# Questions

**1. a) According to the UML Class Diagram in Fig. 2 is it possible for the recordWithdrawal operation to be inherited and used directly in the ChildAccount class? Describe what problems may arise in doing so.**

The function will be successfully inherited by the ChildAccount class, provided that both the SavingsAccount and ChildAccount publicly inherit from their base classes, and the function itself is public as shown.

**b) The C++ CashPoint class includes a BankAccount pointer as one of its data members. Explain why it is not possible to use a BankAccount instance instead of a pointer to implement this relationship?**

It has to be a pointer to a BankAccount because it could be any type of account inheriting from the BankAccount, which can be done with pointers using dynamic casting, but can't be done with instances.

**c) In the C++ implementation given, what is the nature of relationship between the CashCard and List<string> classes, how is it represented in UML and what C++ mechanism is involved in its C++ implementation?**

The CashCard and List<string> share a composition relationship, meaning that when the CashCard is created or destroyed, the List<string> is as well. It is represented in UML by a filled diamond on the CashCard linked by a line to the List. This is implemented as a new instance of the List template class, of type string, within the CashCard class.

**d) Is the UserInterface class is an abstract class? How do you know?**

The UserInterface isn't an abstract class because an abstract class is a base class that can be inherited from but not instantiated, but there is a constant instance of the UserInterface within CashPoint class. Abstract classes must also have a pure virtual function, which UserInterface does not.

**e) Why is the Date::currentDate() function is declared as static? How does this mechanism work?**

This is done so that the current date can be accessed by any class without needing an instance of the Date class, and instead using the scope operator :: to reference the Date class.

**f) Explain the issues, if any, with the following four statements (where t1 is a valid Time instance) and describe which changes are needed for them to be valid and equivalent?**

**Time t ( t1 + Time( 0, 0, 12));**

**Time t ( Time( 12) + t1);**

**Time t ( t1 + 30);**

**Time t ( 12 + t1);**

The bottom statement (Time t(12 + t1) wouldn't work because operator overloading would only allow the '+' to be after the Time class, effectively calling the Time class's '+' function, whereas if the '+' is a prefix, it is the default '+' operator which has no knowledge of the Time class.

**g) The purpose of the TransactionList::deleteGivenTransaction function is to delete the first occurrence, if there is one, of a given transaction from a non-empty transaction list. Is the following version correct? If not, explain why it isn’t and show how it could be amended to work correctly.**

**void TransactionList::deleteGivenTransaction( const Transaction& tr) {**

**assert( \*this.size() != 0);**

**if ( mostRecentTransaction() == tr)**

**\*this = olderTransactions();**

**else {**

**Transaction firstTr( newestTransaction());**

**olderTransactions().deleteGivenTransaction( tr))**

**this->addNewTransaction( firstTr);**

**}**

**}**

This code will not work for two reasons. Firstly, the line "olderTransactions().deleteGivenTransaction( tr))" has a missing semicolon and an extra bracket at the end. Secondly, the function olderTransactions returns a constant transaction list, meaning that it can't then go on to delete a transaction from the list.

**h) What differences would it make to declare a method such as BankAccount::prepareFormattedAccountDetails as virtual and explain why you might want to do this?**

This would allow each individual class to have its own version of the function, and the program will call the correct version depending on the type of bank account. This would mean that, for example, if a particular type of bank account has an overdraft or any other extra information, then this will be displayed as well, but only at the appropriate time.

**i) Would the expression p\_theActiveAccount\_->getOverdraftLimit() be valid if the pointer p\_theActiveAccount\_ were to currently points to a CurrentAccount instance? If not, what could be done, if anything, to solve this problem?**

This wouldn’t work as when p\_theActiveAccount was pointed to a CurrentAccount, it truncated the CurrentAccount class information. To make this statement work, you would have to dynamically cast the pointer to a currentAccount like so:

p\_theActiveAccount = dynamic\_cast<CurrentAccount> (p\_theActiveAccount)

and then call the function from the dynamically casted pointer.

**j) Could the function CashPoint::m1\_produceBalance() have equally been rewritten as follow? If not explain why not.**

**void CashPoint::m1\_produceBalance() const {**

**theUI\_.showProduceBalanceOnScreen( p\_theActiveAccount\_->balance\_);**

**}**

The function wouldn’t work like this because balance\_ is a private member of the active account. One solution to this would be to use the getBalance() function to return the value of balance\_:

theUI\_.showProduceBalanceOnScreen( p\_theActiveAccount\_->getBalance());

# Sequence Diagram for method m2\_withdrawFromBankAccount()

theCashPoint : CashPoint

m2\_withdrawFromBankAccount()

2: transAuthorised:=

canWithdraw(amountToWithdraw): bool

theUI : UserInterface

transactions: TransactionList

p\_theActiveAccount

: \*BankAccount

1.1: readInPositiveAmount(): double

2.1: borrowable(): double

/

3: [transAuthorised] recordWithdrawal(amountToWithdraw)

/

3.1: addNewTransaction( aTransaction)

3.2: updateBalance(double)

4: showWithdrawelOnScreen (transAuthorised,

amountToWithdraw)

aTransaction : Transaction

1: amountToWithdraw:=

readInWithdrawalAmount(): double

# Relationship Diagram

May not be needed with bank account relationship

UserInterface

Node<ITEM>

CashPoint

<<bind>>

<string>

<<bind>>

<Transaction>

TransactionList

CashCard

List

Item

Transaction

Time

Date

BankAccount

CurrentAccount

ISAaccount

ChildAccount

SavingsAccount

# Class Diagram for Different Account Types

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| BankAccount |  | CurrentAccount | | |  |
| - accountType: String |  | - overdraftLimit: Double | | |  |
| - accountNumber: String |  | + getOverdraftLimit():Double | | |  |
| - sortCode: String |  |  |  |  |  |
| - transactions: TransactionList |  | SavingsAccount | | |  |
| - creationDate: Date |  | - minimumBalance : Double | | |  |
| - balance: Double |  | + setMinimumBalance(): Double | | |  |
| + getAccountType(): string |  | + getMinimumBalance(): Double | | |  |
| + getAccountNumber() : String |  |  |  |  |  |
| + getSortCode(): String |  | ChildAccount | | |  |
| + getTransactions(): TransactionList |  | - maximumPaidIn : Double | | |  |
| + getCreationDate(): Date |  | - minimumPaidIn : Double | | |  |
| + getBalance() : Double |  | + getMaximumPaidIn(): Double | | |  |
| + canWithdraw( a: Double) : Boolean |  | + getMinimumPaidIn(): Double | | |  |
| + canDeposit( a: Double) : Boolean |  |  |  |  |  |
| + recordDeposit( a: Double) |  | ISAaccount | | | |
| + recordWithdrawal( a: Double) |  | - maximumYearlyDeposit: Double | | | |
| + recordDeletionOfTransactionUpToDate( d: Date) |  | - currentYearlyDeposit: Double | | | |
| + prepareFormattedStatement(): String |  | - endDepositPeriod: Date | | | |
| + produceAllDepositTransactions(): String x Double |  | + getMaximumYearlyDeposit(): Double | | | |
| + produceMostRecentTransactions( n: Int): String x Double |  | + getCurrentYearlyDeposit(): Double | | | |
| + produceTransactionsForAmount( a: double): String x Int |  | - updateCurrentYearlyDeposit (a: Double) | | | |
| + produceTransactionsForTitle( t: String): String x Int |  | + getEndDepositPeriod(): Date | | | |
| + produceTransactionsForDate( d: Date): String x Int |  |  |  |  |  |
| + produceTransactionsUpToDate( d: Date): String x Int |  |  |  |  |  |
| +isEmptyTransactionList(): Bool |  |  |  |  |  |
| + borrowable(): Double |  |  |  |  |  |
| + readInBankAccountFromFile(s: string) |  |  |  |  |  |
| + storeBankAccountInFile(s: string) |  |  |  |  |  |
| + setTransactions( t: TransactionList) |  |  |  |  |  |
| + setBalance( a: Double) |  |  |  |  |  |
| + setCreationDate( d: Date) |  |  |  |  |  |
| + setSortCode( s: String) |  |  |  |  |  |
| + setAccountNumber( s: String) |  |  |  |  |  |
| + setAccountType( s: String) |  |  |  |  |  |
| - updateBalance( a: double) |  |  |  |  |  |
| - prepareFormattedAccountDetails(): string |  |  |  |  |  |