Student Information
Name **:** **ANA MARIE FOLEY**
Email : ann.foley45@mail.dcu.ie
Student No. : 21268947
Programme Name : EE402 OOP with Embedded Systems
==

INTRODUCTION

The programme is based from a Loyalty Rewards Point System where the main arguments would be the Name, Transaction Amount, Membership Date, Points and Rewards. Where the derived class would be "Member" and the base Class is the "Customer". In this programme, I have considered all "Customers" would be members and therefore, the Pure Virtual Functions required for implementation would be to CheckMembershipStatus if its expired or not, CheckPoints and/or Convert Points from a given transaction amount.

To illustrate the tasks in the assignment, the Source Code will reflect some changes.
==
SOURCE CODE

```
#include <iostream>
#include <vector>
#include <list>

using namespace std;
using std::string;

class AbstractCustomer {
        virtual void CheckMembershipStatus()=0;
        virtual void CheckPoints()=0;
};

class Customer:AbstractCustomer {
        private:

                int MembershipDate;
                double TxnAmt;

        protected :
                string Name;
                double EarnedPoints;
        public:
                Customer (string name, int yyyymmdd, double txnAmt, double earnedPoints){
        Name = name;
        EarnedPoints = earnedPoints;
        setMembershipDate(yyyymmdd);
        setTxnAmt(txnAmt);
}
```

```
void setMembershipDate(int yyyymmdd){
        MembershipDate = yyyymmdd;
}
        int getMembershipDate () {
        return MembershipDate;
}
void setTxnAmt (double txnAmt){
        TxnAmt = txnAmt;
}
        double getTxnAmt(){
        return TxnAmt;
}
void CheckMembershipStatus(){
        bool ValidDate = true;
        if (MembershipDate >= 20211015){
        cout << "Your Membership Date is " << MembershipDate << ". Not Expired"<< endl;
        }else{
        cout << "Expired Membership " << MembershipDate << " . Renew Your Membership"
        << endl;
        }
}
void CheckPoints() {
        bool WithPoints = true;
        if (EarnedPoints > 0 ){
                cout << "Your Earned Points are " << EarnedPoints << ". Choose your
        rewards" << endl;
        } else {
                cout << "You do not have points yet. Make sure to convert transactions to
        points!"<< endl;
        }
    }
};

class Member : public Customer {
public:
        double EarnedPoints;
        double txnAmt;
        Member(string name, int yyyymmdd, double txnAmt, double earnedPoints)
        :Customer(name, yyyymmdd, txnAmt, earnedPoints){
        }
void CheckRewards() {
        cout << "***Rewards Choice****" << endl;
        vector <string> RewardsList;
        RewardsList.push_back ("KeyChain  = 100 Points");
        RewardsList.push_back ("Wallet       = 500 Points");
        RewardsList.push_back ("Air Pod      = 1000 Points");
for (int r=0; r< RewardsList.size(); r++){
        cout << RewardsList[r] << endl;
        }
    }
};
int main()
{
        Member m1= Member ("Nick", 20211027, 1000, 1000);
        Member *ptrM1=&m1;
        cout << "Earned Points of Member1: " << endl; //Pointer
        ptrM1->CheckPoints(); //dynamic pointer
        m1.CheckMembershipStatus();
        m1.CheckRewards();
}
```

==

TASKS

2. Separate Compilation. Illustrated below would be the separate compilation of the Source Code. There is one base class with 2 derived classes. The files are as follows:

- -> Loyalty.h
- ->Customer.h
- ->Customer.cpp
- ->Rewards.h
- ->Rewards.cpp
- ->LoyaltyReward.cpp

File : Loyalty.h

```
#include <iostream>

class AbstractCustomer {
            virtual void CheckMembershipStatus()=0;
            virtual void CheckPoints()=0;
};
```

File : Customer.h

```
#include  Loyalty.h
using namespace std;
using std::string;
```

```
class Customer:AbstractCustomer {
        private:
                int MembershipDate;
                double TxnAmt;
        protected :
                string Name;
                double EarnedPoints;
        public:
                Customer (string name, int yyyymmdd, double txnAmt, double earnedPoints){
                Name = name;
                EarnedPoints = earnedPoints;
                setMembershipDate(yyyymmdd);
                setTxnAmt(txnAmt);
}
------------------------------------------------------------------------
Implementation File : Customer..cpp
------------------------------------------------------------------------
#include Loyalty.h
#include Customer.h

void setMembershipDate(int yyyymmdd){
        MembershipDate = yyyymmdd;
}
        int getMembershipDate () {
        return MembershipDate;
}
void setTxnAmt (double txnAmt){
        TxnAmt = txnAmt;
}
        double getTxnAmt(){
        return TxnAmt;
}
void CheckMembershipStatus(){
        bool ValidDate = true;
        if (MembershipDate >= 20211015){
        cout << "Your Membership Date is " << MembershipDate << ". Not Expired"<< endl;
        }else{
        cout << "Expired Membership " << MembershipDate << " . Renew Your Membership"
        << endl;
        }
}
void CheckPoints() {
        bool WithPoints = true;
        if (EarnedPoints > 0 ){
                cout << "Your Earned Points are " << EarnedPoints << ". Choose your
        rewards" << endl;
        } else {
```

```
                    cout << "You do not have points yet. Make sure to convert transactions to
          points!"<< endl;
          }
     }
};
---------------------------------------------------------------------------
Header file : Rewards.h
---------------------------------------------------------------------------
#include  Loyalty.h
#include Customer.h

class Member : public Customer {
public:
          double EarnedPoints;
          double txnAmt;
          Member(string name, int yyyymmdd, double txnAmt, double earnedPoints)
          :Customer(name, yyyymmdd, txnAmt, earnedPoints){
          }
---------------------------------------------------------------------------
Implementation File: Rewards.cpp
---------------------------------------------------------------------------
#include  Loyalty.h
#include Customer.h
#include Rewards.h

#include <vector>
#include <list>

void CheckRewards() {
          cout << "***Rewards Choice****" << endl;
          vector <string> RewardsList;
          RewardsList.push_back ("KeyChain  = 100 Points");
          RewardsList.push_back ("Wallet       = 500 Points");
          RewardsList.push_back ("Air Pod      = 1000 Points");

          for (int r=0; r< RewardsList.size(); r++){
          cout << RewardsList[r] << endl;
          }
     }
};

---------------------------------------------------------------------------
LoyaltyRewards.cpp
---------------------------------------------------------------------------
#include Loyalty.h
#include Customer.h
#include Rewards.h
```

```
int main()
{
        Member m1= Member ("Nick", 20211027, 1000, 1000);
        Member *ptrM1=&m1;
        cout << "Earned Points of Member1: " << endl; //Pointer
        ptrM1->CheckPoints(); //dynamic pointer
        m1.CheckMembershipStatus();
        m1.CheckRewards();
}
```

1. **Inheritance.** The Derived Classes, Customer.h, Customer.cpp, Rewards.h and Rewards.cpp inherited from the Base Class AbstractCustomer.

3. **One Abstract Class.** The Abstract Class containing the Pure Virtual Functions would be the AbstractCustomer which would require the derived class to implement the functions as stated therein.

4. **Access Specifiers.** Except for the header file of the Abstract Class, the Customer.h, Rewards.h both contains access specifier. The private specifier contains the MembershipDate and TxnAmount which will require access to the said arguments while Name and EarnedPoints are both moved from protected where other classes to put mechanisms in accessing both the arguments but would be simpler to use and understand. The rest were classified public for easier access by the classes.

5. **Passing an object to a function by value and reference.** To illustrate Pass an object to a function, where the object m1=Member is accessing the member function through calling void Convert Points; please see below comment to illustrate Pass by Reference.

```
class Member : public Customer {
    public:
    double EarnedPoints;
    double TxnAmt;

    Member(string name, int yyyymmdd, double txnAmt, double earnedPoints)
    :Customer(name, yyyymmdd, txnAmt, earnedPoints){
    }
    void ConvertPoints(double TxnAmt, double EarnedPoints) {
       if (EarnedPoints > 0){
          EarnedPoints = TxnAmt/5;
          cout << "You converted " << TxnAmt << " to " << EarnedPoints<< endl ;
       }
    };
    int main()
    {
    Member m1= Member ("Nick", 20211027, 1000.0, 100.0);
    m1.ConvertPoints(1500, 150);  //m1 object is accessing the member function
    }                             //Pass by Value
```

Output: "You converted 1500 to 300 points".
300 points (1500/5)

```
//For Pass By Reference, the formula in the ConvertPoints has been modified to
reflect the changes; the new value is referenced from the calculation of the provided
information adding to the old points.

class Member : public Customer {
public:
double EarnedPoints;
double TxnAmt;

Member(string name, int yyyymmdd, double txnAmt, double earnedPoints)
:Customer(name, yyyymmdd, txnAmt, earnedPoints){
}
void ConvertPoints(double TxnAmt, double EarnedPoints) {
   if (EarnedPoints > 0){
         EarnedPoints = TxnAmt/5+EarnedPoints;
         EarnedPoints;
      cout << "You converted " << TxnAmt << " to " << EarnedPoints<< endl ;
   }
};

int main()
{
Member m1= Member ("Nick", 20211027, 1000.0, 100.0);
m1.ConvertPoints(1500, 150);
}
```

Output : "You converted 1500 to 450 points".
This adds 150 points in the Argument.

6. Below is the illustration of calling a method on an object that is **const qualified and** the same object to function by **constant reference.**

```
class Member : public Customer {
public:
double EarnedPoints;
double TxnAmt;

Member(string name, int yyyymmdd, double txnAmt, double earnedPoints)
:Customer(name, yyyymmdd, txnAmt, earnedPoints){
}

void ConvertPoints(const double& TxnAmt, double& EarnedPoints) {
      const_cast<double&>(EarnedPoints) = const_cast<double&>(TxnAmt)/5;
      EarnedPoints;
}
```

```
int main()
{
Member m1= Member ("Nick", 20211027, 1000.0, 100.0);

double aTxnAmt = 1000;
double aEarnedPoints = 150;
m1.ConvertPoints(aTxnAmt, aEarnedPoints);
cout << "Your " << aTxnAmt << " new Transaction is " << aEarnedPoints << "
equivalent points" << endl;
}
```

Output :

“Your 1000 new transaction is 200 equivalent points”

7. **New and Delete** for the allocation of an object with operations on the object with operations on the object using pointers.

```
class Customer {
        int membershipDate;

       public:
        // constructor initializes Date of Membership  to 2021 Nov 12
        Customer() : membershipDate(20211012) {}

        void getMembershipDate() {
           cout << "Member since = " << membershipDate << endl;
        }
     };
     int main() {

        // declare Customer object
        Customer* ptr = new Customer();

        // call MembershipDate() function
        ptr->getMembershipDate();

        // ptr memory is released
        delete ptr;

        return 0;
     }
```