OOP Assignment 2

Group Report

Group UT1

Chad Simpson, Luke Ward, Shaun Webb

***Question 1 :***

**a)**

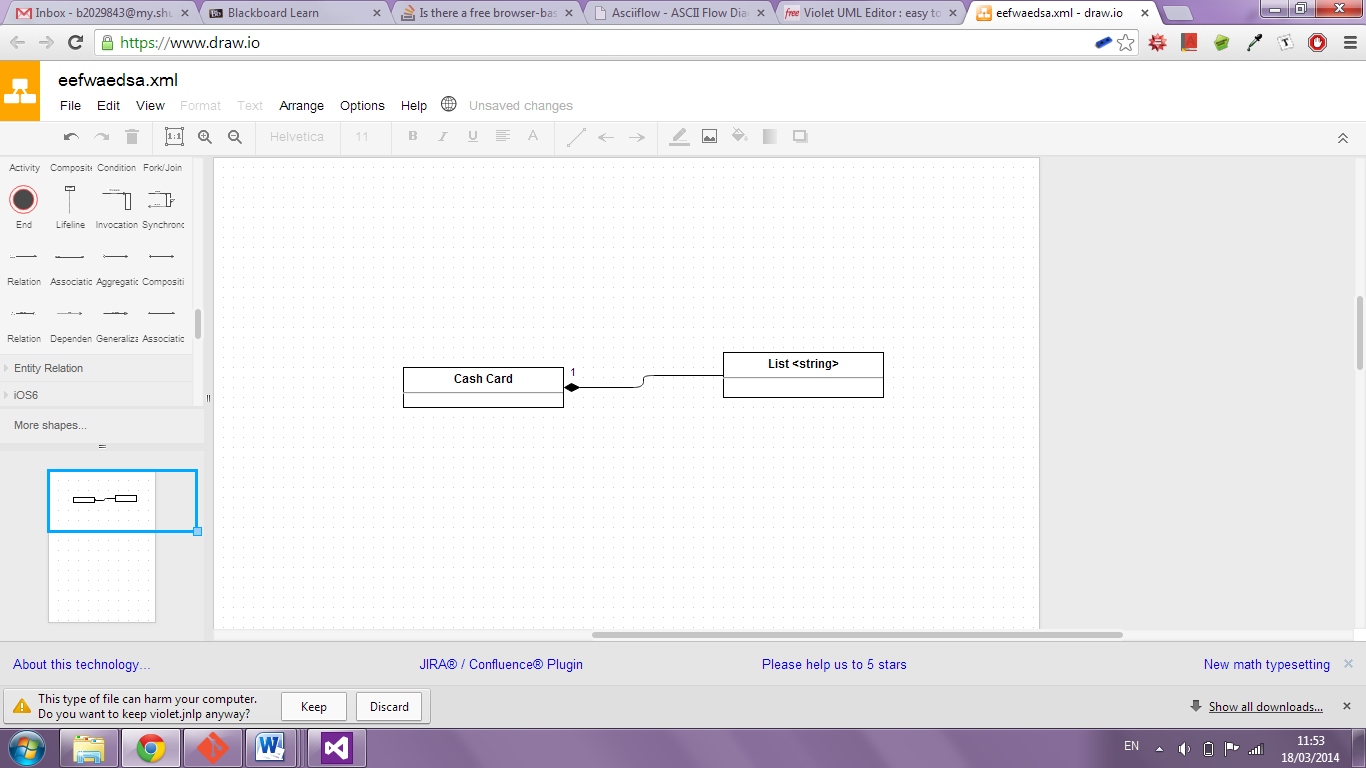
It is possible for the canWithdraw function to be inherited by the ChildAccount class, due to the fact that it is a child of the SavingsAccount class, which is a child of the BankAccount class.

One of the possible problems that would arise with using the canWithdraw function in the ChildAccount class is that it has a minimumBalance due to the fact that it is a child of the SavingsAccount. The BankAccount version of the canWithdraw function will not be taking into account the data member minimumBalance of the SavingsAccount classes, which then opens the possibility of withdrawing below the minimumAmount.

**b)**

**c)**

The relationship between the CashCard class and the List<string> classes is composition. The C++ mechanism that is involved in the implementation of it is that the List<string> is declared inside of the CashCard class.



**d)**

The UserInterface class is not an abstract class; this is due to the lack of pure virtual functions within the class. Abstract classes are useful as base classes only, and must have at least one pure virtual function present.

The only situation in which it would be appropriate for UserInterface to be an abstract class is if the end product needed different GUIs for different platforms, such as mobile and PC. In this case, you would need UserInterface as a base abstract class which the other UI classes could use as a parent class.

For the purpose of this program however, the only use will be on the ATM platform exclusively, negating the need for having UserInterface as an abstract class.

**e)**

In this program the function Date::currentDate() is required to be accessed outside of its class. Declaring the function as static allows currentDate to be called without using a class member object. In this program this is useful as it allows the current date to be accessed without creating a new Date object.

**f)**

***Line 1***: Time t (t1 + Time(0, 0, 12));

Which functions are called?

* *Basic Constructor –* The line calls the basic constructor for the Time class, using the three integers supplied in the arguments.
* *Addition Operator – T*he first line is also using the addition operator to add two Time objects together (t1 and (Time(0, 0, 12)).
* *Copy Constructor –* The result of the addition is then used by copy constructor which is called by Time t.

Issues – None.

***Line 2***: Time t (Time(12) + t1);

Which functions are called?

* *Conversion Constructor* – First of all, the second line calls the conversion constructor in Time(12).
* *Addition Operator* – Similarly to the first line, it is also using the addition constructor to add the two Time objects together.
* *Copy Constructor* – The second line lastly calls the copy constructor on the two Time objects, creating Time t from the sum of their products.

Issues – None.

***Line 3***: Time t (t1 + 12);

Which functions are called?

* *Addition Operator –* The addition operator is called on the two arguments of the constructor for Time t. The reason why it is possible to add an integer to a Time object in this manner is because the copy constructor called will first convert the time in t1 to an integer via the inSeconds function, which then allows the second parameter to be added.
* *Copy Constructor –* Lastly, line 3 calls the copy constructor on the product of the two parameters.

Issues – None.

***Line 4***: Time t (12 + t1);

Which functions are called?

* *Addition Operator –* The addition operator is called on the two parameters of the constructor of Time t. In this case, it will not compile due to the fact that it is trying to add an integer to a Time object without first converting the Time object to an integer.
* *Copy Constructor –* Time t tries to call the copy constructor on the two parameters it has been given.

Issues – Due to the fact that the addition operator adds from left to right, when it tries to add the integer to the Time first rather than trying to add the Time to the integer, it turns out to be a type mismatch.

**g)**

The following method would not work correctly, this is due to the line olderTransactions().deleteGivenTransaction(tr). The function olderTransaction returns a constant TransactionList object as part of its method. As this is constant the function deleteGivenTransaction cannot be used to change the object.

Even if the previous line could work the function as a whole would still be unsuccessful, this is due to olderTransaction returning a copy transaction list. These changes are not then applied to the main transaction list.

**h)**

Having BankAccount::prepareFormattedAccountDetails as virtual would allow the function to be dynamically binded to different types at run time. In this case this would be useful for the different types of child bank accounts to output their details in the same format using the same parent function.

**i)**

This expression would not work due to p\_theActiveAccount\_ being a pointer of type BankAccount while pointing to an object of the derived class CurrentAccount.

The getOverdraftLimit function is a method of the currentAccount class, due to the hierarchy the BankAccount pointer would be unable to access the function of the derived class.

**j)**

This function could not be written as such, the line p\_theActiveAccount->balance\_

would not work as balance is a private data member of the class BankAccount. Instead of accessing the data member in this style the p\_theActiveAccount should use an accessor function such as getBalance().