Padaria Baker

Tema 4 – Gestão de Padarias

Projeto realizado no âmbito da UC de Algoritmos e Estruturas de dados do 2.º ano do MIEIC da FEUP por:

- Adelaide Santos – up201907487 (1/3)
- Ângela Coelho – up201907549 (1/3)
- Bruno Mendes - up201906166 (1/3)

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Parte I

Desenvolver um programa capaz de gerir uma rede de padarias, permitindo aos clientes realizar encomendas remotamente e ir levantá-las à loja pretendida.

Parte II

A partir da solução desenvolvida na parte I, desenvolver um programa em que os produtos da loja estão guardados numa árvore binária de pesquisa, os trabalhadores num tabela de dispersão e as encomendas numa fila de prioridade.

PROBLEMA

Solução

- Reestruturar o programa desenvolvido na parte I e utilizar estruturas de dados não lineares (filas de prioridade, tabelas de dispersão e BSTs) de modo a que a informação seja guardada de forma mais eficiente.

- Implementar código com base em operações de CRUD (Create, Read, Update & Delete).

- Implementar listagens totais ou parciais com critérios a definir pelo utilizador (não aplicável a filas de prioridade).

Algoritmos relevantes e operadores associados às estruturas de dados não lineares

- find
- replace (c++ 20)
- Operador () : HT
- Operador == : HT, BST
- Operador < : PQ, BST

HT – Tabela de dispersão
PQ – Fila de prioridade
BST – Árvore binária de pesquisa

```
void WorkerManager::remove(Worker *worker) {
    auto position = _workers.find(worker);
    if(position == _workers.end()) throw PersonDoesNotExist(worker->getName(), worker->getTaxId());
    _workers.erase(position);
}
```

```
bool Product::operator<(const Product &rhs) const {
    if (getTimesIncluded() != rhs.getTimesIncluded()){
        return getTimesIncluded() < rhs.getTimesIncluded();
    }
    if (getCategory() != rhs.getCategory()){
        return getCategory() < rhs.getCategory();
    }
    return getName() < rhs.getName();
}
```

```
bool Order::operator<(const Order &o2) const {
    if (wasDelivered() != o2.wasDelivered()){
        return wasDelivered();
    }
    if (_client->getMeanEvaluation() != o2._client->getMeanEvaluation()){
        return _client->getMeanEvaluation() > o2._client->getMeanEvaluation();
    }
    return _client->getNumDiscounts() > o2._client->getNumDiscounts();
}
```

Exemplo: ler e escrever clientes

```cpp
void ClientManager::read(const std::string &path) {
    std::ifstream file(path);
    if(!file) throw FileNotFound(path);

    std::string name, premium;
    unsigned long taxID = Person::DEFAULT_TAX_ID;
    unsigned points = 0;
    Credential credential;

    for(std::string line; getline( &: file,  &: line); ){
        util::stripCarriageReturn( &: line);
        if (line.empty()) continue;

        std::stringstream ss(line);
        ss>>name>>taxID>>premium>>points>>credential.username>>credential.password;

        std::replace(name.begin(), name.end(),  old_value: '-',  new_value: ' ');
        Client* client = add(name, taxID,  premium: premium == "premium", credential);
        client ->setPoints(points);
    }
}
void ClientManager::write(const std::string &path) {
    std::ofstream file(path);
    if(!file) throw FileNotFound(path);

    std::string nameToSave, premiumToSave;
    for(const auto & client: _clients){
        nameToSave = client->getName();
        std::replace(nameToSave.begin(), nameToSave.end(),  old_value: ' ',  new_value: '-');
        premiumToSave=(client->isPremium())? "premium" : "basic";
        file << nameToSave << " " << client->getTaxId() << " " << premiumToSave << " "
        << client->getPoints() << " " << client->getCredential().username << " "
        << client->getCredential().password<<'\n';
    }
}
```

Estrutura de ficheiros

```cpp
std::string Store::read(const std::string &dataFolderPath) {
    try {
        boss.read( path: dataFolderPath + "/boss.txt");
        locationManager.read( path: dataFolderPath + "/locations.txt");
        productManager.read( path: dataFolderPath + "/products.txt");
        clientManager.read( path: dataFolderPath + "/clients.txt");
        workerManager.read( path: dataFolderPath + "/workers.txt");
        orderManager.read( path: dataFolderPath + "/orders.txt");
    }
    catch (std::exception& e){
        return "Import failed!\n" + std::string(e.what());
    }
    return "Import succeeded.";
}

std::string Store::write(const std::string& dataFolderPath) {
    try {
        boss.write( path: dataFolderPath + "/boss.txt");
        locationManager.write( path: dataFolderPath + "/locations.txt");
        productManager.write( path: dataFolderPath + "/products.txt");
        clientManager.write( path: dataFolderPath + "/clients.txt");
        workerManager.write( path: dataFolderPath + "/workers.txt");
        orderManager.write( path: dataFolderPath + "/orders.txt");
    }
    catch (std::exception& e){
        return "Export failed!\n" + std::string(e.what());
    }
    return "Export succeeded.";
}
```

```
Alfredo-Machado 23554 basic 300 machado mymachado
Angela-Coelho 324564 premium 280 angela angela123
Bruno-Mendes 879789 premium 280 bdmendes mendes
```

CRUD

Completas

```
Bread* addBread(std::string name, float price, bool small = true);
```

```
Cake* addCake(std::string name, float price, CakeCategory category = CakeCategory::GENERAL);
```

```
Order* add(Client* client, const std::string& location = Order::DEFAULT_LOCATION, const Date &date = {});
```

CREATE

READ

DELETE

UPDATE

```
void read(const std::string& path);
```

```
Client* getClient(unsigned long taxID) const;
```

```
Product* get(const std::string &name, float price);
```

```
std::string read(const std::string& dataFolderPath);
```

```
void remove(Worker* worker);
```

```
void remove(unsigned long position);
```

```
void removeProduct(Product* product);
```

```
void updateTotalPrice();
```

```
void setPremium(bool premium);
```

```
void addPoints(unsigned points);
```

```
void raiseSalary(float percentage);
```

```
void setDeliverLocation(const std::string& location);
```

Listagens

```
Note: Orders at the top have higher delivery priority.
Delivered orders are kept at the bottom for historical reasons.

   CLIENT              REQUESTED           DELIVERED           LOCATION
1. Angela Coelho       12/12/2020 14:45    Not Yet             Lisboa
2. Angela Coelho       12/06/2020 18:34    19:04 (5 points)    Head Office
```

```
std::priority_queue<OrderEntry> OrderManager::get(Worker *worker) const {
    if (!_workerManager->has(worker)) throw PersonDoesNotExist(worker->getName(), worker->getTaxId());
    std::priority_queue<OrderEntry> filtered, orders = _orders;
    for(; !orders.empty(); orders.pop()){
        const auto& orderEntry = orders.top();
        if(*orderEntry.getOrder()->getWorker() == *worker) filtered.push(orderEntry);
    }
    return filtered;
}
```

```
   NAME              TAX ID        SALARY         TO DELIVER     RATING         LOCATION
1. Senhor            2928348       900.00 euros   0 orders       None yet       Lisboa          <--
2. Osvaldo Ribeiro   32534         1500.00 euros  1 orders       5.00 points    Porto
3. Julia Mendes      2345          2001.00 euros  0 orders       4.00 points    Lisboa          <--
4. Maria Gil         7978          2000.00 euros  1 orders       3.00 points    Lisboa          <--
```

```
tabHWorker WorkerManager::getByLocation(const std::string &location) {
    tabHWorker res;
    for (const auto& w : _workers) if (w->getLocation() == location) res.insert(w);
    return res;
}
```

```
Note: Orders at the top have higher delivery priority.
Delivered orders are kept at the bottom for historical reasons.

   CLIENT              WORKER              REQUESTED           DELIVERED           LOCATION
1. Angela Coelho       Osvaldo Ribeiro     12/12/2020 14:45    Not Yet             Lisboa
2. Alfredo Machado     Maria Gil           12/12/2020 14:45    Not Yet             Lisboa
3. Bruno Mendes        Maria Gil           23/11/2020 12:32    13:02 (3 points)    Porto
4. Angela Coelho       Osvaldo Ribeiro     12/06/2020 18:34    19:04 (5 points)    Head Office
5. Angela Coelho       Julia Mendes        23/11/2020 12:32    13:02 (3 points)    Head Office
6. Alfredo Machado     Julia Mendes        12/06/2020 18:34    19:04 (5 points)    Head Office
```

```
std::priority_queue<OrderEntry> OrderManager::getAll() const {
    return _orders;
}
```

Outras funcionalidades

```
Date::Date(int day, int month, int year, int hour, int minute)
    _time(){
    _time.tm_hour = hour;
    _time.tm_min = minute;
    _time.tm_year = year - 1900;
    _time.tm_mon = month - 1;
    _time.tm_mday = day;
    _time.tm_isdst = -1; // determine daylight saving flag
    if (!isValid()) throw InvalidDate(getCompleteDate());
}
```

Uso da struct tm de C

```
void Date::addMinutes(int minutes) {
    _time.tm_min += minutes;
    std::time_t ntime = mktime(&_time);
    localtime_r(&ntime,&_time);
}
```

```
BAKERY STORE - LOGIN

What is your role on the company?
-> Worker
-> Client
-> Boss

client
```

Sistema de login

Funcionalidade destaque

Get Less Busy Worker

```
Worker* WorkerManager::getLessBusyWorker(const std::string& location) {
    if (_workers.empty()) throw StoreHasNoWorkers();

    auto orderComp = [location](const Worker *worker1, const Worker *worker2) {
        if (worker1->getLocation() == location && worker2->getLocation() != location){
            return true;
        }
        if (worker1->getLocation() != location && worker2->getLocation() == location){
            return false;
        }
        return worker1->getUndeliveredOrders() < worker2->getUndeliveredOrders();
    };
    Worker* lessBusyWorker = *std::min_element( first: _workers.begin(), last: _workers.end(), orderComp);
    if (lessBusyWorker->getUndeliveredOrders() == Worker::MAX_ORDERS_AT_A_TIME) throw AllWorkersAreBusy();
    return lessBusyWorker;
}
```

Principais dificuldades

Eliminação de elementos nas filas de prioridade → Remover a encomenda a ser entregue e voltar a adicionar

```
void OrderManager::deliver(Order *order, int clientEvaluation, bool updatePoints, int deliverDuration) {
    remove(order, updateWorkerOrders: false, destroy: false);
    order->deliver(clientEvaluation, updatePoints, deliverDuration);
    order->getWorker()->removeOrderToDeliver();
    _orders.push(OrderEntry(order));
}
```

Exemplos de execução

```
BAKERY STORE

------------------------------------------------------------
Welcome to the Bakery Store management app.
You start with a blank store. Import data or start fresh.
At any screen, type 'back' to go back.
------------------------------------------------------------

Available commands:
-> import data - import data from files
-> export data - export data to files
-> manage store - enter store management
```

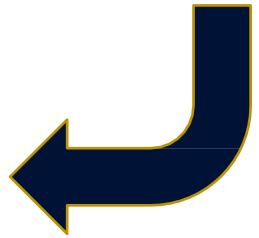
```
IMPORT DATA
--------------------------------
'data' folder path: ../../data

Import succeeded.
Press enter to go back.
```

Exemplos de execução

```
BAKERY STORE - LOGIN

What is your role on the company?
-> Worker
-> Client
-> Boss

client
```

```
BAKERY STORE - LOGIN

-------------------------------------------------------
   NAME                 TAX ID          LOGGED IN
1. Alfredo Machado      23554           No
2. Angela Coelho        324564          No
3. Bruno Mendes         879789          No
-------------------------------------------------------

Available commands:
-> login <index> - login in person's account

login 2
```

```
BAKERY STORE - AUTHENTICATION

-------------------------------------------------------
Dear Angela Coelho, please enter your credentials.
Default is 'client', 'client'.
-------------------------------------------------------

Username: angela

Password: angela123
```



Exemplos de execução

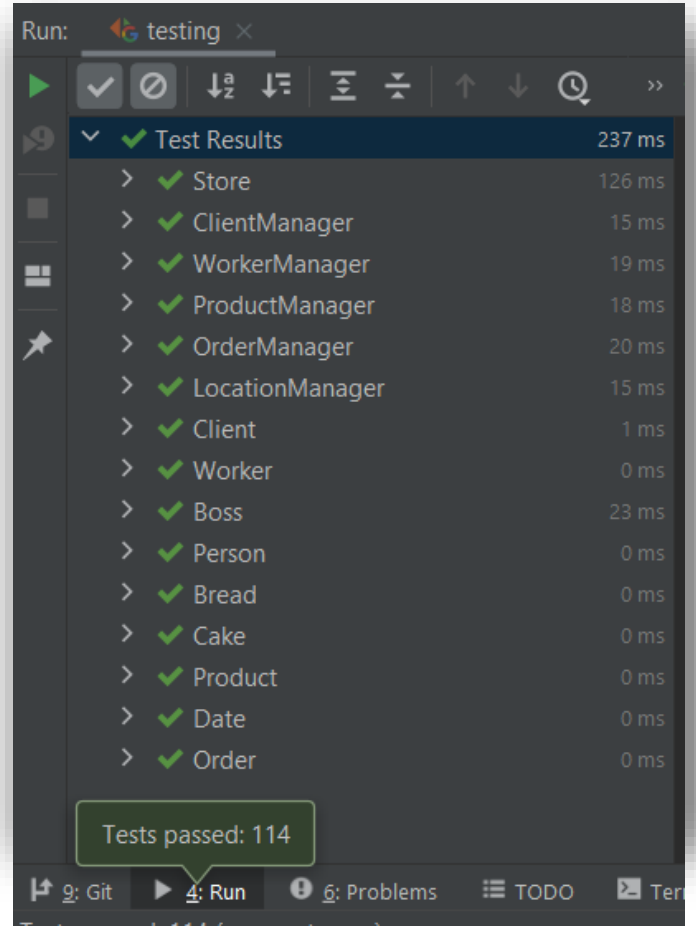
```
BAKERY STORE - ANGELA COELHO'S PERSONAL AREA

------------------------------------------------------------
Angela Coelho
Username: angela
Password: *********

Tax ID: 324564
Status: Premium
Accumulated: 280 points
Benefited from: 2 discounts

Feedback: 4.00 points
------------------------------------------------------------

Available commands:
-> edit account - change personal details
-> new order - request something new
-> manage orders - review and evaluate past requested orders
-> logout - exit and request credential next time

new order
```

```
AVAILABLE STOCK
------------------------------------------------------------------------------------------
   NAME                CATEGORY        UNIT PRICE
1. Bolo de bolacha     Pie             2.20 euros
2. Pao de lo           Sponge          3.20 euros
3. Pao da avo          Big Bread       1.00 euros
4. Regueifa            Big Bread       3.00 euros
5. Bolo crocante       Crunchy Cake    15.00 euros
6. Tarte de morango    Pie             25.00 euros
7. Pao de cereais      Small Bread     0.50 euros
8. Pao de sementes     Small Bread     0.80 euros
9. Bolo esponja        Sponge          30.00 euros
------------------------------------------------------------------------------------------

ORDER DETAILS
------------------------------------------------------------------------------------------
Requested by Angela Coelho (mean evaluation: 4.00; past discounts: 2) on 03/01/2021 19:51
To be delivered by Senhor (who works at Lisboa) at Head Office

No products added

0.00 euros (With discount)
------------------------------------------------------------------------------------------

Available commands:
-> add <product_number> <quantity> - add product from stock
-> remove <product_number> - remove product from order
-> change location - set new deliver location
```

Exemplos de execução

```
AVAILABLE STOCK
-------------------------------------------------------------------------------------------
   NAME                 CATEGORY        UNIT PRICE
1. Bolo de bolacha      Pie             2.20 euros
2. Pao de lo            Sponge          3.20 euros
3. Pao da avo           Big Bread       1.00 euros
4. Regueifa             Big Bread       3.00 euros
5. Tarte de morango     Pie             25.00 euros
6. Pao de cereais       Small Bread     0.50 euros
7. Pao de sementes      Small Bread     0.80 euros
8. Bolo esponja         Sponge          30.00 euros
9. Bolo crocante        Crunchy Cake    15.00 euros
-------------------------------------------------------------------------------------------

ORDER DETAILS
-------------------------------------------------------------------------------------------
Requested by Angela Coelho (mean evaluation: 4.00; past discounts: 2) on 03/01/2021 20:01
To be delivered by Senhor (who works at Lisboa) at Head Office

   Product description  Category        Unit price      Quantity
1. Bolo crocante        Crunchy Cake    15.00 euros     2

28.50 euros (With discount)
-------------------------------------------------------------------------------------------

Available commands:
-> add <product_number> <quantity> - add product from stock
-> remove <product_number> - remove product from order
-> change location - set new deliver location
```

Google Tests



Travis CI