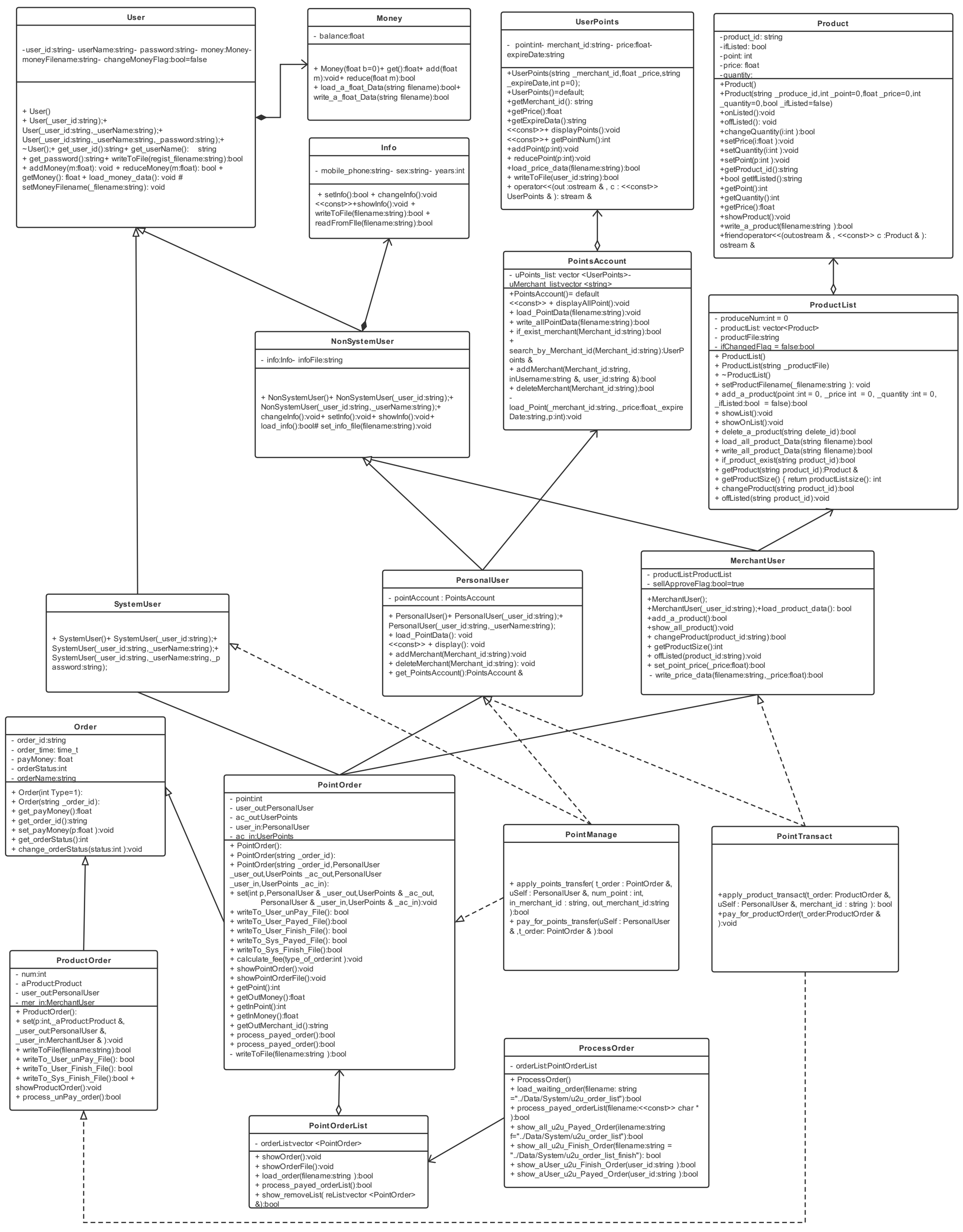
Design specifications

Catalogue

I. Class design

1. Class diagram



2. User Basic Info

The Info class describes, modifies and stores basic information about the user. Data members include mobile_phone for the user's mobile phone number, sex for the user, age for the user, and functions for registering the initial setInfo, changeInfo, showInfo, and writeInfo to file. The writeToFile function writes the information to a file.

public:

bool setInfo(); //setInfo.

void changeInfo(); //modify basic information

void showInfo();; //output information

bool writeToFile(string filename); //write the information to a file

bool readFromFIle(string filename); //read information from a file

3. User wallet type Money

The User Wallet Money class, which describes and manages the user's money, contains data members representing the balance of the wallet, as well as member functions for adding, subtracting reduce, assigning set, getting get balance and reading the balance values from a file load_a_float_Data and writing write_a_float_Data to a file. functions.

public:

Money(float b=0); // constructor, initializing the balance.

float get(){return balance;}; //Get the balance

void add(float m){balance=balance+m;}; //increase the balance

void reduce(float m); //reduce the balance

bool load_a_float_Data(string filename); //writes the balance value to a file.

bool write_a_float_Data(string filename); //read the balance value from the file.

4. User category

4.1. User base class User

The user base class User is used to describe the properties and functions common to all users. The data members are the user name userName, the user ID user_id and the password password, which are used for user login, and the member functions to read them. moneyFilename, and the flag changeMoneyFlag, which indicates whether the balance has changed, as well as the wallet management functions.

public:

User(); default constructor, which automatically reads the amount of money from the file when the object is generated.

User(string _user_id); parameterized constructor that automatically reads the amount of money from a file when the object is generated.

User(string _user_id,string _userName); parameterized constructor that automatically reads the amount of money from the file when the object is generated;, used to generate the user object when logging in.

User(string _user_id,string _userName,string _password); parameterized constructor that automatically reads the amount of money from the file when the object is generated; used to generate the user object when registering.

~User(); destructor, which automatically writes the amount of money to a file when the object dies.

string get_user_id(); //read the user ID.

string get_userName(); //read the user name.

string get_password(); //read the password.

bool writeToFile(string regist_filename); //write the newly registered user information to a file

/ Funds Management

void addMoney(float m); // top up and add funds.
void reduceMoney(float m); //spend and reduce money.
float getMoney(); //get the balance.
void load_money_data(); //read the amount from the file
protected:
void setMoneyFilename(string _filename); //set the path to the money storage file

4.2. System user class systemUse

systemUser class: Derived from User, describes properties and functions specific to platform users; has two friend classes PointManage, ProductManage.
public:

systemUser(); the default constructor, which automatically sets the path to the funds storage file when the object is generated.

systemUser(string _user_id,string _userName); parameterized constructor that automatically sets the path to the funds storage file when the object is generated.

systemUser(string _user_id,string _userName); parameterized constructor that automatically sets the path to the funds storage file when the object is generated.

4.3. Non-SystemUser class nonSystemUser

non-SystemUser class: derived from User, describes attributes and functions common to both individual and merchant users; data members include info, an object of the Info class used to represent basic user information, infoFile, a file name describing the information storage file, and basic information management functions.
public:

NonSystemUser(); //default constructor.
NonSystemUser(string _user_id):User(_user_id); //with-parameter constructor.
NonSystemUser(string _user_id,string _userName):User(_user_id,_userName); //referenced constructor
/ Manage basic information
void changeInfo(string filename); //modify the information and write the data to a file
void setInfo(string filename); //Set the information and write the data to the file
void showInfo(); //output information

bool load_info(); //read information from a
fileprotected:
void set_info_file(string filename); //set the path to the information file

4.4. PersonalUser class:

PersonalUser class: derived from nonSystemUser, describing properties and functions specific to personal users; data members include the PointsAccount class object pointAccount, representing a list of personal user points accounts; three other friend classes PointManage, ProductManage, PointOrder.

public:
PersonalUser():NonSystemUser(); // default constructor
PersonalUser(string _user_id); // parameterized constructor
PersonalUser(string _user_id,string _userName); // Parametric constructor, used for login
//manage points
void load_PointData(); //read the user's imported points information from a file
void display()const; //display the user's current points details
void addMerchant(string Merchant_id); //Add an associated merchant and import the user's points data for that merchant
void deleteMerchant(string Merchant_id); //delete the associated merchant and delete the user's points data at that merchant
PointsAccount & get_PointsAccount(); //Get a list of points accounts

4.5. MerchantUser classMerchantUser

MerchantUser class MerchantUser: Derived from nonSystemUser, describes properties and functions specific to the merchant user; data members are the productList object ProductList, which represents the list of products owned by the merchant, and the flag sellApproveFlag, which indicates whether the merchant has permission to sell products (default is true).
public:
MerchantUser(); //default constructor
MerchantUser(string _user_id); //referenced constructor
//Product management
bool load_product_data(); //Read product information from a file
bool add_a_product(); //add a new product

void show_all_product(); //Show all products
bool changeProduct(string product_id); //Modify product information
int getProductSize(); //Total number of products
void offListed(string product_id); //Unshelve the specified product
//manage points
bool set_point_price(float _price); //set_point_price
private:
bool write_price_data(string filename,float _price); //write the price of the points to the file

5. User points category

5.1. Points account base class UserPoints

The points base class, UserPoints, describes the properties and functions of a user's points account. The data members are points for the number of points, merchant_id for the account to which the points belong, price for the price of the points set by the merchant, expireDate (string type) for the expiry time of the points, and functions to get and modify them. member functions, and also overloads the function with the operator <<.

public:
UserPoints(string _merchant_id,float _price,string _expireDate,int p=0);//referenced constructor
UserPoints()=default; //default constructor
string getMerchant_ id(); //get the merchant ID of the merchant to which the points belong
float getPrice(); //get the price of the points
string getExpireData(); //get the expiration time of the points
void displayPoints()const; //display the merchant to which the points belong, the number of points, and the expiration time
int getPointNum ()const ; //Get the number of points
void addPoint(int p); //Add points
void reducePoint(int p); //Decrease points
bool load_price_data(string filename); //Read the points price data from a file
bool writeToFile(string user_id); //writes the points information to a file.
friend ostream & operator<<(ostream & out, const UserPoints & c); //Operator<< overloaded to facilitate writing points information to a file

5.2. PointsAccount list class

PointsAccount class, describing the merchant with which the user is associated and the corresponding points account list, the merchant with which the user must first be associated in order to add points to that merchant; the data members contain an array uPoints_list (vector) of multiple user points classes UserPoints and a list of merchant IDs containing multiple associations uMerchant_list.
public:
PointsAccount() = default;//default constructor
void displayAllPoint() const;//display all points under this account
void load_PointData(string filename);//read in existing points data
bool write_allPointData(string filename); //write all the user's points accounts to a file
bool if_exist_merchant(string Merchant_id);//determine if the merchant is already associated
UserPoints & search_by_ Merchant_id(string Merchant_id);//search for and get the specified associated merchant bool
addMerchant(string Merchant_id,string & inUsername,string &user_id);//add an associated merchant account
bool deleteMerchant(string Merchant_id);// Delete the associated merchant account
private:
void load_Point(string _merchant_id,float _price,string _expireDate,int p); // Read the points data (including quantity and expiration date) of this individual user from the file under the merchant account that needs to be associated

6. Commodities

6.1. Product

The product class Product is used to describe the properties of the product, the data members are product_id which indicates the product ID, ifListed(bool) which indicates whether the product is on the

shelf, the number of points contained in the product points, the price of the product price and the quantity of the product quantity, as well as for displaying and modifying the product properties and writing the product properties to the file functions.
public:
Product(); //default constructor
Product(string _produce_id,int _point=0,float _price=0,int _quantity=0,bool _ifListed=false); //with reference constructor
void onListed(); //onListed item;
void offListed(); //offListed item;
bool changeQuantity(int i); //increase/decrease quantity
void setPrice(float i); //set unit price
void setQuantity(int i); //set quantity void
setPoint(int p); //set the number of points
string getProduct_id(); //get the product ID
bool getIfListed(); //get whether the product is on the shelf
int getPoint(); //get the number of points for the product
int getQuantity(); //get the The current stock of the product

float getPrice(); //get the unit price of the product
void showProduct(); //show product properties
bool write_a_product(string filename); //write product properties to a file
friend ostream & operator<&lt ;(ostream & out, const Product & c); //overload < <, to facilitate writing to a file

6.2. ProductList

The product list class ProductList, representing all the products under a merchant, with data members of the array productList consisting of the product class Product, as well as a product ID describing the new product, a file path storing the merchant's products, and a flag ifChangedFlag indicating whether the product information has been modified (default false).
public:
ProductList(); //default constructor
ProductList(string _productFile); //constructor with parameters, automatically reads in all product information from the file
~ProductList(); //Destructor function that writes all products back to the file if their properties are modified

void setProductFilename(string _filename); // set the file path of the product
bool add_a_product(int point = 0, int _price = 0, int _quantity = 0, bool _ifListed = false);// add a product
void showList(); // show
all product attributes in the list
void showOnList(); // show the attributes of the product
on the shelf
bool delete_a_product(string delete_id); // delete the product

bool load_all_product_Data(string filename); //read all product information from a file
bool write_all_product_Data(string filename); //write all product data from the product list to a file
bool if_product_exist(string product_id) bool if_product_exist(string product_id); // Search for products by product ID
Product &getProduct(string product_id); // Get the product class by product ID
int getProductSize(); // Count the total number of products in the list
bool changeProduct(string product_id); //modify the properties of the specified product
void offListed(string product_id); //drop the specified product

7. Orders

7.1. Order base classOrder

Order base class Order, used to describe the properties common to both credit and product orders. The data members are order_id for the order ID, order_time for the transaction time, payMoney for the fee, orderStatus for the order transaction status (1: not paid; 2: paid; 3: completed;), orderName for the order type, and the member functions to read and modify them. orderName for the order type and the member functions for reading and modifying them.
public:
Order(int Type=1); //constructor
Order(string _order_id); //referenced constructor
float get_payMoney(); //get the handling fee
string get_order_id(); //get the order ID
void set_ payMoney(float p); //set the handling fee
int get_orderStatus(); //get the order status
void change_orderStatus(int status); //modify order status

7.2. PointOrder (points conversion) class

PointOrder class: Derived from the order base class Order, it is used to describe the unique properties of the points order generated by the user's points conversion. The data members are the point that indicates the number of points transferred out, the PersonalUser object user_out that indicates the user with points out, the UserPoints object ac_out that indicates the user with points out, the PersonalUser object user_in that indicates the user with points in, and the UserPoints object ac_in that indicates the user with points in. UserPoints object ac_in.
public:
PointOrder(); //default constructor
PointOrder(string _order_id); //referenced constructor
PointOrder(string _order_id,PersonalUser _user_out,UserPoints _ac_out , PersonalUser _user_in,UserPoints _ac_in); //referenced constructor
void set(int p,PersonalUser &
_user_out
,UserPoints & _ac_out,
PersonalUser & _ac_in, PersonalUser & _ac_in); //referenced constructor

void set(int p,PersonalUser
& _user_out
,UserPoints & _ac_out,
PersonalUser & _ac_in,UserPoints user_in,UserPoints & _ac_in);//with reference constructor
bool writeTo_User_unPay_File(); //write
 unpaid orders to file
bool writeTo_User_Payed_File();//write paid orders to user file

bool writeTo_User_Finish_File(); //write completed order information to the user file bool writeTo_Sys_Payed_File(); //write paid order information to the system file
bool writeTo_Sys_Finish_File(); //write completed order
void calculate_fee(int type_of_order); //Calculate the handling fee
void showPointOrder(); //Display the order details
void showPointOrderFile(); //Display part of the order
int getPoint(); //Get the number of points transferred
float getOutMoney(); //Get the amount to be spent by the merchant corresponding to the account from which the points were transferred
int getInPoint(); //Get the number of points to be transferred out based on the price of the transferred in, points

float getInMoney(); //get the amount to be earned by the
merchant corresponding to the account to which the points were transferred string getOutMerchant_id(); //get the merchant ID corresponding to the account to which the points were transferred
bool process_payed_order(); //used by the system user to process
paid, unprocessed orders
private:
bool writeToFile(string filename); //write the contents of the order to a file

7.3. PointOrderList class

PointOrderList: Derived from PointOrder, this class is used to describe and manipulate a list of point orders.
public:
void showOrder(); //shows the contents of all orders in the paid orders list;
void showOrderFile(); //shows the contents of the orders section of the paid orders list read from a file.
bool load_order(string filename); // the data of the point order read in from the file to generate the order list
bool process_payed_orderList(); // for the system user to process the
paid, unprocessed orders
bool show_removeList(vector & lt;PointOrder> & reList);//shows the orders in the order list that are not in the reList list

7.4. ProductOrder (credit transaction)

ProductOrder: Derived from the order base class Order, this class is used to describe the properties

specific to the order generated by the user's points transaction, in which the user selects the product and quantity of points to be purchased; the data members are num, which indicates the quantity of the product purchased, aProduct, which indicates the product purchased, and mer_in, which indicates the user who placed the order. PersonalUser object user_out, MerchantUser object mer_in, which is the merchant to which the product belongs.
public:
ProductOrder()
; void set(int p,Product & _aProduct,PersonalUser & _user_out,MerchantUser & _user_in);// set the order content
bool writeToFile(string filename); // Write the order contents to the specified file bool writeTo_User_unPay_File(); // Write the unpaid order to the file
bool writeTo_User_Finish_File();//write completed order information to the user file
bool writeTo_Sys_Finish_File(); //write completed order information to the system file
void showProductOrder();//show
product order content
bool process_unPay_order();//process
unpaid orders

8. Management

8.1. MerchantManage

MerchantManage class MerchantManage: the user describes and manipulates the list of merchants, the data members contain the list mUser (vector) consisting of the merchant class MerchantUser and the query and display member functions for the list.
public:
bool show_a_MerchantInfo(string merchant_id)
; bool show_all_Merchant_Name(string filename=". /Data/Merchant/registration")
; bool if_merchant_exist(string user_id);

private:
vector <MerchantUser> mUser_list;

8.2. PointManage class

The PointManage class, used for point transactions, encapsulates a number of functions involved in the conversion of user points.
public:
bool apply_points_transfer(PointOrder & t_order,PersonalUser & uSelf,int num_point,string in_merchant_id,string out_ merchant_id);//for the user to apply
for points transfer to form an order;
bool pay_for_points_transfer(PersonalUser & uSelf,PointOrder & t_order);//for the user to pay the handling fee

8.3. Order processing class ProcessOrder

Order processing class ProcessOrder, encapsulating multiple system order processing related functions, including orderList object orderList which represents the order information of the order list class PointOrderList
Contains order data loading and order transaction execution.
Member function bool process_payed_orderList(char * filename) is used to execute an order transaction, with the argument being the name of the storage file for the completed order processed, calling the data member orderList's interface: the processing order list member function
public:
ProcessOrder() = default;
// System user call
bool load_waiting_order(string filename=". /Data/System/u2u_order_list"); // uncompleted order data load bool
process_payed_orderList(const char * filename); // order transactions are executed and completed orders are stored to the specified file.
bool show_all_u2u_Payed_Order(string filename=". /Data/System/u2u_order_list"); //
Show all outstanding orders for credit conversion

bool show_all_u2u_Finish_Order(string filename = ". /Data/System/u2u_order_list_finish"); // show all points conversion completed orders // individual user call
bool show_aUser_u2u_Finish_Order(string user_id); //
Show all outstanding orders for points conversion
bool show_aUser_u2u_Payed_Order(string user_id); //Show all completed orders for points conversion

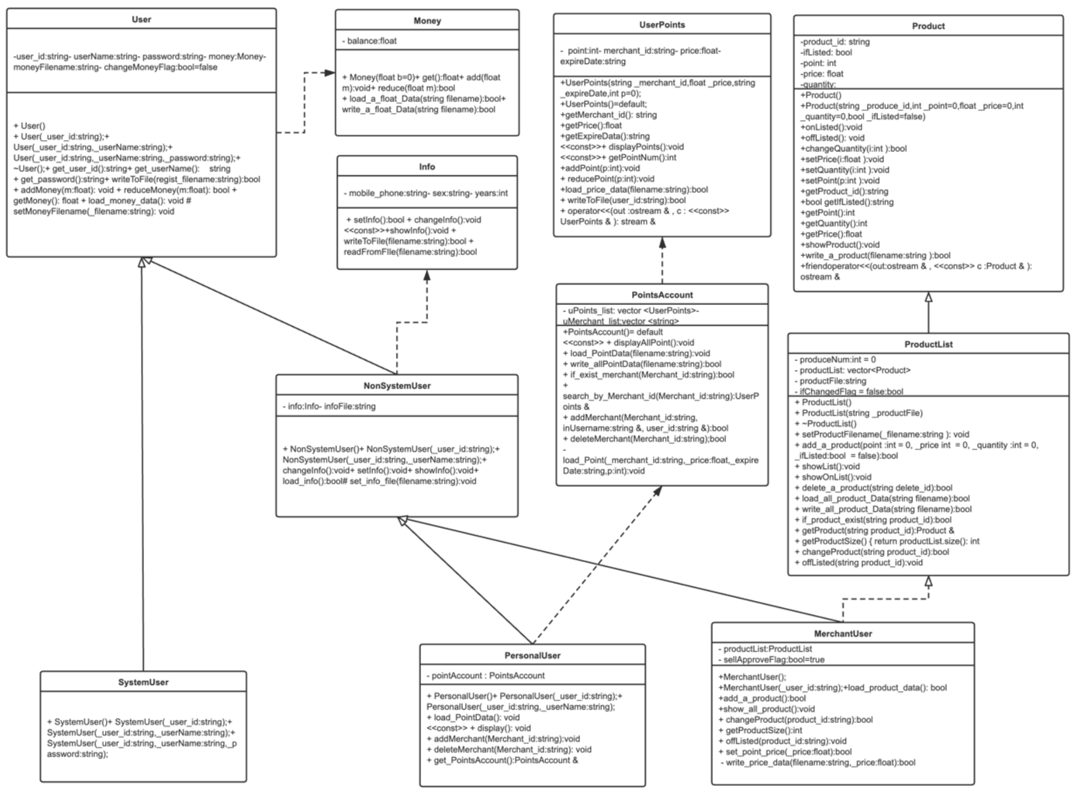
8.4. PointTransact

PointTransact: Used to purchase point products from merchants on the shelves; encapsulates several functions involving point transaction functions.
public:
bool apply_product_transact(ProductOrder & t_order,PersonalUser &uSelf,string merchant_id); //generate points transaction order and store
void pay_for_ productOrder(ProductOrder & t_order); //pay for the points transaction order and store

II. Main technical difficulties and implementation options

1. User category

There are three roles in this system: personal user, merchant user and platform user; they have some common attributes (username, user ID, password, wallet), and personal user and merchant user also have common attributes (basic information). Therefore, the user base class User is designed to contain common attributes such as username, user ID, password, wallet class Money object, etc., and derives the system user class systemUser, which is the platform user, and the non-system user class, which contains the object member info that represents basic information, and then derives the personal user class and merchant user class on the non-system user class NonSystemUser. class and the merchant user class. The user class diagram is as follows: The individual user class has an object member pointAccount that represents the list of points accounts, and the merchant user class has an object member productList that represents the list of products.


User
-user_id:string- userName:string- password:string- money:Money-moneyFilename:string- changeMoneyFlag:bool=false
+ User()
+ User(_user_id:string);+ User(_user_id:string,_userName:string);+ User(_user_id:string,_userName:string,_password:string);+ ~User();+ get_user_id():string+ get_userName(): string
+ get_password():string+ writeToFile(regist_filename:string):bool
+ addMoney(m:float): void + reduceMoney(m:float): bool + getMoney(): float + load_money_data(): void # setMoneyFilename(_filename:string): void
Money
- balance:float
+ Money(float b=0)+ get():float+ add(float m):void+ reduce(float m):bool
+ load_a_float_Data(string filename):bool+ write_a_float_Data(string filename):bool
Info
- mobile_phone:string- sex:string- years:int
+ setInfo():bool + changeInfo():void <<const>>+showInfo():void + writeToFile(filename:string):bool + readFromFIle(filename:string):bool
UserPoints
- point:int- merchant_id:string- price:float- expireDate:string
+UserPoints(string _merchant_id,float _price,string _expireDate,int p=0);
+UserPoints()=default;
+getMerchant_id(): string
+getPrice():float
+getExpireData():string
<<const>>+ displayPoints():void
<<const>>+ getPointNum():int
+addPoint(p:int):void
+ reducePoint(p:int):void
+load_price_data(filename:string):bool
+ writeToFile(user_id:string):bool
+ operator<<(out :ostream & , c : <<const>> UserPoints &): stream &
Product
-product_id: string
-ifListed: bool
-point: int
-price: float
-quantity:
+Product()
+Product(string _produce_id,int _point=0,float _price=0,int _quantity=0,bool _ifListed=false)
+onListed():void
+offListed(): void
+changeQuantity(i:int):bool
+setPrice(i:float):void
+setQuantity(i:int):void
+setPoint(p:int):void
+getProduct_id():string
+bool getIfListed():string
+getPoint():int
+getQuantity():int
+getPrice():float
+showProduct():void
+write_a_product(filename:string):bool
+friendoperator<<(out:ostream & , <<const>> c :Product &): ostream &
PointsAccount
- uPoints_list: vector <UserPoints>- uMerchant_list:vector <string>
+PointsAccount()= default
<<const>> + displayAllPoint():void
+ load_PointData(filename:string):void
+ write_allPointData(filename:string):bool
+ if_exist_merchant(Merchant_id:string):bool
+ search_by_Merchant_id(Merchant_id:string):UserPoints &
+ addMerchant(Merchant_id:string, inUsername:string &, user_id:string &):bool
+ deleteMerchant(Merchant_id:string);bool
- load_Point(_merchant_id:string,_price:float,_expireDate:string,p:int):void
ProductList
- produceNum:int = 0
- productList: vector<Product>
- productFile:string
- ifChangedFlag = false:bool
+ ProductList()
+ ProductList(string _productFile)
+ ~ProductList()
+ setProductFilename(_filename:string): void
+ add_a_product(point :int = 0, _price int = 0, _quantity :int = 0, _ifListed:bool = false):bool
+ showList():void
+ showOnList():void
+ delete_a_product(string delete_id):bool
+ load_all_product_Data(string filename):bool
+ write_all_product_Data(string filename):bool
+ if_product_exist(string product_id):bool
+ getProduct(string product_id):Product &
+ getProductSize() { return productList.size(): int
+ changeProduct(string product_id):bool
+ offListed(string product_id):void
NonSystemUser
- info:Info- infoFile:string
+ NonSystemUser()+ NonSystemUser(_user_id:string);+ NonSystemUser(_user_id:string,_userName:string);+ changeInfo():void+ setInfo():void+ showInfo():void+ load_info():bool# set_info_file(filename:string):void
SystemUser
+ SystemUser()+ SystemUser(_user_id:string);+ SystemUser(_user_id:string,_userName:string);+ SystemUser(_user_id:string,_userName:string,_password:string);
PersonalUser
- pointAccount : PointsAccount
+ PersonalUser()+ PersonalUser(_user_id:string);+ PersonalUser(_user_id:string,_userName:string);
+ load_PointData(): void
<<const>> + display(): void
+ addMerchant(Merchant_id:string):void
+ deleteMerchant(Merchant_id:string): void
+ get_PointsAccount():PointsAccount &
MerchantUser
- productList:ProductList
- sellApproveFlag:bool=true
+MerchantUser();
+MerchantUser(_user_id:string);+load_product_data(): bool
+add_a_product():bool
+show_all_product():void
+ changeProduct(product_id:string):bool
+ getProductSize():int
+ offListed(product_id:string):void
+ set_point_price(_price:float):bool
- write_price_data(filename:string,_price:float):bool

2. Points conversion

The points conversion involves four users: the individual user, the platform and the two merchants involved in the points exchange; the points conversion is processed asynchronously: the individual user generates the order, pays the handling fee and then needs to wait for the system to process the order; the specific implementation process is as follows.

1) Individual user-generated orders and automatic determination of whether the user has sufficient points to transfer from their account.
2) The individual user pays the handling fee and changes the user and system balance. (b) After successful payment, the system does not immediately execute the entire transaction, only freezes the user's transfer account points and does not increase them, while distributing the order to the system pending list/files, pending system processing.
3) When the system processes an order, it reads in all pending orders in the file and processes them one by one, increasing the user's transfer account points, decreasing the balance of the merchant to which the transfer account belongs and increasing the balance of the merchant to which the transfer account belongs, bringing in system revenue due to the difference, while increasing the system balance.

The class is designed as follows.

1) The PointManage class is used for user point transactions and encapsulates the member functions apply_points_transfer and pay_for_points_transfer, PointManage is a friend class of PersonalUser, its member functions can be called directly A private member of PersonalUser, which changes the user's points account.
2) The order processing class ProcessOrder contains the object orderList of the order list class PointOrderList, which represents the order list, and encapsulates a number of system order processing-related functions such as order data loading and order transaction execution; ProcessOrder, as a friend class of PointOrderList, has member functions that can Directly call the private members of PointOrderList, directly call the private members of PointOrderList orderList and its interface, read the order data.

3. Points Trading

A Points transaction is a purchase by a user of points from a merchant, in the form of a purchase of a merchant's product containing a certain number of points.

The points transaction involves two users: the individual user and the merchant; the points transaction is a single-step process: the merchant racks up the products, the individual user purchases the products to generate an order, and the transaction is executed immediately after the total amount of the order is paid. The specific implementation process and class design is as follows.

1) Merchants on the shelves, through the product class, productList class to manage all the merchant's products, parameters for the merchant id named file name, automatically read the stored product information, Flags mark the deletion of goods or modify the goods, add goods to the end of the file to add a line, (destructor) when Flags is true, automatically rewrite the file. It is also possible to query the existence of an item based on its item id, as well as returning the item class.

2) merchant to buy products, implemented by the pointTransact class, pointTransact contains generate transaction order apply_a_product_order and pay_for_productOrder member functions; apply_a_product_order generates transaction order, the parameters are the order The reference to the ProductOrder class object that represents the order information and operation; the ProductOrder class object is then passed to the pay_for_productOrder pay order total function, which calls the member function of the ProductOrder class object that implements the payment function.