1. **Client**
   1. Description
      1. User interacts with this component
      2. Takes user inputs
      3. Requests services from Server – requests stored in Connector 1
      4. Displays results of services to User
      5. Implemented using a thread
      6. Suspended if Connector1 Message Buffer is full
      7. Suspended until response received from ATM Server
   2. Methods
      1. Withdraw Funds
      2. Check Balance
2. **Connector 1** 
   1. Description
      1. Synchronous connector between client and server
      2. Passive Class
      3. Single Message Buffer, Single Response Buffer
      4. Connector 1 used for Client Server interactions
      5. Connector 2 (HeartBeatResponseMessageQueue)
      6. Connector 3 (HeartBeatRequestMessageQueue)
   2. **Methods**
      1. Send (called by Client – sends service request and receives response)

Public send (in message, out response)

Place message (request) from Client in buffer

Message Buffer Full is set to True (there is a message)

Signal

While the Response Buffer is empty – suspend / wait

Response taken from Response Buffer when it is filled

Response Buffer Full is set to False

* + 1. Receive (called by Server (and Monitor) – reads service request)

Public receive (out message)

While the Message Buffer is full – suspend / wait

Message taken from Message Buffer when it is filled

Message Buffer Full is set to False (no message)

* + 1. Reply (called by Server – sends response)

Public reply (in response)

Put response in response buffer

Response Buffer Full set to True (there is a response)

Signal

* + 1. isMessage (Not sure this is necessary for this class!?)

Message Buffer Check

Check if there is a message in the message buffer

Aka the status of Message Buffer Full Variable (True or False)

* 1. **Variables**
     1. Message Buffer (Service Request from Client)
     2. Response Buffer (Response from Server)
     3. Message Buffer Full Boolean (is there a message?) (needed???)
     4. Response Buffer Full Boolean (is there a response?) (needed???)

1. **Server** 
   1. Description
      1. Unit of Mitigation for fault tolerance
      2. Monitored using Heartbeat pattern
      3. Recovered the errors using Failover pattern
      4. Reads service requests from Connector 1
      5. Processes Requests
      6. Sends response to Client through Connector
      7. Suspended if Connector1 Message Buffer is empty
      8. Checks regularly for a message from Monitor requesting Heartbeat through Connector3 (HeartBeat Request Message) – does this check after processing a service request from Client
   2. Pattern
      1. Check Connector 1 (complete request)
      2. Check Connector 3 (respond through Connector 2 if necessary)
      3. Repeat pattern
   3. 2 Components **(NOTE: LEAST AMOUNT OF INFORMATION FOR IMPLEMENTATION)**
      1. Non-Fault tolerant component (runs first)
      2. Fault tolerant component (runs second, placed by error processor)
      3. In Non-fault tolerant component create conditions such that Server thread cannot check the Heartbeat Request Message and thus stops sending heartbeat response
   4. When ATM Server is created
      1. Create 5 checking accounts and debit cards
      2. Create 2 lists (checkingAccountList and debitCardList)
2. **Account**
   1. Description
      1. Passive Class
      2. Cannot have a negative balance
   2. Methods
      1. Open (accountnumber)
      2. readBalance ()
      3. credit (amount)
      4. debit (amount)
      5. getAccountNumber()
   3. Variables

accountNumber (integer)

balance (float initialized to 0)

1. **CheckingAccount**
   1. Description
      1. Passive Class
      2. Inherits from the Account class
      3. Cannot have a negative balance
   2. Methods
      1. Credit (amount)
      2. readLastDepositAmount()
   3. Variables
      1. lastDepositAmount
2. **DebitCard**
   1. Description
      1. Passive Class
   2. Methods
      1. DebitCard (cardId, PIN, accountNo, dailyDebitTotal)
      2. validatePIN (cardId, PIN)

checks if debit card Id matches the PIN

If it does – return account number

If it does not match – return 0

* + 1. updateDailyDebitTotal (amount)

updates daily debit total whenever money is withdrawn

* + 1. checkDailyDebitLimit (amount) – returns True or False

checks if withdrawing amount is over daily limit of $300.00

Returns True if the amount is not over daily limit

Returns False if the amount is over the daily limit

* 1. Variables
     1. cardId (integer)
     2. PIN (integer)
     3. accountNo (integer)
     4. dailyDebitTotal (float)
     5. dailyLimit = $300.00 (suggested, not in handout)

1. **WithdrawalTransaction**
   1. Description
      1. Passive Class
      2. Needs to display error message why the function does not execute successfully
   2. Methods
      1. withdrawFunds (cardId, PIN, amount, checkingAccountList, debitCardList)

withdraws money from an account

1st checks if PIN and cardId match

2nd checks if the cardId is not over daily debit limit

If not over limit, withdraws amount from corresponding account

Updates total daily debit for the cardId

(note: don’t allow withdrawal over current balance)

\*\*\*question? – in document it says the data type of amount is “Real”?

1. **CheckBalanceTransaction**
   1. Description
      1. Passive Class
      2. Needs to display error message why the function does not execute successfully
   2. Methods
      1. checkBalance(cardId, PIN, checkingAccountList, debitCardList)

checks the current balance of the account

1st checks if cardId matches PIN

Reads balance on the account

1. Connectors 2/3 (aka “Monitor MessageQueue”)
   1. Description
      1. Different class for connectors 2 and 3
   2. Methods
      1. Send

Wait while the messageCount = maxCount

Place Message in the buffer

Increment messageCount

If messageCount =1 then signal

* + 1. Receive

Wait while messageCount = 0

Remove message from buffer

Decrement messageCount

If messageCount = maxCount-1 then signal

* + 1. isMessage

check if a message is in the messageQueue

* 1. Variables
     1. Private maxCount
     2. Private MessageCount=0

1. **Monitor**
   1. Description
      1. Checks the thread of Server 1 regularly to see if its alive
      2. Uses “Heartbeat Pattern”
      3. 2 Connectors between Monitor and Server
      4. After 3 heartbeat requests if there is no response from server, notifies the error processing component
2. **Error Processor**
   1. Design using Failover Pattern
   2. Replaces non-fault Server component with fault-tolerant server component

**Definitions in our book:**

Unit of Mitigation

Heartbeat Pattern

Failover Pattern

Thread