According to the UML Class Diagram in **Fig. 2** is it possible for the **canWithdraw** operation to be inherited and used directly in the **ChildAccount** class? Describe what problems may arise **when implementing this design** and what can be done to solve these.

1 a) Yes, the canWidthdraw function can be used on the ChildAccount class because ChildAccount inherits this function and the function is public. When implementing this design, there are problems. For example, in the base class canWidthdraw function, it compares the amount to withdraw is less than the balance. In the ChildAccount class, this will be implemented differently since this will need to take in account the balance AND the minimum paid in and max paid in. So, the minimum amount of money must be deposited into the ChildAccount and cannot deposit more than the maximumPaidIn

The C++ **ATM** class includes a **BankAccount** pointer as one of its data members. Explain why, in this case study, using a **BankAccount** instance instead of a pointer to implement this relationship, would not be appropriate. Give specific examples **from your final solution** to illustrate your answer.

1 b) make sense to use pointer instead of instance because if there is a BankAccount instance, copying another subclass to a base class means object slicing occurs. Some of the information from the subclasses of BankAccount is lost in the bank account pointer. However, using a pointer means the BankAccount pointer can point to a subclass and this means down casting can occur so information from the subclass is recovered and can be accessed.

In the C++ implementation given, what is the nature of relationship between the **Card** and **List<string>** classes, how should it be represented in UML and what C++ mechanisms are involved in its C++ implementation?

1 c) The relationship between Card and a List<String> class is composition. This should be represented in UML with a black filled diamond going to the Card class. In order to implement this, the whole class (Card) has an instance of the part class (List of type string) as one of its data members. The part instance is created in the constructor of the whole instance and destroyed when the whole instance goes out of scope.

Is the **UserInterface** class an abstract class? How do you know? If not, should it be?

1 d) The **UserInterface** is not an abstract class because there is no pure virtual function that exists in the class. For example, there is no function that is marked as virtual and there is no “= 0” at the end of the function declaration. This should not be an abstract class because there is no class that inherits from user interface so it wouldn’t make any sense to make the class abstract.

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

1 e)