.out.println("Account Number: " + accountNumber);
51         System.out.println("Account Holder: " + accountHolderName);
52         System.out.println("Balance: ₹" + balance);
53         System.out.println("Email: " + email);
54         System.out.println("Phone Number: " + phoneNumber);
55         System.out.println(x: "=====\n");
56     }
57 }
```



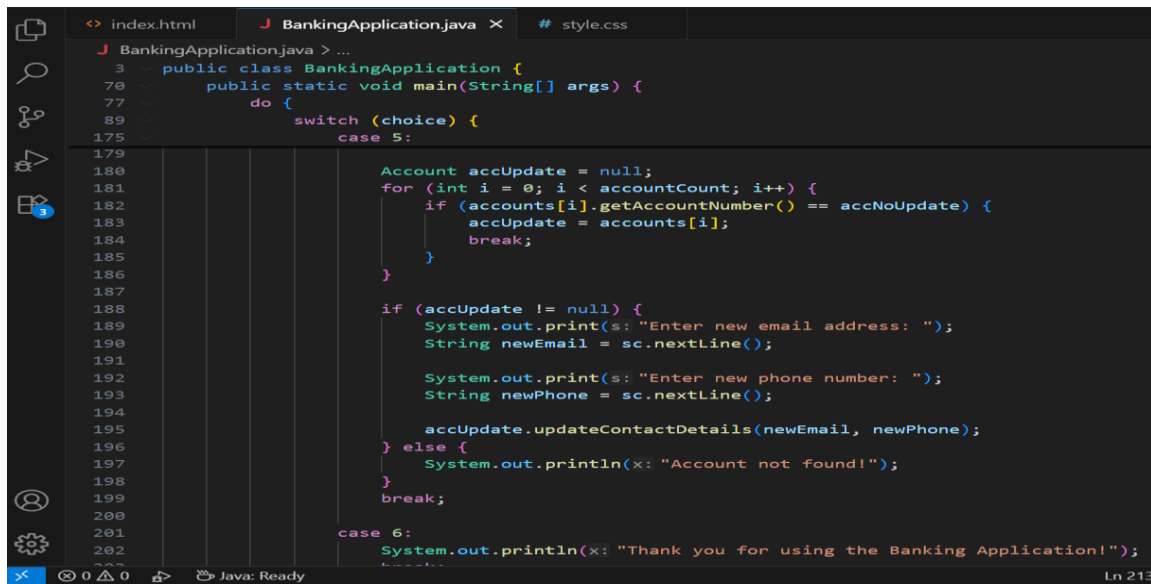
```
index.html  J BankingApplication.java  # style.css
J BankingApplication.java > ...
3  public class BankingApplication {
70  public static void main(String[] args) {
77      do {
80          System.out.println(x: "2. Deposit money");
81          System.out.println(x: "3. Withdraw money");
82          System.out.println(x: "4. View account details");
83          System.out.println(x: "5. Update contact details");
84          System.out.println(x: "6. Exit");
85          System.out.print(s: "Enter your choice: ");
86          choice = sc.nextInt();
87          sc.nextLine(); // consume newline
88
89          switch (choice) {
90              case 1:
91                  System.out.print(s: "Enter account holder name: ");
92                  String name = sc.nextLine();
93
94                  System.out.print(s: "Enter initial deposit amount: ");
95                  double balance = sc.nextDouble();
96                  sc.nextLine();
97
98                  System.out.print(s: "Enter email address: ");
99                  String email = sc.nextLine();
100
101                  System.out.print(s: "Enter phone number: ");
102                  String phone = sc.nextLine();
103
104                  accounts[accountCount] = new Account(nextAccountNumber++, name, balance, email, phone);
105                  System.out.println("Account created successfully with Account Number: " + accounts[accountCount].getAccountNumber());
```

Ln 213, Col 1 Spaces: 4 UTF-8 CRLF () Java Go Live



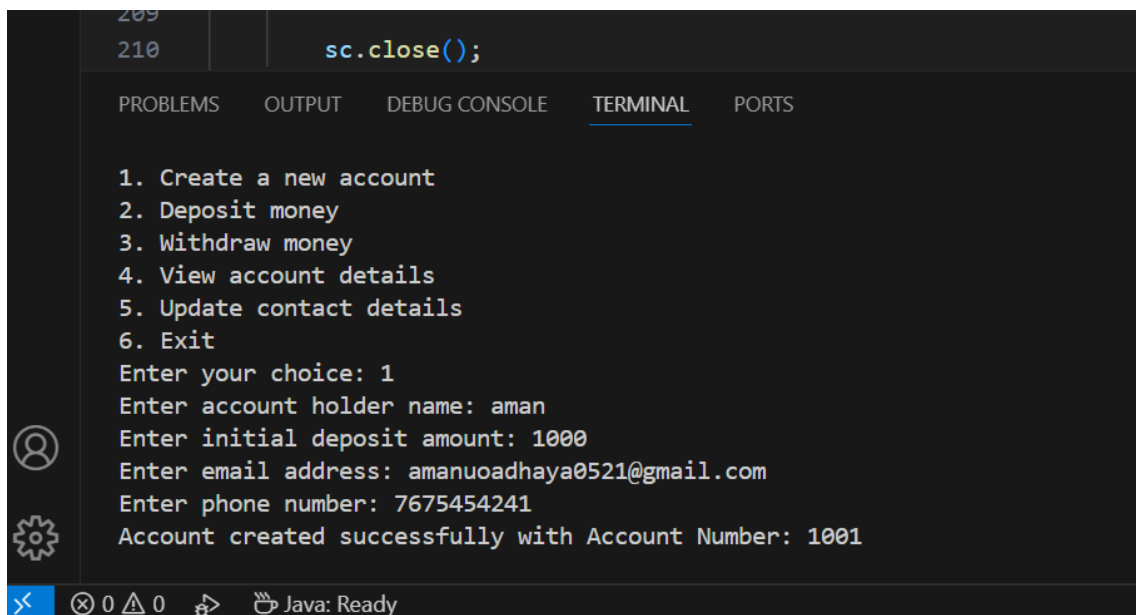
```
index.html  J BankingApplication.java  # style.css
J BankingApplication.java > ...
3  public class BankingApplication {
6      static class Account {
47          public void displayAccountDetails() {
55          }
56
57          // Update Contact Details
58          public void updateContactDetails(String email, String phoneNumber) {
59              this.email = email;
60              this.phoneNumber = phoneNumber;
61              System.out.println(x: "Contact details updated successfully!");
62          }
63
64          public int getAccountNumber() {
65              return accountNumber;
66          }
67      }
68
69      // ===== Main Application (User Interface) =====
70      public static void main(String[] args) {
71          Scanner sc = new Scanner(System.in);
72          Account[] accounts = new Account[100];
73          int accountCount = 0;
74          int nextAccountNumber = 1001;
75
76          int choice;
77          do {
78              System.out.println(x: "\n===== Welcome to the Banking Application =====");
79              System.out.println(x: "1. Create a new account");
```

Ln 213, Col 1



```
<> index.html  BankingApplication.java  # style.css
J BankingApplication.java > ...
3  public class BankingApplication {
70      public static void main(String[] args) {
77          do {
89              switch (choice) {
175                  case 5:
179                      Account accUpdate = null;
180                      for (int i = 0; i < accountCount; i++) {
181                          if (accounts[i].getAccountNumber() == accNoUpdate) {
182                              accUpdate = accounts[i];
183                              break;
184                          }
185                      }
186                  }
187
188                  if (accUpdate != null) {
189                      System.out.print(s: "Enter new email address: ");
190                      String newEmail = sc.nextLine();
191
192                      System.out.print(s: "Enter new phone number: ");
193                      String newPhone = sc.nextLine();
194
195                      accUpdate.updateContactDetails(newEmail, newPhone);
196                  } else {
197                      System.out.println(x: "Account not found!");
198                  }
199                  break;
200
201                  case 6:
202                      System.out.println(x: "Thank you for using the Banking Application!");
203                      break;
204              }
205          }
206      }
207  }
```

## OUTPUT



```
209
210      sc.close();
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Update contact details
6. Exit
Enter your choice: 1
Enter account holder name: aman
Enter initial deposit amount: 1000
Enter email address: amanuoadhaya0521@gmail.com
Enter phone number: 7675454241
Account created successfully with Account Number: 1001
Java: Ready
```

## Explanation —

### 1) High-level design

- The application models bank accounts with an `Account` class and provides a console menu (create, deposit, withdraw, view, update, exit).
  - Data storage: a `Map<Integer, Account>` (`HashMap`) stores accounts keyed by account number for fast lookup.
  - Money is represented with `BigDecimal` (not `double`) to avoid floating-point rounding errors.
  - Input/validation is centralized in helper methods to keep the `main` loop clean and robust.
- 

## 2) `Account` class — data & behavior

Key fields:

- `private final int accountNumber;` — immutable account id once created.
- `private final String accountHolderName;` — holder name set once.
- `private BigDecimal balance;` — monetary balance stored precisely with `BigDecimal`.
- `private String email;` and `private String phoneNumber;` — contact info editable.

Constructor:

- Accepts `BigDecimal` initial balance and sets scale (`setScale(2, RoundingMode.HALF_EVEN)`) to represent currency with two decimal places and a standard rounding mode.

Behavior methods:

- `deposit(BigDecimal amount):` checks `amount > 0` using `compareTo`, adds to balance, re-applies scale, prints confirmation.
- `withdraw(BigDecimal amount):` checks positivity and sufficient funds (`amount.compareTo(balance) <= 0`), subtracts, re-applies scale, prints result; otherwise prints errors.
- `displayAccountDetails():` prints account fields (safe console display).
- `updateContactDetails(String, String):` updates email/phone.
- `getAccountNumber():` accessor for map key operations.

Why `BigDecimal`?

- `double` is binary floating point and can't exactly represent many decimal monetary values → rounding errors. `BigDecimal` stores exact decimal values and is the right tool for financial arithmetic.

Why `final` on some fields?

- `final` provides immutability guarantees for `accountNumber` and `accountHolderName`, preventing accidental reassignment after construction.
- 

## 3) Input & validation helpers

Helpers centralize parsing and validation so `main` stays readable:

- `readAmount(String prompt)`: loops until user enters a valid decimal number; returns `BigDecimal` with scale 2. Handles bad formats (catches `NumberFormatException`) rather than crashing.
- `readInt(String prompt)`: safely parses integers (menu choice, account numbers) using a loop and `Integer.parseInt`.
- `readNonEmpty(String prompt)`: enforces non-empty strings for name/email/phone.
- `isValidEmail(String)` and `isValidPhone(String)`: simple regex checks (not perfect but prevents obvious bad input). Phone accepts 7–15 digits; email uses a `\S+@\S+\.\S+`-style pattern.

Benefits:

- Centralized validation reduces duplicated code and the chance of inconsistent checks.
  - Defensive reading prevents `InputMismatchException` and makes the UX friendlier.
- 

## 4) Storage: `Map<Integer, Account>` (HashMap)

- Accounts are stored in a `HashMap` keyed by `accountNumber`.
  - Lookup complexity: average  $O(1)$  to get an account by account number (`accounts.get(accNo)`).
  - Contrast with previous array + linear search ( $O(n)$ ): map makes deposit/withdraw/view/update operations scale much better.
- 

## 5) Main loop & operations

- Menu loop reads a user choice (safe via `readInt`) and switches on it.
- Create account:
  - Read name, initial deposit, email, phone with validation.
  - Build `Account` with `nextAccountNumber` and `accounts.put(nextAccountNumber, acc)`.
  - Increment `nextAccountNumber`.
- Deposit / Withdraw / View / Update:
  - Read account number with `readInt`.

- `accounts.get(accNo)` **returns the** `Account` **or** `null`.
  - If found, call the relevant `Account` method (`deposit/withdraw/display/update`).
  - If not found, print `"Account not found!"`.
- Exit closes scanner and ends loop.