Case Study 1

```cpp
#include <bits/stdc++.h>
using namespace std;

#define RS "â‚¹"   // â€¦ Macro for Rupee symbol

// ---------- Base Class ----------
class Account {
protected:
    int accountNumber;
    string name;
    double balance;
public:
    Account(int accNo, string accName, double bal)
        : accountNumber(accNo), name(accName), balance(bal) {}

    virtual void deposit(double amount) {
        balance += amount;
        cout << RS << amount << " deposited. New balance: " << RS << balance << "\n";
    }

    virtual void withdraw(double amount) {
        if (amount > balance) {
            cout << "Insufficient balance!\n";
        } else {
            balance -= amount;
            cout << RS << amount << " withdrawn. New balance: " << RS << balance << "\n";
        }
    }

    virtual void applyInterestOrPenalty() = 0;  // Pure virtual

    // Operator overloading for fund transfer
    void operator+(Account &receiver) {
        double amount;
        cout << "Enter amount to transfer from " << name
             << " to " << receiver.name << ": " << RS;
        cin >> amount;

        if (amount > balance) {
            cout << "Transfer failed. Insufficient funds.\n";
        } else {
            balance -= amount;
            receiver.balance += amount;
            cout << "Transferred " << RS << amount << " successfully.\n";
        }
    }

    virtual void display() const {
        cout << "Account No: " << accountNumber
             << " | Name: " << name
             << " | Balance: " << RS << balance << "\n";
    }

    int getAccNo() const { return accountNumber; }
    double getBalance() const { return balance; }
    string getName() const { return name; }
};

// ---------- Derived Classes ----------
class SavingsAccount : public Account {
public:
    SavingsAccount(int accNo, string accName, double bal)
        : Account(accNo, accName, bal) {}

    void applyInterestOrPenalty() override {
        double interest = balance * 0.05;  // 5% annual
        balance += interest;
        cout << "5% interest added. New balance: " << RS << balance << "\n";
    }
};

class CurrentAccount : public Account {
public:
    CurrentAccount(int accNo, string accName, double bal)
        : Account(accNo, accName, bal) {}

    void applyInterestOrPenalty() override {
        if (balance < 1000) {
            balance -= 500;
            cout << "Balance < " << RS << "1000. Penalty of " << RS << "500 applied. Balance: " << RS << balance << "\n";
        } else {
            cout << "No penalty applied.\n";
        }
    }
};

// ---------- Bank Class ----------
class Bank {
    vector<Account*> accounts;
    int nextAccNo = 1001;
public:
    void createAccount() {
        string name;
        int type;
        double initialDeposit;

        cout << "Enter name: ";
        cin >> name;
        cout << "Account type (1 - Savings, 2 - Current): ";
        cin >> type;
        cout << "Enter initial deposit: " << RS;
        cin >> initialDeposit;

        if (type == 1)
            accounts.push_back(new SavingsAccount(nextAccNo++, name, initialDeposit));
        else
            accounts.push_back(new CurrentAccount(nextAccNo++, name, initialDeposit));

        cout << "Account created successfully!\n";
    }

    Account* findAccount(int accNo) {
        for (auto acc : accounts)
            if (acc->getAccNo() == accNo) return acc;
        return nullptr;
    }

    void depositMoney() {
        int accNo; double amount;
        cout << "Enter account number: "; cin >> accNo;
        cout << "Enter deposit amount: " << RS; cin >> amount;

        Account* acc = findAccount(accNo);
        if (acc) acc->deposit(amount);
        else cout << "Account not found!\n";
    }

    void withdrawMoney() {
        int accNo; double amount;
        cout << "Enter account number: "; cin >> accNo;
        cout << "Enter withdrawal amount: " << RS; cin >> amount;

        Account* acc = findAccount(accNo);
        if (acc) acc->withdraw(amount);
        else cout << "Account not found!\n";
    }

    void transferFunds() {
        int fromAcc, toAcc;
        cout << "Enter sender account no: "; cin >> fromAcc;
        cout << "Enter receiver account no: "; cin >> toAcc;

        Account* sender = findAccount(fromAcc);
        Account* receiver = findAccount(toAcc);

        if (sender && receiver) {
            *sender + *receiver; // operator overloading
        } else {
            cout << "Invalid account number(s).\n";
        }
    }

    void applyYearEnd() {
        for (auto acc : accounts) {
            acc->applyInterestOrPenalty();
        }
    }

    void showAllAccounts() {
        for (auto acc : accounts) acc->display();
    }

    // â€¦ Show balance of specific account
    void showBalance() {
        int accNo;
        cout << "Enter account number: ";
        cin >> accNo;

        Account* acc = findAccount(accNo);
        if (acc) {
            cout << "Account No: " << acc->getAccNo()
                 << " | Name: " << acc->getName()
                 << " | Balance: " << RS << acc->getBalance() << "\n";
        } else {
            cout << "Account not found!\n";
        }
    }

    // â€¦ Delete an account
    void deleteAccount() {
        int accNo;
        cout << "Enter account number to delete: ";
        cin >> accNo;

        for (auto it = accounts.begin(); it != accounts.end(); ++it) {
            if ((*it)->getAccNo() == accNo) {
                delete *it; // free memory
                accounts.erase(it);
                cout << "Account deleted successfully.\n";
                return;
            }
        }
        cout << "Account not found!\n";
    }

    ~Bank() {
        for (auto acc : accounts) delete acc;
    }
};

// ---------- Main ----------
int main() {
    Bank bank;
    int choice;

    do {
        cout << "\n------ Bank Menu ------\n";
        cout << "1. Create Account\n";
        cout << "2. Deposit Money\n";
        cout << "3. Withdraw Money\n";
        cout << "4. Transfer Funds\n";
        cout << "5. Apply Interest/Penalty\n";
        cout << "6. Show All Accounts\n";
        cout << "7. Show Balance\n";      // â€¦ New Option
        cout << "8. Delete Account\n";    // â€¦ New Option
        cout << "0. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch(choice) {
            case 1: bank.createAccount(); break;
            case 2: bank.depositMoney(); break;
            case 3: bank.withdrawMoney(); break;
            case 4: bank.transferFunds(); break;
            case 5: bank.applyYearEnd(); break;
            case 6: bank.showAllAccounts(); break;
            case 7: bank.showBalance(); break;
            case 8: bank.deleteAccount(); break;
            case 0: cout << "Exiting...\n"; break;
            default: cout << "Invalid choice!\n";
        }
    } while (choice != 0);

    return 0;
}
```