EOOP - preliminary project

Date: 21.04.2023 **Semester:** Spring 2023

Author and Group: Michał Sarosiek (324842) Group 105

Subject (Keyword): Second hand shop

Description of the project

Overview of the project

A data bank stores information about the crucial aspects of shop management. It contains a list of shops, their inventory, employees and customers.
It provides a wide range of functionality such as:

- management of multiple shops
- storing data specific to each category (money, names, IDs, salary)
- general operations (selling items, management of lists, modifying shop data)
- various operations on employees (hiring, firing, raising salary, modifying personal information)
- various operations on customers (adding, removing, modifying data)
- various operations on the inventory (adding items, removing items, modifying data)
- presenting all data

It is additionally equipped with preventative measures against illegal data operations.

Class and data structures overview

There exist the following classes: ShopNetwork, SecondHandShop, Person, Employee, Customer, Item. There also exist two-directional lists which can store an unlimited amount of data. They correspond to each category of the ShopNetwork (shopList, employeeList, customerList, inventory).

- ShopNetwork – the general class
  -It is the class which interacts with the user. It offers a wide range of operations on the other classes. It contains a list of all employees, customers and shops in the network. It has the ability to modify these lists and pass data to the other classes. It offers a variety of printing operations. It also identifies and corrects incorrect arguments given by the user.
- SecondHandShop
  -It serves as a way to navigate through all the functionality of a particular shop. It contains the lists specific to each shop: customerList, employeeList and the inventory. It has the ability to add and remove elements from them. It can also perform operations such as: modifying all the data within classes assigned to it and printing it. It additionally stores the name of the shop which serves as a way to identify it.
- Person
  -It is the base class of Employee and Customer. It contains a name, personID and a list of pointers to SecondHandShop. It is equipped with the ability to easily create unique IDs for those classes. The IDs are always incremented by 1. ID serves as a primary method of identification.
- Employee

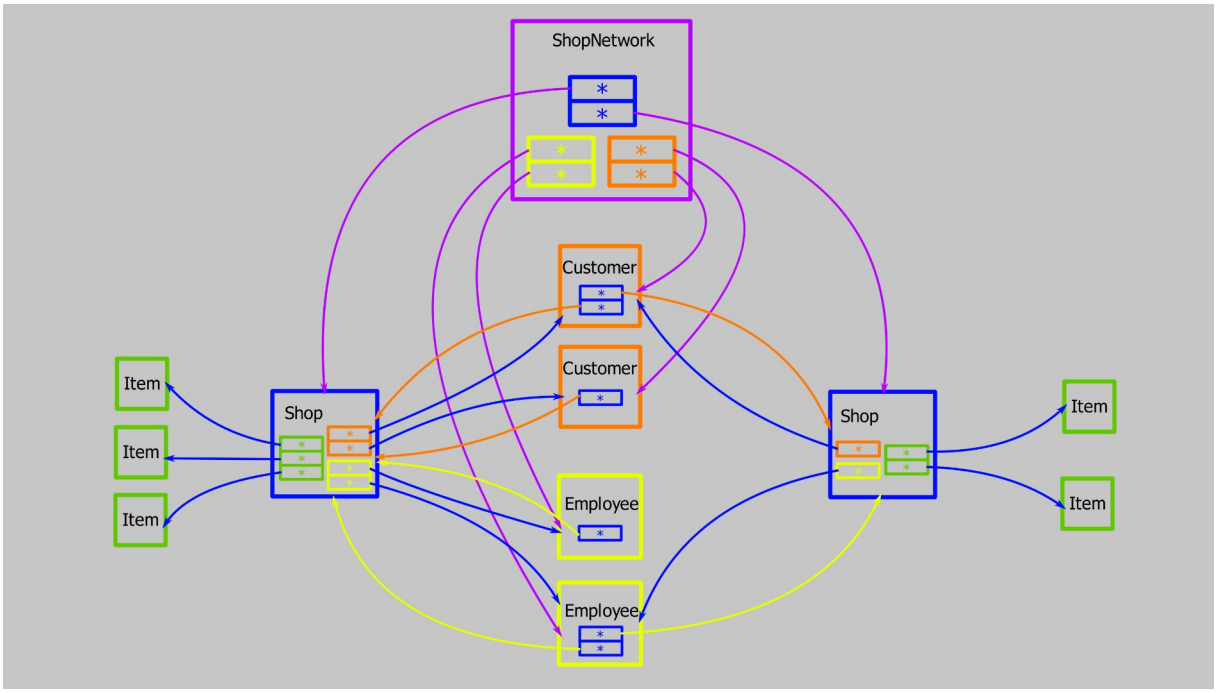
-Inherited from the Person class. It adds salary to the arguments from the Person class. The SecondHandShop uses this class and data contained within to properly pay salaries. Such operations result in the reduction of a particular shop's money. A single employee can be assigned to multiple shops.

- Customer
  -Inherited from the Person class. It adds money to the arguments from the Person class. Customers can buy items contained in the inventory. Such operations have an effect on the money of the shop and the customer. A single customer can be assigned to multiple shops.
- Item
  -it is the class which contains data about a particular item (itemID, itemName, price, quantity). The inventory (a list of Item objects) provides customers with the knowledge of items for purchase. This data is exclusive to every shop.

Restrictions, limits, assumptions

1. Money, salary and raise percentage cannot be lower than 0 or contain decimal digits lower than 0.01. Numbers that don't meet the 2nd condition are rounded to the closest viable number.
2. Operations which would result in a negative money balance are not carried out (e.g. paying salary, buying items).
3. Customers and employees always have unique IDs. Each consecutive ID is incremented by 1.
4. Shop names have to be unique.
5. Each Employee can work at max 3 shops. It can be adjusted upon shop creation.
6. Customers can't buy items from shops they are not assigned to.
7. Customers can't buy items if there are no employees assigned to the shop
8. Two items from the same shop can't have the same itemID. Adding items with the same itemID and name results in the increase in quantity and change in price.
9. Item quantity can't be lower than 0 (ignores operations which would result in such a state). If quantity drops to 0, item is removed from the list.
10. Item price can't be lower than 0.01 or contain decimal digits lower than 0.01.

Case study (a memory map)



Declaration of the classes

```
// the general class; responsible for the interaction with user
class ShopNetwork{
    int maximumShopsForEmployee;
    list<Employee*> employeeList;
     /*list of pointers to Employee class; unlimited list
       of all Employees in the network*/
    list<Customer*> customerList;
     /*list of pointers to Customer class; unlimited list
       of all Customers in the network*/
    list<SecondHandShop*> shopList;
     /*list of pointers to SecondHandShop class; unlimited list
       of all Shops in the network*/
    Customer* findCustomer(int customerID);
     /*checks if customer exists; if yes - returns a pointer to customer;
       if not - returns nullptr*/
    Employee* findEmployee(int employeeID);
     /*checks if employee exists; if yes - returns a pointer to employee;
      if not - returns nullptr*/
    SecondHandShop* findShop(string shopName);
     //checks if shop exists; if yes - returns a pointer to shop; if not - returns nullptr
    void FixMoney(float& money);
     /*a helping method - changes money to the correct amount if decimal digits
       are lower than 0.01 (rounded to the closest viable number);
       does nothing if it's already correct*/
public:
    ShopNetwork(int maximumShopsForEmployee=3);
     /*constructor; maximumShopsForEmployee is 3 by default
      -it's the number of shops an employee can be assigned to*/
    ~ShopNetwork();
     //destructor
    bool addShop(string shopName, float money=0);
     /*creates a shop and adds it to the shop list; money is 0 by default; uses fixMoney;
       if shop of this name already exists or money is lower than 0 it fails
       and notifies user; returns 1 if this method was successful and 0 if not*/
    bool removeShop(string shopName);
     /*removes a shop from the shop list and removes it and its inventory from memory;
       removes all pointers to this shop from customers and employees;
       fails and notifies user if shop doesn't exist;
       returns 1 if this method was successful and 0 if not*/
    int addEmployee(string name, float salary);
     /*Creates an Employee and adds it to the general employee list; returns id
       of the created employee; fails and notifies user if salary is lower than 0;
       uses fixMoney; Returns 0 if method was unsuccessful*/
    void assignEmployee(string shopName, int employeeID);
     /*Assigns a previously created Employee to a shop; Employee can be in max 3 shops;
       if attempted to assign more, the operation fails and notifies user;
       the same happens if employee or shop doesn't exist*/
    void unassignEmployee(string shopName, int employeeID)
     /*removes an employee from a shop employee list;
       fails and notifies user if parameters are wrong*/
```

```
bool removeEmployee(int employeeID);
 /*removes an Employee from all Employee lists and removes from memory;
   fails and notifies user if employee doesn't exist
   returns 1 if this method was successful and 0 if not*/
int addCustomer(string name, float money=0);
 /*creates a Customer and adds it to the general customer list; money defaults to 0;
   fails and notifies user if money is lower than 0; uses fixMoney
   returns id of the created customer. Returns 0 if method was unsuccessful*/
void assignCustomer(string shopName, int customerID);
 /*assigns a previously created customer to a shop;
   amount of shops assigned is unlimited;
   fails and notifies user if customer/shop doesn't exist*/
void unassignCustomer(string shopName, int customerID);
 /*removes a customer from a shop customer list;
   fails and notifies user if parameters are wrong;*/
bool removeCustomer(int customerID);
 /*removes a customer from all customer lists and removes from memory;
   fails and notifies user if customer doesn't exist;
   returns 1 if this method was successful and 0 if not*/
void paySalary(string shopName);
 /*pays Salary to all employees of a certain shop;
   fails and notifies user if shop doesn't exist*/
void paySalary(string shopName, int employeeID);
 /*pays salary from a shop to an employee;
   fails and notifies user if parameters are wrong*/
void changeSalary(int employeeID, int newSalary);
 /*changes the salary of an employee to the new amount;
   fails and notifies user if number is negative;*/
void giveRaise(int employeeID, float percentage);
 /*gives a raise to a certain employee based on the given percentage;
  fails and notifies user if percentage is lower than 0;
  uses fixMoney on percentage and the resulting salary*/
bool addItem(string shopName, int itemID, string itemName, float price, int quantity);
 /*adds an item to the inventory of a certain shop; fails and notifies user
   if itemID or quantity are invalid; uses fixMoney on price; itemID is unique;
   Adding items with the same itemID and name results in the increase in quantity
   and change in price; returns 1 if this method was successful and 0 if not*/
bool removeItem(string shopName, int itemID, int quantity=1);
 /*removes an item from a certain shop; quantity is 1 by default, can't remove more
   than the current number, if attempted fails and notifies user;
   if quantity reaches 0, item gets removed from the list;
   returns 1 if this method was successful and 0 if not*/
bool buyItem(string shopName, int customerID, int itemID, int quantity=1);
 /*lets a customer buy an item from a certain shop - deducts the item's price
   from customer's money; quantity is 1 by default, can't remove more
   than the current number, if attempted notifies user and stops;
   such an item has to exist in the Inventory of the given shop;
   fails and notifies user if given data is incorrect;
   returns 1 if this method was successful and 0 if not*/
void changeItemPrice(string shopName, int itemID, float newPrice)
 /*changes the item price in a certain shop; fails and notifies user if:
   newPrice is lower than 0.01 ir item/shop doesn't exist; uses fixMoney*/
void changeMoney(int customerID, float newMoney);
 /*Changes the stored amount of customer's money; fails and notifies user if newMoney
   is lower than 0 or customer doesn't exist; uses fixMoney*/
void changeMoney(string shopName, float newMoney);
 /*Changes the stored amount of shop's money; notifies the user if newMoney
   is lower than 0 or customer doesn't exist and stops; uses fixMoney*/
void updateMoney(int customerID, float amount);
 /*Lowers or increases the customer's money based on the amount;
   amount can be negative; uses fixMoney; fails and notifies user
   if resulting money balance would be negative or customer doesn't exist*/
void updateMoney(string shopName, float amount);
 /*Lowers or increases the shop's money based on the amount;
   amount can be negative; uses fixMoney; fails and notifies user
   if resulting money balance would be negative or shop doesn't exist*/
void changeShopName(string shopName, string newShopName);
 /*changes the shop name; fails and notifies user
   if shop doesn't exist or name is already taken*/
void printShopList();
 //prints the list of shops
void printShopListOfEmployee(int employeeID);
 /*prints the list of shops an employee belongs to;
  fails and notifies user if employee doesn't exist*/
```

```
    void printShopListOfCustomer(int customerID);
     /*prints the list of shops a customer belongs to;
      fails and notifies user if customer doesn't exist*/
    void printEmployeeList();
     //prints the list of all employees in the network
    void printEmployeeList(string shopName);
     /*prints the list of all employees in a particular shop;
      fails and notifies user if shop doesn't exist*/
    void printCustomerList();
     //prints the list of all customers in the network
    void printCustomerList(string shopName);
     /*prints the list of all customers in a particular shop;
      fails and notifies user if shop doesn't exist*/
    void printInventory(string shopName);
     //prints the inventory; fails and notifies user if shop doesn't exist
    void printAllShopData();
     //prints all shop data (all shops with their employees, customers and items)
    void printAllShopData(string shopName);
     /*prints all shop data for a particular shop;
       fails and notifies user if shop doesn't exist*/
    void printAllNetworkData();
     //prints all network data (full list of employees, Customers and shops)
};
//shop data
class SecondHandShop{
    string shopName;
     //Name used as identification; unique
    float money;
     //cannot be lower than 0 or contain decimal digits lower than 0.01;
    list<Employee*> employeeList;
     //list of pointers to Employee class
    list<Customer*> customerList;
     //list of pointers to Customer class
    list<Item*> inventory;
     //list of pointers to Item class
    Customer* findCustomer(int customerID);
     /*checks if customer is assigned to this shop; if yes
      - returns a pointer to customer; if not - returns nullptr*/
    Employee* findEmployee(int employeeID);
     /*checks if employee is assigned to this shop; if yes
      - returns a pointer to employee; if not - returns nullptr*/
public:
    SecondHandShop(string shopName);
     //constructor
    ~SecondHandShop();
     /*destructor; properly deallocates memory;
       If called deletes all objects in the lists
       unless they have pointers to other shops*/
    string getName() const;
     //returns shopName
    float getMoney() const;
     //returns money
    void changeShopName(string newShopName);
     //changes the shop name
    void hireEmployee(Employee* emp);
     /*adds an employee to the list of employees;
      does nothing if Employee* is nullptr*/
    bool fireEmployee(int employeeID);
     /*removes an employee from the list of employees;
      returns 0 if employee isn't in the list, 1 if successful*/
    bool paySalary();
     /*pays salary to every shop employee; if this operation would result
       in negative money balance it fails and returns 0; returns 1 if successful*/
    bool paySalary(int employeeID);
     /*pays salary to a specific employee; if this operation would result
       in negative money balance or employee isn't in the list it fails and returns 0;
       returns 1 if successful*/
    bool changeSalary(int employeeID, float newSalary);
     /*changes the salary for a specific employee; this method replaces the old salary
      with the new one; returns 0 if employee isn't in the list, 1 if successful*/
    bool giveRaise(int employeeID, float percentage);
     /*raises an employee's salary by a specified percentage;
       returns 0 if employee isn't in the list, 1 if successful*/
```

```
    void addCustomer(Customer* cust);
     /*adds a customer to the list of customers; does nothing if Customer* is nullptr*/
    bool removeCustomer(int customerID);
     /*removes a customer from the list of customers;
       returns 0 if customer isn't in the list, 1 if successful*/
    bool addItem(int itemID, string itemName, float price, int quantity);
     /*adds an item to the inventory; checks itemID if invalid returns 0;
       itemID is unique; Adding items with the same itemID and name results
       in the increase in quantity and change in price*/
    bool removeItem(int itemID, int quantity=1);
     /*quantity is 1 by default, can't remove more than the current number,
       if attempted fails and returns 0; if quantity reaches 0, item gets removed
       from the list; returns 1 if method was successful;*/
    void changeMoney(float newMoney);
     //Changes the stored amount of money;
    bool updateMoney(float amount);
     /*Lowers or increases the amount of money based on the amount;
       if this operation would result in negative money balance it fails
       and returns 0; returns 1 if successful*/
    void printEmployeeList();
     //prints the list of employees
    void printCustomerList();
     //prints the list of customers
    void printInventory();
     //prints the inventory
    void printAll();
     //prints all data
};

class Person{
    static int id;
     //serves as a counter for personID; increments by 1 after Person creation
    int personID;
     //ID of a person; unique - increments of 1; serves as a way to identify
    string name;
     //the name of the person; not unique
    list<SecondHandShop*> shopList;
     //list of pointers to the SecondHandShop class
public:
    Person(string name);
     //constructor;
    ~Person();
     //destructor
    int getPersonID() const;
     //returns personID
    string getName() const;
     //returns name
    void updateName(string newName);
     //updates the name
    void joinShop(SecondHandShop* shop);
     //adds a pointer to shop; does nothing if shop=nullptr
    void leaveShop(string shopName);
     //removes a pointer to shop; does nothing if shop doesn't exist
    void print();
     //prints person data
    void printShopNames();
     //prints names of all shops person belongs to
};
//customer data;
class Customer: public Person{
    float money;
     //can't be lower than 0; decimal digits can't be lower than 0.01
public:
    Customer(string name, float money=0);
     //constructor; money defaults to 0;
    ~Customer();
     //destructor
    float getMoney() const;
     //returns money
```

```
    void changeMoney(float money);
     /*Changes the stored amount of money;*/
    bool updateMoney(float amount);
     /*Lowers or increases the amount of money based on the amount;
       if this operation would result in negative money balance it fails
       and returns 0; returns 1 if successful*/
    void print();
     //prints customer data
};

//Employee data
class Employee: public Person{
    float salary;
     /*can't be lower than 0; only accepts values
       with decimal digits not lower than 0.01*/
public:
    Employee(string name, float salary=0);
     //constructor; salary=0 by default
    ~Employee();
     //destructor; checks validity, notifies user if not
    float getSalary() const;
     //returns salary
    void changeSalary(float newSalary);
     //changes salary
    void print();
     //prints employee data
};


//Item data;items are exclusive to every shop
class Item{
    int itemID;
     //can't be lower than 1; unique to a specific shop
    string itemName;
    float price;
     //can't be lower than 0.01; can't contain decimal digits lower than 0.01
    int quantity;
     //can't be lower than 1; if set to 0 then Item is removed
public:
    Item(int itemID, string itemName, float price, int quantity);
     //constructor
    ~Item();
     //destructor
    int getItemID() const;
     //returns itemID
    string geItemName() const;
     //returns itemName
    float getPrice() const;
     //returns price
    int getQuantity() const;
     //returns quantity
    void changePrice(int newPrice);
     //changes price;
    void changeQuantity(int quantity);
     //changes quantity
    bool updateQuantity(int amount);
     /*Lowers or increases quantity based on the amount; returns 0 if
      this operation would result in negative quantity; returns 1 if successful*/
    void print();
     //prints item data
};
```

Functional test cases

```
bool addShop(string shopName, float money=0);
 /*
 adding a shop when:
 -everything is correct
 -R1: money is negative
 -R1: money decimal digits are lower than 0.01
 -R4: shop with this name already exists
 -adding multiple shops
 -adding a shop after removal
 */
bool removeShop(string shopName);
 /*
 -removing a shop
 -removing a shop which doesn't exist
 -removing a shop twice
 -removing a shop which has Employees, Customers and Items assigned
 */
int addEmployee(string name, float salary);
 /*
 -adding an employee
 -adding multiple employees
 -adding after removal
 -R1: salary is negative
 -R1: salary decimal digits are lower than 0.01
 */
void assignEmployee(string shopName, int  employeeID);
 /*
 -assigning an employee to a shop
 -assigning to a shop which doesn't exist
 -assigning to multiple shops
 -assigning to a shop already assigned (should do nothing)
 -R5: assigning an employee to a 4th shop
 */
void unassignEmployee(string shopName, int employeeID)
 /*
 -unassigning an employee
 -unassigning an employee twice
 -unassigning an employee who doesn't exist
 -unassigning an employee who's not assigned to the shop
 -unassigning an employee from a shop which doesn't exist
 -unassigning when employee nor shop exist
 */
bool removeEmployee(int employeeID);
 /*
 removing when:
 -employee exists
 -employee doesn't exist
 -R3: employee twice
 -employee is assigned to a shop
 -employee is assigned to multiple shops
 */
```

```
int addCustomer(string name, float money);
 /*
 adding when:
 -no customers exist
 -multiple customers exist
 -after removal
 -R1: money is negative
 -R1: money contains decimal digits which are lower than 0.01
 */
void assignCustomer(string shopName, int customerID);
 /*
 assigning when:
 -everything correct
 -shop doesn't exist
 -customer does't exist
 -shop and customer don't exist
 -customer already assigned to a shop
 -customer already assigned to multiple shops
 -customer already assigned to the shop we try to assign to
 -after unassigning
 */
void unassignCustomer(string shopName, int customerID);
 /*
 unassign when:
 -everything correct
 -shop doesn't exist
 -customer does't exist
 -shop and customer don't exist
 -customer not assigned
 -customer assigned to a different shop
 -unassigning twice
 */
bool removeCustomer(int customerID);
 /*
 remove a customer when:
 -everything correct
 -customer doesn't exist
 -R3: customer twice
 -customer is assigned to a shop
 -customer is assigned to multiple shops
 */
void paySalary(string shopName);
 /*
 Paying salary when:
 -everything correct
 -shop doesn't exist
 -shop has no employees
 -shop has 1 employee
 -shop has multiple employees
 -R2: operation would result in a negative money balance
 */
void paySalary(string shopName, int employeeID);
 /*
 Pay salary when:
 -everything correct
 -shop doesn't exist
 -employee doesn't exist
 -shop has no employees
 -shop has multiple employees
 -R2: operation would result in a negative money balance
 */
```

```
void changeSalary(int employeeID, int newSalary);
 /*
 changing salary when:
 -everything is correct
 -employee doesn't exist
 -R1: salary lower than 0
 -R1: salary has a decimal digit lower than 0.01
 */
void giveRaise(int employeeID, float percentage);
 /*
 Giving a raise when:
 -everything is correct
 -employee doesn't exist
 -R1: percentage lower than 0
 -R1: percentage has a decimal digit lower than 0.01
 */
bool addItem(string shopName, int itemID, string itemName, float price, int quantity);
 /*
 adding an item when:
 -everything is correct
 -shop doesn't exist
 -after removal
 -R10: price is lower than 0.01
 -R10: price has a decimal digit lower than 0.01
 -R8: quantity is lower than 1
 -R8: itemID is already in the list, itemName is different
 -R8: itemId is already in the list, itemName is the same
 */
bool removeItem(string shopName, int itemID, int quantity);
 /*
 removing an item when:
 -everything is correct
 -shop doesn't exist
 -removing twice
 -item doesn't exist
 -quantity is lower than 1
 -R9: operation would result in quantity lower than 0
 -R9: operation would result in quantity equal to 0
 */
bool buyItem(string shopName, int itemID, int quantity, int customerID);
 /*
 Buying an item when:
 -everything is correct
 -shop doesn't exist
 -item doesn't exist
 -customer doesn't exist
 -buying twice
 -R6: customer isn't assigned to the shop
 -R7: There are no employees assigned to the shop
 -quantity is lower than 1
 -R9: operation would result in quantity lower than 0
 -R9: operation would result in quantity equal to 0
 -R2: operation would result in a negative money balance
 */
void changeItemPrice(string shopName, int itemID, float newPrice)
 /*
 changing item price when:
 -everything is correct
 -shop doesn't exist
 -shop doesn't contain the item
 -R10: newPrice is lower than 0.01
 -R10: newPrice has a decimal digit lower than 0.01
 */
```

```
void changeMoney(int customerID, float newMoney);
 /*
 changing money when:
 -everything is correct
 -customer doesn't exist
 -R1: newMoney is negative
 -R1: newMoney decimal digits are lower than 0.01
 */
void changeMoney(string shopName, float newMoney);
 /*
 changing money when:
 -everything is correct
 -shop doesn't exist
 -R1: newMoney is negative
 -R1: newMoney decimal digits are lower than 0.01
 */
void updateMoney(int customerID, float amount);
 /*
 updating money when:
 -everything is correct, amount is positive
 -customer doesn't exist
 -amount is negative and balance doesn't end up negative (should work)
 -R2: amount is negative and balance would end up negative
 -R1: amount decimal digits are lower than 0.01
 */
void updateMoney(string shopName, float amount);
 /*
 updating money when:
 -everything is correct, amount is positive
 -shop doesn't exist
 -amount is negative and balance doesn't end up negative (should work)
 -R2: amount is negative and balance would end up negative
 -R1: amount decimal digits are lower than 0.01
 */
void changeShopName(string shopName, string newShopName);
 /*
 changing shopName when:
 -everything is correct
 -shop doesn't exist
 -new name is already taken
 */
void printShopList();
 /*
 printing when:
 -no shops
 -one shop
 -multiple shops
 -after shop removal
 */
void printShopListOfEmployee(int employeeID);
 /*
 printing when:
 -employee exists
 -employee doesn't exist
 -no shops assigned
 -one shop assigned
 -multiple shops assigned
 -after unassigning a shop
 */
```

```
void printShopListOfCustomer(int customerID);
 /*
 printing when:
 -customer exists
 -customer doesn't exist
 -no shops assigned
 -one shop assigned
 -multiple shops assigned
 -after unassigning a shop
 */
void printEmployeeList();
 /*
 printing when:
 -no employees
 -one employee
 -multiple employees
 -after employee removal
 */
void printEmployeeList(string shopName);
 /*
 printing when:
 -no employees assigned
 -one employee assigned
 -multiple employees assigned
 -after unassigning an employee
 */
void printCustomerList();
 /*
 printing when:
 -no customers
 -one customer
 -multiple customers
 -after customer removal
 */
void printCustomerList(string shopName);
 /*
 printing when:
 -no customers assigned
 -one customer assigned
 -multiple customers assigned
 -after unassigning a customer
 */
void printInventory(string shopName);
 /*
 printing when:
 -no items
 -one item
 -multiple items
 -after item removal
 */
void printAllShopData();
 /*
 -printing (it utilizes previous methods)
 */
void printAllShopData(string shopName);
 /*
 -shop doesn't exist
 -shop exists (it utilizes previous methods)
 */
void printAllNetworkData();
 /*
 -printing (it utilizes previous methods)
 */
```