# TUTORIAL ⎯ Preparation for Task 1c – Group Assignment

## OBJECTIVES

The objective of this tutorial is to check and understand the C++ code given to you for the Cash Point case study and to help you with some of the questions for Task 1c.

## Instructions

1. Download the **OOP\_ATM\_Task1c\_SKELETON.zip** file.   
   To test it, enter **1** when prompted for a command, **01** for the card number, and enter the bank account details (e.g. **001** for the account number).
2. Study the C++ code given carefully and **draw the Sequence Diagram** for this command. It should start with the **mAcct3\_depositToBankAccount** message sent to the object ‘**theCashPoint**’ and represent the interactions that take place in the system when the deposit transaction has been requested.  
   Do not represent the details of what happens in the **List** class's operations and do not show more than 3 levels of nested interactions.

NOTE: It is also possible in UML to use some (sort of) 'local' variables and associate them with a value returned form a message using the := symbol. Typically these are used as guard *for other messages, but can also be used to give more information about some parameters* (as in the following version of the same sequence diagram). Remember that UML Sequence Diagrams focus on portraying dynamic interactions between objects in the system. They do not intend to specify the full detail of the operation.

1. Look at the following Use Case and related (partial) Sequence Diagram and implement a new "Show All Deposit Transactions" command (**option 5**) that matches the information on the UML specification documents (Use Case and Sequence Diagram) given below (i.e., implement the CashPoint::mAcct5\_showAllDepositsTransactions function and any sub functions it calls).
2. Write iterative versions of the TransactionList functions getAllDepositTransactions and getTotalTransactions.
3. Write alternative recursive versions of these functions too.

**"Showing All Deposit Transactions" Use Case**

1. The system shows the options available for processing the account currently open.
2. The customer chooses to see a list of all the deposit transactions on the account.
3. The system displays the current date and time (e.g., "ALL DEPOSIT TRANSACTIONS REQUESTED AT 12:03:45 ON 20/02/2018").
4. The system displays the details of all the credit transactions in the bank account (with a positive amount) together with their cumulated amount (may be £0.00).
5. The system indicates that the operation has been successful and goes back to step 1.

**Alternate courses:**

at 4a. If there are no transactions in the bank account the system displays an appropriate message (e.g., "NO TRANSACTIONS IN BANK ACCOUNT") and goes back to step 1.

at 4b. If there are no deposit transactions in the bank account the system displays an appropriate message (e.g., "NO DEPOSITTRANSACTIONS IN BANK ACCOUNT") and goes back to step 1

Fig a: Use Case describing the 'Showing All Deposit Transactions' option.

theATM : ATM

theActiveAccount

: BankAccount

mAcct5\_showAllDepositsTransactions()

2: [not noTransaction] produceAllDepositTransactions():

string x double

theUI : UserInterface

transactions: TransactionList

2.2: total:= getTotalTransactions():

double

2.1: trl := getAllDepositTransactions():

TransactionList

1: noTransaction:= isEmptyTransactionList(): boolean

: Time

3.1: currentTime(): Time

3.2: currentDate(): Date

: Date

3: showAllDepositsOnScreen

(noTransaction, str, total)

trl: TransactionList

2.3: str:= toFormattedString(): string

Fig. b: (Partial) Sequence Diagram showing All Deposit Transactions.

*NOTE: This diagram is slightly simplified: it does not show all of the operations involved in the creation of* ***trl****, the temporary instance of the* ***Transaction List*** *class that takes place in* ***message 2****. This will also be acceptable for the diagrams you produce for Task 1c.*