Jakov Andreski and Norbert Strobel

Computing 2 Homework Problems

Problem Set 4 (Associations)

Association is a simple structural relationship between classes. It can have many meanings such as "has-a" or "uses". All objects can have their own lifecycle and there need not be any owner. In fact, compositions and aggregations are special associations where there is ownership and possibly life-cycle dependency (see diagram in the lecture notes from Monday, 13.05.2019). In class, we also discussed an example where an object of type `Customer` was using an object of type `Account` to do banking (see UML diagram below).

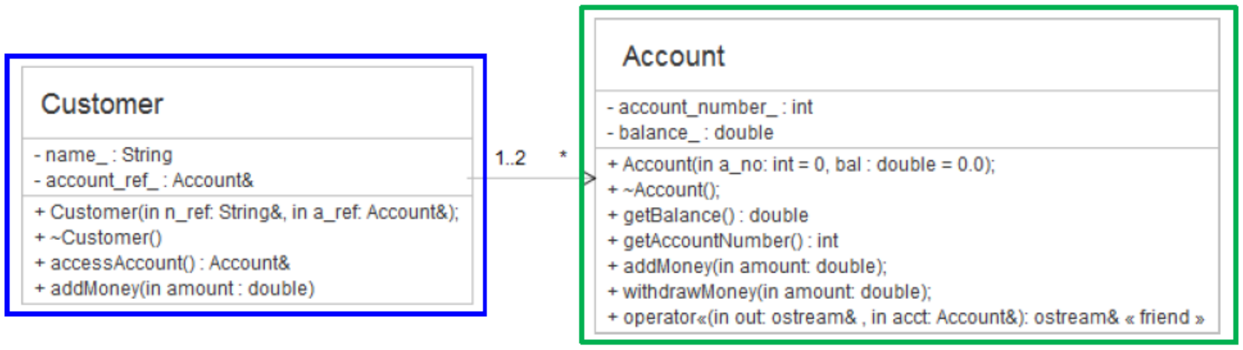

Customer
- name_ : String
- account_ref_ : Account&
+ Customer(in n_ref: String&, in a_ref: Account&);
+ ~Customer()
+ accessAccount() : Account&
+ addMoney(in amount : double)
1..2
*
Account
- account_number_ : int
- balance_ : double
+ Account(in a_no: int = 0, bal : double = 0.0);
+ ~Account();
+ getBalance() : double
+ getAccountNumber() : int
+ addMoney(in amount: double);
+ withdrawMoney(in amount: double);
+ operator«(in out: ostream& , in acct: Account&): ostream& « friend »

Figure 1: UML diagram showing an association between class `Customer` and class `Account`. An object of type `Customer` can be associated with an arbitrary number of accounts (multiplicity: *). The UML diagram shows that an object of type Account has at most two `Customer` objects associated with it, two in case of a couple who has a joint account (multiplicity: 1..2). The direction of the arrow from `Customer` to `Account` indicates a unidirectional association. It means that the `Customer` object is aware of and can use an `Account` object. However, an `Account` object does not know about a `Customer` object.

Associations are implemented either using pointers or references on the sending side (here objects of type `Customer`) that point to or reference objects on the receiving side (here objects of type `Account`). For example, objects of type `Customer` can call ("use") member functions of the class `Account` this way.

There are also bidirectional associations where both objects are aware of each other. In fact, had we included information about the customer in the Account class, we would have established a bidirectional association. This would be indicated using a line with arrows on both ends.

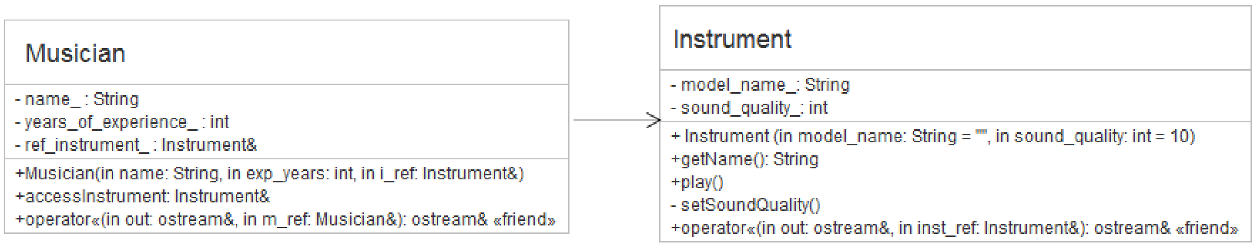
The important pieces of code of our unidirectional example are:

```
Account new_account(4711, 100); // creating account object

Customer new_customer("Customer", new_account); // passing account object as a reference to constructor
new_customer.accessAccount().addMoney(100); // Customer object uses addMoney() function of Account
new_customer.addMoney(20);
new_customer.accessAccount().withdrawMoney(50); // Customer object uses withdrawMoney() of Account
cout << new_customer.accessAccount();
```

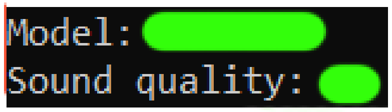
Jakov Andreski and Norbert Strobel

In this homework, you are asked to implement an association between a class `Musician` and a class `Instrument` as shown in the UML diagram below:


Musician
- name_ : String
- years_of_experience_ : int
- ref_instrument_ : Instrument&
+Musician(in name: String, in exp_years: int, in i_ref: Instrument&)
+accessInstrument: Instrument&
+operator«(in out: ostream&, in m_ref: Musician&): ostream& «friend»
Instrument
- model_name_: String
- sound_quality_: int
+ Instrument (in model_name: String = "", in sound_quality: int = 10)
+getName(): String
+play()
- setSoundQuality()
+operator«(in out: ostream&, in inst_ref: Instrument&): ostream& «friend»

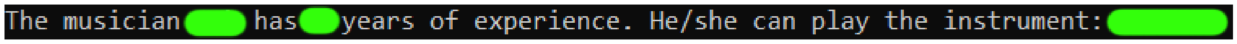
Your tasks are:

1. Write `class Instrument` such that it has private variables `string name_` and `int sound_quality_`. The variable `sound_quality_` should have a range of 1-100. If the value is out of range, default it to 10. Create a suitable constructor and overload the output stream operator (<<) such that it prints out as shown below.



```
Model:
Sound quality:
```

2. Write `class Musician` with private variables `string name_` and `int years_of_experience_`. Create a suitable constructor, and once again, overload the output stream operator such that it prints out as shown below.



```
The musician  has  years of experience. He/she can play the instrument:
```

3. Now, we want to create a **unidirectional association** between `class Instrument` and `class Musician`. Essentially, `Musician` should be aware of and use the `Instrument`, while `Instrument` is not aware of `Musician`. Additionally, a single instrument should be usable by multiple `Musicians`. How can this be done? Alter the code as necessary to achieve this.
4. Lastly, create a simple *main.cpp* file to show that the code works.

Challenge Problem: make it so that the musician can play multiple instruments. Create appropriate functions that allow more instruments to be added/subtracted.