Part 2:

a) We have built a C++ application that simulates bank transaction. User is given a choice of creating either a regular account or a VIP account; both accounts can have any amount of deposit but regular account has a 1000 withdraw limit while VIP account has 2000 withdraw limit. We have applied the template pattern to this problem and below is a partial UML diagram before applying any design pattern.

Member functions approve(), disapprove(), maxLimit() should be private or protected -- they should not be part of the public interface. And amt_ is protected (to match the implementation)
BankAccount (Inheritance)
- `amt_`: int

+ approve(int): void virtual
+ disapprove(): void virtual
+ maxLimit(): int virtual
+ deposit(int): void virtual
+ withdraw(int): void (template method)

RegularAccount
- `maxLimit`
+ approve(int): void
+ disapprove(): void
+ maxLimit(): int
+ deposit(int): void

VIPAccount
- `maxLimit`
+ approve(int): void
+ disapprove(): void
+ maxLimit(): int
+ deposit(int): void

Note that maxLimit is underlined, to indicate it is a static data member