Templates

Templates allow functions and classes to operate with generic data types, allowing the code to be reusable and type safe. In this system templates are used to handle financial operations (like deposits, withdrawals, and balance updates) these may use different types like int float and double.

This system includes the following:

template <typename T>

T addAmount(T a, T b) {

return a + b;

}

template <typename T>

T subtractAmount(T a, T b) {

if (b > a) {

throw std::runtime\_error("Insufficient balance");

}

return a - b;

}

Some examples where it is used are in the Customer::deposit() and Customer::withdraw() functions:

balance = addAmount(balance, amount); // Generic deposit

balance = subtractAmount(balance, amount); // Generic withdrawal

Some benefits of using templates are flexibility (easily switching between different numeric types), maintainability (one function handles all numeric cases), and reusability (templates are reusable across different banking components).

Exception handling

Exception handling prevents the program from crashing unexpectedly and allows for graceful recovery from runtime errors. It improves the reliability of the system by handling edge cases like overdrawing an account.

The key exception is thrown in the template subtractAmount function:

if (b > a) {

throw std::runtime\_error("Insufficient balance");

}

And in the Customer::withdraw() method the exception is caught and handled:

try {

balance = subtractAmount(balance, amount);

addTransaction("Withdraw", amount);

std::cout << "Withdrawn: " << amount << "\n";

return true;

} catch (const std::exception& e) {

std::cerr << "Withdrawal failed: " << e.what() << "\n";

return false;

}

Some benefits of exception handling is that it prevents crashes, gives clear feedback, and centralizes error logic for better maintainability.

Some of the challenges were ensuring exceptions don't crash app which I solved by wrapping risky operations in try-catch. Making arithmetic work with types which was solved by using templates to generalize logic.

To summarize I Provided scalable, generic, and safe operations with templates and this Simplifies financial logic without hardcoding numeric types as well. Also I added robust error handling to ensure the app remains usable even when errors occur with exception handling.