**Slip 17: Sample Solutions and Explanations**

**Q1. ATM Simulation: Deposit, Withdraw, Check Balance**

**Approach**

* Create an ATM class with a balance attribute.
* Implement methods for deposit, withdraw (with validation), and checking balance.
* Use a menu-driven approach in main to perform operations.

**Code**

#include <iostream>  
using namespace std;  
  
// [ATM Class Definition]  
class ATM {  
 double balance;  
public:  
 ATM(double b = 0) : balance(b) {}  
 void deposit(double amt) { balance += amt; }  
 void withdraw(double amt) {  
 if(amt > balance) cout << "Insufficient funds\n";  
 else balance -= amt;  
 }  
 void checkBalance() { cout << "Balance: " << balance << endl; }  
};  
  
int main() {  
 ATM atm(1000);  
 int choice; double amt;  
 do {  
 cout << "1.Deposit 2.Withdraw 3.Balance 0.Exit: ";  
 cin >> choice;  
 switch(choice) {  
 case 1: cout << "Amount: "; cin >> amt; atm.deposit(amt); break;  
 case 2: cout << "Amount: "; cin >> amt; atm.withdraw(amt); break;  
 case 3: atm.checkBalance(); break;  
 }  
 } while(choice != 0);  
 return 0;  
}

**Explanation**

* The ATM class manages the account balance and provides methods for deposit, withdrawal (with validation), and balance inquiry.
* The menu-driven loop in main allows the user to perform operations until exit.

**Syntax Definitions**

* **class**: A user-defined type that groups data and functions.
* **switch-case**: A control structure for multi-way branching.
* **do-while loop**: Executes the loop body at least once, then repeats as long as the condition is true.

**Q2. Student Class: Accept, Display, and Search by Marks**

**Approach**

* Create a Student class with attributes: roll number, name, and marks.
* Accept details for n students and store them in a vector.
* Search for students with marks above a given threshold and display their details.

**Code**

#include <iostream>  
#include <vector>  
using namespace std;  
  
// [Student Class Definition]  
class Student {  
 int roll;  
 string name;  
 double marks;  
public:  
 void accept() {  
 cout << "Roll: "; cin >> roll;  
 cout << "Name: "; cin >> name;  
 cout << "Marks: "; cin >> marks;  
 }  
 void display() { cout << roll << " " << name << " " << marks << endl; }  
 double getMarks() { return marks; }  
};  
  
int main() {  
 int n;  
 cout << "Number of students: ";  
 cin >> n;  
 vector<Student> students(n);  
 for(auto &s : students) s.accept();  
 double threshold;  
 cout << "Enter marks threshold: "; cin >> threshold;  
 cout << "Students with marks above threshold:\n";  
 for(auto &s : students)  
 if(s.getMarks() > threshold) s.display();  
 return 0;  
}

**Explanation**

* The Student class encapsulates student data and provides methods to accept and display it.
* The program reads n students, then displays those with marks above the specified threshold.
* The getMarks method is used for filtering.

**Syntax Definitions**

* **vector**: A dynamic array from the C++ Standard Library.
* **auto**: Automatically deduces the type of the variable from its initializer.

**Q3. ATM Simulation System (Case Study)**

**Approach**

* Implement ATM class with deposit, withdraw, and balance methods.
* Validate each operation for errors and demonstrate object usage in main.

**Code**

#include <iostream>  
using namespace std;  
  
class ATM {  
 double balance;  
public:  
 ATM(double b = 0) : balance(b) {}  
 void deposit(double amt) { balance += amt; }  
 void withdraw(double amt) {  
 if(amt > balance) cout << "Insufficient funds\n";  
 else balance -= amt;  
 }  
 void checkBalance() { cout << "Balance: " << balance << endl; }  
};  
  
int main() {  
 ATM atm(1000);  
 atm.deposit(500);  
 atm.withdraw(200);  
 atm.checkBalance();  
 return 0;  
}

**Explanation**

* The ATM class manages the account balance and provides methods for deposit, withdrawal (with validation), and balance inquiry.
* The program demonstrates object usage and method calls for typical ATM operations.

**Syntax Definitions**

* **class**: A user-defined type that groups data and functions.
* **Constructor with default argument**: Allows object creation with or without an initial balance.