

# 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)



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

# PROBLEMA

# Solução

---

## OOP

Utilização de classes adequadas para a representação das entidades envolvidas.

## Herança e Polimorfismo

Utilização de conceitos de herança e polimorfismo.

## Exceções

Tratamento de exceções que possam ocorrer durante a execução do programa.

## Pesquisa e ordenação

Utilização de algoritmos de pesquisa e ordenação para as diferentes funcionalidades pretendidas.

## Ficheiros

Escrita e leitura de ficheiros para armazenar os dados do programa.



# Algoritmos relevantes

## *algorithm*

Neste trabalho, recorreremos a vários tipos de algoritmos da biblioteca *algorithm* da stl, tais como:



**sort**



**find / find\_if**




**replace (C++ 20)**



**all\_of**




**transform**



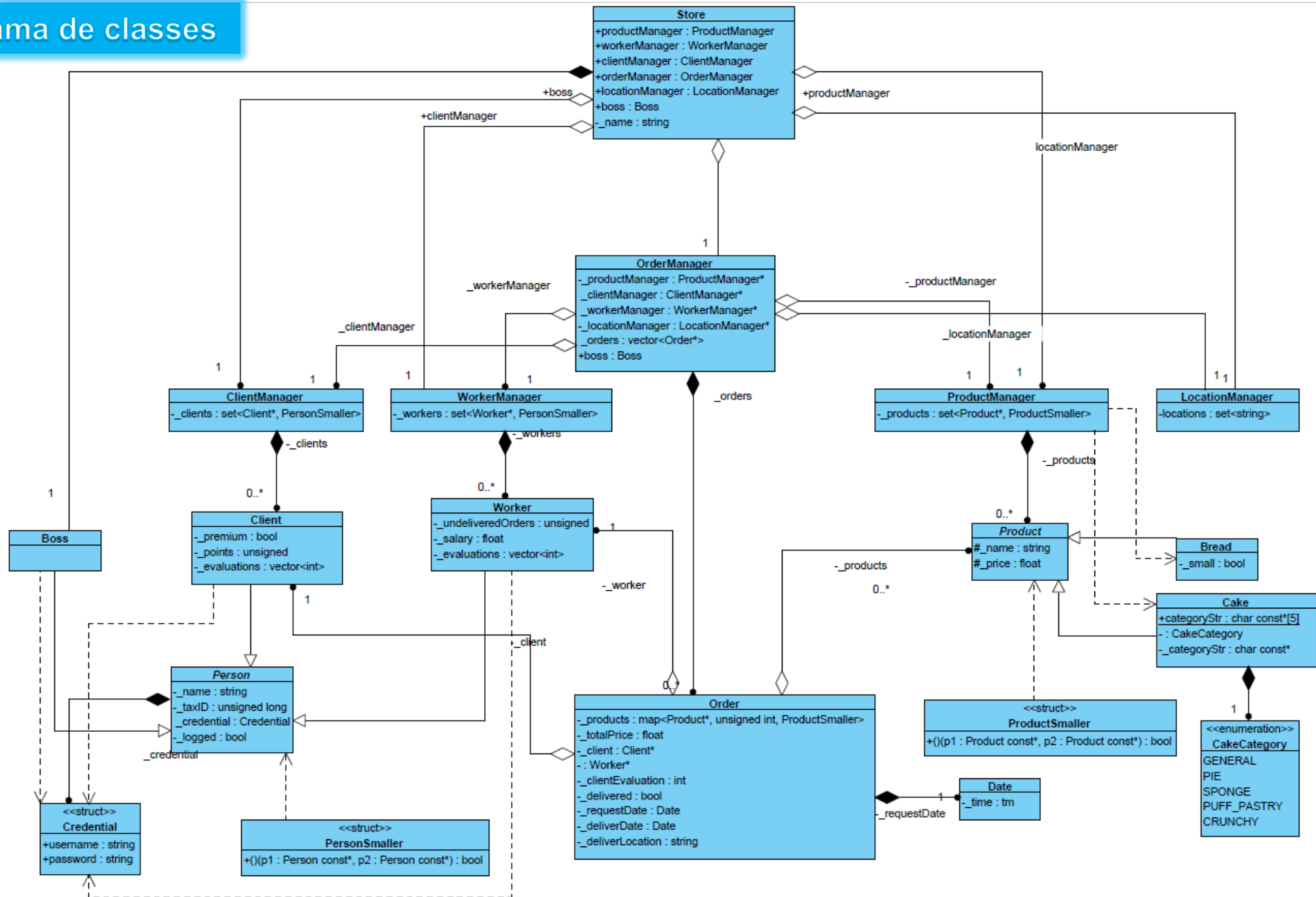
```
for(std::string line; getline( & file, & line); ){
    std::stringstream ss(line);
    ss >> name >> taxID >> salary >> credential.username >> credential.password;
    std::replace(name.begin(), name.end(), old_value: '-', new_value: ' ');
    add(name, taxID, salary, credential);
}
```

```
void OrderManager::sort() {
    std::sort( first: _orders.begin(), last: _orders.end());
}
```

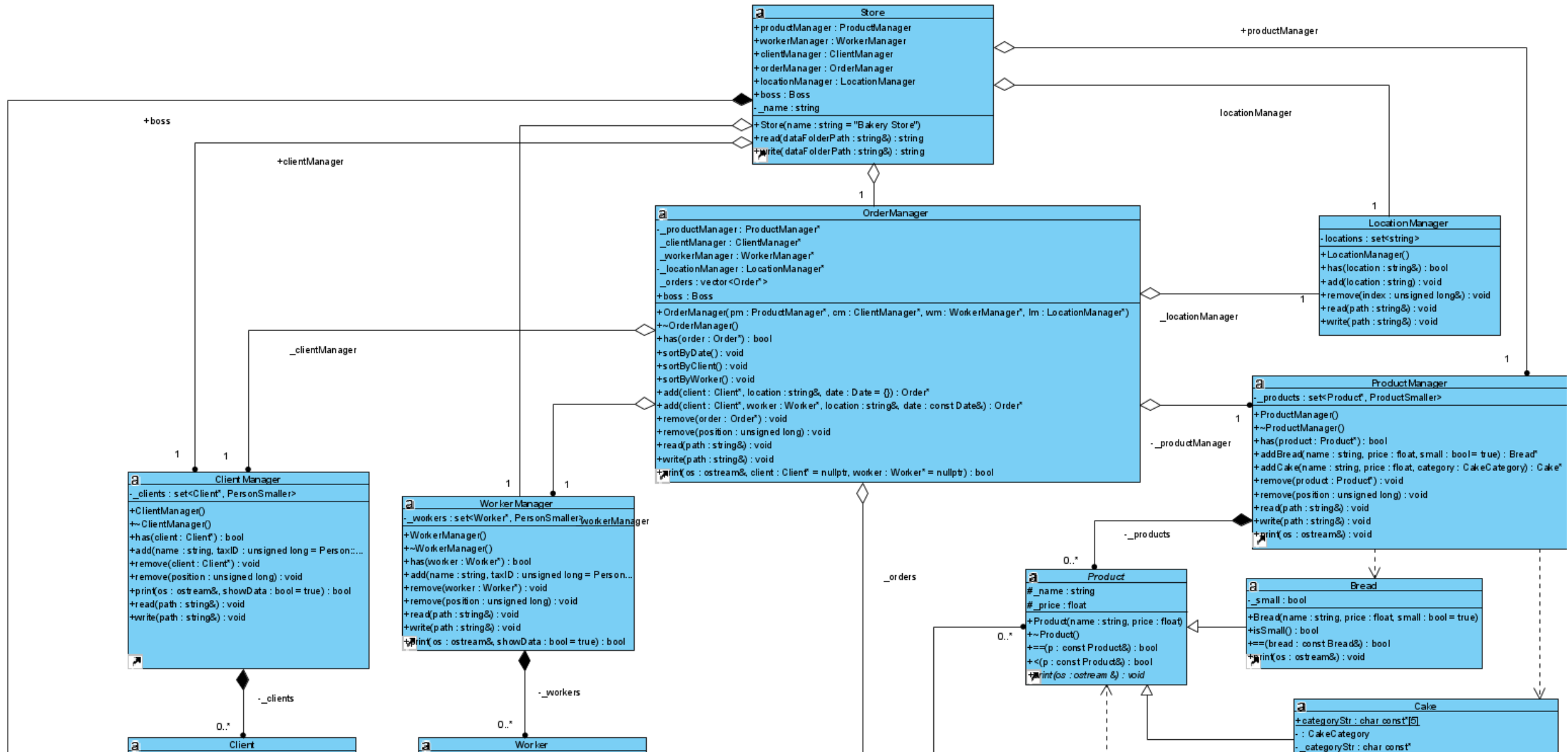


```
void OrderManager::remove(Order *order) {
    auto position = std::find( first: _orders.begin(), last: _orders.end(), order);
    if (position == _orders.end()) throw OrderDoesNotExist();
    if (order->wasDelivered()) throw std::invalid_argument("It's not possible to delete a delivered order");
    order->getWorker()->removeOrderToDeliver();
    _orders.erase(position);
}
```

# Diagrama de classes

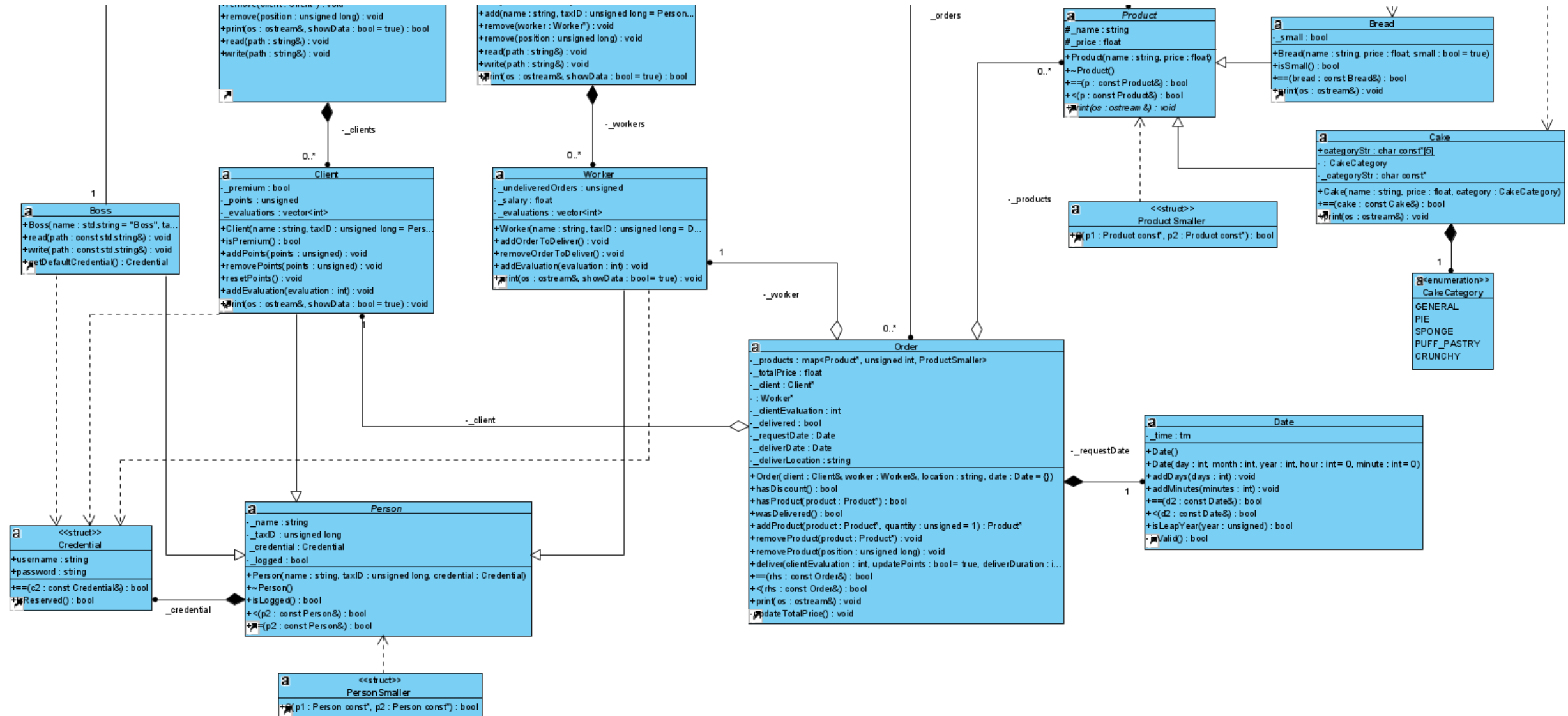


# Diagrama de classes





# Diagrama de classes



## Exemplo: ler e escrever clientes

```
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

```
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
```



## Tratamento de exceções (alguns exemplos)

```
* Class relative to the exception of an invalid person position on some list.
*/
class InvalidPersonPosition : public std::invalid_argument{
public:
    /**
        * Creates a new InvalidPersonPosition exception object.
        *
        * @param position the position
        * @param size the persons list size
        */
    InvalidPersonPosition(unsigned long position, unsigned long size);
};
```

```
StoreHasNoWorkers::StoreHasNoWorkers() :
    std::logic_error("This store has no workers!"){
}
```

```
PersonDoesNotExist::PersonDoesNotExist(const std::string& name, unsigned long taxID) :
    std::logic_error(name + ", with number " + std::to_string(taxID) + ", does not exist!"){
}
```

```
InvalidOrderEvaluation::InvalidOrderEvaluation(int evaluation, const Client &client):
    std::invalid_argument(client.getName() + " gave an invalid evaluation of " + std::to_string(evaluation) + " to this order; should be between 0 and 5!"){
}
```

```
OrderWasAlreadyDelivered::OrderWasAlreadyDelivered(const Client &client, const Worker &worker, const Date &date):
    std::logic_error("The order, requested by " + client.getName() + " on " + date.getCalendarDay() + ", was already delivered by " + worker.getName() + "!"){
}
```

```
void LoginMenu::selectPerson(PersonRole role) {
    bool hasPersons = false;
    printLogo( detail: "Login");

    std::cout << SEPARATOR;
    if (role == PersonRole::WORKER) hasPersons = _store.workerManager.print( & std::cout, showData: false);
    else if (role == PersonRole::CLIENT) hasPersons = _store.clientManager.print( & std::cout, showData: false);
    std::cout << SEPARATOR << "\n";

    if (hasPersons){
        const std::vector<std::string> options = {
            "login <index> - login in person's account"
        };
        printOptions(options);
    }

    for (;;) {
        try {
            std::string input = readCommand();
            if (input == BACK) return;
            else if (hasPersons && validInput1Cmd1ArgDigit(input, cmd: "login")) {
                unsigned long personPosition = std::stoul( str: to_words(input).at( n: 1)) - 1;
                if (role == PersonRole::WORKER) login( person: _store.workerManager.get(personPosition));
                else login( person: _store.clientManager.get(personPosition));
                break;
            }
            else printError();
        }
        catch (std::exception& e){
            std::cout << e.what() << SPACE;
            continue;
        }
    }
}
```

# 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 = {});
```

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

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

```
std::set<Cake*, ProductSmaller> getCakes() const;
```

```
std::set<Bread*, ProductSmaller> getBreads() const;
```

CREATE



READ



DELETE



UPDATE



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

```
void removePoints(unsigned points);
```

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

```
void updateTotalPrice();
```

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

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

```
Worker * setSalary(unsigned position, float salary);
```

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

## 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

## Exemplos de execução

## Pesquisa

```
bool OrderManager::has(Order *order) const {
    auto comp = [order](const Order* o2){
        return *order == *o2;
    };
    return std::find_if( first: _orders.begin(), last: _orders.end(), comp) != _orders.end();
}
```

```
bool WorkerManager::has(Worker *worker) const {
    return _workers.find(worker) != _workers.end();
}
```

```
bool ProductManager::has(Product *product) const {
    return _products.find(product) != _products.end();
}
```

```
bool ClientManager::has(Client *client) const {
    return _clients.find(client) != _clients.end();
}
```

## Listagem

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

```
Order* OrderManager::get(unsigned long position, Client* client, Worker* worker) const {
    std::vector<Order*> filtered;
    if (client != nullptr && worker != nullptr)
        throw std::invalid_argument("Can't choose both worker and client");
    else if (client != nullptr) filtered = get(client);
    else if (worker != nullptr) filtered = get(worker);
    else filtered = getAll();

    if (position >= filtered.size()) throw InvalidOrderPosition(position, filtered.size());
    return filtered.at(position);
}
```

```
Worker* WorkerManager::get(unsigned long position) {
    if (position >= _workers.size()) throw InvalidPersonPosition(position, size: _workers.size());
    auto it = _workers.begin(); std::advance(& it, position);
    return *it;
}
```

## Funcionalidade destaque

### Polimorfismo e Herança entre as classes *Product*, *Bread* e *Cake*

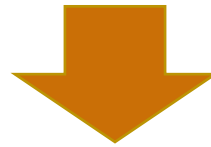
```
bool Product::operator==(const Product &p) const{  
    return _name == p.getName() && _price == p.getPrice();  
}
```

```
bool Cake::operator==(const Cake &cake) const{  
    return getName() == cake.getName() && getCategory() == cake.getCategory()  
    && getPrice() == cake.getPrice();  
}
```

```
bool Bread::operator==(const Bread &bread) const{  
    return _name == bread.getName() && _price == bread.getPrice() && _small == bread.isSmall();  
}
```

## Principais dificuldades

Dificuldade de acesso às outras classes a partir da classe *Order*. Para resolver, criámos Managers para as classes interagirem mais facilmente umas com as outras.



```
Order* OrderManager::add(Client *client, const std::string& location, const Date &date) {  
    if (!_clientManager->has(client)) throw PersonDoesNotExist(client->getName(), client->getTaxId());  
    if (!_locationManager->has(location)) throw LocationDoesNotExist(location);  
    auto* order = new Order(&*_client, &*_workerManager->getLessBusyWorker(), location, date);  
    _orders.push_back(order);  
    return 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. I  
Press enter to go back. ■
```

## Exemplos de execução

### BAKERY STORE - LOGIN

What is your role on the company?

-> Worker

-> Client

-> Boss

client



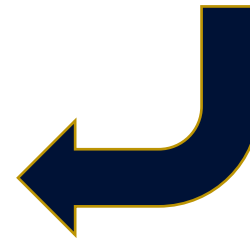
### BAKERY STORE - LOGIN

	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

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



BAKERY STORE - ANGELA COELHO'S PERSONAL AREA - EDIT ORDER

AVAILABLE STOCK

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

-----  
ORDER DETAILS

Requested by Angela Coelho on 20/11/2020 01:25

To be delivered by Julia Mendes at Head Office

No products added

0.00€ (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

add 2 4

## Exemplos de execução

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

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

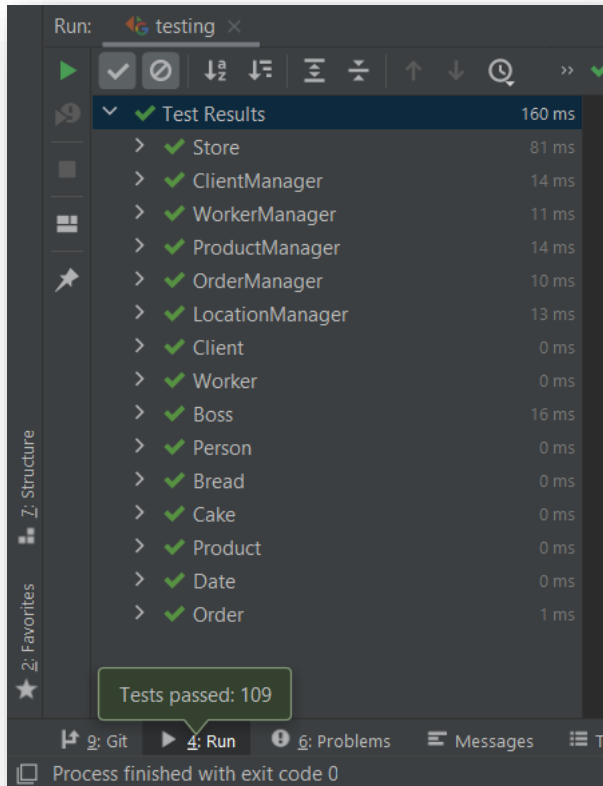
ORDER DETAILS
-----
Requested by Angela Coelho on 20/11/2020 01:25
To be delivered by Julia Mendes at Head Office

      Product description  Category      Unit price      Quantity
1. Bolo de bolacha       Pie           2.20 euros      4

8.36€ (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

build passing codecov 56% codefactor A